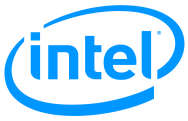
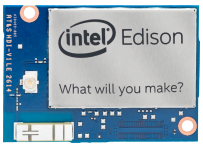
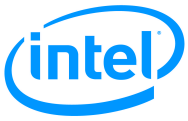




IoT Platform
Intel® Edison Tutorial 4: GPIO and I2C Interfaces



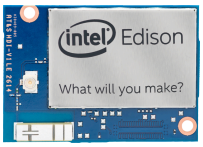
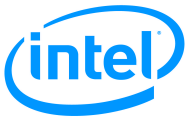
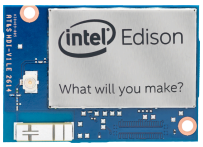


Table of Contents

Revision history		
Version	**Date**	**Comment**
1.0	9/24/2015	Initial release





Introduction

In this tutorial, you will:

1. Be introduced to MRAA library,
2. Learn to access the GPIOs on the Edison in Arduino and C,
3. Learn to set an interrupt on a GPIO in C,
4. Learn to read the analog input on the Edison in Arduino and C, and
5. Learn to implement I2C between the Edison and the Arduino Uno.

Things Needed

1. An Intel Edison with Arduino-compatible breakout,
2. A micro USB cable,
3. A Grove Starter kit, and
4. A PC or Mac

I/O Programming on Intel Edison

With the Arduino-compatible breakout board, I/O programming can be easily done by writing Arduino sketches. However, only one sketch can be uploaded to an Edison board and it is overwritten when a new sketch is uploaded. Instead, we can write a program that is saved in the Edison's internal memory (4GB of eMMC flash) and execute it whenever we need. For instance, you can implement the simple web server example from the previous tutorial in C rather than as an Arduino sketch. Then, write another web server application that controls another pin in a different port. You can simultaneously run these applications and open two browsers and control different components.

As you have discovered in the previous tutorials, the Yocto Embedded Linux image enables C development on the Edison. In this tutorial, we will discuss about how to do I/O programming in C, and we will compare C implementations with the equivalent Arduino sketches. In addition, you can check out the **Appendix** if you want to learn about I/O programming on the Intel Edison in Python and JavaScript.

MRAA Library

I/O programming on the Edison is made easier with a library called MRAA. Libmraa is a C/C++ library with bindings to javascript & python to interface with the IO on Galileo, Edison & other platforms, with a structured and sane API where port names/numbering matches the board that you are on [1]. This library allows developers control low-level communication protocol by high-level language. The Yocto Embedded Linux image includes MRAA library. Let's first check the version of MRAA library.

1. **$ mkdir tutorial4_examples**
2. **$ cd tutorial4_examples**
3. **$ vi check_mraa_version.c**
4. Then, type the following C code.

```c
#include <stdio.h>
#include <mraa.h>

int main()
{
    printf("MRAA version: %s\n", mraa_get_version());
    return 0;
}
```

Figure 1 C Code

5. **$ gcc -lmraa -o check_mraa_version check_mraa_version.c**
   You need to add "-lmraa" in order to include the mraa library.
6. **./check_mraa_version**

GPIO (General Purpose Input/Output)

To demonstrate GPIO access on the Edison, we will try the following LED blink example. In this example, we will compare Arduino sketch and C implementation.

1. Plug an LED into an LED socket. The LED has two legs. The longer leg is the anode (positive) and the shorter leg is the cathode (negative). Connect the longer leg to the socket pin with "+" sign. (Caution: LEDs have polarity.)

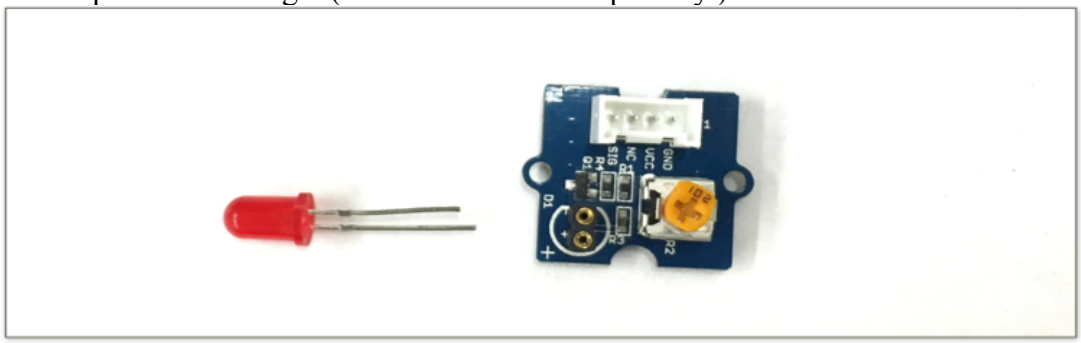


Figure 2 LED

2. Insert the Grove Base Shield into the Edison board and connect other components as shown in the pictures below.

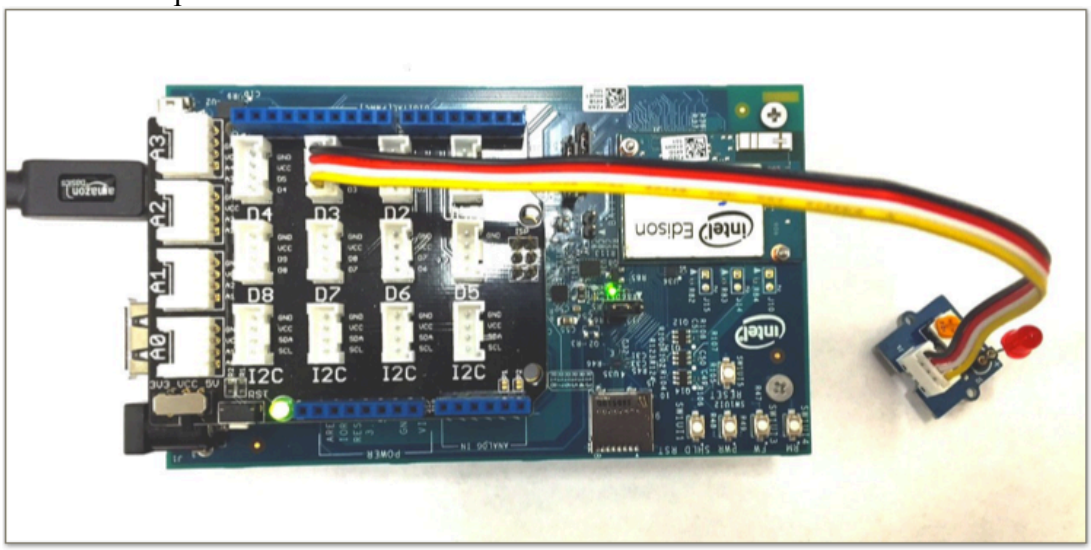


Figure 3 Hardware Setup

3. Upload the following Arduino sketch.

```
//A modified version of File->Examples->01.Basics->Blink

int led = 3;

//the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

//the loop routine runs over and over again forever:
void loop() {
  //turn the LED on (HIGH is the voltage level)
  digitalWrite(led, HIGH);
  // wait for a second
  delay(1000);
  // turn the LED off by making the voltage LOW
  digitalWrite(led, LOW);
  // wait for a second
  delay(1000);
}
```

Figure 4 GPIO Arduino Sketch

4. The LED should start blinking.
5. Open a new sketch File -> New and upload the blank sketch to overwrite the sketch above.
6. SSH into the Edison and navigate to ~/tutorial4_examples.

7. **$ vi blink.c**
8. Type the following C code.

```
#include <signal.h>
#include <mraa/gpio.h>

//a flag for the while loop
sig_atomic_t volatile run_flag = 1;

//signal handler for Ctrl-C
void do_when_interrupted(int sig)
{
     //if an interrupt signal invoked, set the flag to "false"
     if (sig == SIGINT)
          run_flag = 0;
}

int main()
{
     //declare led as a GPIO type
     mraa_gpio_context led;
     //initialize pin 3 (D3) for led
     led = mraa_gpio_init(3);
     //set the direction of the pin as output
     mraa_gpio_dir(led, MRAA_GPIO_OUT);

     signal(SIGINT, do_when_interrupted);

     while (run_flag) {
          //turn on the LED by setting the output as "high"
          mraa_gpio_write(led, 1);
          //wait for 1 second
          sleep(1);
          //turn off the LED by setting the output as "low"
          mraa_gpio_write(led, 0);
          sleep(1);
     }
     //close the GPIO context
     mraa_gpio_close(led);
     return 0;
}
```

Figure 5 GPIO C code

9. **$ gcc -lmraa -o blink blink.c**
10. **$ ./blink**
11. Press **Ctrl-C** to quit.

More information on the MRAA Library's GPIO API can be found at http://iotdk.intel.com/docs/master/mraa/gpio_8h.html

GPIO Interrupt

The previous section introduced how to use a GPIO to control an LED. The C implementation of the blink program includes a signal handler. When the user invokes an interrupt signal by pressing Ctrl-C, do_when_interrupted function is called. If the signal invoked is SIGINT (interrupt signal), run_flag becomes 0. run_flag is the while loop's condition and the program exits the loop if run_flag is 0 (logical false). In this section, we will use the Grove button component to interrupt instead of Ctrl-C.

1. Set up the hardware as shown in the figure below.

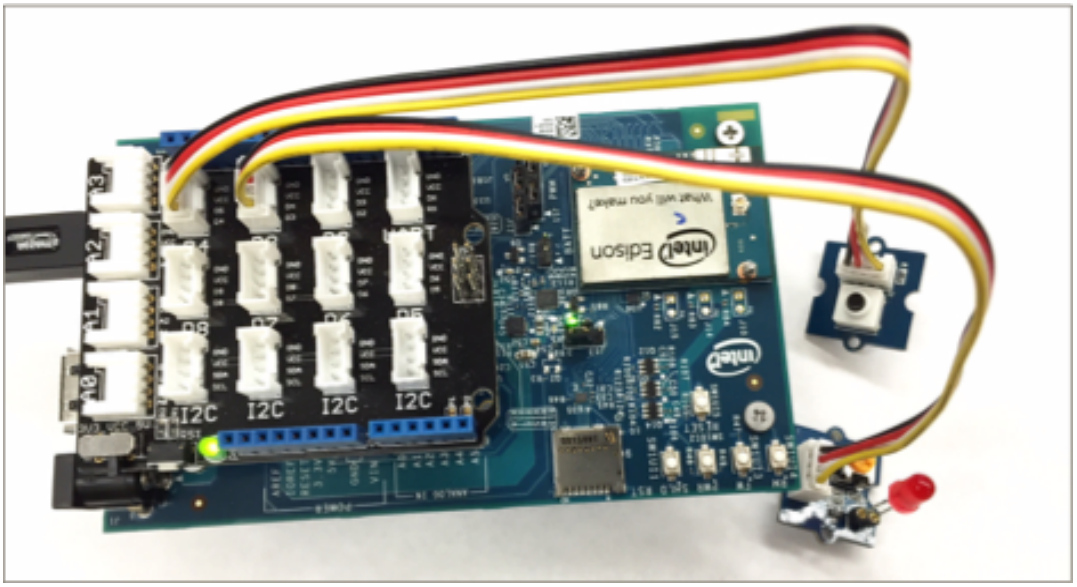
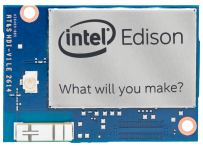


Figure 6 Hardware Setup

2. SSH into the Edison and navigate to ~/tutorial4_examples.
3. $ vi gpio_interrupt.c
4. Type the following C code




```
#include <unistd.h>
#include <mraa/gpio.h>

static volatile int run_flag = 1;

//handler for GPIO interrupt
void do_when_interrupted()
{
      run_flag = 0;
}

int main()
{
      //declare led and button as GPIO contexts
      mraa_gpio_context led, button;
      led = mraa_gpio_init(3);
      //initialize pin 4 for button
      button = mraa_gpio_init(4);
      mraa_gpio_dir(led, MRAA_GPIO_OUT);
      //set the direction of the pin as INPUT
      mraa_gpio_dir(button, MRAA_GPIO_IN);

      //set an interrupt on the pin for button
      mraa_gpio_isr(button, MRAA_GPIO_EDGE_RISING,
&do_when_interrupted, NULL);

      while (run_flag) {
            mraa_gpio_write(led, 1);
            sleep(1);
            mraa_gpio_write(led, 0);
            sleep(1);
      }

      mraa_gpio_close(led);
      return 0;
}
```

Figure 7 GPIO Interrupt C code

5. **$ gcc -lmraa -o gpio_interrupt gpio_interrupt.c**
6. **$ ./gpio_interrupt**
7. Press the button.
   • As you can see, the program exits when you press the button.
8. Let's modified the code for a different behavior.
9. **$ vi gpio_interrupt.c**
10. Change the line of code, "mraa_gpio_isr(button, MRAA_GPIO_EDGE_RISING, &do_when_interrupted, NULL);" to "mraa_gpio_isr(button, MRAA_GPIO_EDGE_FALLING, &do_when_interrupted, NULL);".
11. **$ gcc -lmraa -o gpio_interrupt gpio_interrupt.c**
12. **$ ./gpio_interrupt**

13. Press and hold the button for a few seconds.
14. Release the button.
    • The program exits when you release the button.

The interrupt on the GPIO is triggered by the signal edge. There are two kinds of signal edges: a rising edge and a falling edge. These edges are illustrated in the figure below.

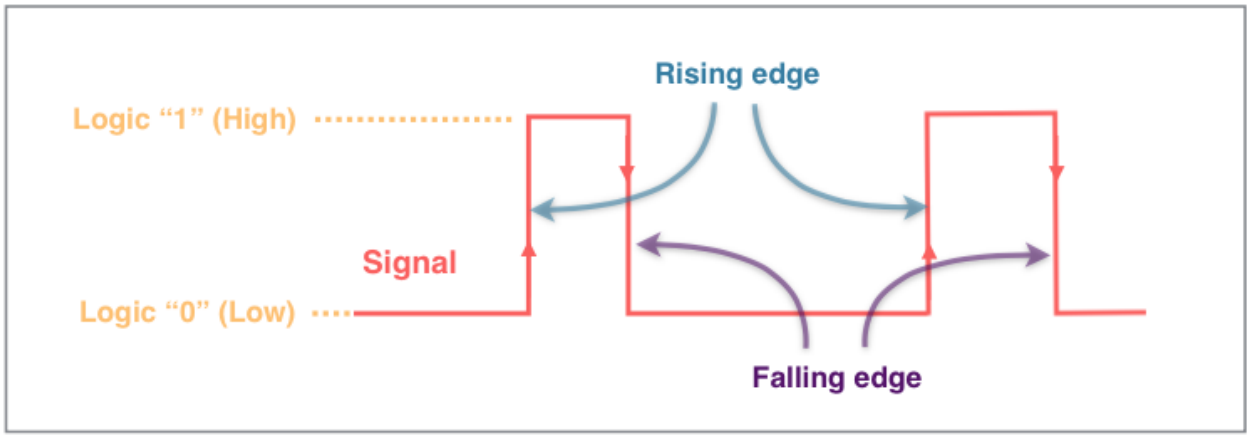


Figure 8 Signal Edges

When the button is not pressed, the signal is low. As soon as the button is pressed, the signal "rises" to high. Such signal transition from low to high is called a "rising edge".

The C code above before the modification has a GPIO interrupt setup for rising edges. The rising edge on the GPIO pin triggers the GPIO interrupt and the handler for the interrupt is called. The handler sets run_flag to 0; hence the program exits out of the while loop.

The modification changes the GPIO interrupt setup. The interrupt is now triggered by the falling edge. When the button is being pressed, the signal is high. As soon as the button is released, the signal "falls" to low. Such signal transition from high to low is called a "falling edge". In the modified C code, the interrupt handler is called when a falling edge is detected on the GPIO pin. The handler sets run_flag to 0; hence the program exits out of the while loop.

More information on the GPIO Interrupt in the MRAA Library can be found at http://iotdk.intel.com/docs/master/mraa/gpio_8h.html.

Analog Input

To demonstrate how to read the analog input on the Edison, we will try the following example from Grove Starter Kit V2.0.

1. Assemble the Edison, a base shield, and a rotary angle sensor as shown in the picture below.

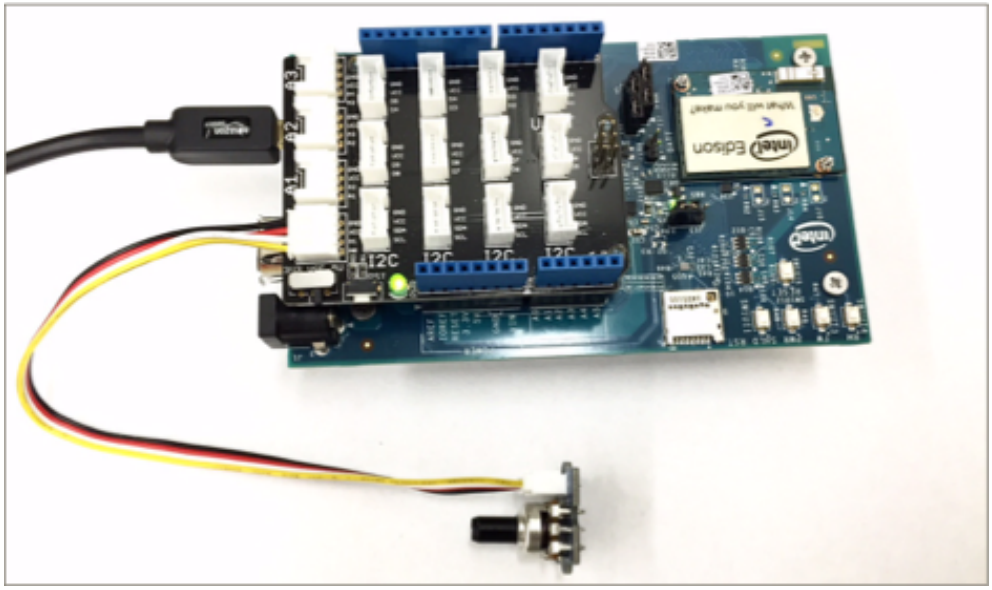


Figure 9 Hardware Setup

2. Open Arduino software and upload the following sketch.

```
// demo of Starter Kit V2.0
const int potentiometer = A0;

void setup()
{
    Serial.begin(9600); //start serial connection for output
    pinMode(potentiometer, INPUT);
}

void loop()
{
    int value = analogRead(potentiometer);
    Serial.println(value); //print value to serial
    delay(1000);
}
```

Figure 10 Analog Input Arduino Sketch

3. Click on Serial Monitor on the upper right corner of Arduino software window.

Figure 11 Serial Monitor

4. Rotate the sensor knob and look at the printed value on the serial monitor.

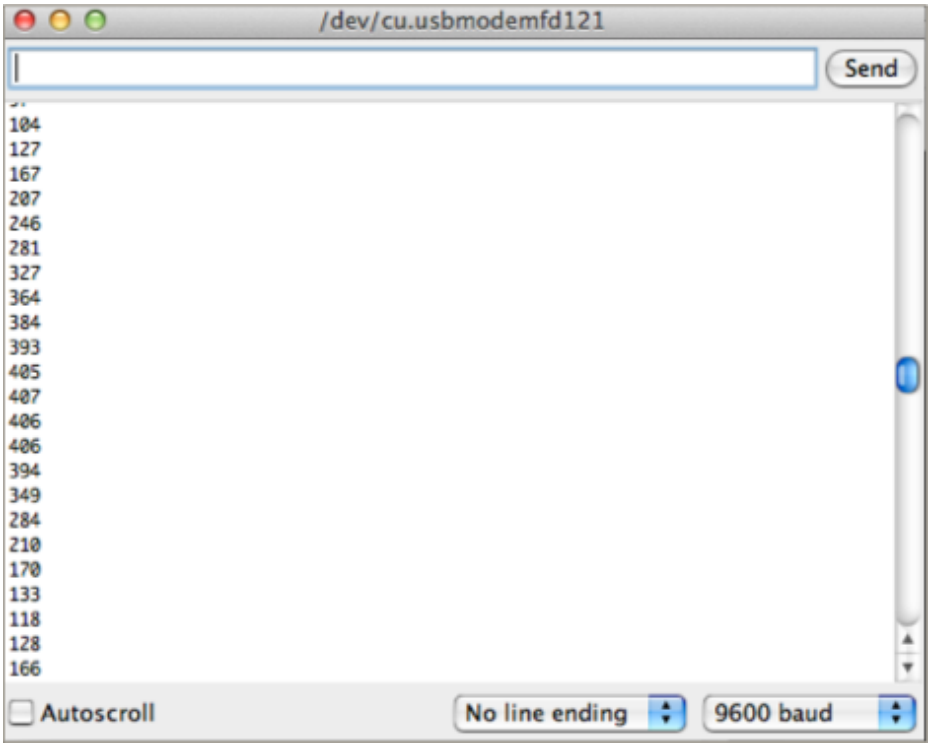
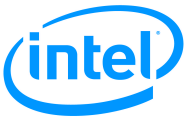
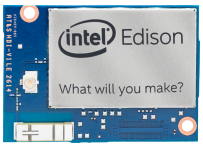


Figure 12 Serial Monitor

5. Don't close the serial monitor, then SSH into the Edison.
6. **$ vi rotary.c**
7. Enter the following C code on the next page.
8. Compile this with **$ gcc -lmraa -o rotary rotary.c**
9. Then execute with **$ ./rotary**
10. Slowly rotate the knob and compare the output to the printed values on the serial monitor.
11. Press **Ctrl-C** to quit.






```c
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <mraa/aio.h>

sig_atomic_t volatile run_flag = 1;

void do_when_interrupted(int sig)
{
     if (sig == SIGINT)
          run_flag = 0;
}
int main()
{
     uint16_t value;
     //declare rotary as an analog I/O context
     mraa_aio_context rotary;
     rotary = mraa_aio_init(0);

     while(run_flag){
          //read the rotary sensor value
          value = mraa_aio_read(rotary);
          printf("%d\n", value);
          sleep(1);
     }

     mraa_aio_close(rotary);

     return 0;
}
```

Figure 13 Analog Input C code

More information on the MRAA Library's AIO API can be found at http://iotdk.intel.com/docs/master/mraa/aio_8h.html.

I2C (Inter-Integrated Circuit)

In the 1980s, Philips developed I2C, which is a low-bandwidth, short distance protocol for on board communications [3]. The I2C-bus is a de facto world standard that is now implemented in over 1000 different ICs manufactured by more than 50 companies [4]. All devices (master and slaves) are connected to serial data (SDA) and serial clock (SCL) as shown in the figure below.

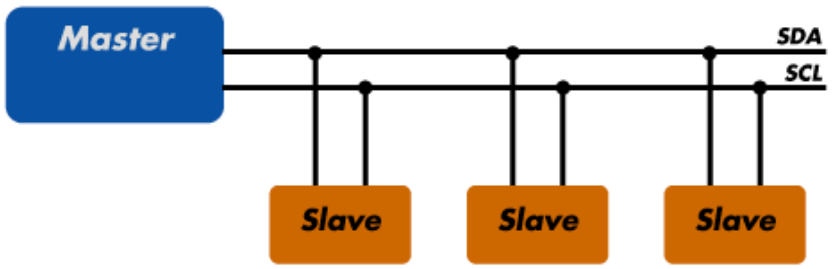


Figure 14 I2C (source: http://www.totalphase.com/support/articles/200349156)

To demonstrate how I2C works, we will set up an I2C between the Edison and the Arduino Uno.

1. Make sure both the Edison and the Arduino Uno are completely powered off (No power is supplied).

2. Connect the Edison and the Arduino Uno as shown in the picture below.

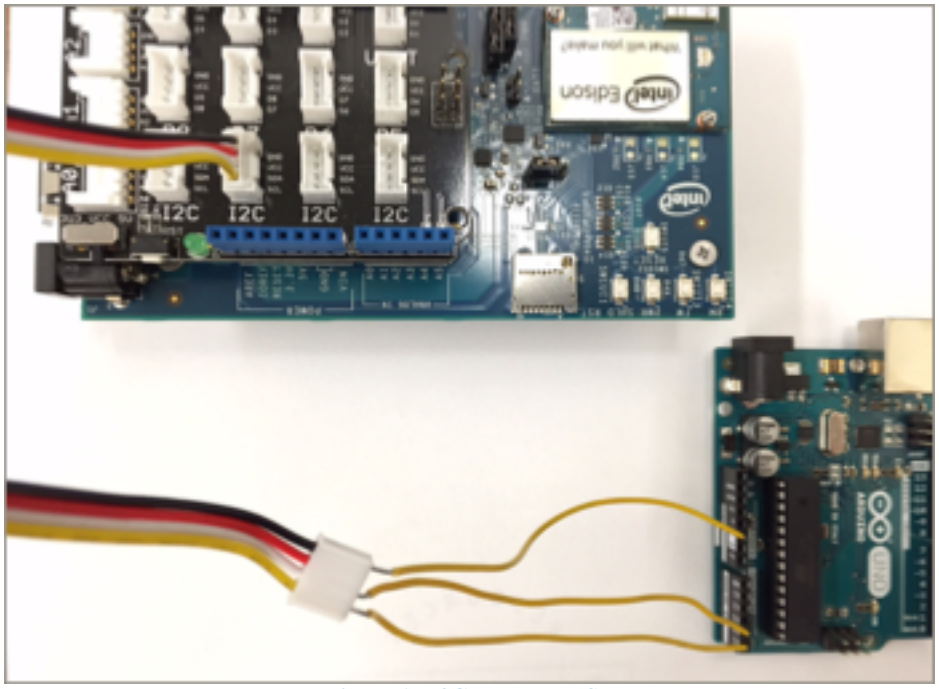
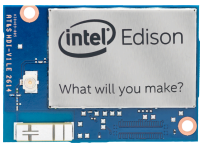


Figure 15 I2C Hardware Setup

1) Plug a cable onto any of I2C port of the base shield on the Edison.
2) Connect three wires to three pins on other end of the cable.
   - black (ground), white (SDA), yellow (SCL)
3) Connect these wires to the Arduino Uno.
   - black (ground) to Arduino GND
   - white (SDA) to Arduino A4 (SDA)
   - yellow (SCL) to Arduino A5 (SCL)

3. Connect a USB cable to the Arduino Uno and your PC.
4. Open Arduino IDE.
5. Go to **Tools -> Board** and select Arduino Uno.
6. Go to **Tools -> Port** and select **/dev/cu.usbmodemxxxxx** (Arduino Uno). (Windows users: select the COM port associated with the Arduino Uno)
7. Upload the following sketch to the Arduino Uno.




```
// Wire Slave Sender
// by Nicholas Zambetti <http://www.zambetti.com>
// Demonstrates use of the Wire library
// Sends data as an I2C/TWI slave device
// Refer to the "Wire Master Reader" example for use with this
// Created 29 March 2006
// This example code is in the public domain.

#include <Wire.h>

void setup()
{
  Wire.begin(2);                // join i2c bus with address #2
  Wire.onRequest(requestEvent); // register event
}

void loop()
{
  delay(100);
}

// function that executes whenever data is requested by master
// this function is registered as an event, see setup()
void requestEvent()
{
  Wire.write("hello "); // respond with message of 6 bytes
                        // as expected by master
}
```

Figure 16 I2C Slave Arduino Sketch

8. Once uploading is done, unplug the USB cable.
9. Connect a micro USB cable to the multi-gadget USB port on the Edison and your PC.
10. From Tools -> Board and Tools -> Port, select the Edison.
11. Upload the following sketch to the Edison.

```
// Wire Master Reader
// by Nicholas Zambetti <http://www.zambetti.com>
// Demonstrates use of the Wire library
// Reads data from an I2C/TWI slave device
// Refer to the "Wire Slave Sender" example for use with this
// Created 29 March 2006
// This example code is in the public domain.

#include <Wire.h>

void setup()
{
  Wire.begin();   // join i2c bus (address optional for master)
  Serial.begin(9600);  // start serial for output
}

void loop()
{
  Wire.requestFrom(2, 6); //request 6 bytes from slave device 2

  while(Wire.available()) // slave may send less than requested
  {
    char c = Wire.read(); // receive a byte as character
    Serial.print(c);      // print the character
  }
  delay(500);
}
```

Figure 17 I2C Master Arduino Sketch

12. Open the serial monitor.
13. Turn on the Arduino Uno by re-connecting the USB cable.
14. You will see “hello” message being printed repetitively.

Figure 18 Output

15. If you don’t see the message, unplug the USB cable from the Arduino Uno, then re-plug.
16. If it still does not work, repeat steps 2 - 13.
17. Upload the blank sketch to the Edison to overwrite the I2C sketch.
18. Unplug the USB cable from Arduino Uno.
19. SSH into the Edison.

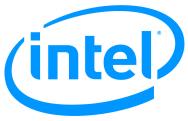
20. **$ cd tutorial4_examples**
21. **$ vi i2c.c**
22. Type the following C code.

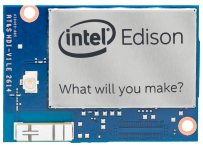
```
#include <stdio.h>
#include <unistd.h>
#include <mraa/i2c.h>

//I2C address is 2
#define I2C_ADDR 2

int main()
{
     char* message;
     int i;

     //initialize MRAA
     mraa_init();
     //declare i2c as an i2c context
     mraa_i2c_context i2c;
     i2c = mraa_i2c_init(0);
     //set the i2c address
     mraa_i2c_address(i2c, I2C_ADDR);

     for (i=0; i<20; i++) {
          //read 6 bytes (characters) of data via I2C
          mraa_i2c_read(i2c, message, 6);
          printf("%.6s\n", message);
          sleep(1);
     }
     //de-initialize the i2c context
     mraa_i2c_stop(i2c);
     return 0;
}
```

Figure 19 I2C C Code

23. **$ gcc -lmraa -o i2c i2c.c**
24. Turn on the Arduino Uno by connecting the USB cable.
25. **$ ./i2c**
26. Compare the output.

More information on the MRAA Library's I2C API can be found at
http://iotdk.intel.com/docs/ master/mraa/i2c_8h.html.





Appendix

You may find more MRAA library API and examples in Python and JavaScript.
Python: http://iotdk.intel.com/docs/master/mraa/python/index.html
JavaScript: http://iotdk.intel.com/docs/master/mraa/node/modules/mraa.html

References

1. http://iotdk.intel.com/docs/master/mraa/
2. http://www.seeedstudio.com/wiki/Grove_-_Starter_Kit_Plus
3. http://www.totalphase.com/support/articles/200349156
4. http://www.nxp.com/documents/user_manual/UM10204.pdf
5. http://arduino.cc/en/Tutorial/MasterReader