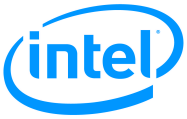
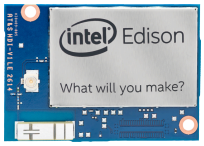
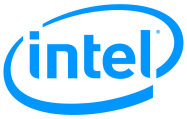




Intel® Edison Tutorial: PWM, SPI and UART



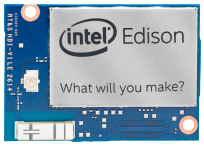
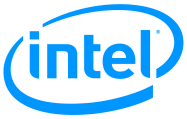
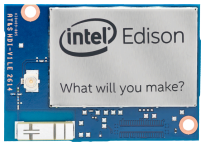


Table of Contents





Introduction

In this tutorial, users will learn how to:

1. Control the brightness of an LED using a PWM signal.
2. Control a servo using a PWM signal.
3. Write C-code to enable communication using the SPI protocol.
4. Write C-code to enable communication using the UART protocol.

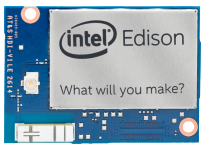
Prerequisite Tutorials

Users should ensure they are familiar with the documents listed below before proceeding.

1. Intel Edison Tutorial – Introduction, Linux Operating System Shell Access
2. Intel Edison Tutorial – Introduction to Linux
3. Intel Edison Tutorial – Introduction to Vim
4. Intel Edison Tutorial – GPIO, Interrupts and I2C Interfaces

List of Required Materials and Equipment

1. 1x Intel Edison Kit
2. 2x USB 2.0 A-Male to Micro B Cable (micro USB cable)
3. 1x powered USB hub OR an external power supply
4. 1x Grove – Starter Kit for Arduino
5. 1x Male to Male Flexible Breadboard Jumper Cable
6. 1x Personal Computer



PWM – Introduction

Pulse-Width Modulation (PWM) is a **modulation scheme** that enables a **digital signal** to represent **analog data**. PWM has a variety of use cases including control of complex circuits, enabling custom brightness levels for Light Emitting Diodes (LED's) and controlling servos.

PWM is a clever scheme to enables digital circuits that can only output either logical "high" (e.g. +5V) or logical "low" (e.g. 0V) to produce an analog output somewhere between "high" and "low". The circuit produces this "in between" value by switching between "high" and "low" very quickly. For example, the Intel Edison is capable of producing a 9.6MHz PWM signal. This means the Intel Edison is capable of generating a new pulse approximately every 105 nanoseconds. The signal can be averaged to deliver this "in between" value. This is typically achieved through some kind of low pass filtering circuitry.

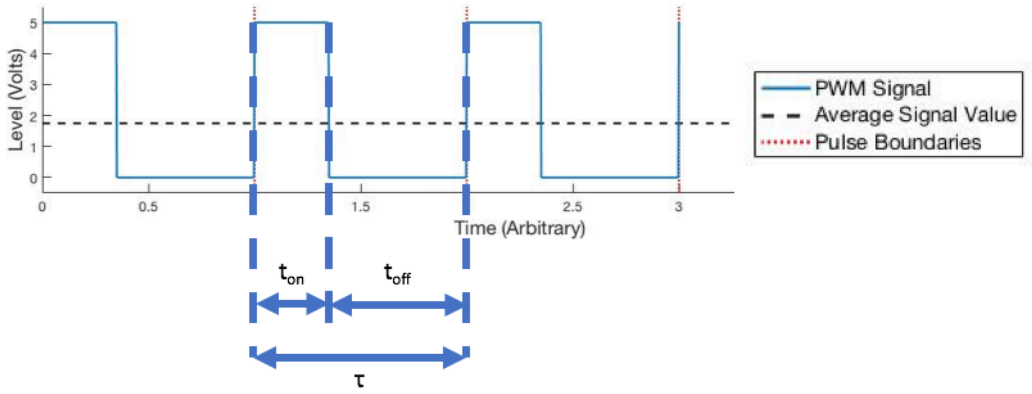


Figure 1: Key metrics of a PWM signal

From the waveform, there are three key values of interest:

1. t_{on} – the amount of time the circuit is outputting a logical level "high".
2. t_{off} – the amount of time the circuit is outputting a logical level "low".
3. τ – the period of the signal.

Any two of these values can be used to calculate the duty cycle of the PWM signal.

$$D = \frac{t_{on}}{\tau} = 1 - \frac{t_{off}}{\tau} = \frac{t_{on}}{t_{on} + t_{off}} \quad [Unitless]$$

If the PWM signal **is a square wave as shown above**, the average value of the signal may be calculated from the below formula. It is important to note that the formula below **may not apply if the shape is different from** Figure 1.

$$\overline{y} = D \times (V_{max} - V_{min}) \; [V]$$

Figure 2 shows how adjusting the duty cycle changes the average value of the signal.

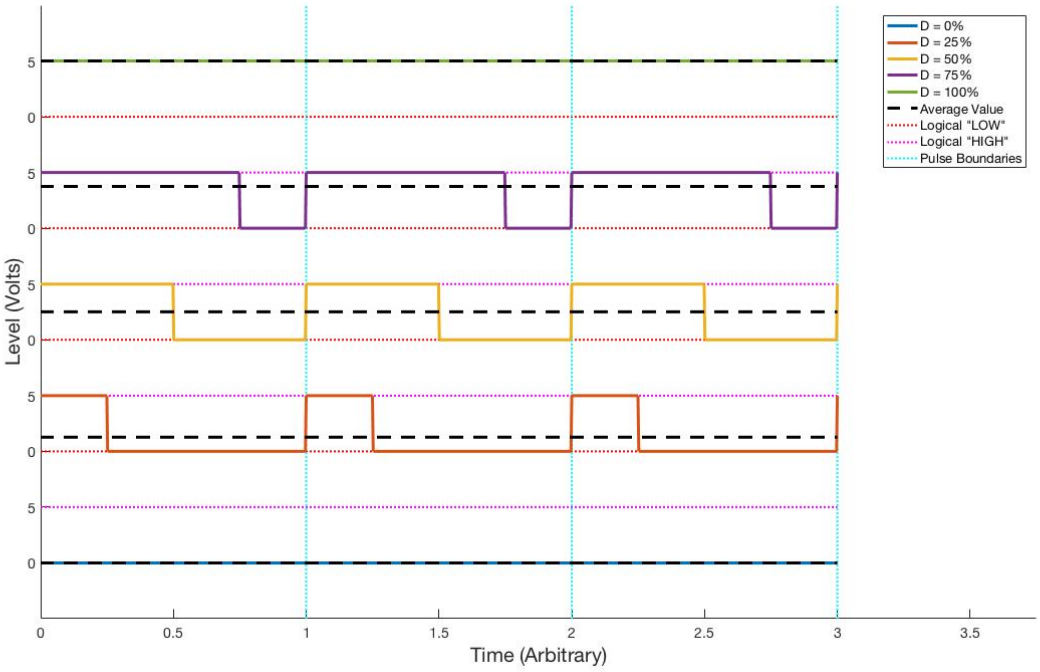


Figure 2: Visualization of how a PWM signal can be averaged to get an analog output from a digital circuit

PWM – LED Control

Follow the below steps in order generate a PWM signal to control the brightness of an LED. There is a total of 6 digital pins on the Edison Board for Arduino that are capable of generating a PWM signal. These pins are D11, D10, D9, D6, D5, and D3.

1. Power down the Intel Edison.
2. Remove all USB and power cables connected to the Intel Edison.
3. Insert the Grove Base shield into the breakout. Ensure it is operating at 5V.
4. Connect an LED to D6 (pin 6) using an LED socket.

   Note: digital pins with "~" before the number are available for PWM.

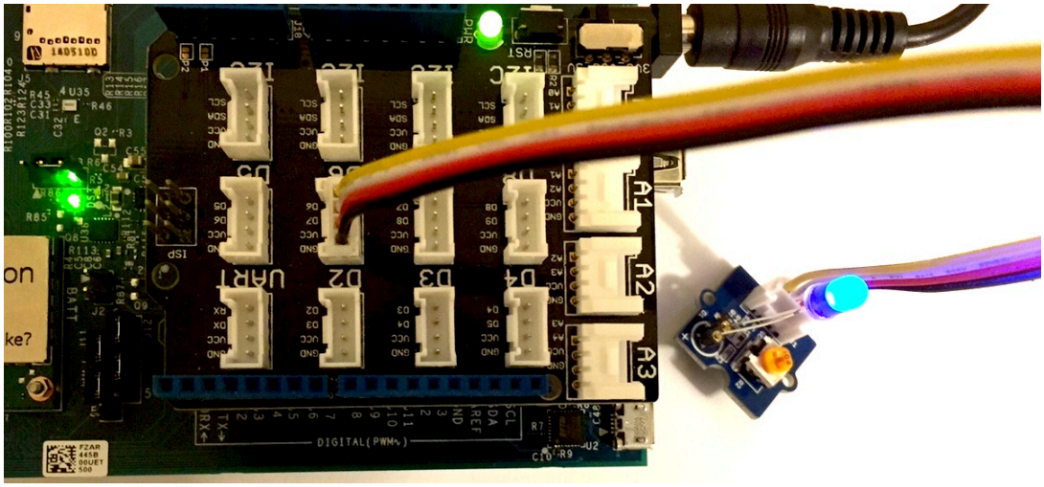


Figure 3: Hardware configuration to demonstrate control of an LED with a PWM signal

5. Connect the USB and power cables to the Intel Edison.
6. Access the shell on the Intel Edison. For more information, refer to the document labelled ***Intel Edison Tutorial – Introduction, Shell Access and SFTP***.
7. Issue the commands shown below.

   $ mkdir ~/tutorial5_examples
   $ cd ~/tutorial5_examples

8. Navigate to **/home/root/tutorial5_examples** directory.
9. **$ vi pwm_led.c.**
10. Enter the C code from Figure 4 into the Vim editor.

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

#define BUFF_SIZE 256

int main()
{
    double brightness;
    char user_input[BUFF_SIZE];
    // declare pwm as a mraa_pwm_context variable (Pulse Width Modulation)
    mraa_pwm_context pwm;

    // initialize the PWM interface
    pwm = mraa_pwm_init(6);

    // set the period of the PWM signal (microseconds)
    mraa_pwm_period_us(pwm, 200);
    // begin outputting the PWM signal
    mraa_pwm_enable(pwm, 1);

    while(1) {
        // display some text for the user
        printf("Enter the desired brightness value (0-100): ");
        // read the user input on standard input
        scanf("%s", user_input);
        // convert the string supplied to a double
        brightness = atof(user_input);
        // error check the value entered by the user
        if (brightness < 0 || brightness > 100) {
            // display an error message if the value is out of bounds
            fprintf(stderr, "Error. Value (%f) is not valid.\n",
                    brightness);
        }
        // the input is valid
        else {
            // display a message to the user
            printf("Setting brightness to (%f)\n", brightness);
            // scale the brightness
            brightness = brightness/100.0;
            // generate the appropriate PWM signal
            mraa_pwm_write(pwm, brightness);
        }
    }
    return 0;
}
```

Figure 4: C code source file pwm_led.c

11. **$ gcc -lmraa -o pwm_led pwm_led.c**
12. **$ ./pwm_led**

PWM – Servo Control

A servo motor is a device that contains gears that control the rotation of a shaft. Typical hobby servo motors have shafts which can be positioned at various angles (usually between 0 and 180 degrees). This angle can be precisely controlled by providing an encoded PWM signal to the input of the servo motor.

For most servos, a 0.5ms pulse width results in the full left position. A pulse width of 2.5ms results in the servo actuating the shaft such that it is in the full right position. The servo also expects the period of the PWM signal to be approximately 20ms. Follow the steps below to implement a servo control system.

1. Power down the Intel Edison.
2. Remove all USB and power cables connected to the Intel Edison.
3. Insert the Grove Base shield into the breakout. Ensure it is operating at 5V.
4. Connect a servo to D6 and a rotary angle sensor to A0.
5. Connect a power supply, capable of providing sufficient current to drive the servo, to the Intel Edison. If the Intel Edison reboots during operation, consider using a power supply capable of providing more current.

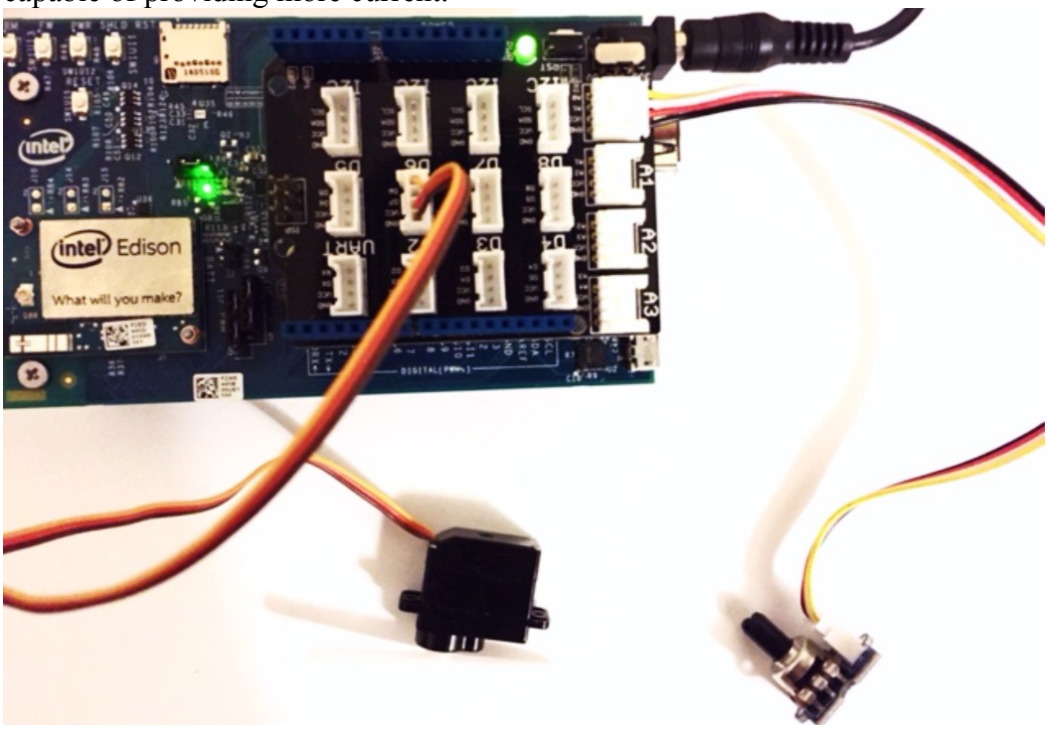


Figure 5: Hardware configuration to demonstrate controlling a servo motor via rotary angle sensing input

6. Navigate to the directory labelled **~/tutorial5_examples**.
7. **$ vi pwm_servo.c**
8. Enter the C code from Figure 6 into the Vim editor.

```c
#include <stdio.h>
#include <signal.h>
#include <mraa/pwm.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()
{
    mraa_pwm_context pwm;
    mraa_aio_context rotary;
    uint16_t rotary_value;
    float value = 0.0f;

    signal(SIGINT, &do_when_interrupted);

    pwm = mraa_pwm_init(6);
    rotary = mraa_aio_init(0);

    mraa_pwm_period_ms(pwm, 20);
    mraa_pwm_enable(pwm, 1);

    while (run_flag) {
        // read the rotary value
        rotary_value = mraa_aio_read(rotary);
        // convert this from an integer to a float
        value = ((float) rotary_value) / 1023.0;

        // scale as appropriate
        value = value/13.33 + 0.025;
        // display the PWM duty cycle to the user
        printf("%f\n", value);
        // actuate the servo by changing the PWM signal
        mraa_pwm_write(pwm, value);
        // pause and continue
        usleep(50000);
    }
    mraa_pwm_write(pwm, 0.025f);
    return 0;
}
```

Figure 6: C code source file pwm_servo.c

9. **$ gcc -lmraa -o pwm_servo pwm_servo.c**
10. **$ ./pwm_servo**
11. Turn the rotary angle sensor to control the servo.
    Note: If the Intel Edison reboots during operation, use the 9V power supply to ensure that the Edison platform is provided with sufficient power supply current.

SPI

This section will provide instructions on sending values to a device using the SPI protocol.

1. Power the Intel Edison down.
2. Remove all power and USB cables from the Intel Edison.
3. Connect a wire from digital pin D11 (MOSI) to digital pin D12 (MISO) as shown below.

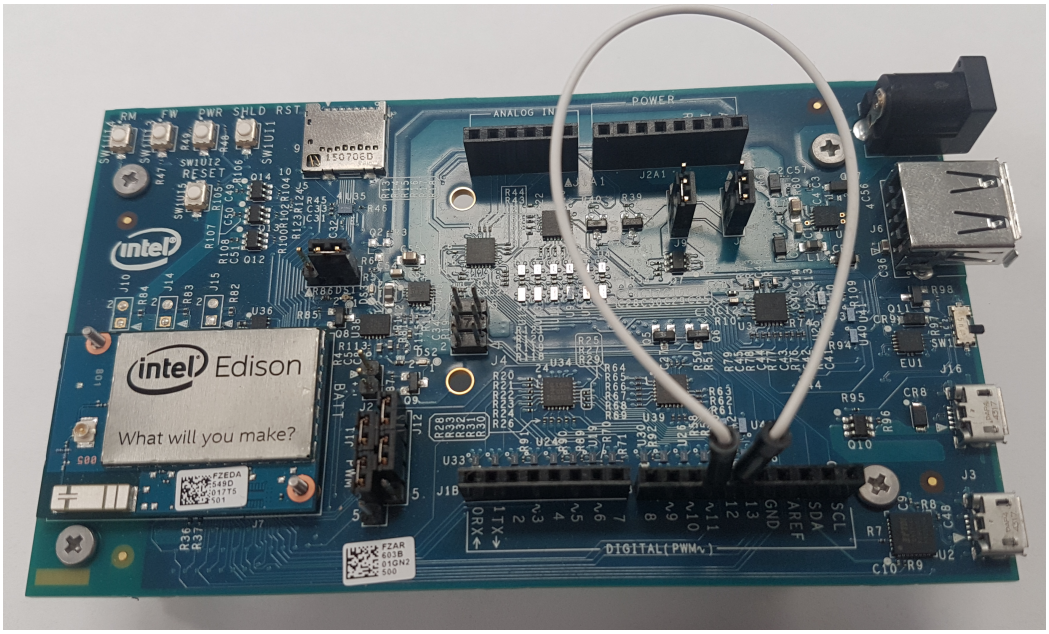
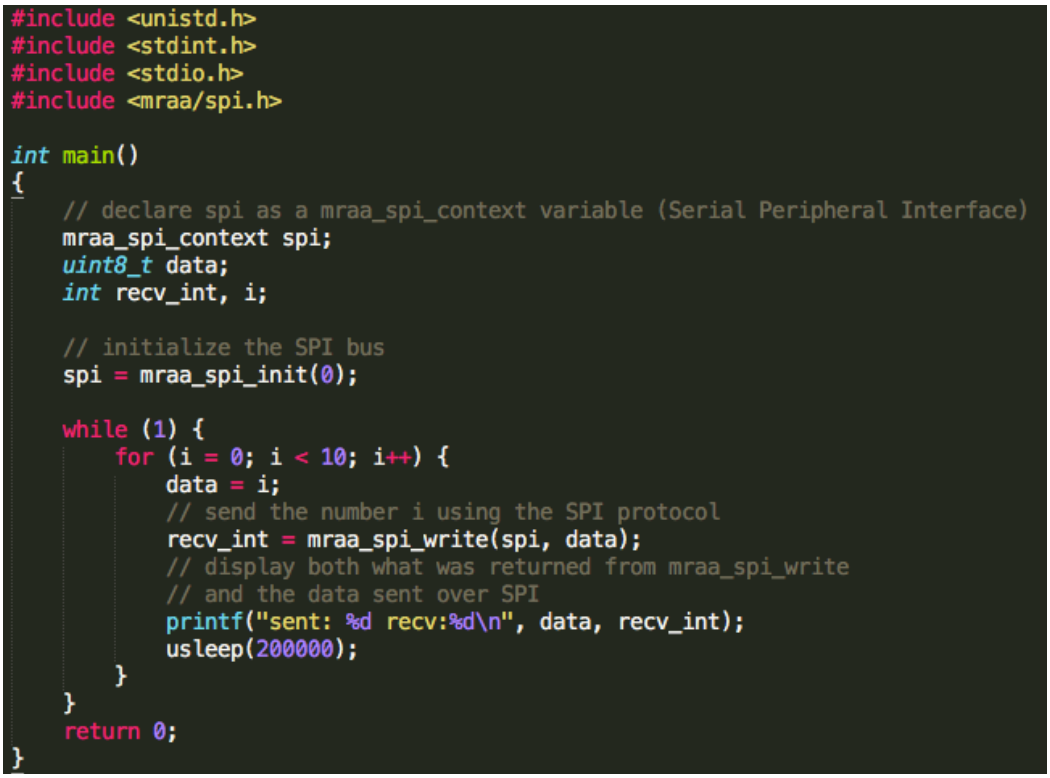


Figure 7: Hardware configuration to demonstrate communication using the SPI protocol

4. Access the shell on the Intel Edison.
5. **$ vi spi.c**
6. Enter the C code from Figure 8 into the Vim editor.



```c
#include <unistd.h>
#include <stdint.h>
#include <stdio.h>
#include <mraa/spi.h>

int main()
{
    // declare spi as a mraa_spi_context variable (Serial Peripheral Interface)
    mraa_spi_context spi;
    uint8_t data;
    int recv_int, i;

    // initialize the SPI bus
    spi = mraa_spi_init(0);

    while (1) {
        for (i = 0; i < 10; i++) {
            data = i;
            // send the number i using the SPI protocol
            recv_int = mraa_spi_write(spi, data);
            // display both what was returned from mraa_spi_write
            // and the data sent over SPI
            printf("sent: %d recv:%d\n", data, recv_int);
            usleep(200000);
        }
    }
    return 0;
}
```

Figure 8: C code source file spi.c

7. **$ gcc –lmraa –o spi spi.c**
8. **$ ./spi**
9. Notice how the data being sent is the same data being received.
10. Issue the **ctrl-C** keystroke to terminate the **spi** process.

UART

This section will provide instructions on sending and receiving strings using the UART protocol.

1. Power the Intel Edison down.
2. Remove all power and USB cables from the Intel Edison.
3. Connect a wire from digital pin 0 (RX) to digital pin 1 (TX) as depicted below.

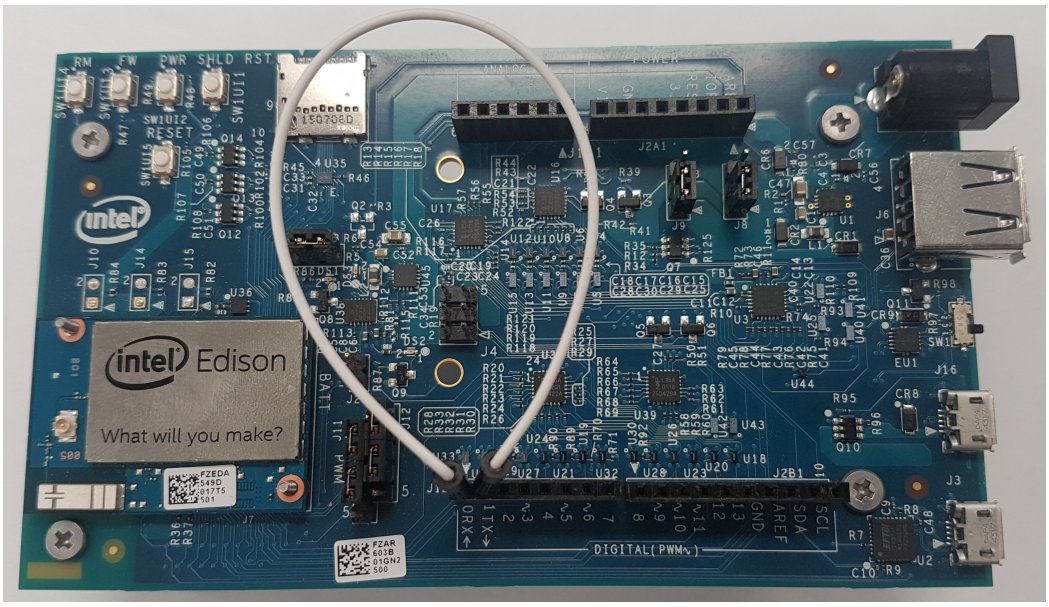


Figure 9: Hardware configuration to demonstrate communication using the UART protocol

4. Access the shell on the Intel Edison.
5. **$ vi uart.c**
6. Enter the C code from Figure 10 into the Vim editor.

```c
#include <stdio.h>
#include <string.h>
#include <mraa/uart.h>

#define BUFF_SIZE 256

int main()
{
    // declare uart as a mraa_uart_context variable (Universal Asynchronous Receiver/Transmitter)
    mraa_uart_context uart;
    int i;
    char msg[BUFF_SIZE];

    // initialize the UART interface
    uart = mraa_uart_init(0);
    // set the baudrate to 9600
    mraa_uart_set_baudrate(uart, 9600);

    while (1) {
        // clear contents of character character array msg
        memset(msg, 0, sizeof(msg));
        // place the string "Hello World!" in the character array msg
        sprintf(msg, "Hello World!");
        // send the character array msg using the UART protocol
        i = mraa_uart_write(uart, msg, strlen(msg));
        // the mraa library buffers messages
        // call mraa_uart_flush to flush the buffer (ie: send a message immediately)
        mraa_uart_flush(uart);
        // clear the buffer again
        memset(msg, 0, sizeof(msg));
        // read what is being sent through the UART protocol
        i = mraa_uart_read(uart, msg, sizeof(msg));
        // display what is being sent to stdout
        printf("UART MSG: %s\n", msg);
        usleep(100000);
    }

    mraa_uart_stop(uart);
    mraa_deinit();
    return 0;
}
```

Figure 10: C code source file uart.c

7. **$ gcc –lmraa –o uart uart.c**
8. **$ ./uart**
9. Notice how the data being sent is the same data being received.
10. Issue the **ctrl-C** keystroke to terminate the **uart** process.