Creating applications with the MCU SDK for the Intel® Edison board: Overview

This guide contains steps to create, build, and debug applications for the generic microcontroller unit (MCU) on an Intel® Edison board using the MCU SDK. The MCU SDK is an Eclipse*-based software development kit for Windows*, Mac* OS X*, and Linux*, used to create applications for the MCU. It is separate from the Eclipse SDK for the Intel® Atom™ processor.

- Creating applications with the MCU SDK for the Intel(R) Edison board
- Installing the MCU SDK
- Creating, building, and working with MCU projects
- Blinking an LED using the MCU
- Using the MCU SDK and API: Code examples
- MCU API
- Sample scripts for the MCU SDK
- Troubleshooting and Resources

Creating applications with the MCU SDK for the Intel® Edison board

This guide contains steps to create, build, and debug applications for the generic microcontroller unit (MCU) on an Intel® Edison board using the MCU SDK. The MCU SDK is an Eclipse*-based software development kit for Windows*, Mac* OS X*, and Linux*, used to create applications for the MCU. It is separate from the Eclipse SDK for the Intel® Atom™ processor. The components of the MCU include:

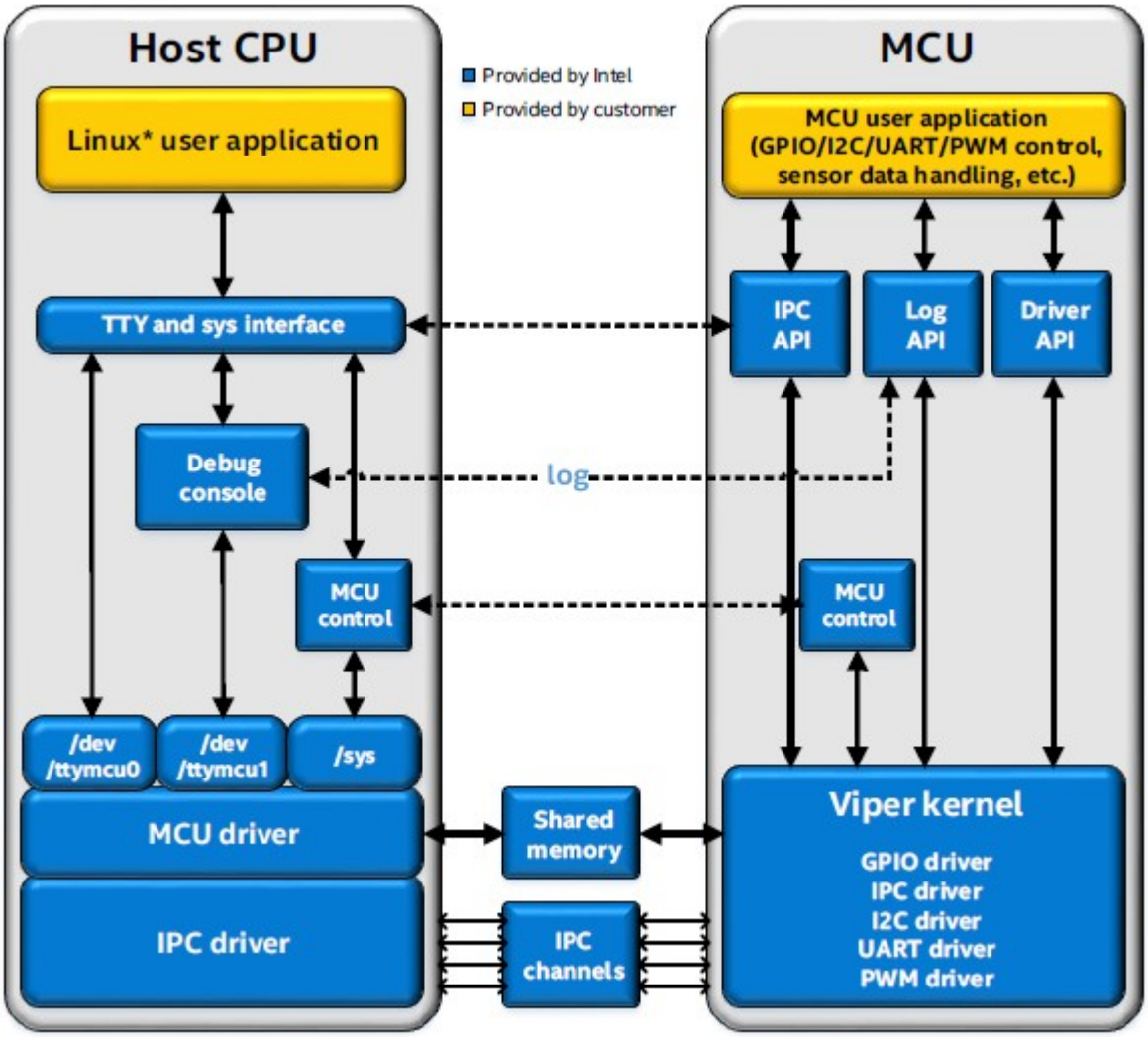
- Scripts and tools to set environment variables, download program binaries, monitor the MCU device trace, and so on
- Documentation for the MCU API. The MCU API is separate from the API used to program the Intel Atom processor
- A launcher to run Eclipse and apply updates to the MCU SDK
- Cygwin to run Linux scripts on Windows

The MCU consists of a host CPU and the MCU itself. The MCU provides developers with simple and real-time peripheral control capabilities for GPIOs, UARTs, and I^2C interfaces. For Intel Edison devices, the host CPU is an Intel Atom processor running Linux, and the MCU is a Minute Intel® architecture CPU running Viper*, a Wind River real-time operating system (RTOS) that provides basic OS function support, including thread scheduling, memory management, interrupt dispatch, and more.

The MCU application runs as an application above the Viper kernel and independently controls the peripherals that connect to the MCU. For example, it can manage the GPIO ports, control and

collect sensor data, and communicate with the host Intel Atom processor. The MCU can also be used to save power by keeping the Intel Atom processor on your board asleep and having the MCU wake the processor when needed.

The MCU driver provides IPC protocol support and allows you to download the MCU application to your device.



The MCU subsystem

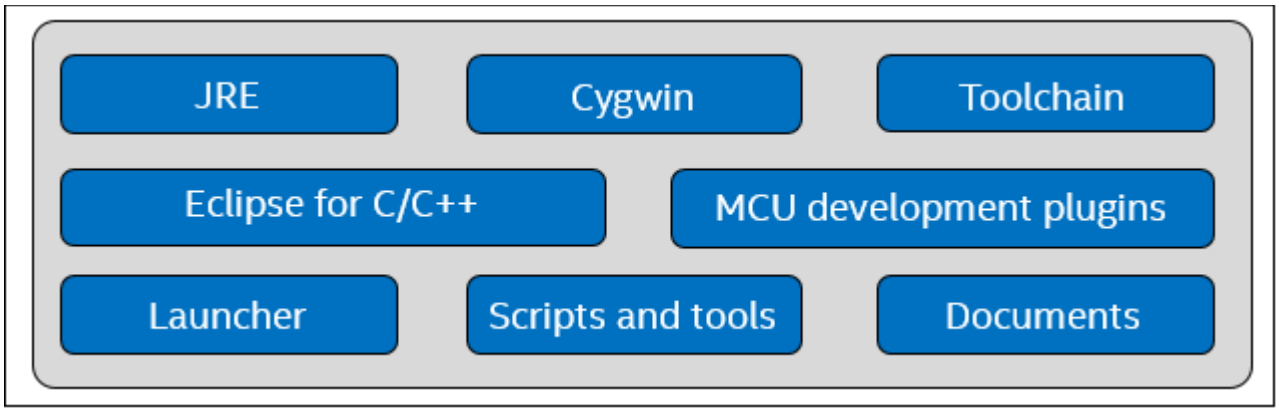
The MCU subsystem features a low-power Minute Intel architecture (IA) 486-based CPU system, with Intel® Pentium® architecture compatibility, I/O (IPC, I^2C, GPIO, HSU, DMA), and SRAM. The MCU subsystem features include:

- The Minute Intel architecture is based on the 486 with Pentium IA ISA compatibility added.
- In the fully active D0 device power state, the CPU frequency is 100 MHz; in S0ix “active idle” sleep states, the CPU frequency drops to 38.4 MHz.
- SRAM: 192 KB SRAM for both code and data.
- IPC: The MCU provides an IPC mechanism that allows subsystem control as well as thread-based communication between the MCU and the Intel architecture core.
- I^2C: I^2C8 and I^2C9 in the system controller unit (SCU) cluster are assigned for MCU usage.

- UART: The MCU can access UART1 and UART2, which may be used as a Linux* kernel console as well.
- GPIO: The MCU can access all GPIO ports.
- The MCU shares all GPIOs with the host Intel architecture.
- PWM: MCU shares all PWMs with the host Intel architecture.
- MCU provides a microsecond-level (1 µs) high accuracy delay API. For details, see the Auxiliary API section.
- MCU provides a tracing capability with different log levels for debugging purposes.
- The MCU SDK is an Eclipse-based SDK for users to create, build, download, and debug MCU applications on multiple operating systems, including Windows, Mac OS X, and Linux.

If you are designing a whole system, make sure that the host system and MCU do not conflict with each other when accessing hardware resources.

The components of the MCU SDK are pictured below:



Where:

- **JRE** is the Java Runtime Environment
- **Cygwin** is provided to run Linux scripts on Windows
- **Toolchain** is the gcc cross-compile toolchain for Intel® Edison Linux* OS
- **Launcher** is the MCU SDK launcher user interface
- **Scripts and tools** are provided to set environment variables, download binary, get trace from device, and more

MCU applications

An MCU application for the Intel Edison board is a user application that runs on Viper*, a real-time operating system provided by Wind River. The MCU application's binary file is stored on Linux* RootFS and downloaded by the MCU driver during Linux* kernel bootup. An MCU application can independently control GPIOs and UARTs, collect and preprocess sensor data, and communicate with the host CPU (an Intel Atom processor on the Intel Edison board), which can enter a low power state while the MCU is still running. Intel provides an MCU SDK and a set of APIs to help you develop MCU applications.

Host side interfaces

TTY Channels and several sysfs nodes listed below are defined and exposed for user-space Linux applications to communicate with the MCU side.

TTY interfaces

- **/dev/ttymcu0**: A message/data transmission channel between the host and the MCU. The host application can send and receive data to/from the MCU via this interface and an MCU application can communicate with host via the host_send and host_receive APIs.
- **/dev/ttymcu1**: This channel is defined as an interface to get MCU log messages.

Sysfs interfaces

- **/sys/devices/platform/intel_mcu/control**: A write-only control node to load the MCU application. Currently this node is for internal use only.
- **/sys/devices/platform/intel_mcu/fw_version**: The version of the MCU SDK that is used to build MCU applications.
- **/sys/devices/platform/intel_mcu/log_level**: A read/write node to set and get the current MCU application log level. Supported input strings include fatal, error, warning, info, and debug.

The MCU SDK

The MCU SDK includes an Eclipse plugin for MCU application development. It includes the prebuilt tools to build, download, and debug MCU applications. Template source code is included and will be deployed while creating projects. The main features of the MCU SDK include:

- Creating an MCU project
- Building an MCU application
- Downloading an MCU application to target device
- Showing debug messages

Known limitations

The following are known limitations regarding MCU implementation on the Intel® Edison platform:

- The maximum message size for interprocess communication between the CPU and the MCU is currently limited to 255 bytes.
- The current Viper OS release does not support unloading an MCU user application after it is loaded. The user must reboot the device to unload the downloaded binary.
- The MCU application update requires a Linux reboot.
- Maximum MCU binary size is limited to 120 KB currently.
- Due to the limitations of the current Viper OS release, the MCU OS tick value is 10 ms.
- You cannot install the SDK in any directory that has spaces in the path name. (For example, Program Files or My Documents is not allowed.)
- There is no floating point support in MCU. (You can build a floating point code using MCU,

but it will generate runtime exceptions.)

- The MCU is limited to 2xI^2C.
- The MCU application is limited to single-thread usage.
- You cannot boot the MCU without first booting Linux.
- There is no access coordination between the Intel Atom processor and the MCU. Both can configure the same I/O at once. This can potentially cause conflicts and must be user-managed.
- The MCU SDK (Eclipse) is different and separate from the C/C++ Eclipse SDK for the Intel Atom processor.
- The MCU I/O API is not aligned with the API for the Intel Atom processor. You must rewrite code to move from the Intel Atom processor to the MCU.
- The MCU does not have a watchdog timer.
- Currently, the SPI feature is not supported for the MCU.
- It may take a long time to launch the MCU SDK on Windows under some network configurations due to Eclipse or Cygwin limitations.
- Pin muxing (Arduino extension board configuration) must be done through the Linux interface before starting the MCU. For details, see Setting the Intel® Edison board up to test system on a chip (SoC) GPIOs.
- PWM runtime power management must be disabled from the Linux side in order to access this module. For details, see Controlling PWM ports.
- There are no programmable timer interrupts available on the MCU.

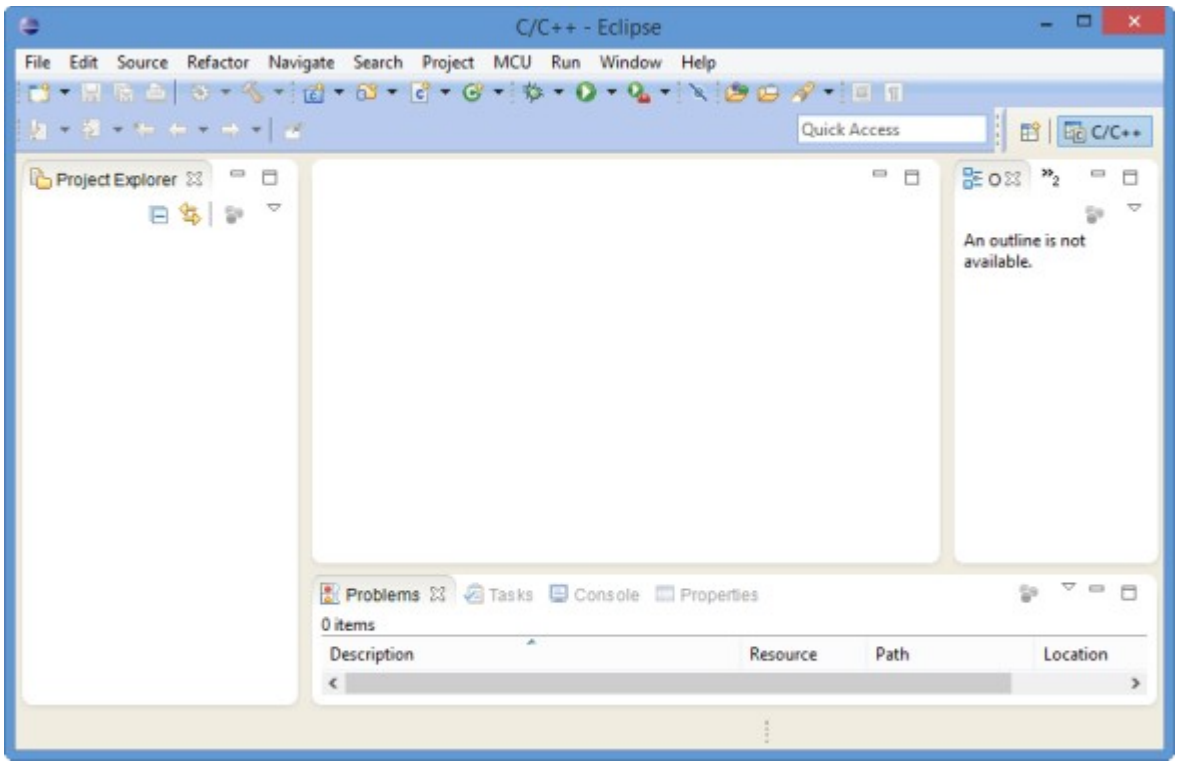
Installing the MCU SDK

This section contains steps to install the MCU SDK on your system. See the appropriate link below:

- Installing the MCU SDK on Windows*
- Installing the MCU SDK on Mac* OS X*
- Installing the MCU SDK on Linux*

Once you have installed the MCU SDK, the next steps are to run the MCU SDK version of Eclipse* and copy the MCU SDK scripts onto your board, detailed in the following sections:

- Running Eclipse
- Getting the MCU SDK scripts onto your board
- Configure your MCU launching service to run scripts automatically (optional)



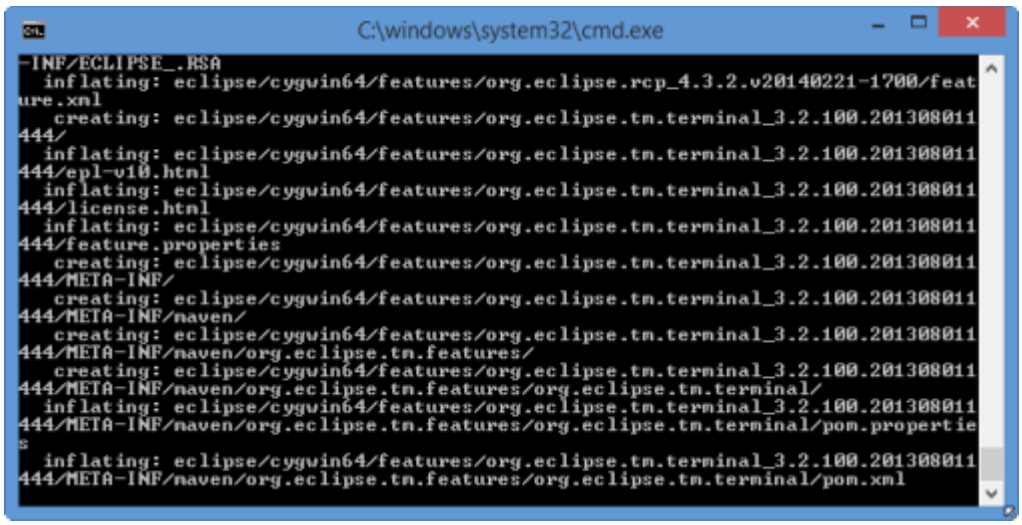
Prerequisites

- You have set up your Intel Edison board and established a serial communication session with it. For steps, see the appropriate link below:
    - Getting Started with the Intel® Edison board on Windows*
    - Getting Started with the Intel® Edison board on Mac*
    - Getting Started with the Intel® Edison board on Linux*
- Your system and your board are on the same local network.

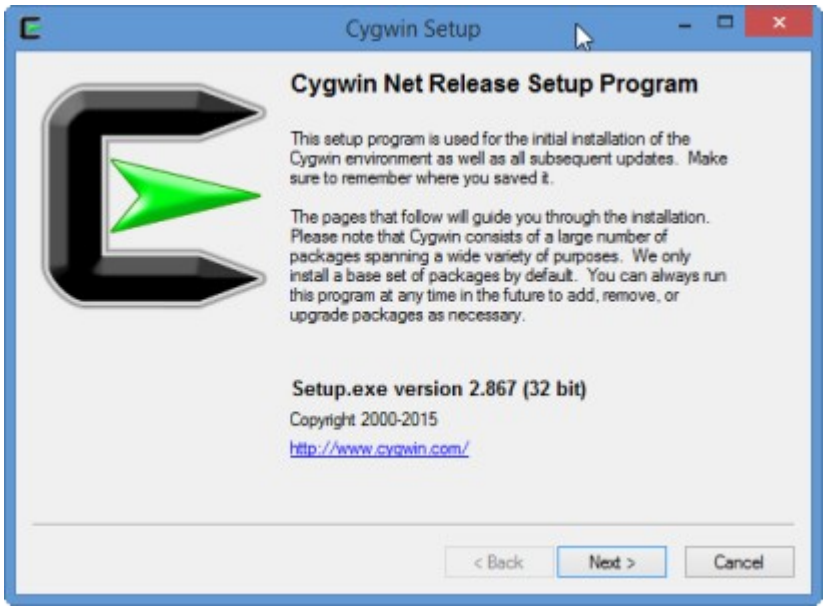
Installing the MCU SDK on a Windows* system

This section contains steps to install the MCU SDK on a Windows system.

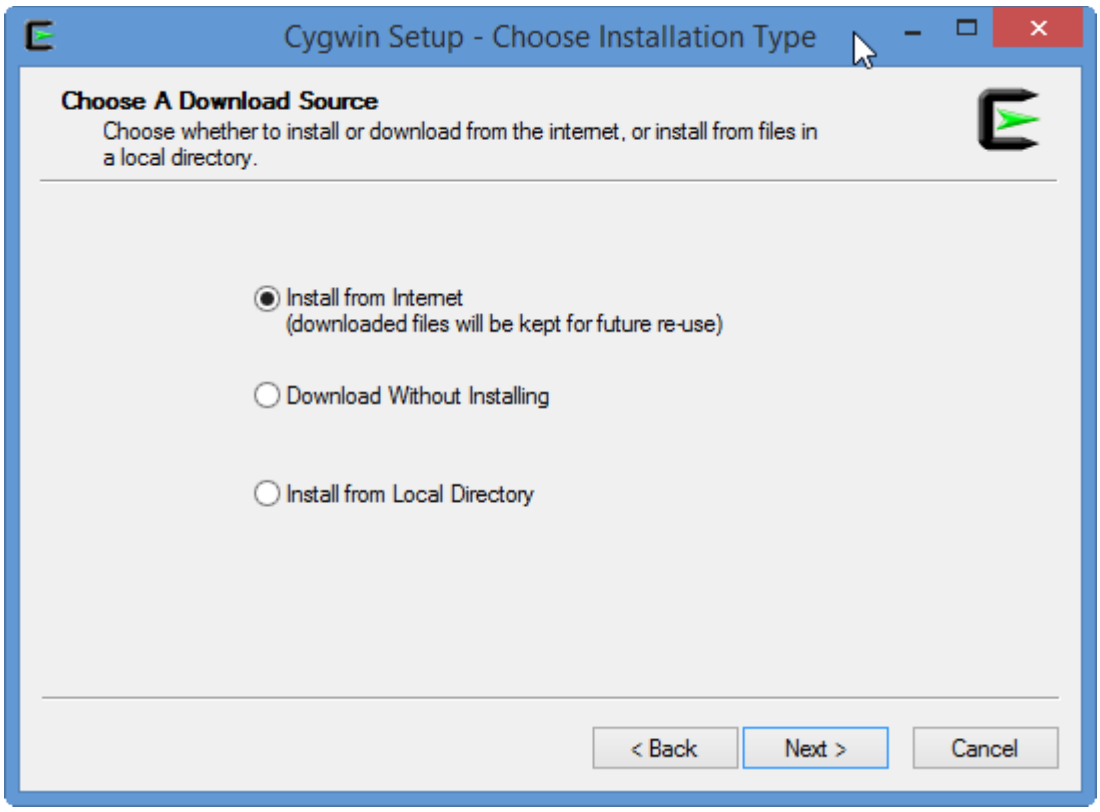
1. Download the SDK installation package from the Intel® Edison Software Downloads page.
2. Extract the installation package. The folder where you extract the MCU SDK files is also your installation folder. Do not select a location that has spaces in the folder path name, such as Program Files or My Documents.
3. In the extracted folder, double-click **setup.bat**. Wait for the extraction process to finish.



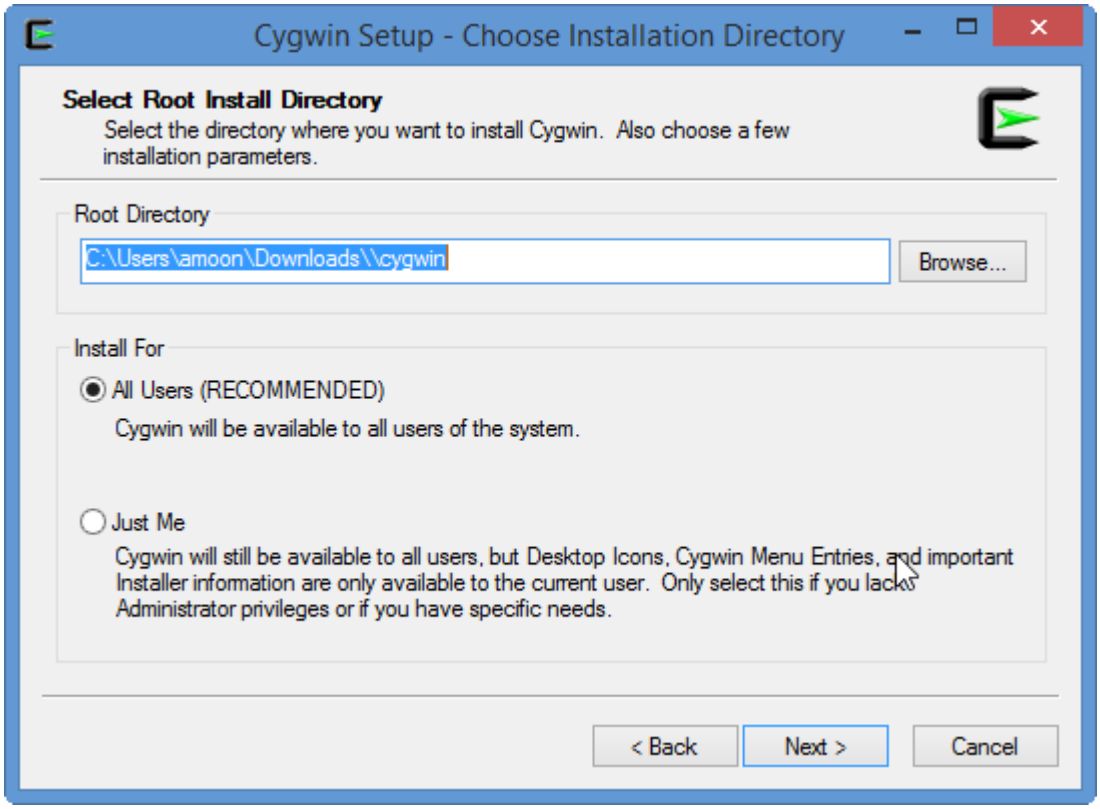
4. When the User Account Control dialog box opens, click **Yes** to continue the installation. Click **Next**. The Cygwin Setup window opens.



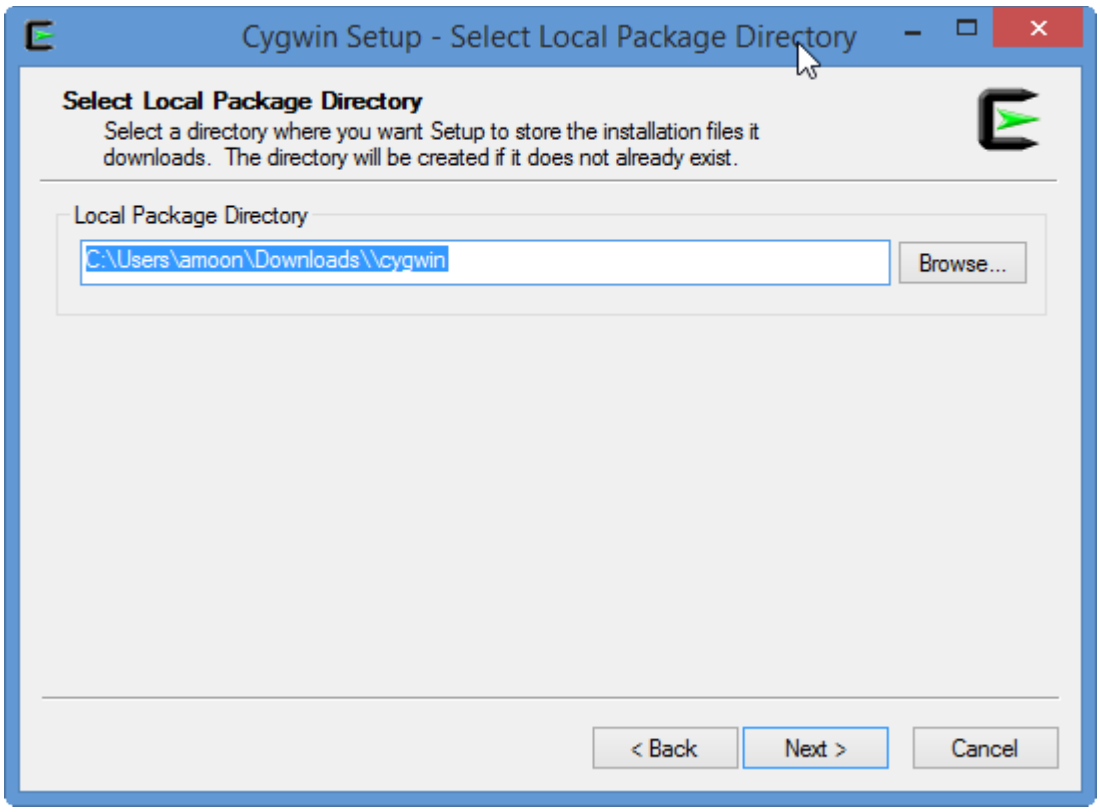
5. Select **Install from Internet** and click **Next**.



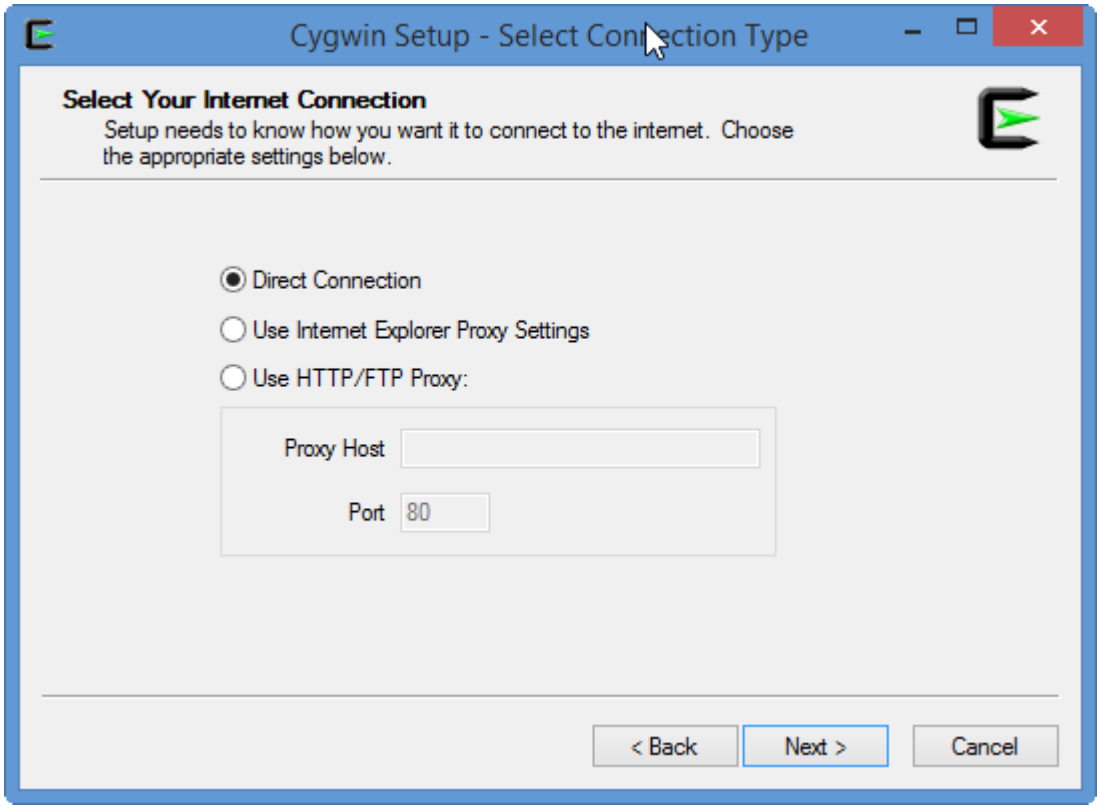
6. On the Choose Install Directory page, click **Next**.



7. On the Select Local Package Directory page, click **Next**.



8. Under Select Your Internet Connection, select the appropriate internet connection settings and click **Next**.

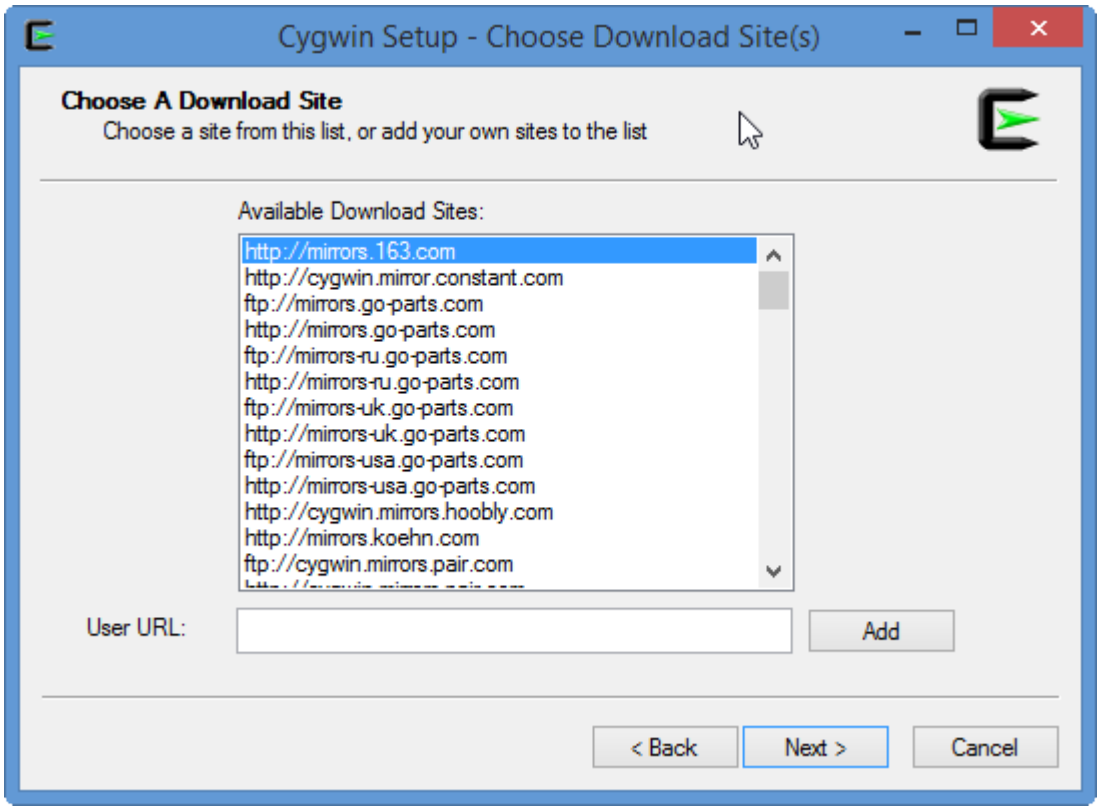


9. From the Available Download Sites list, choose a mirror site to download the Cygwin* tools

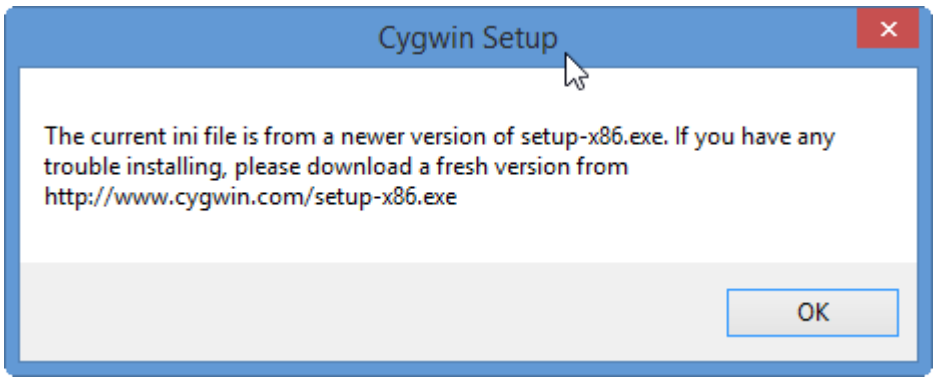
from.

If no sites are listed, copy the URL of a mirror site from the following location: https://cygwin.com/mirrors.html. Copy and paste the URL in the **User URL** field, then click **Add**.

10. Click **Next**.



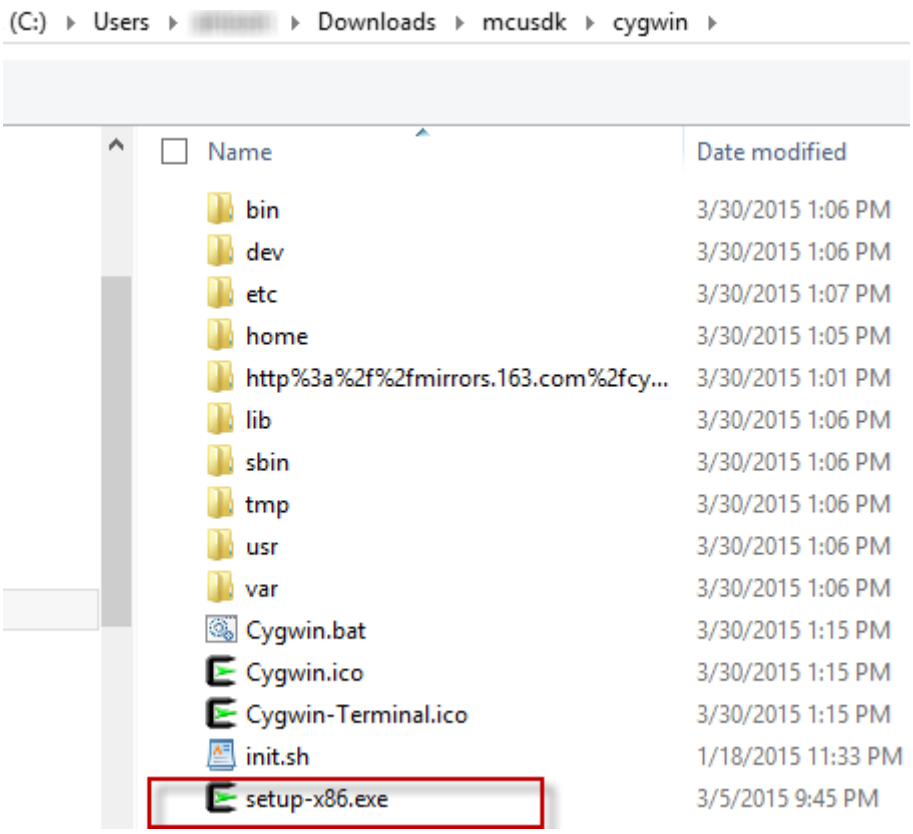
11. If a "Current ini file is from a newer version of setup-x86.exe" message is displayed, your installation files contain an older version of the Cygwin setup program. Exit the installer, then download and run the latest Cygwin setup program by doing the following:



a. Download the latest version of the installer from the following location:

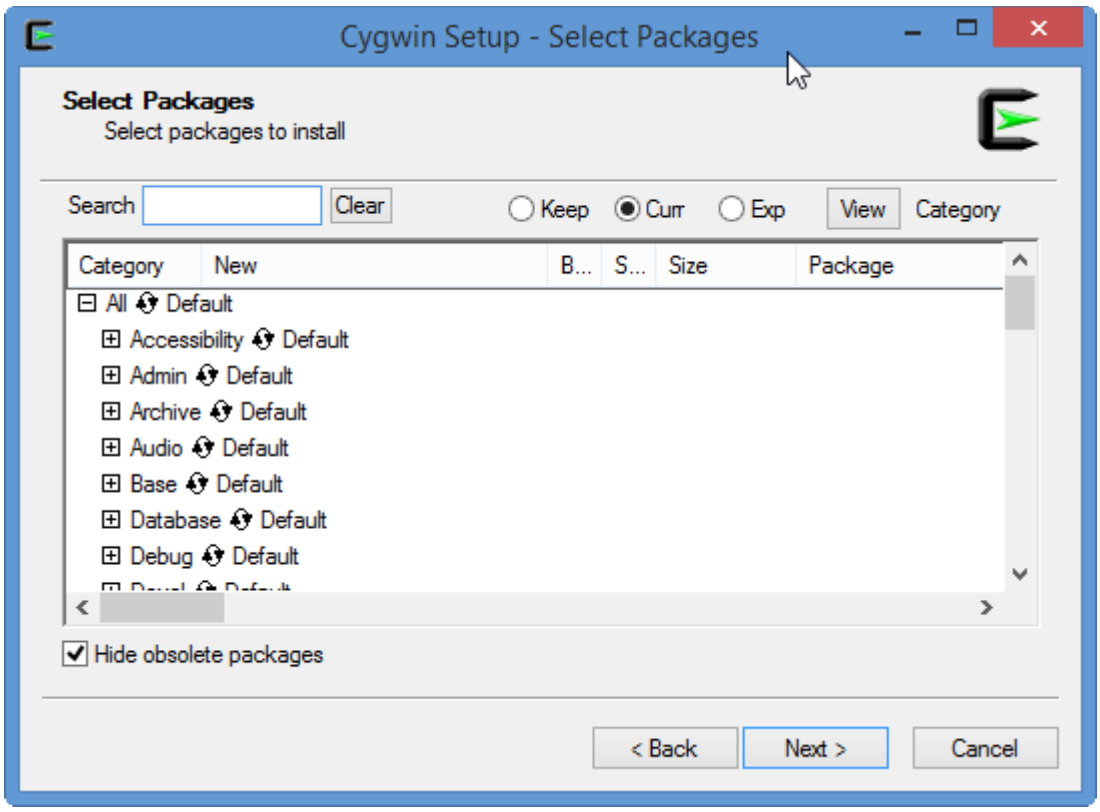
http://www.cygwin.com/setup-x86.exe

b. In your MCU SDK installation folder, open the **cygwin** folder. Replace the existing `setup-x86.exe` file with the newer version you just downloaded.

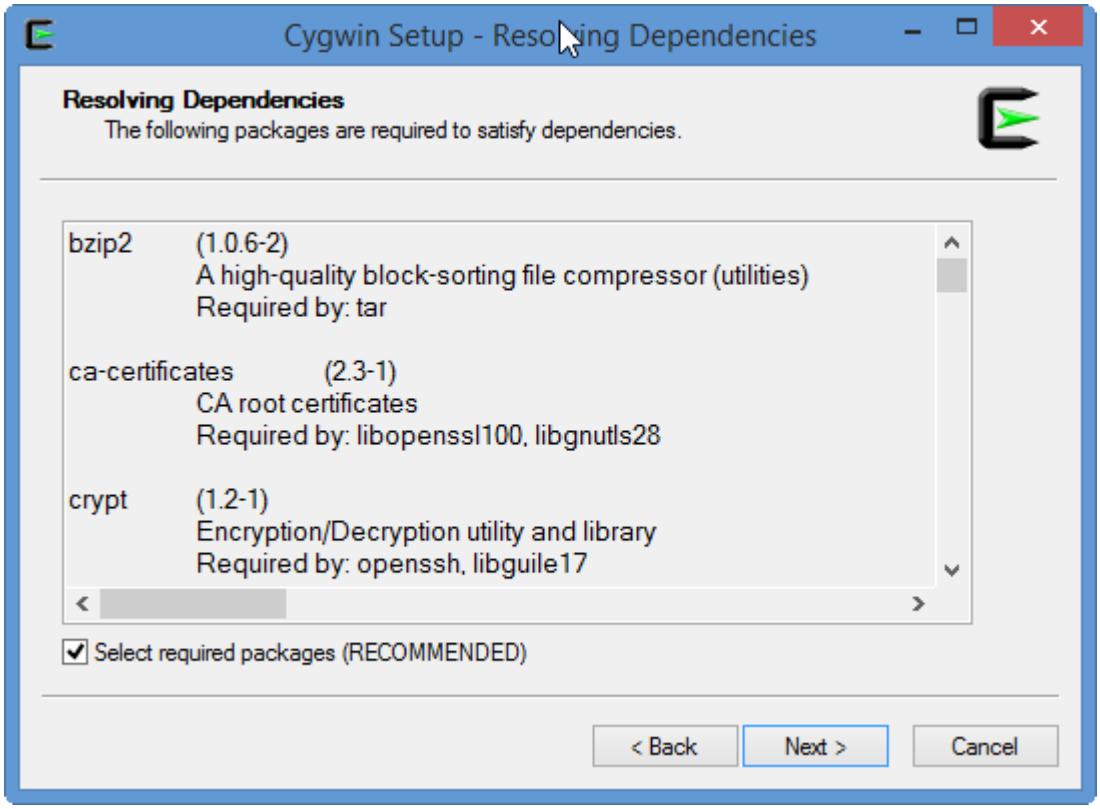


c. Exit the Cygwin installer, if it is still open. Navigate back to your installation folder and double-click **cygwin_setup.bat**. Repeat the appropriate steps above to install Cygwin with your new installer.

12. Click **Next**.



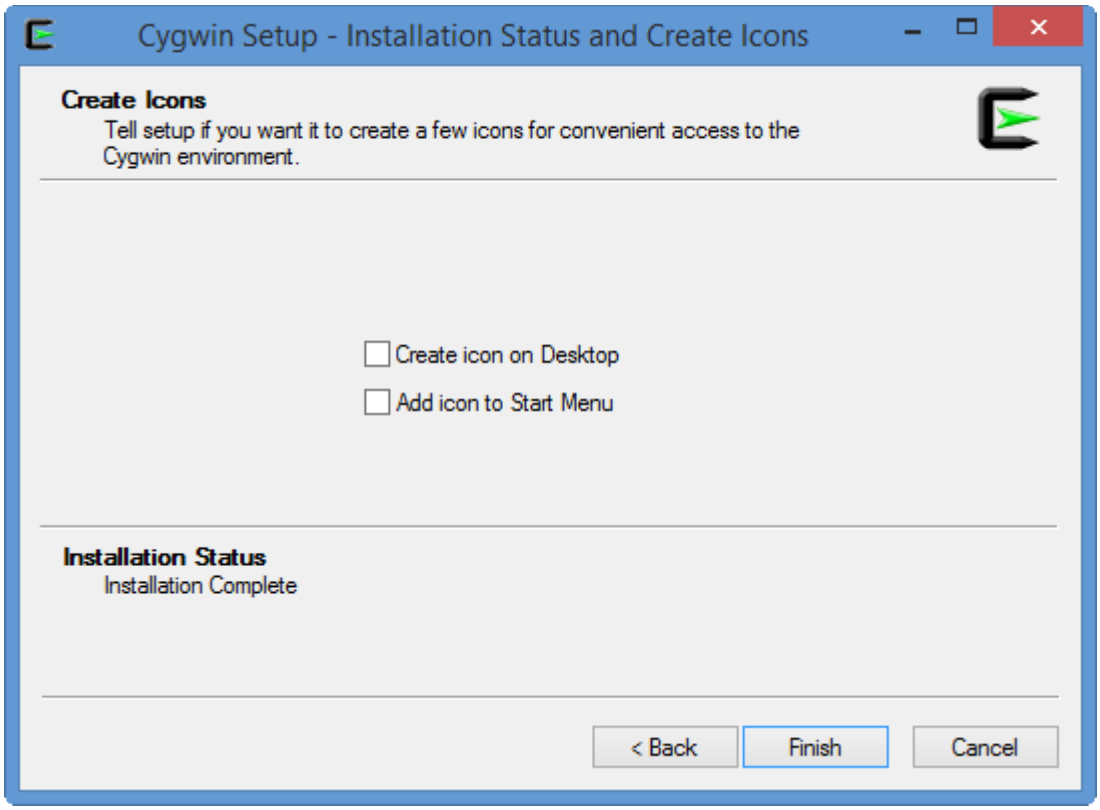
13.Click **Next**.



14.Under Create Icons, do the following:

a. Clear the **Create icon on Desktop** check box.

b. Clear the **Add icon to Start Menu** check box.
c. Click **Finish**.



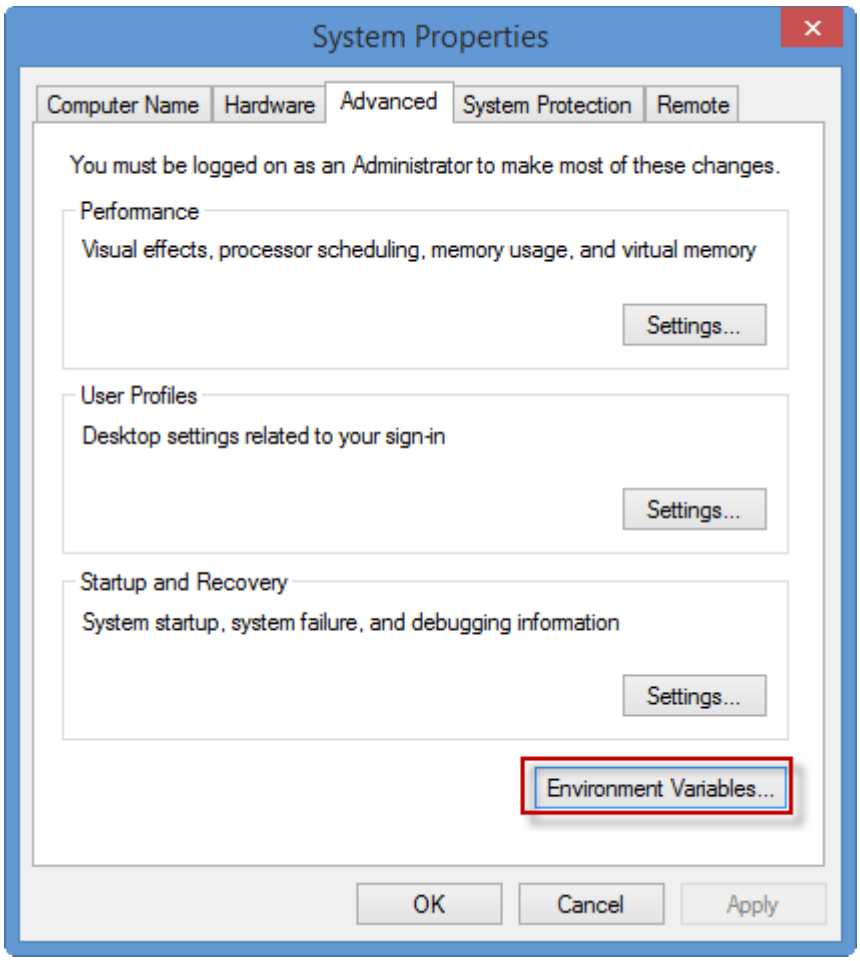
15. Next, download the appropriate JRE installation package for your system, as listed in the table below. The JRE installation packages can be found in the following location: http://www.oracle.com/technetwork/java/javase/downloads/jre7-downloads-1880261.html

    Be sure to download the Java SE Runtime Environment 7u75 version of the JRE.

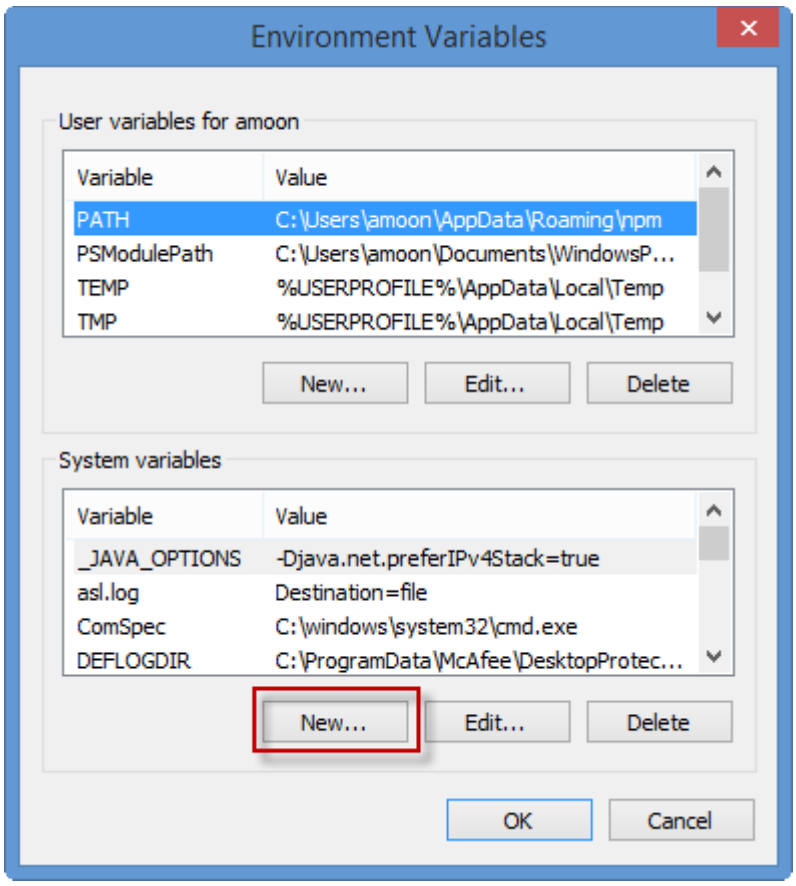
If you have this version of the MCU SDK...	Download this version of the JRE
Windows32	Windows x86
Windows64	Windows x64

16. Double-click the JRE installation package, then follow the on-screen instructions to install the JRE.

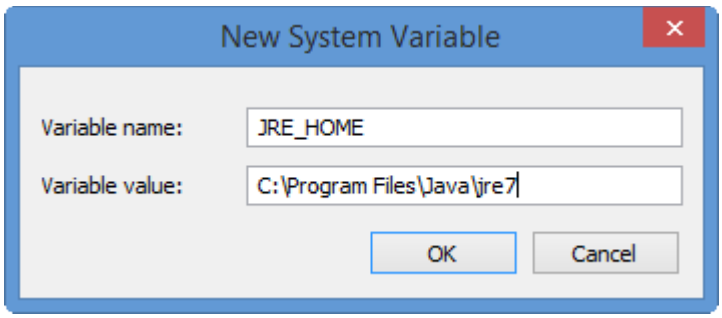
17. Once you have finished installing the JRE, click the **Start** menu, then right-click **Computer**. Select **Properties**.
18. Click **Advanced System Settings**. Navigate to the Advanced System Properties and click **Environment Variables**.



19.Under System variables, click **New**.



20. In the **Variable name** field, type `JRE_HOME`. In the **Variable value** field, type the location where you installed Java (for example, `C:\Program Files\Java\jre7`). Click **OK**.



21. Click **OK** to create your variable.
22. Click **OK** to apply your changes.
23. Continue with the steps in the Running Eclipse section.

Cygwin version 1.7.34 includes changes in Windows account management that may lead to a delay of more than 20 seconds on some network configurations. To minimize this delay, do the following:

1. Navigate to *`MCU_SDK_ROOT`*`\cygwin\bin`, where *`MCU_SDK_ROOT`* is your MCU SDK installation folder.
2. Double-click **mintty.exe**. Based on your network environment, it may take up to 30 seconds to open a new Cygwin window.

3. Enter the following commands to generate the passwd and group files:

```
./mkpasswd.exe -c > /etc/passwd
./mkgroup.exe -c > /etc/group
```

4. In *MCU_SDK_ROOT*`\cygwin\etc`, open the nsswitch.conf file in a simple text editor. Edit the passwd and group settings, as follows:

```
#       /etc/nsswitch.conf
…
# db_shell: cygwin desc
# db_gecos: cygwin desc
passwd: files
group: files
```

5. Close the Cygwin window.
6. Double-click the **mintty.exe** file to verify that the amount of time required to open a new Cygwin window has decreased.

Installing the MCU SDK on a Mac* OS X* system

For a Mac OS X system, follow these steps:

1. Download the MCU SDK installation package from the Intel® Edison Software Downloads page, then extract the contents of the package.
2. Download and install the Java installation package from the following location: https://support.apple.com/kb/DL1572?locale=en_US.
3. Continue with the steps in the Running Eclipse section below.

Installing the MCU SDK on a Linux* system

For a Linux system, follow these steps:

1. Download the MCU SDK installation package from the Intel® Edison Software Downloads page, then extract the contents of the package.
2. Download and install the JRE installation package corresponding to your system. Be sure to

download the Java SE Runtime Environment 7u75 version:
http://www.oracle.com/technetwork/java/javase/downloads/jre7-downloads-1880261.html

If you have this version of the MCU SDK...	Download this version of the JRE
Linux32	Linux x84
Linux64	Linux x64

3. Define the environment variable `JRE_HOME` as the folder path where you installed the JRE.
4. Add `export JRE_HOME=/[your_JRE_installation_dir]/jre1.7.0_75` to the .bashrc file in your home directory, then reboot your terminal.
5. Continue with the steps in the Running Eclipse section below.

Running Eclipse

Now you are ready to run Eclipse.

Prerequisites

- Your board should be connected to your system in the same local network.
- For systems with Linux or Mac OS X, be sure to set your host network IP address to 192.168.2.1 and your subnet mask to 255.255.255.0.

1. Navigate to your MCU SDK installation folder, then do one of the following:
    - For Windows: Double-click **mcusdk.exe**.
    - For Mac OS X and Linux: Click **mcusdk**.
2. The MCU SDK Launcher opens. Click **Launch**. It may take some time to load the appropriate Eclipse modules.

3. The first time you run Eclipse, a welcome page is displayed. Close the welcome page to view your workspace.

Loading the MCU SDK scripts onto your board

The MCU SDK includes scripts (`.sh` f files) to set up the GPIO configuration for the Arduino expansion board, disable PWM management, and more. These scripts are necessary to control PWM ports, read the value of an I2C device, and more. This section contains steps to copy the scripts to your board. For more information about the available scripts and download links, see the MCU Development Guide.

1. Navigate to *`MCU_SDK_HOME`*`/cygwin/home/`*`username`*, where *`MCU_SDK_HOME`* is your MCU SDK installation folder and *`username`* is your user name. Copy the .sh files into this location.
2. Navigate to `MCU_SDK_HOME/cygwin/bin/`. Double-click **mintty.exe** to run it. A Cygwin window opens.
3. To copy the scripts onto your board, enter the command:

```
scp ~/*.sh root@192.168.2.15:~
```

4. If prompted to continue connecting, type `yes` and press Enter. All the MCU SDK scripts are copied to your board.

```
/myscripts
$ scp ~/*.sh root@192.168.2.15:~
The authenticity of host '192.168.2.15 (192.168.2.15)' can't be established.
ECDSA key fingerprint is SHA256:I51UoCi+dnmop23W5vvz3Ob/9D7fRF1po8f1fdSrJU0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.2.15' (ECDSA) to the list of known hosts.
init_DIG.sh                                   100% 7157     7.0KB/s   00:00
init_i2c8.sh                                  100%  946     0.9KB/s   00:00
init_mcu_PWM.sh                               100% 1451     1.4KB/s   00:00
set_DIG.sh                                    100% 2206     2.2KB/s   00:00
```

5. Make your scripts executable by entering the command:

   ```
   ssh root@192.168.2.15 "chmod +x ~/*.sh "
   ```

Configure your MCU launching service to run scripts automatically (optional)

You can have MCU SDK scripts run automatically when you reboot your board. Scripts run in this manner do not have to be rerun each time you reboot your board manually. To automatically run one or more scripts, do the following:

1. In a serial communication session with your board, open the `mcu_fw_loader.sh` file using a text editor such as vi.

   ```
   vi /etc/intel_mcu/mcu_fw_loader.sh
   ```

2. For each script that you want to run, type `/folderpath/scriptname.sh`, where *`folderpath`* is the location where the script is stored and *`scriptname`* is the name of the script, as shown in the example below. Each script should be on its own line. Be sure to include any required arguments for running the script.
   For example, the following file is configured to run `init_i2c8.sh` and `init_DIG.sh`:

   ```
   #!/bin/sh
   #author: JiuJin Hong (jiujinx.hong@intel.com)
   if [ ! -d "/sys/devices/platform/intel_mcu" ];then
           exit
   fi

   if [ ! -f "/lib/firmware/intel_mcu.bin" ];then
           exit
   fi

   /home/root/init_i2c8.sh
   /home/root/init_DIG.sh -o 7 -d output

   echo "on" > /sys/devices/pci0000\:00/0000\:00\:17.0/power/control
   echo "load mcu app" > /sys/devices/platform/intel_mcu/control
   ```

Creating, building, and working with MCU projects

To create MCU projects for your Intel® Edison board, use the version of Eclipse* provided with the

MCU SDK. Here, you'll find information on the following topics:

- Creating a new MCU project
- Building your project
- Downloading your application to your board
- Uninstalling your application
- Connecting Eclipse to a different board
- Debugging MCU projects
- Updating the MCU SDK

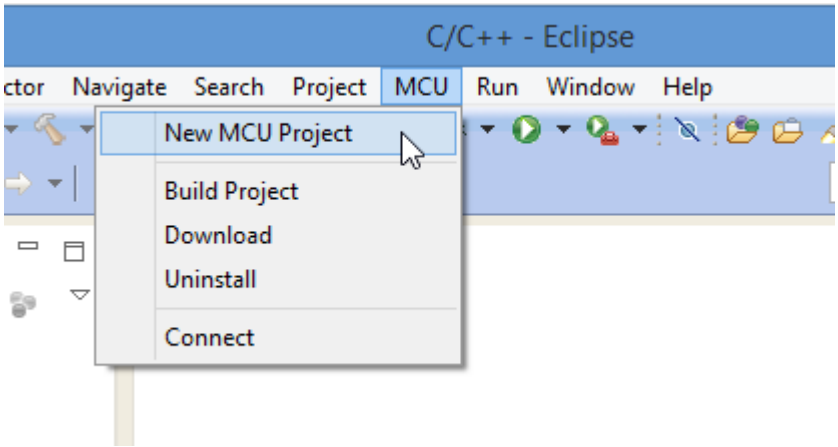
Prerequisites

- To download, uninstall, or debug MCU projects, your board must be connected to your system. For steps to set up and connect your board, see the appropriate link below:
    - Getting Started with the Intel® Edison board on Windows*
    - Getting Started with the Intel® Edison board on Mac*
    - Getting Started with the Intel® Edison board on Linux*
- You have already installed the MCU SDK version of Eclipse.
- Your system and your board are connected to the same local network. If you have a Mac* OS X* or Linux* system, you must set the IP address of your system to 192.168.2.1 and the subnet mask to 255.255.255.0. For steps to connect to your board, see Intel® Edison board: Connecting Ethernet over USB.

Creating a new MCU project

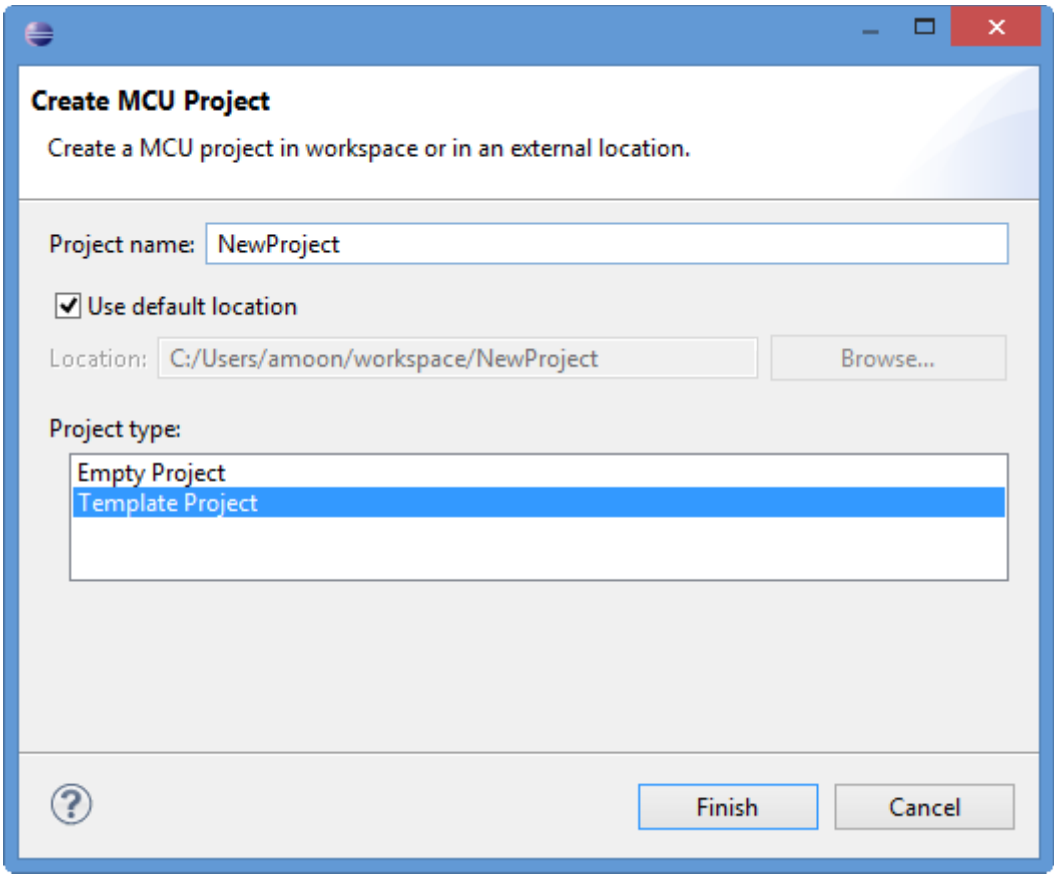
This section contains steps to create a new MCU project.

1. Choose **MCU** > **New MCU Project**.

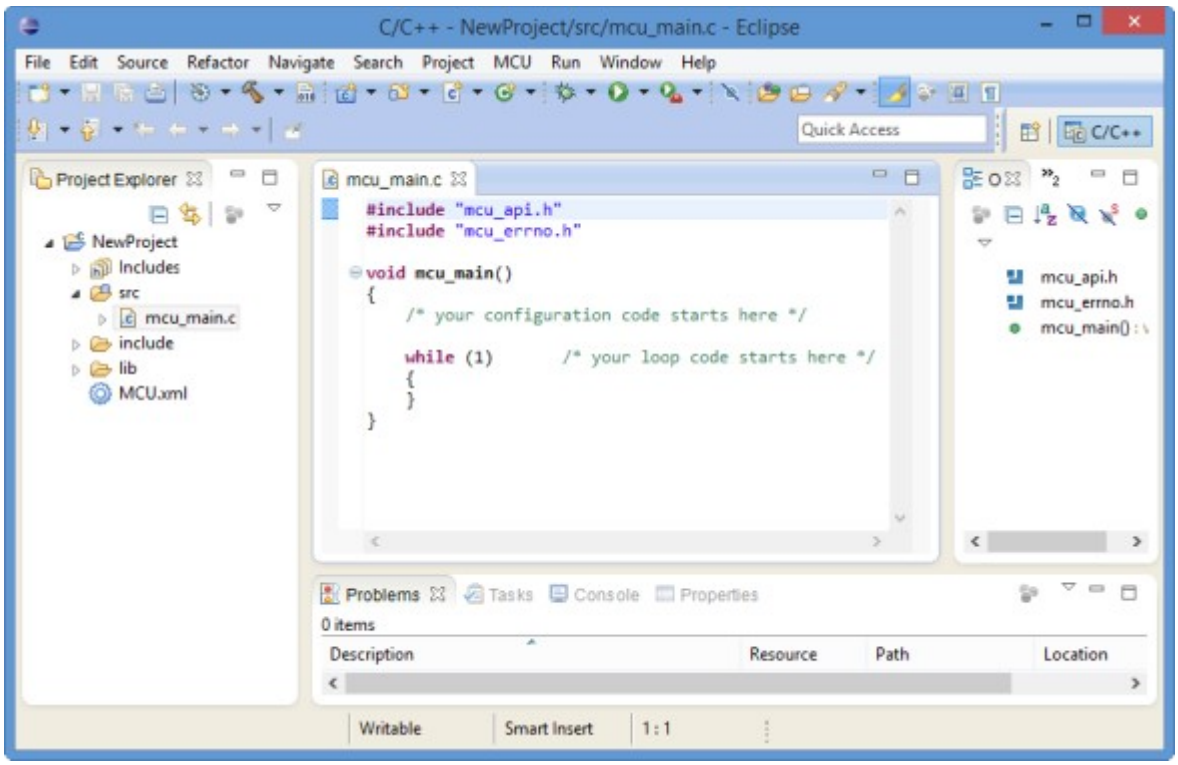


2. In the **Project name** field, type a name for the project.
3. You can define the location in which to store your project files. Clear the **Use default location** check box and click **Browse**. Navigate to and select the location to save the project files, then click **Open**.
4. In the **Project type** list, choose the type of project to create, as follows:

- To create a new project with a default main file and header files (recommended), select **Template Project**.
- To create a blank project, select **Empty project**.



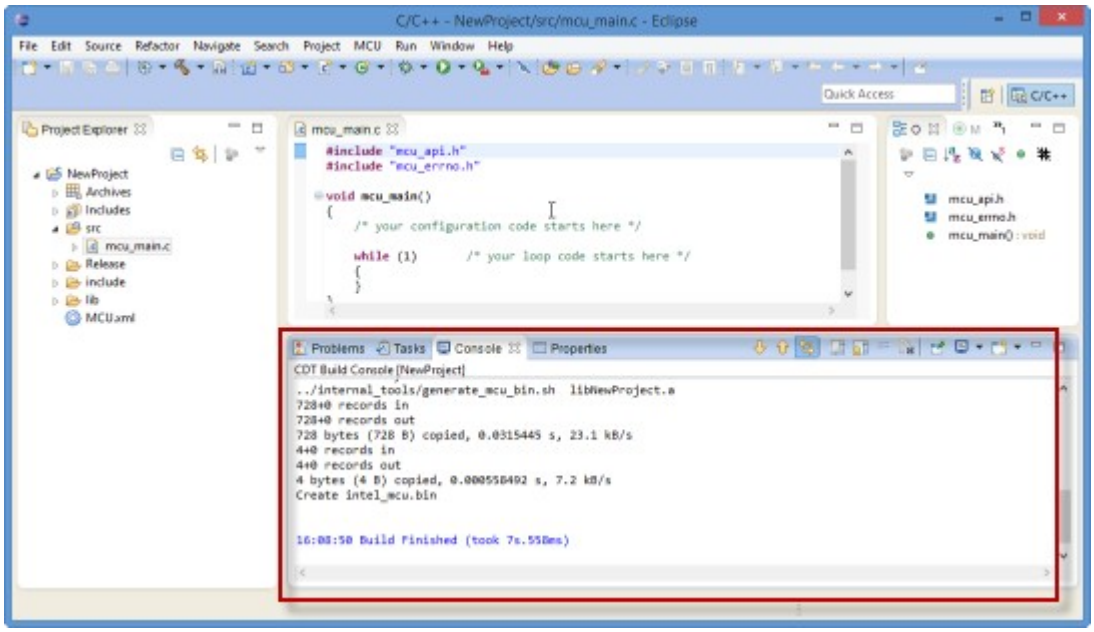
5. Click **Finish** to create your project. If you chose to create your project using a template, your default main file and header files are created for you, as shown in the image below.



Building your project

Once you've entered the code for your project, you're ready to build it by following these steps:

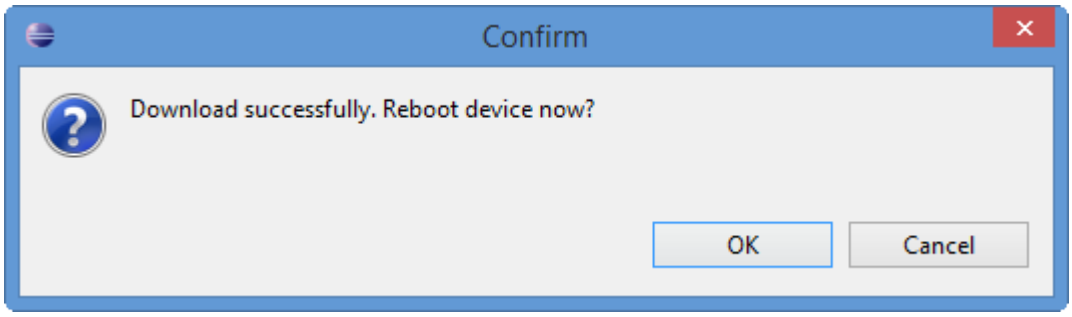
1. Choose **MCU > Build Project**.
2. A log of the build process is shown on the Console tab. Verify that your project builds successfully and a Build Finished message is displayed.



Downloading your application to your board

Now you are ready to download your application onto your board.

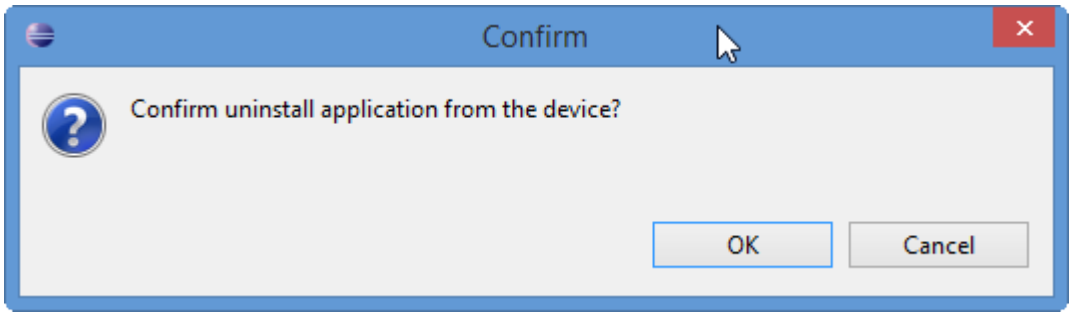
1. Choose **MCU** > **Download**. Your program is downloaded to your device.
2. Click **OK** to reboot and update your device. It takes one to two minutes for your board to restart.



Uninstalling your application

Once you have downloaded an application to your board, you must uninstall the application before loading the next version onto the board. Steps are below:

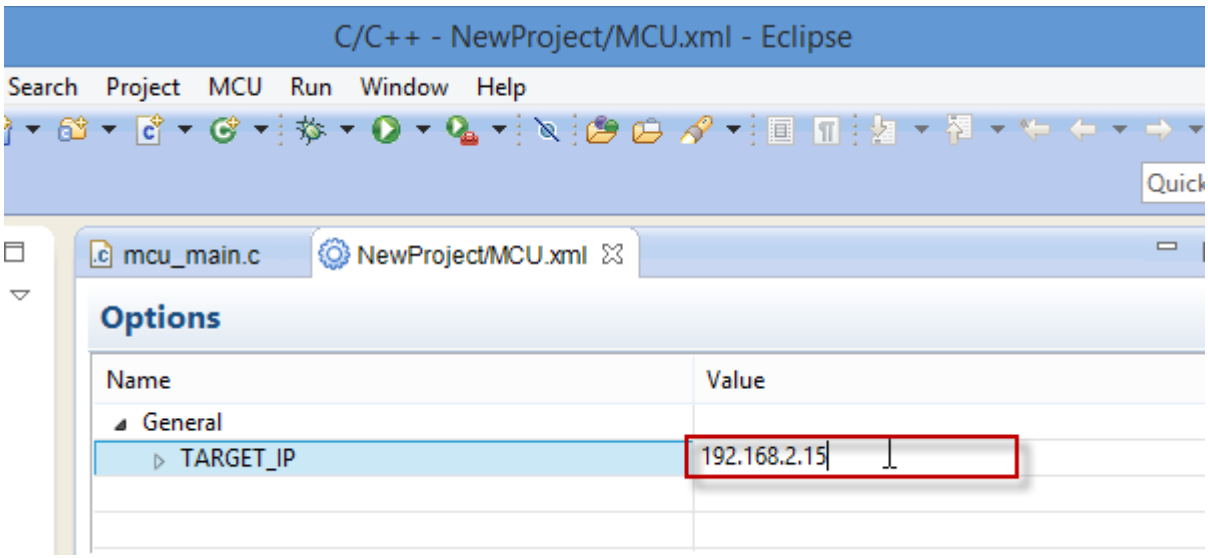
1. Choose **MCU** > **Uninstall**.
2. Click **OK** to uninstall the program and reboot your device. It takes one to two minutes for your board to restart.



Connecting Eclipse to a different board

By default, Eclipse connects to the Intel Edison board with the IP address 192.168.2.15. If you have multiple boards in the network, you can connect to a different board by specifying its IP address in Eclipse.

1. In the Project Explorer tab, right-click the **MCU.xml** file for the project, then choose **Open With** > **MCU Config Editor**.
2. The IP address of your board is displayed in the Value column next to TARGET_IP. Type the IP address of the device you want to access in the Value column and press Enter.

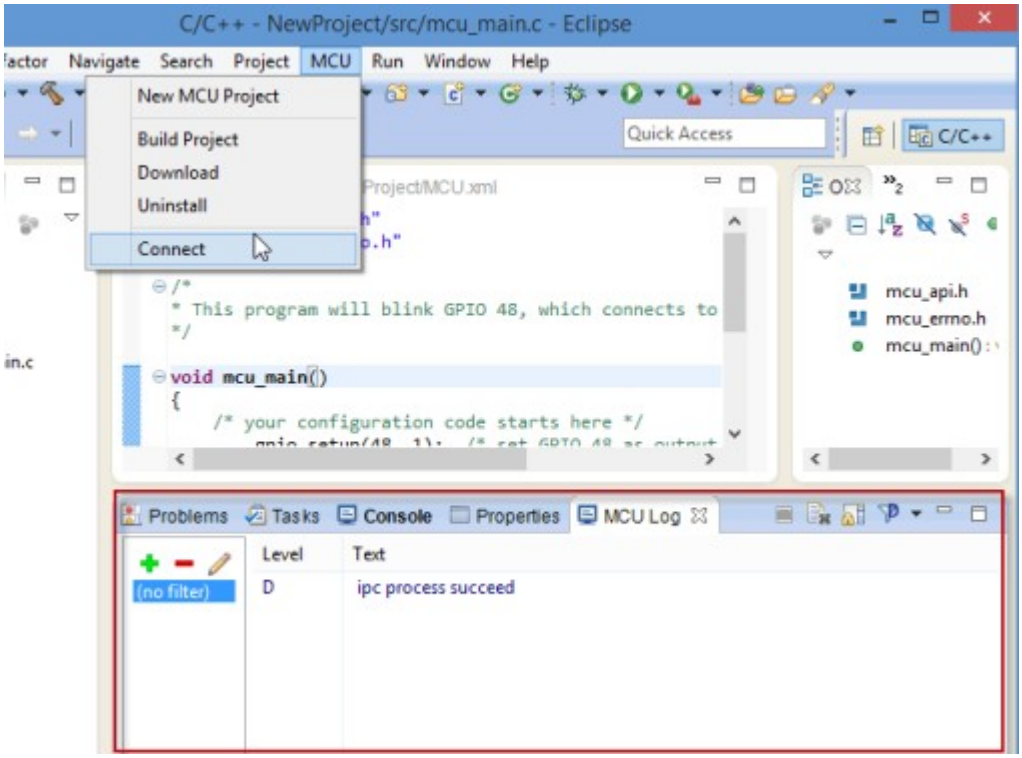


3. Choose **File** > **Save** to save your changes.

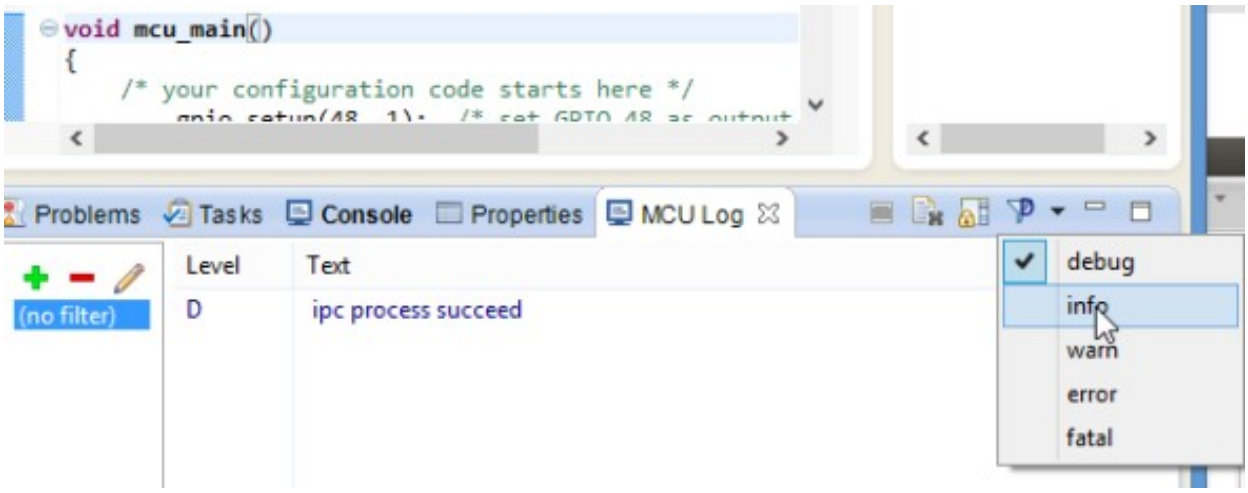
Debugging MCU projects

You can monitor the MCU's device trace via the MCU log tab for debugging purposes. You can also filter the messages displayed by log level. For example, you can display only warnings or fatal errors.

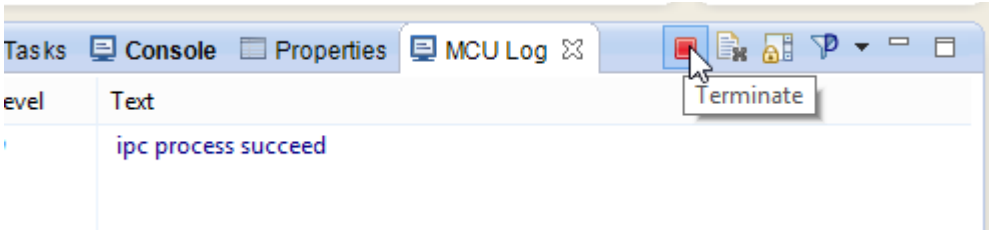
1. Choose **MCU** > **Connect**. The device trace is displayed on the MCU Log tab.



2. You can filter the entries displayed in the device trace. From the **Log Level** drop-down list, select the details you want to monitor.



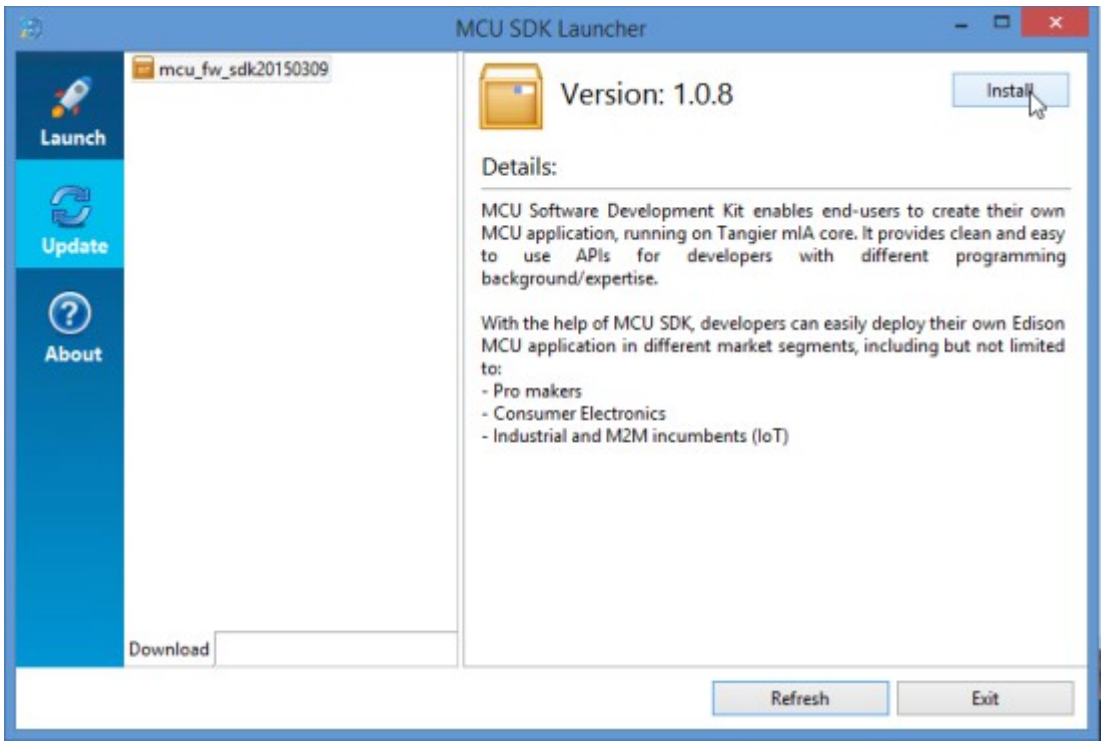
3. To stop monitoring the device trace, click the **Terminate** icon.



Updating the MCU SDK

When a new version of the MCU SDK is released, follow these steps to update it using the MCU SDK Launcher.

1. Download the .zip file containing the updated MCU SDK. Copy the .zip file to your *MCU_SDK_HOME*`/download/` folder, then do one of the following:
    - Windows: Double-click **mcusdk.exe**.
    - Linux/Mac OS X: Click **mcusdk**.
2. The MCU SDK Launcher opens. From the list on the left, click **Update**.
3. Select the package to install. Click **Install**. The MCU SDK files are updated.

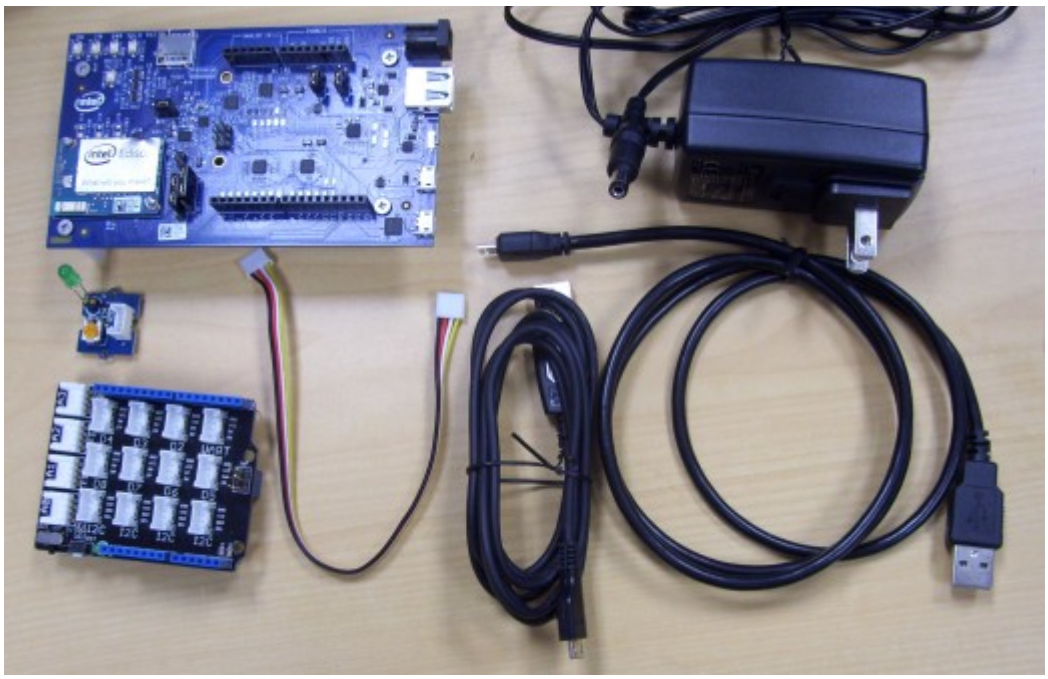


Blinking an LED using the MCU

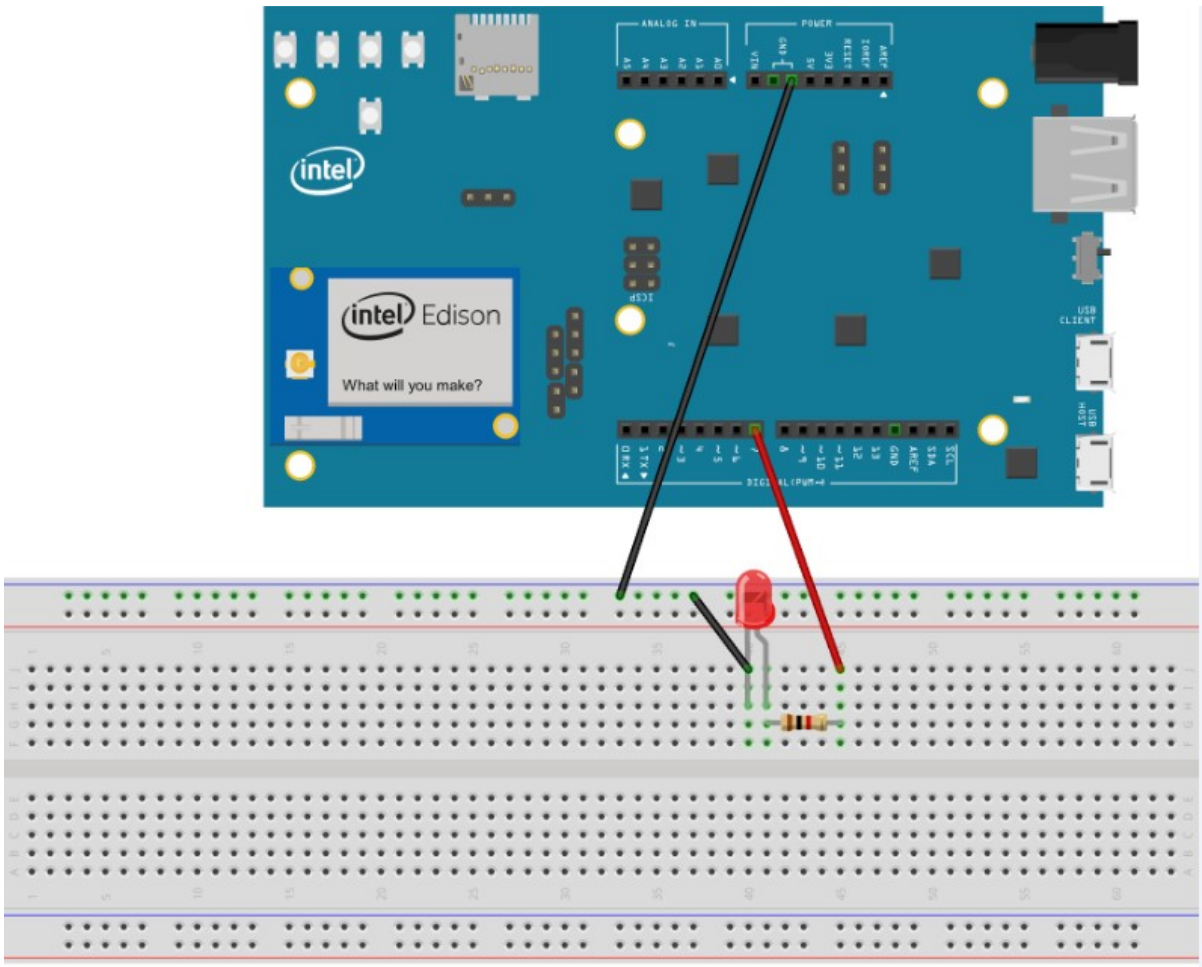
This section contains steps to blink an LED using the MCU on your Intel® Edison board.

Prerequisites

- You have set up your Intel® Edison board with the Arduino expansion board. For steps, see Assembling the Intel® Edison board with the Arduino expansion board.
- You have set up your Intel Edison board and established a serial communication session with it. For steps, see the appropriate link below:
    - Getting Started with the Intel® Edison board on Windows*
    - Getting Started with the Intel® Edison board on Mac*
    - Getting Started with the Intel® Edison board on Linux*
- You have installed the MCU SDK version of Eclipse*. For steps, see Installing the MCU SDK.
- You should have the following:
    - An Intel Edison board and Arduino expansion board
    - 1 micro USB cable
    - 1 DC power supply for your Intel Edison board
    - Grove* - Base Shield
    - Grove* - LED Socket Kit
    - Grove* Cable

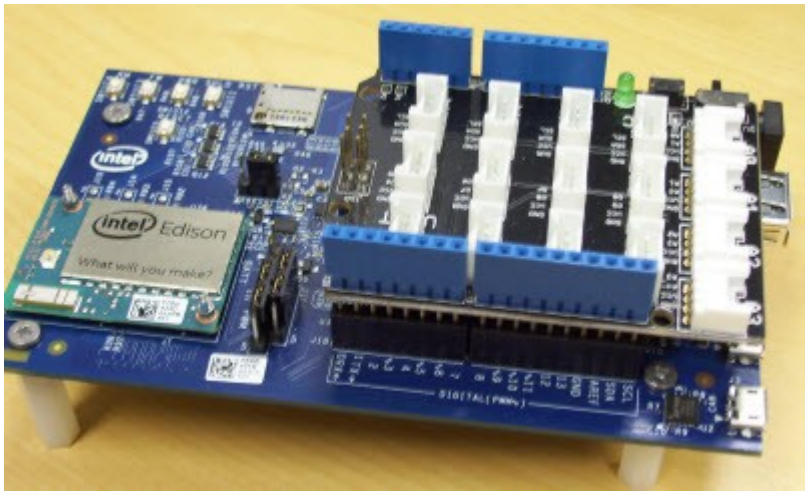


You can also create your LED circuit using a breadboard, jumper wires, and 5mm LEDs, as shown in the diagram below:

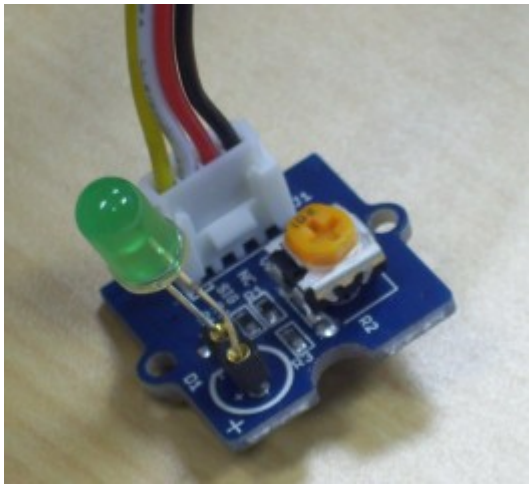


Set up your board and LED

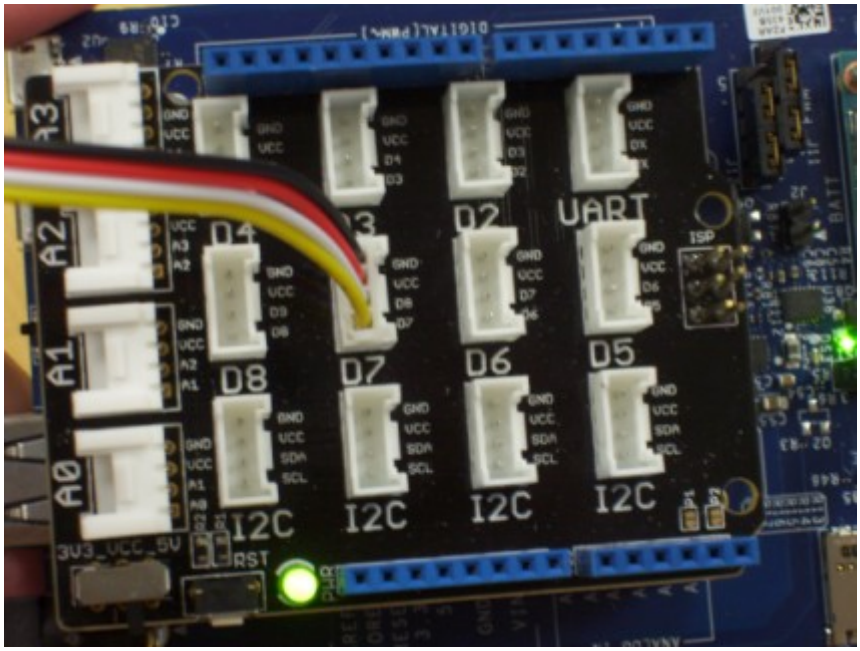
1. Line up the pins on the bottom of your base shield to fit the contacts on your board, as shown in the image below. Once aligned, gently push the base shield down towards your board until it is securely attached.



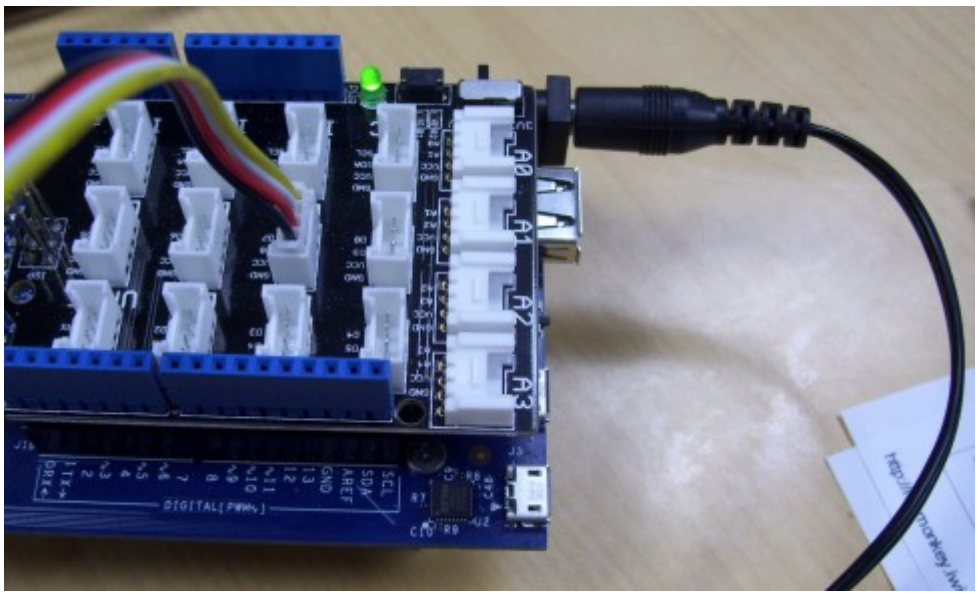
2. Slide the connector on one end of the cable into the socket on your LED socket kit until you feel the cable snap into place.



3. Slide the other end of the cable into the D7 socket on your base shield until you feel the cable snap into place.

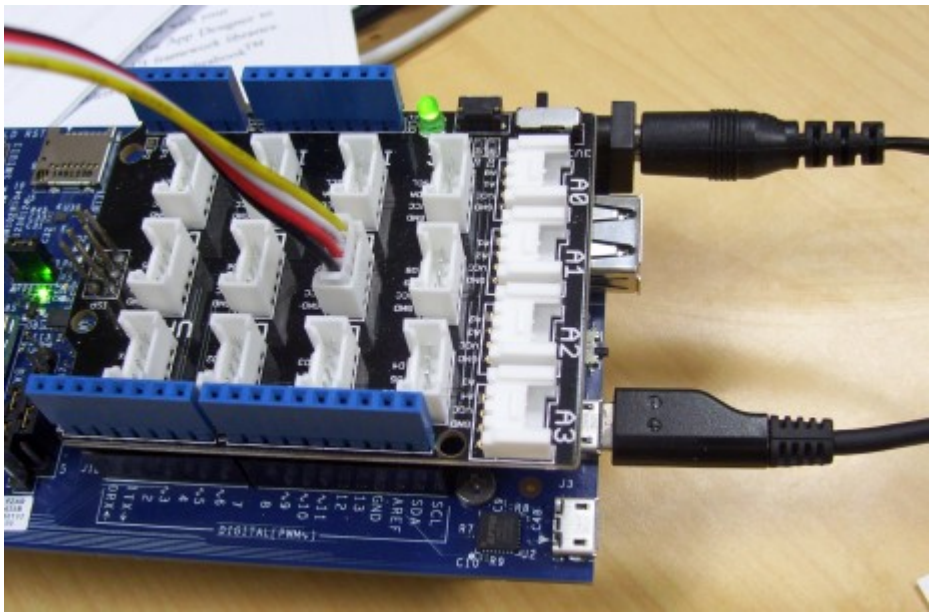


4. Plug in your board using the DC power supply.



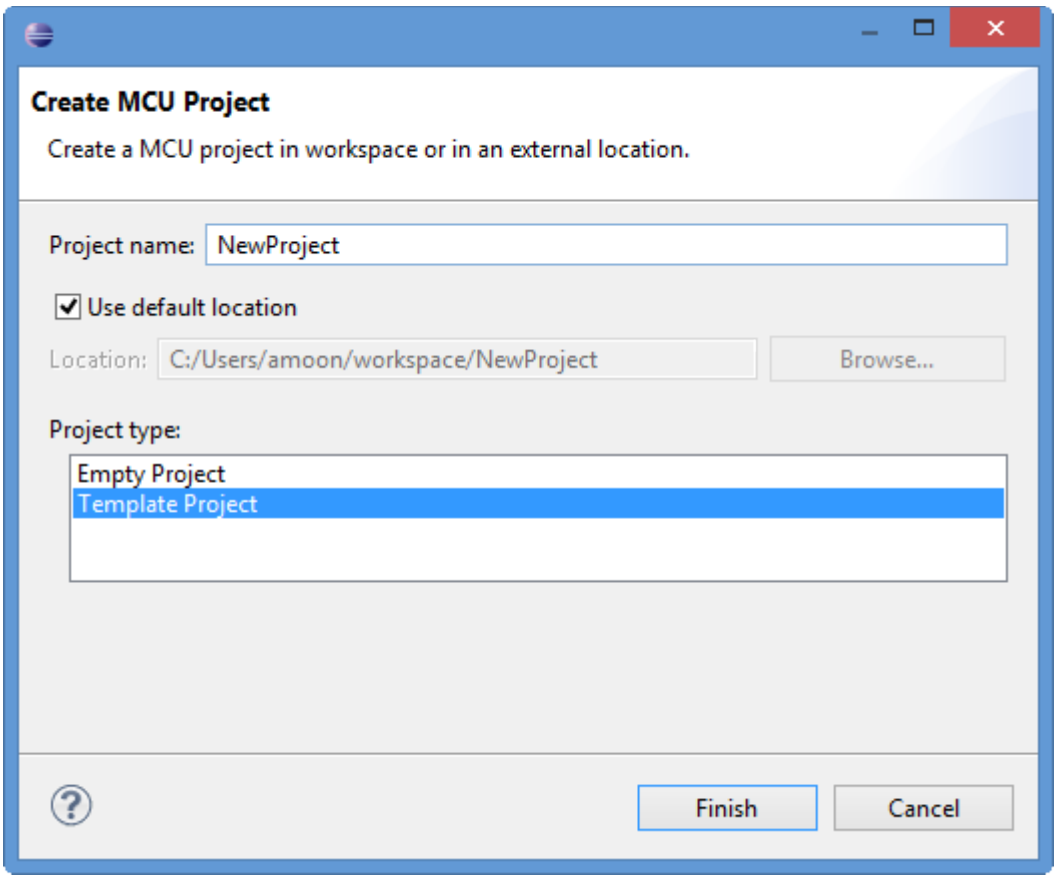
|

5. Plug the micro USB cable in to the port closer to the middle of the board, as shown below. Plug the other end in to your system.



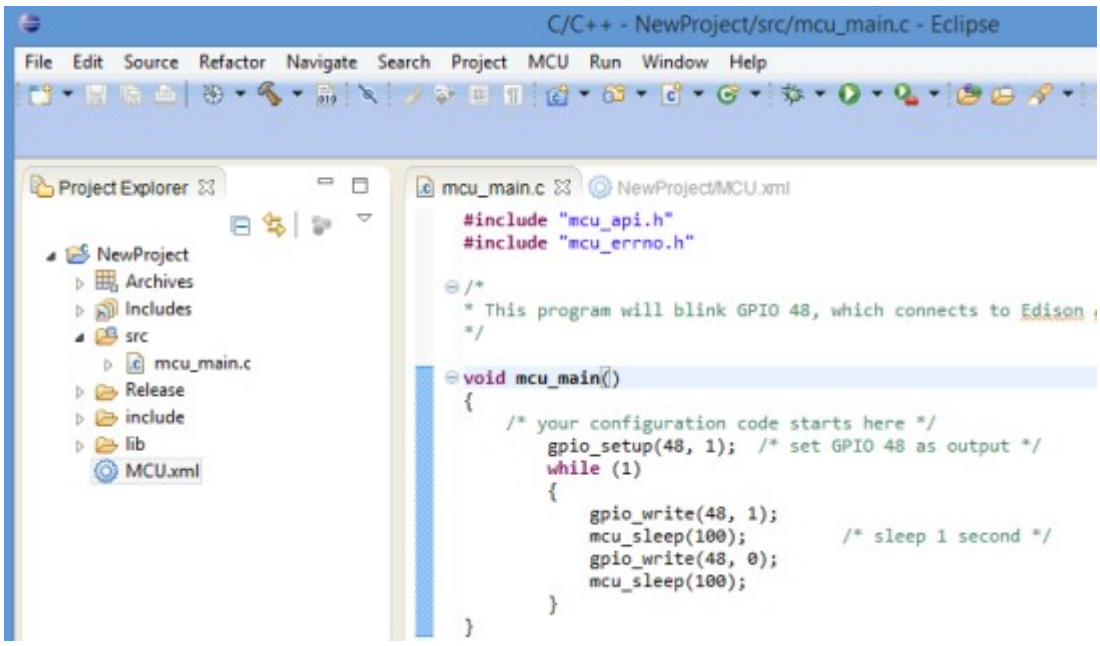
Create a project

1. In Eclipse, choose **MCU > New MCU Project**.
2. In the **Project name** field, type a name for the project.



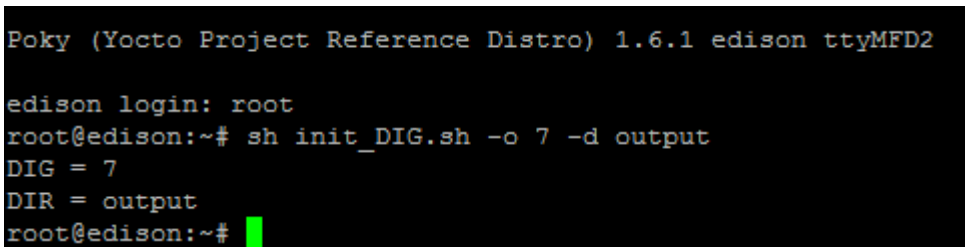
3. In the **Project type** list, select **Template Project**.
4. Click **Finish** to create your project. Your default main file and header files are created for you.
5. In your main file, paste the following code:

```
#include "mcu_api.h"
#include "mcu_errno.h"
/*
* This program will blink GPIO 48, which connects to Edison Arduino board
DIG 7
*/
void mcu_main()
{
    /* your configuration code starts here */
    gpio_setup(48, 1);  /* set GPIO 48 as output */
    while (1)
    {
        gpio_write(48, 1);
        mcu_sleep(100);         /* sleep 1 second */
        gpio_write(48, 0);
        mcu_sleep(100);
    }
}
```

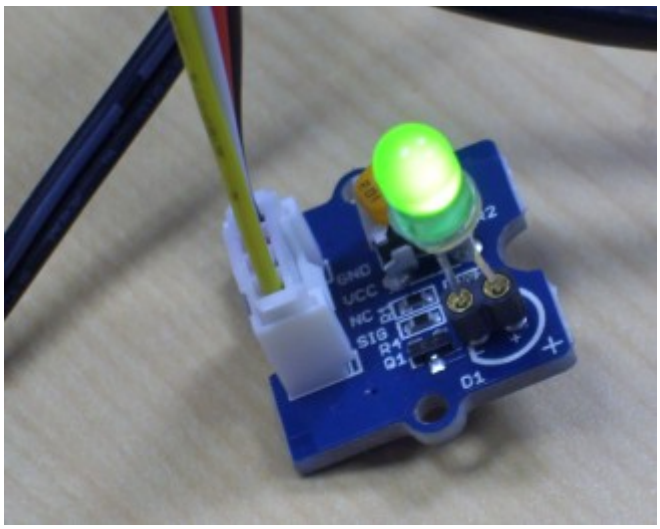


6. To build your project, choose **MCU** > **Build Project**. A log of the build process is shown on the Console tab. Verify that your project builds successfully and a Build Finished message is displayed.
7. Choose **MCU** > **Download**. Your program is downloaded to your device.
8. Click **OK** to reboot and update your device. Wait one or two minutes for the board to boot up.
9. In a serial communication session with your board, log in. To set up the GPIO configuration for the Arduino expansion board, enter the following command:

```
sh init_DIG.sh -o 7 -d output
```



The LED blinks on and off every second.



Note that the GPIO numbering for the CPU/MCU is different from the numbering for the Arduino DIG port, as shown in the table below:

Arduino DIG port	MCU GPIO port
0	130
1	131
2	128
3	12
4	129
5	13
6	182
7	48
8	49
9	183
10	41
11	43
12	42
13	40

Using the MCU SDK and API: Code examples

This section provides code examples for using the MCU SDK and accompanying API. See the links below:

- Reading the value of an I2C device and sending it back to the host
- Sending data via UART
- Controlling PWM ports
- Registering an interrupt handler to a GPIO port
- Communicating between the host and the MCU
- Waking up the host CPU using the MCU
- Getting debug messages from the MCU

- Setting the Intel® Edison board up to test system on a chip (SoC) GPIOs

Requirements

The examples below assume that you have the following:

- You have set up your Intel® Edison board with the Arduino expansion board. For steps, see Assembling the Intel® Edison board with the Arduino expansion board.
- You are familiar with the process of establishing a serial communication session with your board. For steps, see the appropriate link below:
    - Getting Started with the Intel® Edison board on Windows*
    - Getting Started with the Intel® Edison board on Mac*
    - Getting Started with the Intel® Edison board on Linux*
- You have installed the MCU SDK version of Eclipse*. For steps, see Installing the MCU SDK.
- Some of the examples may require additional hardware, such as breadboards, resisters, and jumper wires, depending on the device you are connecting to.

Reading the value of an I^2C device and sending it back to the host

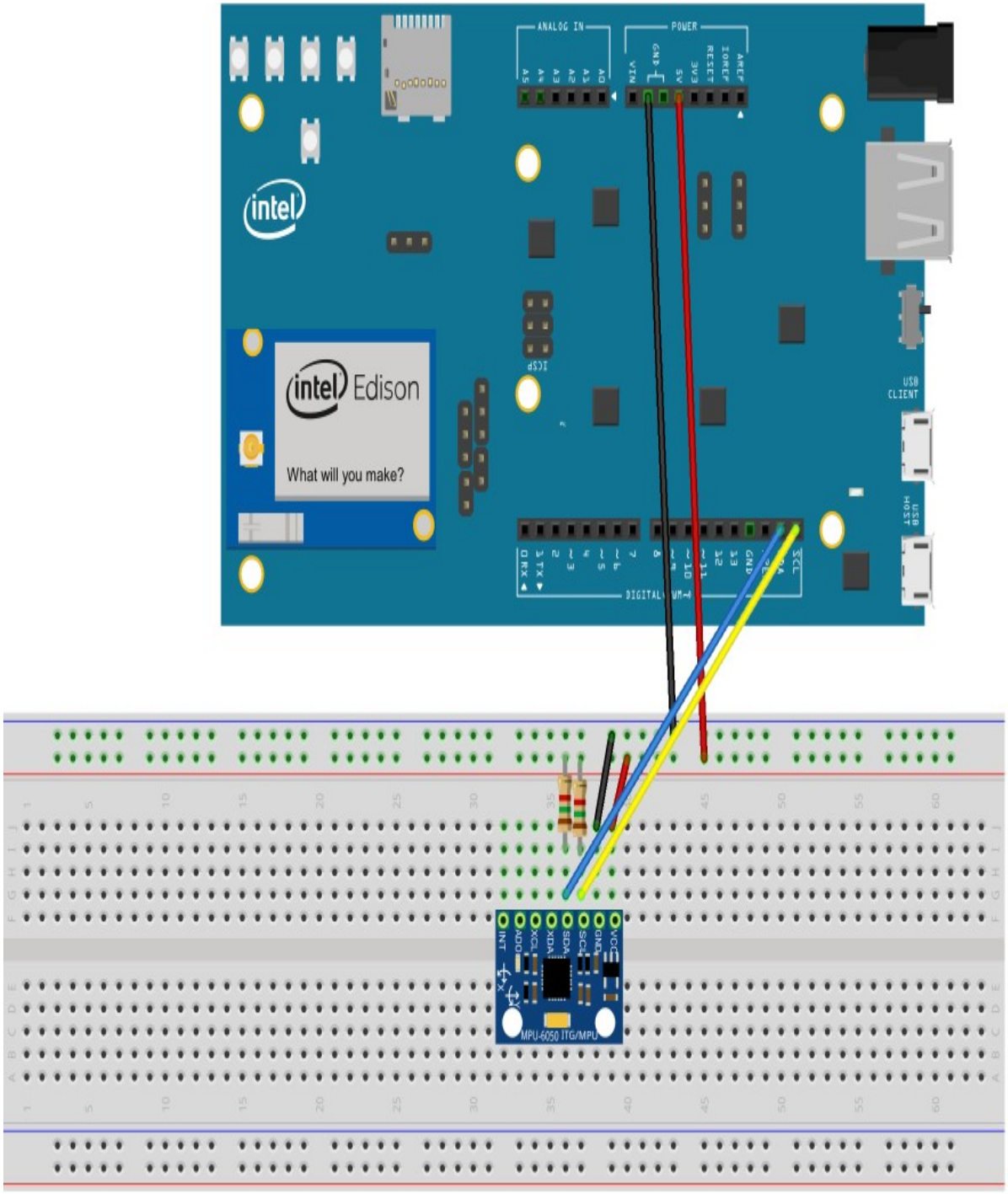
This example reads the WHOAMI register value of an MPU6050 sensor via the I^2C interface and sends it back to the host. You may need some extra hardware for this setup, including a breadboard, the correct value of resistors (as per the I^2C device spec), and a few jumper wires.

1. Create a new project from a template. For steps, see Creating a new MCU project.
2. In your main file, paste the following code:

```
#include "mcu_api.h"
#include "mcu_errno.h"

/* the I2C address of MPU6050 */
#define MPU6050_ADDR    0x68
/* the register address of WHOAMI register, which is used to verify
the identity of the device */
#define MPU6050_WHOAMI          ((unsigned char)0x75)
#define MPU6050_WHOAMI_VALUE    ((unsigned char)0x68)

static unsigned char mpu6050_read_whoami()
{
    unsigned char id;
    int res;
    res = i2c_read(MPU6050_ADDR, MPU6050_WHOAMI, &id, 1);
    if (res)
    {
        debug_print(DBG_ERROR, "mpu6050_i2c_read whoami fail...");
        return 0Xff;
    }
    if (id != MPU6050_WHOAMI_VALUE)
    {
        debug_print(DBG_ERROR, "wrong chip ID 0x%x, expected 0x%x\n",
                    id, MPU6050_WHOAMI_VALUE);
```

```
        return 0xff;
    }
    return id;
}

void mcu_main()
{
    unsigned char device_id;
    char buf[16];
    int len;
    while (1)
    {
        mcu_sleep(300);
        debug_print(DBG_INFO, "after sleep 3 seconds\n");
        device_id = mpu6050_read_whoami();
        len = mcu_snprintf(buf, 16, "id = %d\n", device_id);
        host_send((unsigned char*)buf, len);
    }
}
```

3. Set up an I^2C circuit between the MPU6050 sensor and the Intel Edison board. Connect the SDA, SCL, VCC and GND ports of the I^2C interface, as shown in the image below. In our example, we have used an MPU-6050 triaxial gyroscope + accelerometer sensor module for Arduino.



4. To build your project, choose **MCU > Build Project**. A log of the build process is shown on the Console tab. Verify that your project builds successfully and a Build Finished message is displayed.
5. Choose **MCU > Download**. Your program is downloaded to your device.
6. Click **OK** to reboot and update your device. Wait one or two minutes for the board to boot up.
7. Run `init_i2c8.sh` from the Edison shell.
8. Run `cat /dev/ttymcu0` to fetch the data from the MCU that the host_send() API sent, as shown in the example below:

```
root@edison:~# cat /dev/ttymcu0
id = 104
```

```
id = 104
id = 104
```

You should be able to see the output every 3 seconds.

Sending data via UART

Because the host and MCU share UART2 (for example, ttyMFD2), it is possible to send data to UART2 via the MCU. To create an MCU project to send data via UART, do the following:

1. Create a new project from a template. For steps, see Creating a new MCU project.
2. In your main file, paste the following code:

```
#include "mcu_api.h"
#include "mcu_errno.h"
/*
* This program sends a string to UART2 every 3 seconds
*/
void mcu_main()
{
    /* your configuration code starts here */
    uart_setup(2, 115200);

    while (1)
    {
        uart_write(2, (unsigned char*)"hello mcu\r\n", 11);
        mcu_sleep(300);
    }
}
```

3. To build your project, choose **MCU > Build Project**. A log of the build process is shown on the Console tab. Verify that your project builds successfully and a Build Finished message is displayed.
4. Choose **MCU > Download**. Your program is downloaded to your device.
5. Click **OK** to reboot and update your device. Wait one or two minutes for the board to boot up.
6. If it is not already connected, connect the Arduino serial port to your computer.
7. If you are successful, you will see the following output from the MCU:

```
hello mcu
hello mcu
hello mcu
```

Controlling PWM ports

The MCU can access all PWM ports, from PWM 0 to PWM 3. The PWM period can vary from 104 ns to 218,453,000 ns. This example shows how the MCU can control the PWM port.

1. Create a new project from a template. For steps, see Creating a new MCU project.
2. In your main file, paste the following code:

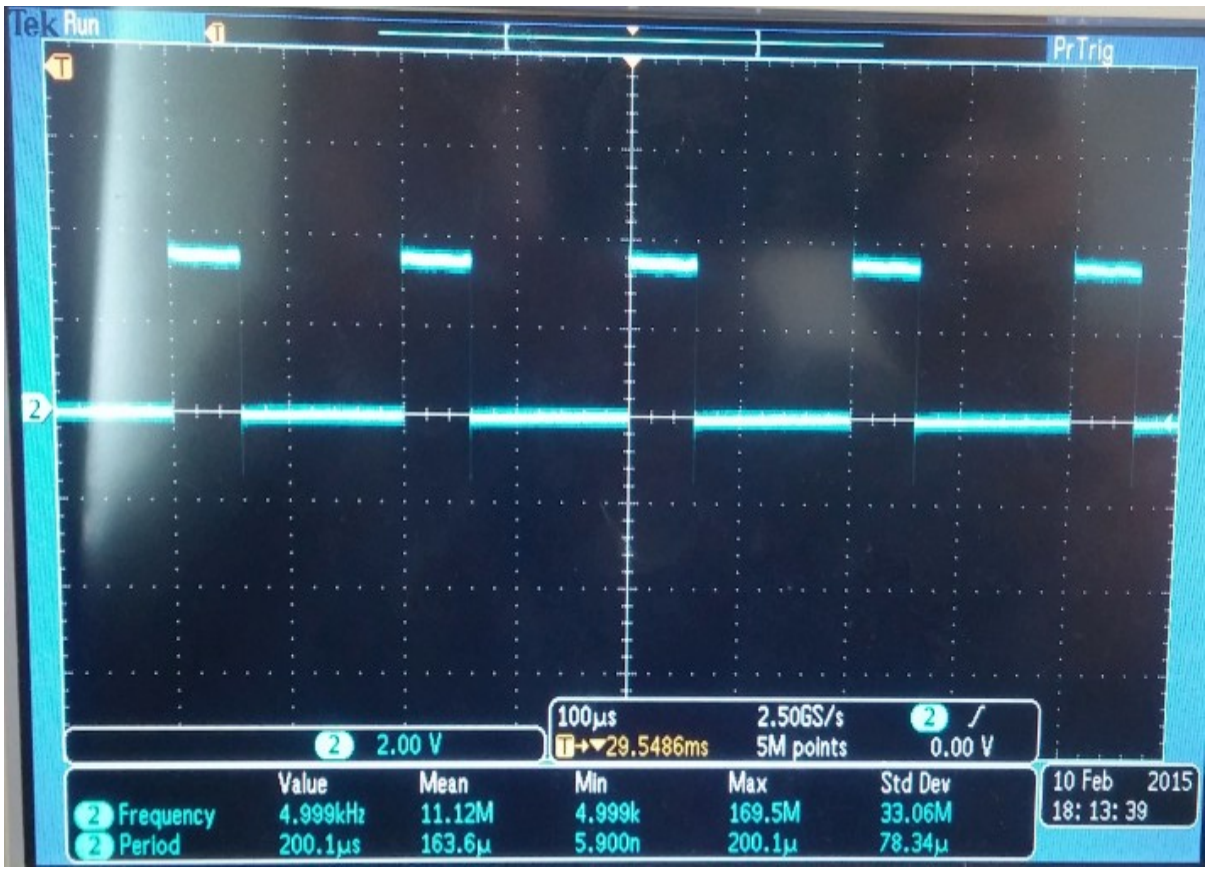
```
#include "mcu_api.h"
#include "mcu_errno.h"

#define PORT 0
```

```
void mcu_main()
{
        /* Configure PWM and enable it */
        pwm_configure(PORT, 60000, 200000);
        pwm_enable(PORT);

        while (1)
        {
                mcu_sleep(2000);
                debug_print(DBG_INFO, "I am still alive...\n");
        }
}
```

3. Edit the `/etc/intel_mcu/mcu_fw_loader.sh` file to disable the PWM power management on the Linux* side, as shown below. This enables the MCU to configure the PWM modules.

```
root@edison:/etc/intel_mcu# cat mcu_fw_loader.sh
#!/bin/sh
#author: JiuJin Hong (jiujinx.hong@intel.com)
if [ ! -d "/sys/devices/platform/intel_mcu" ];then
        exit
fi
if [ ! -f "/lib/firmware/intel_mcu.bin" ];then
        exit
fi
echo "on" > /sys/devices/pci0000\:00/0000\:00\:17.0/power/control
echo "load mcu app" > /sys/devices/platform/intel_mcu/control
```

4. To build your project, choose **MCU** > **Build Project**. A log of the build process is shown on the Console tab. Verify that your project builds successfully and a Build Finished message is displayed.
5. Choose **MCU** > **Download**. Your program is downloaded to your device.
6. Click **OK** to reboot and update your device. Wait one or two minutes for the board to boot up.
7. Run **init_mcu_PWM.sh** to set up the PWM configuration for the Arduino expansion board.
8. Measure the output of DIG 3, which maps to PWM 0. You should see output on the DIG output screen similar to the image below.



The table below shows PWM port numbers mapped to Arduino DIG numbers by default.

PWM port	0	1	2	3
Arduino DIG port	3	5	6	9

Registering an interrupt handler to a GPIO port

This example shows how you can register an interrupt handler for the MCU.

1. Create a new project from a template. For steps, see Creating a new MCU project.
2. In your main file, paste the following code:

```
 #include "mcu_api.h"
#include "mcu_errno.h"


 /*
  * This program use GPIO 48 as an interrupt handler, which connects to
  * Edison Arduino board DIG 7. GPIO 49 (DIG 8)is simulated as an
interrupt
  * input, which triggers the ISR that defined in GPIO 48.
 */

 int my_irq(int req)
 {
   debug_print(DBG_INFO, "Interrupt triggered!\n");
   return IRQ_HANDLED;
 }

 void mcu_main()
```

```
 {
    /* your configuration code starts here */
    gpio_setup(48, 0);  /* set GPIO 48 as input*/
    gpio_setup(49, 1);  /* set GPIO 49 (DIG 8) as output and as an
interrupt trigger */
    gpio_register_interrupt(48, 0, my_irq); /* define the ISR on GPIO 48
*/
    while (1)
    {
debug_print(DBG_INFO, "I am in main loop!\n");
gpio_write(49, 0);
mcu_sleep(300);         /* sleep 3 second */
gpio_write(49, 1);
mcu_sleep(300);         /* sleep 3 second */
    }
 }
```

3. Connect the Arduino digital port 7 to digital port 8 with a wire. DIG 7 is used as an interrupt input and DIG 8 is used to simulate triggering an interrupt.
4. Power on the device.
5. To build your project, choose **MCU > Build Project**. A log of the build process is shown on the Console tab. Verify that your project builds successfully and a Build Finished message is displayed.
6. Choose **MCU > Download**. Your program is downloaded to your device.
7. Click **OK** to reboot and update your device. Wait one or two minutes for the board to boot up.
8. After reboot, run `init_DIG.sh -o 7 -d input` and `init_DIG.sh -o 8 -d output` to set up the GPIO configuration for the Arduino expansion board.

```
root@edison:~# ./tools/init_DIG.sh -o 7 -d input
DIG = 7
DIR = input
root@edison:~# ./tools/init_DIG.sh -o 8 -d output
DIG = 8
DIR = output
root@edison:~#
```

9. Run `cat /dev/ttymcu1` to fetch the debug message from the MCU. You can see log messages from both main thread and handler function of the GPIO interrupt, as shown in the example output below.

```
root@edison:~# cat /dev/ttymcu1
( INFO): I am in main loop!
( INFO): Interrupt triggered!
( INFO): I am in main loop!
( INFO): Interrupt triggered!
( INFO): I am in main loop!
( INFO): Interrupt triggered!
^C
```

Communicating between the host and the MCU

This sample shows how the MCU can retrieve a command from the host and send a response to the host.

1. Create a new project from a template. For steps, see Creating a new MCU project.
2. In your main file, paste the following code:

```
#include "mcu_api.h"
#include "mcu_errno.h"
#include <string.h>

void mcu_main()
{
    /* your configuration code starts here */
    char buf[64];
    int len;

    while (1)       /* your loop code starts here */
    {
        do {
            len = host_receive((unsigned char *)buf, 64);
            mcu_sleep(10);
        } while (len <= 0);
        if (strncmp(buf, "start", 5) == 0)
        {
            debug_print(DBG_INFO, "received start command!\n");
            host_send((unsigned char*)"hello mcu\n", 10);
        }
    }
}
```

3. To build your project, choose **MCU** > **Build Project**. A log of the build process is shown on the Console tab. Verify that your project builds successfully and a Build Finished message is displayed.
4. Choose **MCU** > **Download**. Your program is downloaded to your device.
5. Click **OK** to reboot and update your device. Wait one or two minutes for the board to boot up.
6. Run `echo "start" > /dev/ttymcu0` on the target device.
7. Run `cat /dev/ttymcu0` to retrieve the response from the MCU, as shown in the example output below:

```
root@edison:~# cat /dev/ttymcu0
^C
root@edison:~# echo “start” > /dev/ttymcu0
root@edison:~# cat /dev/ttymcu0
hello mcu
^C
```

As an alternative, you can also get the response using C code. The following is a code example for communicating with the host using host-side C code:

```
#include <stdio.h>   /* Standard input/output definitions */
#include <fcntl.h>   /* File control definitions */
int main()
{
    int fd;
    unsigned char data;
    unsigned char buf[100];
    int i, n;
    fd = open("/dev/ttymcu0", O_RDWR | O_NOCTTY);
    if (fd == -1)
    {
```

```
        printf("open ttymcu0 failed!\n");
        return 1;
    }

    i = 0;

    do
    {
        n = read(fd, &data, 1);
        buf[i] = data;
        i++;
    } while(data != '\n' && i < 100);

    i = (i >=99? 99:i);
    buf[i] = '\0';
    printf("Response from MCU: %s", buf);

    close(fd);
    return 0;
}
```

Waking up the host CPU using the MCU

In situations when power consumption is critical, you can have the host CPU rest in a sleep status while the MCU is kept active to collect data and wake the CPU only when necessary. The MCU can communicate with the host via the host_send and host_receive APIs. The MCU will wake up the host using host_send. This example shows how to set the MCU to wake the CPU every two minutes.

1. Create a new project from a template. For steps, see Creating a new MCU project.
2. In your main file, paste the following code:

```
#include "mcu_api.h"
#include "mcu_errno.h"


void mcu_main()
{
    /* your configuration code starts here */

    while (1)       /* your loop code starts here */
    {
        mcu_sleep(12000);   /* wake up CPU every 2 minutes */
        host_send((unsigned char*)"wake up!\n", 10);
    }
}
```

3. To build your project, choose **MCU** > **Build Project**. A log of the build process is shown on the Console tab. Verify that your project builds successfully and a Build Finished message is displayed.
4. Choose **MCU** > **Download**. Your program is downloaded to your device.
5. Click **OK** to reboot and update your device. Wait one or two minutes for the board to boot up.
6. Check the count of interrupts from the MCU using the `cat /proc/interrupts` command, and the output from `/dev/ttymcu0`, as shown below:

```
Poky (Yocto Project Reference Distro) 1.6.1 edison ttyMFD2

edison login: root
root@edison:~#
root@edison:~# cat /proc/interrupts | grep intel_psh_ipc
 47:          1          1   IO-APIC-fasteoi   intel_psh_ipc
root@edison:~# cat ttymcu0
^C
root@edison:~#
```

Note: To enable power management of the serial port, you might need to run the following command:

```
echo auto >
/sys/devices/pci0000\:00/0000\:00\:04.3/power/control
```

This lets the CPU enter into sleep mode when there is no input from the serial port.

7. Wait for several minutes and check again. You should be able to see more interrupts from the MCU, which will wake up the CPU, and the strings from the MCU on `/dev/ttymcu0`, as shown in the example below.

```
root@edison:~# cat /proc/interrupts | grep intel_psh_ipc
47: 2 2 IO-APIC-fasteoi intel_psh_ipc
root@edison:~# cat ttymcu0
wake up!
wake up!
^C
root@edison:~#
```

Getting debug messages from the MCU

On the Linux side, there is a TTY driver, which can communicate with MCU to get debug messages.
The sample below shows how the MCU can output messages to the Linux side with different log levels. You can get debug messages from the MCU through the TTY node.

1. Create a new project from a template. For steps, see Creating a new MCU project.
2. In your main file, paste the following code:

```
#include "mcu_api.h"
#include "mcu_errno.h"
/*
* This program outputs debug messages to host in different loglevel every
10 seconds
*/
void mcu_main()
{
    /* your configuration code starts here */
    while (1)       /* your loop code starts here */
   {
        debug_print(DBG_DEBUG, "mcu dbg message print in DEBUG
level...\n");
        debug_print(DBG_INFO, "mcu dbg message print in INFO level...\n");
        debug_print(DBG_WARNING, "mcu dbg message print in WARNING
level...\n");
        debug_print(DBG_ERROR, "mcu dbg message print in ERROR
```

```
level...\n");
        debug_print(DBG_FATAL, "mcu dbg message print in FATAL
level...\n");
        mcu_sleep(1000);
    }
}
```

3. To build your project, choose **MCU** > **Build Project**. A log of the build process is shown on the Console tab. Verify that your project builds successfully and a Build Finished message is displayed.
4. Choose **MCU** > **Download**. Your program is downloaded to your device.
5. Click **OK** to reboot and update your device. Wait one or two minutes for the board to boot up.
6. Connect to the Intel Edison board, and run:
    a. `cat /dev/ttymcu1` to retrieve debug message from MCU.
    b. `cat /sys/devices/platform/intel_mcu/log_level` to get the current debug log level in the MCU.
    c. `echo debug > /sys/devices/platform/intel_mcu/log_level` to set the debug log level for the MCU.

```
root@edison:~# cat /dev/ttymcu1
(DEBUG) : mcu dbg message print in DEBUG level...
( INFO) : mcu dbg message print in INFO level...
( WARN) : mcu dbg message print in WARN level...
(ERROR) : mcu dbg message print in ERROR level...
(FATAL) : mcu dbg message print in FATAL level...
root@edison:~#
root@edison:~# cat /sys/devices/platform/intel_mcu/log_level
debug
root@edison:~#
root@edison:~# echo "debug" >
/sys/devices/platform/intel_mcu/log_level
root@edison:~#
```

If you want to get messages from all available log levels from the MCU, you should set log_level to debug, since debug is the highest level. MCU will output the messages with levels lower than the current level set. The MCU log level supports five kinds of levels, which are fatal, error, warning, info, and debug (in order from lowest to highest level).

Setting the Intel® Edison board up to test system on a chip (SoC) GPIOs

This section provides information to set up the tests for the GPIOs on an Intel Edison board. In order to thest the GPIOs, you must upload a series of scripts to your board and enable them to be executed using the chmod command. If you have already uploaded these scripts, as described in Loading the MCU SDK scripts onto your board, you should not need to repeat this portion of the steps.

Note: Only the GPIOs for the SoC that are available on the Arduino shield have been tested.

Requirements

1. In order to test the GPIOs, you need to initialize the board. To do so, download the files named **init_DIG.sh**, **set_DIG.sh**, and **read_DIG.sh** on the host system in the /home/lab/GPIO directory.
2. Open a shell and run the following command to send the files to the board:

   ```
   scp ~/GPIO/*.sh root@192.168.2.15:~
   ssh root@192.168.2.15 "chmod +x ~/*_DIG.sh "
   ```

   Note: Export the IP_ADDR variable (optional). To avoid setting the Intel Edison board's IP address on each test step, you can set it to the current shell environment using the following command:
   `IP_ADDR=192.168.2.15`

   Or you can set it for the profile (every shell) by editing the **~/.profile** file, and adding the following line to the end:
   `export IP_ADDR=192.168.2.15`

   Note: To test the GPIOs, use the script **init_DIG.sh** to set up the multiplexers to have the SoC's GPIO on the shield.
   `root@edison:~# init_DIG.sh -o <number> -d <direction>`

   ...where <number> is the digital output number and <direction> is input or output mode.
3. The following are examples of commands to work with output:
   a. Initializing the DIG3 output:
      `root@edison:~# ./init_DIG.sh -o 3 -d output`
   b. Setting the output to high level:
      `root@edison:~# ./set_DIG.sh -o 3 -v high`
   c. Setting the output to low level:
      `root@edison:~# ./set_DIG.sh -o 3 -v low`
4. The following are examples of commands to work with input:
   a. Initializing the DIG3 output:
      `root@edison:~# ./init_DIG.sh -o 3 -d input`
   b. Reading the input status (low or high):
      `root@edison:~# ./read_DIG.sh -o 3`

MCU API

The MCU API provides an interface that allows you to do the following:

- Control the peripherals that connect to the MCU, including GPIO, I^2C, and UART.
- Communicate with the host system.
- Perform auxiliary functions.

For more information on the MCU API, see

MCU_SDK_HOME/`docs/api_doc/html/index.html`, where *MCU_SDK_HOME* is your MCU SDK installation folder.

Note: The MCU API does not support floating point. Refer to *mcu_api.h* for detailed definitions of each function.

APIs to access HW resources

- APIs to access GPIO
- APIs to access I2C
- APIs to access UART
- APIs to communicate with host
- APIs to control PWM
- Auxiliary API

APIs to access GPIO

`gpio_setup`

Set up the input or output mode for one GPIO port. It is okay to call this API with different parameters for the same GPIO port. For example, you can set up a GPIO as an input and later as an output.

```
void gpio_setup(int gpio, int direction)
```

...where:

- gpio: GPIO port number
- direction: 0 = input, 1 = output

`gpio_register_interrupt`

Register an interrupt handler for one GPIO port.

```
int gpio_register_interrupt(int gpio, int mode, irq_handler_t
isr);
```

...where:

- gpio: GPIO port number
- mode: 0 = falling edge interrupt, 1 = rising edge interrupt
- isr: interrupt handler function
- Return value: 0 if successful

`gpio_read`

Read the value of one GPIO port.

```
int gpio_read(int gpio)
```

...where:

- gpio: GPIO port number

- return value: 0 = low voltage, 1 = high voltage

`gpio_write`

Write the value of one GPIO port.

```
void gpio_write(int gpio, int value)
```

...where:

- gpio: GPIO port number
- value: 0 = low voltage, 1 = high voltage

APIs to access I^2C

MCU can access I^2C devices that connect to I^2C8 bus, which is muxed with the I^2C6 bus. You can easily get the raw data from the I^2C device via MCU API. The MCU uses I^2C-8, whereas the Intel® Atom™ processor uses I^2C-6. You must program pin muxing before using I^2C from Intel® Atom™ or MCU. By default, muxing is I^2C-6, for example, for the Intel® Atom™ chip. For details, see Getting debug messages from the MCU.

`i2c_read`

Read the value of the register from a I^2C device.

```
int i2c_read(int address, int reg, unsigned char *buf, int length)
```

...where:

- address: I2C device address
- reg: register address
- buf: preallocated buffer for output values
- length: length of the buffer
- return value: 0 if successful

Note: We strongly recommend you do not use malloc in Viper. For MCU application development, we only support static buffers.

`i2c_write`

Write the value to the register of an I^2C device.

```
int i2c_write(int address, int reg, unsigned char *buf, int
length)
```

...where:

- address: I^2C device address
- reg: register address
- buf: buffer containing input values
- length: length of the buffer

- return value: 0 if successful

Note: MCU application can only access I^2C8.

APIs to access UART

The MCU can access UART1 as well as UART2, which may be used as Linux kernel console as well. UART2 (ttyMFD2) is shared by both the host and the MCU. You can disable the output from the host side by running `systemctl stopserial-getty@ttyMFD2.service` on the Linux side.

`uart_setup`

Set up the baud rate of a UART device.

```
int uart_setup(unsigned char port, int baudrate)
```

...where:

- port: UART port number:1 or 2
- baudrate: baud rate of the UART
- return value: 0 if successful

Supported baud rate values are:

- 115,200
- 57,600
- 38,400
- 28,800
- 19,200
- 14,400
- 9600
- 7200
- 4800

The other UART parameters are as follows. These cannot be modified.

- 8 bits data
- 1 stop bit
- no parity
- no flow control

`uart_read`

Reads from the UART device.

```
int uart_read(unsigned char port, unsigned char *buf, int length)
```

...where:

- port: UART port number:1 or 2 (currently only port 2 is supported)

- buf: the buffer to be read
- length: length of the buffer
- return value: 0 if successful

`uart_write`

Writes to the UART device.

```
int uart_write(unsigned char port, unsigned char *buf, int length)
```

...where:

- port: UART port number:1 or 2 (currently only port 2 is supported)
- buf: the buffer to be written
- length: length of the buffer
- return value: 0 if successful

APIs to communicate with host

`host_send`

Send raw data from the MCU to the host (/dev/ttymcu0).

```
int host_send(unsigned char *buf, int length)
```

...where:

- buf: buffer containing the data to be sent
- length: length of the buffer
- return value: 0 if successful. Returns an error code in case of failure

`host_receive`

Receive raw data from the host (/dev/ttymcu0). This synchronous API fetches the data from the buffer. The buffer's size is limited. New data will replace the old data if the data exceeds the buffer's size limit.

```
int host_receive(unsigned char *buf, int length)
```

...where:

- buf: the buffer to be received
- length: length of the buffer
- return value: size of data that received

APIs to control PWM

The MCU can access all PWM ports, from PWM 0 to PWM 3. The PWM period can vary from 104 ns to 218,453,000 ns.

`pwm_configure`

This API configures the PWM port.

```
int pwm_configure (unsigned char port, int duty_ns, int period_ns)
```

...where:

- port: PWM port to configure: 0, 1, 2, 3
- duty_ns: active pulse duration of the PWM waveform, in nanoseconds
- period_ns: the whole period of the PWM waveform, in nanoseconds

`pwm_enable`

This API enables the PWM port.

```
int pwm_enable (unsigned char port)
```

...where:

- port: PWM port to enable: 0, 1, 2, 3

`pwm_disable`

This API disables the PWM port.

```
int pwm_disable (unsigned char port)
```

...where:

- port: PWM port to disable: 0, 1, 2, 3

Auxiliary API

`debug_print`

This API prints the MCU debug message to the host debug console, for example /dev/ttymcu1.

```
void debug_print(int level, const char *fmt, ...)
```

...where:

- level: debug level of the output message, which could be one of the following: DBG_FATAL | DBG_ERROR | DBG_WARNING | DBG_INFO | DBG_DEBUG
- fmt: format string

`mcu_sleep`

This API sleeps the MCU application.

```
void mcu_sleep(int ticks)
```

...where:

- ticks: one tick is 10 ms for Edison MCU

`time_ms`

This API returns the number of milliseconds since the MCU has booted. This number will overflow (go back to zero) after 1193 hours.

```
unsigned long time_ms()
```

...where:

- return value: the number of milliseconds.

`time_us`

This API returns the number of microseconds since the MCU has booted. This number will overflow (go back to zero) after 71 minutes.

```
unsigned long time_us()
```

...where:

- return value: the number of microseconds.

`mcu_delay`

This API pauses the application for a period of time; it does not make the application sleep. Always call mcu_sleep if an application needs to sleep for more than one tick (10 ms).

```
int mcu_delay(int delay_us)
```

...where:

- delay_us: the number of microsecond to pause.
- return value: 0 if successful

`mcu_snprintf`

A simplified version of snprintf.

```
void mcu_snprintf(char *buf, unsigned int size, const char
*fmt, ...)
```

...where:

- buf: the buffer point to contain the data
- size: size of buffer
- fmt: format string
- return value: the number of characters that have been written

Note: Only %d, %x, %s are supported in this API.

`api_version`

This API returns the version of the API.

```
int api_version()
```

...where:

- return value: 100 multiples major version number plus minor version number

Sample scripts for the MCU SDK

This section contains the sample scripts for the MCU SDK referenced in the code examples, as well as download links for convenience.

Script	Download Link
init_DIG.sh	http://downloadmirror.intel.com/24910/eng/init_DIG.sh
init_i2c8.sh	http://downloadmirror.intel.com/24910/eng/init_i2c8.sh
init_mcu_PWM.sh	http://downloadmirror.intel.com/24910/eng/init_mcu_PWM.sh
set_DIG.sh	http://downloadmirror.intel.com/24910/eng/set_DIG.sh
read_DIG.sh	http://downloadmirror.intel.com/24910/eng/read_DIG.sh
init_UART1.sh	http://downloadmirror.intel.com/24910/eng/init_UART1.sh

init_DIG.sh

This shell script is available from the following location:
http://downloadmirror.intel.com/24910/eng/init_DIG.sh

```
#!/bin/sh
#author: David Pierret (davidx.pierret@intel.com)
#this script setup Edison/Arduino board to use DIG0 to DIG13 as GPIOs
#
#history
#
# V1 : only use debugfs for SoC gpios
# V2 : only use sysfs for SoC gpios
# V3 : use both debugfs and sysfs for SoC gpios
# V4 : correction of bugs.

DIG=""
DIR=""

usage() {
echo "usage $0 -o [num] -d [direction]" >&2
echo "work only on build edison-latest-99-2014-07-16_14-39-50 and next"
echo "num : number of the digital output 0->13" >&2
echo "direction : 'input' or 'output'" >&2
exit 1
}

export_gpio() {
        if [ $# -ne 1 ]; then
                echo "erreur appel fonction $0"
                exit 2
        fi
        if [ ! -d /sys/class/gpio/gpio$1 ]; then
                echo $1 > /sys/class/gpio/export
        fi
}

while getopts "o:(output)d:(direction)" optname
        do
                case "$optname" in
        "o")
                        DIG=$OPTARG
          ;;
        "d")
                        DIR=$OPTARG
          ;;
        "?")
                        usage
          ;;
        ":")
                        usage
          ;;
```

```
        *)
        # Should not occur
          echo "Unknown error while processing options"
          ;;
                esac
        done

echo "DIG = $DIG"
echo "DIR = $DIR"
# check the dig number var
if [ ! "$DIG" -ge 0 -a "$DIG" -le 13 ]; then
        usage
fi
# check the direction var
if [ "$DIR" != "output" -a "$DIR" != "input" ]; then
        usage
fi

export_gpio 214

echo low > /sys/class/gpio/gpio214/direction

case $DIG in
#gp130 to DIG0
0)
        export_gpio 248
        echo mode0 > /sys/kernel/debug/gpio_debug/gpio130/current_pinmux
        if [ $DIR == "output" ]; then
                echo high > /sys/class/gpio/gpio248/direction #config as output
                echo out > /sys/class/gpio/gpio130/direction
        else
                echo low > /sys/class/gpio/gpio248/direction #config as input
                echo in > /sys/class/gpio/gpio130/direction
        fi
;;

#gp131 to DIG1
1)
        export_gpio 249
        echo mode0 > /sys/kernel/debug/gpio_debug/gpio131/current_pinmux
        if [ $DIR == "output" ]; then
                echo high > /sys/class/gpio/gpio249/direction #config as output
                echo out > /sys/class/gpio/gpio131/direction
        else
                echo low > /sys/class/gpio/gpio249/direction #config as input
                echo in > /sys/class/gpio/gpio131/direction
        fi
;;

#gp128 to DIG2
2)
        export_gpio 250
        echo mode0 > /sys/kernel/debug/gpio_debug/gpio128/current_pinmux
        if [ $DIR == "output" ]; then
                echo high > /sys/class/gpio/gpio250/direction #config as output
                echo out > /sys/class/gpio/gpio128/direction
        else
                echo low > /sys/class/gpio/gpio250/direction #config as input
                echo in > /sys/class/gpio/gpio128/direction
        fi
;;

#gp12 to DIG3
3)
```

```
        export_gpio 251
        export_gpio 12
        echo mode0 > /sys/kernel/debug/gpio_debug/gpio12/current_pinmux
        if [ $DIR == "output" ]; then
                echo high > /sys/class/gpio/gpio251/direction #config as output
                echo out > /sys/class/gpio/gpio12/direction
        else
                echo low > /sys/class/gpio/gpio251/direction #config as input
                echo in > /sys/class/gpio/gpio12/direction
        fi
;;

#gp129 to DIG4
4)
        export_gpio 252
        echo mode0 > /sys/kernel/debug/gpio_debug/gpio129/current_pinmux
        if [ $DIR == "output" ]; then
                echo high > /sys/class/gpio/gpio252/direction #config as output
                echo out > /sys/class/gpio/gpio129/direction
        else
                echo low > /sys/class/gpio/gpio252/direction #config as input
                echo in > /sys/class/gpio/gpio129/direction
        fi
;;

#gp13 to DIG5
5)
        export_gpio 253
        export_gpio 13
        echo mode0 > /sys/kernel/debug/gpio_debug/gpio13/current_pinmux
        if [ $DIR == "output" ]; then
                echo high > /sys/class/gpio/gpio253/direction #config as output
                echo out > /sys/class/gpio/gpio13/direction
        else
                echo low > /sys/class/gpio/gpio253/direction #config as input
                echo in > /sys/class/gpio/gpio13/direction
        fi
;;

#gp182 to DIG6
6)
        export_gpio 254
        export_gpio 182
        echo mode0 > /sys/kernel/debug/gpio_debug/gpio182/current_pinmux
        if [ $DIR == "output" ]; then
                echo high > /sys/class/gpio/gpio254/direction #config as output
                echo out > /sys/class/gpio/gpio182/direction
        else
                echo low > /sys/class/gpio/gpio254/direction #config as input
                echo in > /sys/class/gpio/gpio182/direction
        fi
;;

#gp48 to DIG7
7)
        export_gpio 255
        export_gpio 48
        echo mode0 > /sys/kernel/debug/gpio_debug/gpio48/current_pinmux
        if [ $DIR == "output" ]; then
                echo high > /sys/class/gpio/gpio255/direction #config as output
                echo out > /sys/class/gpio/gpio48/direction
        else
                echo low > /sys/class/gpio/gpio255/direction #config as input
                echo in > /sys/class/gpio/gpio48/direction
```

```
        fi
;;

#gp49 to DIG8
8)
        export_gpio 256
        export_gpio 49
        echo mode0 > /sys/kernel/debug/gpio_debug/gpio49/current_pinmux
        if [ $DIR == "output" ]; then
                echo high > /sys/class/gpio/gpio256/direction #config as output
                echo out > /sys/class/gpio/gpio49/direction
        else
                echo low > /sys/class/gpio/gpio256/direction #config as input
                echo in > /sys/class/gpio/gpio49/direction
        fi
;;

#gp183 to DIG9
9)
        export_gpio 257
        export_gpio 183
        echo mode0 > /sys/kernel/debug/gpio_debug/gpio183/current_pinmux
        if [ $DIR == "output" ]; then
                echo high > /sys/class/gpio/gpio257/direction #config as output
                echo out > /sys/class/gpio/gpio183/direction
        else
                echo low > /sys/class/gpio/gpio257/direction #config as input
                echo in > /sys/class/gpio/gpio183/direction
        fi
;;

#gp41 to DIG10
10)
        export_gpio 258
        export_gpio 240
        export_gpio 263
        echo low > /sys/class/gpio/gpio240/direction
        echo high > /sys/class/gpio/gpio263/direction

        if [ ! -d /sys/class/gpio/gpio41 ]; then
                echo 41 > /sys/class/gpio/export
        fi
        echo mode0 > /sys/kernel/debug/gpio_debug/gpio41/current_pinmux
        if [ $DIR == "output" ]; then
                echo high > /sys/class/gpio/gpio258/direction #config as output
                echo out > /sys/class/gpio/gpio41/direction
        else
                echo low > /sys/class/gpio/gpio258/direction #config as input
                echo in > /sys/class/gpio/gpio41/direction
        fi
;;

#gp43 to DIG11
11)
        export_gpio 259
        export_gpio 241
        export_gpio 262
        export_gpio 43
        echo low > /sys/class/gpio/gpio241/direction
        echo high > /sys/class/gpio/gpio262/direction

        echo mode0 > /sys/kernel/debug/gpio_debug/gpio43/current_pinmux
        if [ $DIR == "output" ]; then
                echo high > /sys/class/gpio/gpio259/direction #config as output
```

```
                echo out > /sys/class/gpio/gpio43/direction
        else
                echo low > /sys/class/gpio/gpio259/direction #config as input
                echo in > /sys/class/gpio/gpio43/direction
        fi
;;

#gp42 to DIG12
12)
        export_gpio 260
        export_gpio 242
        export_gpio 42
        echo low > /sys/class/gpio/gpio242/direction
        echo mode0 > /sys/kernel/debug/gpio_debug/gpio42/current_pinmux
        if [ $DIR == "output" ]; then
                echo high > /sys/class/gpio/gpio260/direction #config as output
                echo out > /sys/class/gpio/gpio42/direction
        else
                echo low > /sys/class/gpio/gpio260/direction #config as input
                echo in > /sys/class/gpio/gpio42/direction
        fi
;;

#gp40 to DIG13
13)
        export_gpio 261
        export_gpio 243
        export_gpio 40
        echo low > /sys/class/gpio/gpio243/direction
        echo mode0 > /sys/kernel/debug/gpio_debug/gpio40/current_pinmux
        if [ $DIR == "output" ]; then
                echo high > /sys/class/gpio/gpio261/direction #config as output
                echo out > /sys/class/gpio/gpio40/direction
        else
                echo low > /sys/class/gpio/gpio261/direction #config as input
                echo in > /sys/class/gpio/gpio40/direction
        fi
;;
*)
        usage
;;
esac

echo high > /sys/class/gpio/gpio214/direction
```

init_i2c8.sh

This shell script is available from the following location:
http://downloadmirror.intel.com/24910/eng/init_i2c8.sh

```
echo 28 > /sys/class/gpio/export
echo 27 > /sys/class/gpio/export
echo 204 > /sys/class/gpio/export
echo 205 > /sys/class/gpio/export
echo 236 > /sys/class/gpio/export
echo 237 > /sys/class/gpio/export
echo 14 > /sys/class/gpio/export
echo 165 > /sys/class/gpio/export
echo 212 > /sys/class/gpio/export
echo 213 > /sys/class/gpio/export
if [ ! -d /sys/class/gpio/gpio214 ]; then
        echo 214 > /sys/class/gpio/export
```

```
fi
echo high > /sys/class/gpio/gpio214/direction
echo low > /sys/class/gpio/gpio204/direction
echo low > /sys/class/gpio/gpio205/direction
echo in > /sys/class/gpio/gpio14/direction
echo in > /sys/class/gpio/gpio165/direction
echo low > /sys/class/gpio/gpio236/direction
echo low > /sys/class/gpio/gpio237/direction
echo in > /sys/class/gpio/gpio212/direction
echo in > /sys/class/gpio/gpio213/direction
echo mode2 > /sys/kernel/debug/gpio_debug/gpio28/current_pinmux
echo mode2 > /sys/kernel/debug/gpio_debug/gpio27/current_pinmux
```

init_mcu_PWM.sh

This shell script is available from the following location:

http://downloadmirror.intel.com/24910/eng/init_mcu_PWM.sh

```
#!/bin/sh
#based on exemple from GSG_GPIO_PinMux_Control_FabB.pdf Intel internal document.
#exporting TRI_STATE_ALL
if [ ! -d /sys/class/gpio/gpio214 ]; then
        echo 214 > /sys/class/gpio/export
fi
echo low > /sys/class/gpio/gpio214/direction
#PWM 0 to DIG3
echo 251 > /sys/class/gpio/export
echo 219 > /sys/class/gpio/export
echo high > /sys/class/gpio/gpio251/direction
echo in > /sys/class/gpio/gpio219/direction

#PWM 1 to DIG5
echo 253 > /sys/class/gpio/export
echo 221 > /sys/class/gpio/export
echo high > /sys/class/gpio/gpio253/direction
echo in > /sys/class/gpio/gpio221/direction

#PWM 2 to DIG6
echo 254 > /sys/class/gpio/export
echo 222 > /sys/class/gpio/export
echo high > /sys/class/gpio/gpio254/direction
echo in > /sys/class/gpio/gpio222/direction

#PWM 3 to DIG9
echo 257 > /sys/class/gpio/export
echo 225 > /sys/class/gpio/export
echo high > /sys/class/gpio/gpio257/direction
echo in > /sys/class/gpio/gpio225/direction

#PWM 4 to DIG10 (PWM2)
#echo 263 > /sys/class/gpio/export
#echo 258 > /sys/class/gpio/export
#echo 225 > /sys/class/gpio/export
#echo high > /sys/class/gpio/gpio214/direction
#echo high > /sys/class/gpio/gpio263/direction
#echo high > /sys/class/gpio/gpio258/direction
#echo in > /sys/class/gpio/gpio226/direction
#echo mode1 > /sys/kernel/debug/gpio_debug/gpio182/current_pinmux
#echo low > /sys/class/gpio/gpio214/direction
#echo 2 > /sys/class/pwm/pwmchip0/export

echo high > /sys/class/gpio/gpio214/direction
```

set_DIG.sh

This shell script is available from the following location:

http://downloadmirror.intel.com/24910/eng/set_DIG.sh

```
#!/bin/sh
#author: David Pierret (davidx.pierret@intel.com)
#this script setup Edison/Arduino board to use DIG0 to DIG13 as GPIOs

DIG=""
VAL=""

usage() {
echo "usage $0 -o [num] -v [value]" >&2
echo "init_DIG.sh must be called before"
echo "num : number of the digital output 0->13" >&2
echo "value : 'low' or 'high'" >&2
exit 1
}

## set_output [mux] [gpio] [value]
## [mux] gpio to check to control level switcher
## [gpio] gpio number to read
## [value] value to apply
set_output() {
        if [ $# -ne 3 ]; then
                echo "erreur appel fonction $0"
                exit 2
        fi
        if [ ! -d /sys/class/gpio/gpio$1 ]; then
                usage
        fi
        if [ ! $(cat /sys/class/gpio/gpio$1/direction) == "out" ]; then
                usage
        fi
#       echo $VAL > /sys/kernel/debug/gpio_debug/gpio130/current_value
        echo $3 > /sys/class/gpio/gpio$2/direction
}

while getopts "o:(output)v:(value)" optname
        do
                case "$optname" in
        "o")
                        DIG=$OPTARG
          ;;
        "v")
                        VAL=$OPTARG
          ;;
        "?")
                        usage
          ;;
        ":")
                        usage
          ;;
        *)
        # Should not occur
          echo "Unknown error while processing options"
          ;;
                esac
        done


echo "DIG = $DIG"
```

```
echo "VAL = $VAL"
# check the dig number var
if [ ! "$DIG" -ge 0 -a "$DIG" -le 13 ]; then
        usage
fi
# check the direction var
if [ "$VAL" != "high" -a "$VAL" != "low" ]; then
        usage
fi

if [ ! -d /sys/class/gpio/gpio214 ]; then
        echo 214 > /sys/class/gpio/export
fi
echo low > /sys/class/gpio/gpio214/direction

case $DIG in
#gp130 to DIG0
0)
        set_output 248 130 $VAL
;;
#gp131 to DIG1
1)
        set_output 249 131 $VAL

;;

#gp128 to DIG2
2)
        set_output 250 128 $VAL
;;

#gp12 to DIG3
3)
        set_output 251 12 $VAL
;;

#gp129 to DIG4
4)
        set_output 252 129 $VAL
;;

#gp13 to DIG5
5)
        set_output 253 13 $VAL
;;

#gp182 to DIG6
6)
        set_output 254 182 $VAL

;;

#gp48 to DIG7
7)
        set_output 255 48 $VAL
;;

#gp49 to DIG8
8)
        set_output 256 49 $VAL
;;

#gp183 to DIG9
9)
```

```
        set_output 257 183 $VAL
;;

#gp41 to DIG10
10)
        set_output 258 41 $VAL
;;

#gp43 to DIG11
11)
        set_output 259 43 $VAL
;;

#gp42 to DIG12
12)
        set_output 260 42 $VAL
;;

#gp40 to DIG13
13)
        set_output 261 40 $VAL
;;

*)
        usage
;;

esac

echo high > /sys/class/gpio/gpio214/direction
```

read_DIG.sh

This shell script is available from the following location:
http://downloadmirror.intel.com/24910/eng/read_DIG.sh

```
#!/bin/sh
#author: David Pierret (davidx.pierret@intel.com)
#this script setup Edison/Arduino board to use DIG0 to DIG13 as GPIOs

DIG=""
VAL=""

usage() {
echo "usage $0 -o [num]" >&2
echo "init_DIG.sh must be called before"
echo "num : number of the digital input 0->13" >&2
exit 1
}

## set_output [mux] [gpio]
## [mux] gpio to check to control level switcher
## [gpio] gpio number to read
read_input() {
        if [ $# -ne 2 ]; then
                echo "erreur appel fonction $0"
                exit 2
        fi
        if [ ! -d /sys/class/gpio/gpio$1 ]; then
                usage
        fi
        if [ ! $(cat /sys/class/gpio/gpio$1/direction) == "out" ]; then
```

```
                usage
        fi
#       cat /sys/kernel/debug/gpio_debug/gpio$2/current_value
        cat /sys/class/gpio/gpio$2/value
}

while getopts "o:(output)" optname
        do
                case "$optname" in
        "o")
                        DIG=$OPTARG
          ;;
        "?")
                        usage
          ;;
        ":")
                        usage
          ;;
        *)
        # Should not occur
          echo "Unknown error while processing options"
          ;;
                esac
        done

# check the dig number var
if [ ! "$DIG" -ge 0 -a "$DIG" -le 13 ]; then
        usage
fi

if [ ! -d /sys/class/gpio/gpio214 ]; then
        echo 214 > /sys/class/gpio/export
fi
echo high > /sys/class/gpio/gpio214/direction

case $DIG in
#gp130 to DIG0
0)
        read_input 248 130
;;
#gp131 to DIG1
1)
        read_input 249 131

;;

#gp128 to DIG2
2)
        read_input 250 128
;;

#gp12 to DIG3
3)
        read_input 251 12
;;

#gp129 to DIG4
4)
        read_input 252 129
;;

#gp13 to DIG5
5)
        read_input 253 13
```

```
;;

#gp182 to DIG6
6)
        read_input 254 182

;;

#gp48 to DIG7
7)
        read_input 255 48
;;

#gp49 to DIG8
8)
        read_input 256 49
;;

#gp183 to DIG9
9)
        read_input 257 183
;;

#gp41 to DIG10
10)
        read_input 258 41
;;

#gp43 to DIG11
11)
        read_input 259 43
;;

#gp42 to DIG12
12)
        read_input 260 42
;;

#gp40 to DIG13
13)
        read_input 261 40
;;

*)
        usage
;;

esac

echo low > /sys/class/gpio/gpio214/direction
```

init_UART1.sh

This shell script is available from the following location:
http://downloadmirror.intel.com/24910/eng/init_UART1.sh

```
#!/bin/sh
#author:Jiujin Hong (Jiujinx.hong@intel.com)
#
#this script setup Edison/Arduino board to use DIG0/DIG1 as UART1 RX/UART1 TX
#
#
```

```
export_gpio() {
        if [ $# -ne 1 ]; then
                echo "erreur appel fonction $0"
                exit 2
        fi
        if [ ! -d /sys/class/gpio/gpio$1 ]; then
                echo $1 > /sys/class/gpio/export
        fi
}

#config DIG0 UART1 RX mode and level shiffter
export_gpio 248
echo mode1 > /sys/kernel/debug/gpio_debug/gpio130/current_pinmux
echo low > /sys/class/gpio/gpio248/direction #config as input
echo in > /sys/class/gpio/gpio130/direction

#config DIG1 UART1 RX mode and level shiffter
export_gpio 249
echo mode1 > /sys/kernel/debug/gpio_debug/gpio131/current_pinmux
echo high > /sys/class/gpio/gpio249/direction #config as output
echo out > /sys/class/gpio/gpio131/direction
```

Troubleshooting and Resources

- Can’t log in to the Intel® Edison board
- Unable to download program or connect to board
- Connected to board but can't view the device trace
- Opening the MCU SDK Launcher and running Eclipse takes a long time on Windows*
- "No device found" error when downloading an application to a board

Can’t log in to the Intel® Edison board

If you can't log in to your board using the `ssh root@192.168.2.15` command, check the following:

- You have installed the correct device drivers on your host machine. For steps, see the appropriate link below:
  - Getting Started with the Intel® Edison board on Windows*
  - Getting Started with the Intel® Edison board on Mac*
  - Getting Started with the Intel® Edison board on Linux*
- You have set your host machine’s IP address to 192.168.2.1 (or 192.168.2.*x*) and subnet mask to 255.255.255.0.

Unable to download program or connect to board

Check that you can log in to the board by entering the following command: `ssh root@192.168.2.15`.

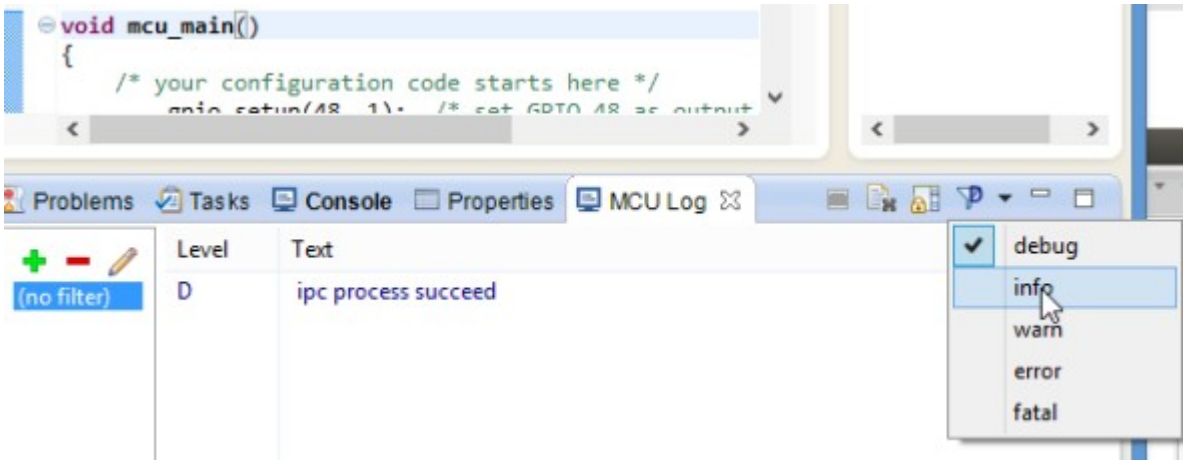
Check that there is enough disk space on the Intel Edison board. To do so, in a a serial communication session, enter the command: `df -h`

```
root@edison:~# df -h
Filesystem                Size      Used Available Use% Mounted on
/dev/root                463.9M    377.1M     51.0M  88% /
devtmpfs                 479.9M         0    479.9M   0% /dev
tmpfs                    480.2M         0    480.2M   0% /dev/shm
tmpfs                    480.2M    516.0K    479.7M   0% /run
tmpfs                    480.2M         0    480.2M   0% /sys/fs/cgroup
systemd-1                  5.7M      5.3M    462.0K  92% /boot
systemd-1                  2.2G     64.1M      2.2G   3% /home
tmpfs                    480.2M      4.0K    480.2M   0% /tmp
tmpfs                    480.2M         0    480.2M   0% /var/volatile
/dev/mmcblk0p5          1003.0K     21.0K    911.0K   2% /factory
/dev/mmcblk0p10            2.2G     64.1M      2.2G   3% /home
/dev/mmcblk0p7             5.7M      5.3M    462.0K  92% /boot
root@edison:~#
```

The disk usage is displayed in the /dev/root row.

Connected to board but can't view the device trace

On the MCU Log tab, from the **Log Level** drop-down list, select **debug**. From the Saved Filters list on the left, remove all log filters.



Check to see if you can log in to the board from a Linux shell using the `ssh root@192.168.2.15` command. If you can't log in, see Can’t log in to the Intel Edison board and Unable to download program or connect to board above.

Opening the MCU SDK Launcher and running Eclipse takes a long time on Windows*

Certain network configurations may lead to a delay when running the Launcher. Try the following:

- Try the workaround for Cygwin version 1.7.34.
- Disable the network connection while you run the Launcher. Once Eclipse opens, reenable the connection.

It may take a long time to run Eclipse when the Edison RNDIS connection is on, as described in Intel® Edison Connecting Ethernet over USB. Try the following:

- Disable the RNDIS connection when running Eclipse. Once Eclipse opens, reenable the connection.
- Pull out the Edison USB cable when running Eclipse. Once Eclipse opens, reconnect the cable.

"No device found" error when downloading an application to a board

You may encounter this error if you have just flashed the board. Make sure you have set up the board correctly and that your system and the board are on the same local network, then perform the following steps:

1. On Linux* and Mac* OS X*, connect to your board by entering the command `ssh root@192.168.2.15` in a Linux shell window before you run Eclipse.
2. Enter `yes` to have your system remember the board.

References

Reference Name	Location
Intel® Edison Board Support Package User Guide	http://www.intel.com/support/edison/sb/CS-035278.htm
Intel® Edison Compute Module Hardware Guide	http://www.intel.com/support/edison/sb/CS-035274.htm
Intel® Edison Breakout Board Hardware Guide	http://www.intel.com/support/edison/sb/CS-035252.htm
Intel® Edison Kit for Arduino* Hardware Guide	http://www.intel.com/support/edison/sb/CS-035275.htm
Intel® Galileo and Intel® Edison Release Notes	https://communities.intel.com/docs/DOC-23388
Intel® Edison Getting Started Guide	Windows*: Getting Started with the Intel® Edison board on Windows* Mac* OS X*: Getting Started with the Intel® Edison board on Mac* Linux*: Getting Started with the Intel® Edison board on Linux*
Intel® Edison Wi-Fi Guide	http://www.intel.com/support/edison/sb/CS-035380.htm
Intel® Edison Bluetooth* Guide	http://www.intel.com/support/edison/sb/CS-035381.htm
Yocto Project Quick Start Guide	http://www.yoctoproject.org/docs/current/yocto-project-qs /yocto-project-qs.html
Yocto Developer Manual	http://www.yoctoproject.org/docs/current/dev-manual /dev-manual.html
Yocto Kernel Developer Manual	http://www.yoctoproject.org/docs/latest/kernel-dev /kernel-dev.html

Terminology

Term	Meaning
API	Application program interface
DMA	Direct memory access
FTP	File transfer protocol
GDB	GNU debugger
GPIO	General purpose input/output
Host CPU	On Intel® Edison, the Intel® Atom™ processor that runs on Linux OS
HSU	High-speed UART
IA	Intel architecture
I2C	Inter-integrated circuit
IDE	Integrated development environment
IPC	Interprocessor communication
LED	Light emitting diode
MCU	Microcontroller unit. On the Intel® Edison board, this is a “Minute IA” 32-bit CPU
MCU application	Application that runs on an MCU. In most cases, it implements the expected features (controlling GPIOs, getting data from sensors, etc.)
OS	Operating system
PWM	Pulse width modulation
RTOS	Real-time operating system
SDK	Software development kit
SRAM	Static random-access memory
SSH	Secure shell
UART	Universal asynchronous receiver/transmitter
Viper	An Intel (Wind River) real-time operating system that runs on the MCU. Viper provides basic OS function support, including thread scheduling, memory management, interrupt dispatch, and so on
D0	Device power state, equivalent to “fully on”. D1 and D2 are intermediate states; D3 is “Off”
S0ix	An “active idle” sleep state, which delivers the same power consumption as S3 sleep, yet provides a quick activate time into full S0 state
SCU	System controller unit
SRAM	Static random access memory or static RAM (compared to DRAM or dynamic RAM)