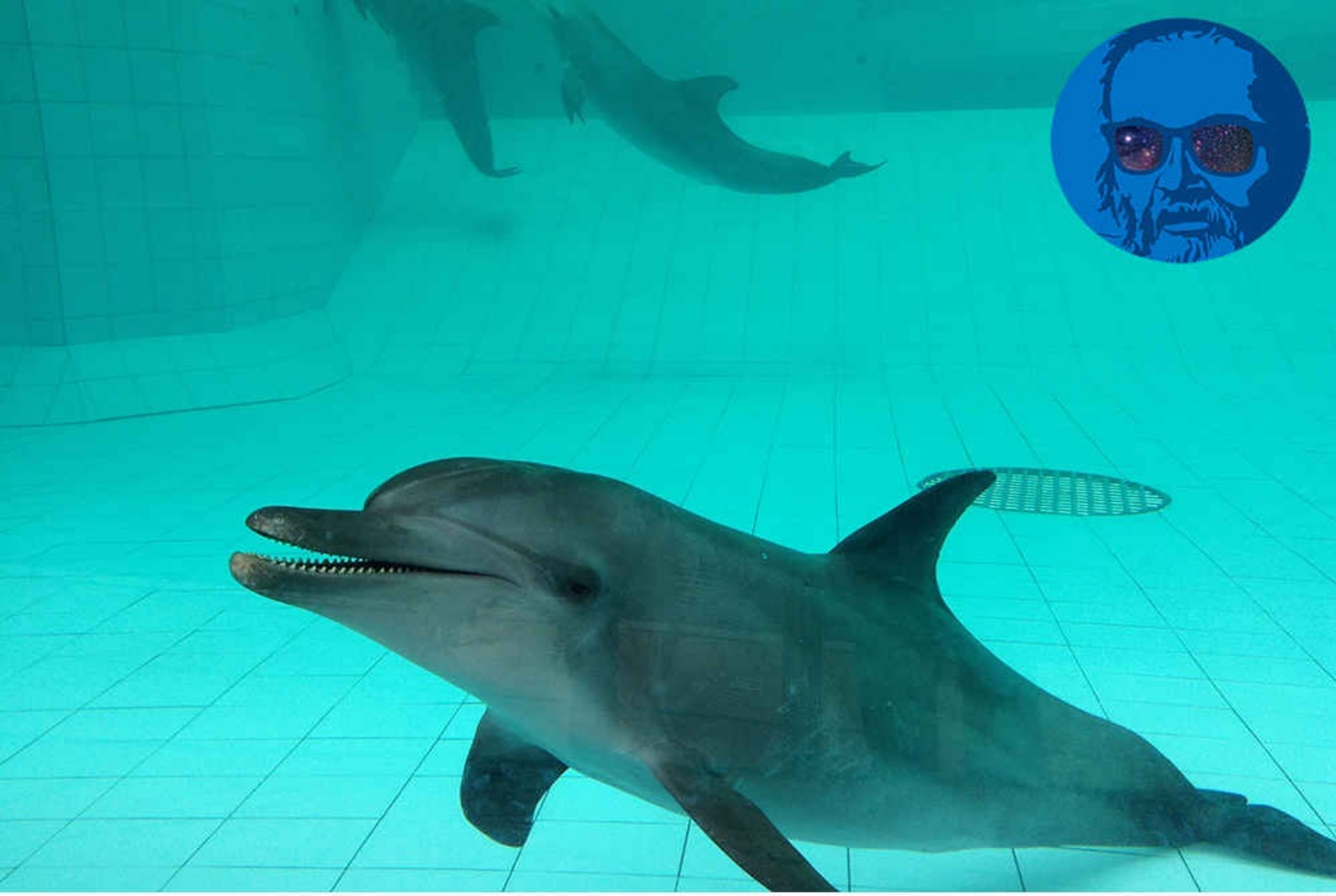


Getting Started with Windows IoT and Intel Galileo

Agus Kurniawan

intel Galileo

Copyright

Table of Contents

Preface

This book was written to help anyone wants to get started in Windows for IoT and Intel Galileo. It describes all the basic elements of the Windows for IoT and Intel Galileo with step-by-step approach. Several code samples are be provided to illustrate how to work with Intel Galileo.

Agus Kurniawan

Depok, April 2015

1. Preparing Development Environment

1.1 Intel Galileo

Intel Galileo is a microcontroller board based on the Intel® Quark SoC X1000 Application Processor, a 32-bit Intel Pentium-class system on a chip (datasheet). It’s the first board based on Intel architecture designed to be hardware and software pin-compatible with Arduino shields designed for the Uno R3. Currently there are two type of Intel Galileo:

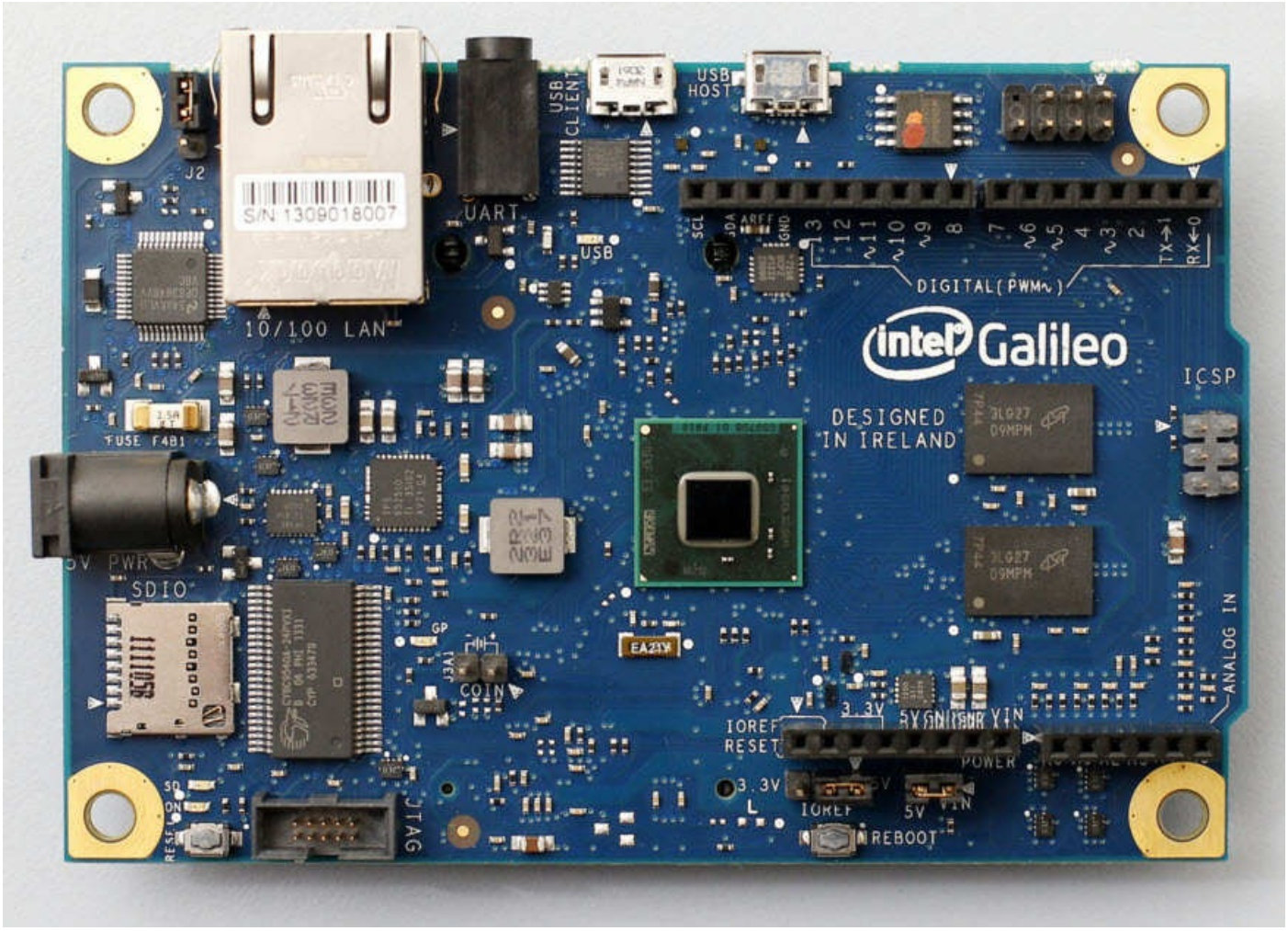
- Intel Galileo generation 1
- Intel Galileo generation 2

In this book, I use Intel Galileo generation 2 for all illustration.

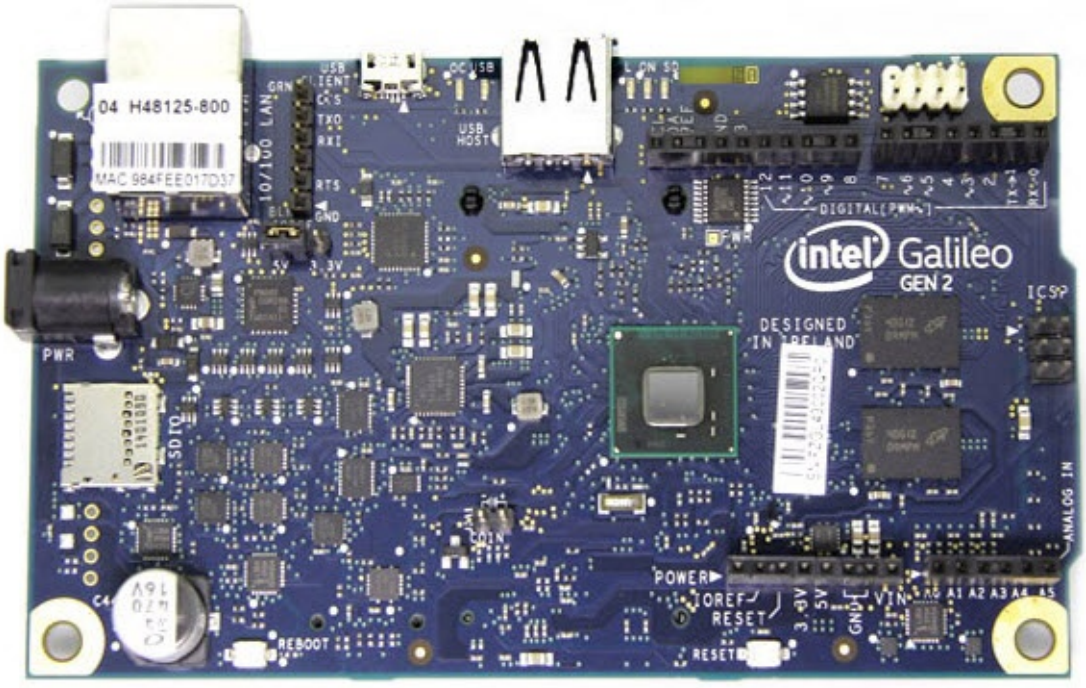
The following is technical specification of Intel Galileodevice:

- Intel® Quark™ SoC X1000 application processor, a 32-bit, single-core, single-thread, Intel® Pentium® processor instruction set architecture (ISA)-compatible, operating at speeds up to 400 MHz.
- Support for a wide range of industry standard I/O interfaces, including a full-sized mini-PCI Express* slot, 100 Mb Ethernet port, microSD* slot, USB host port, and USB client port.
- 256 MB DDR3, 512 kb embedded SRAM, 8 MB NOR Flash, and 8 kb EEPROM standard on the board, plus support for microSD card up to 32 GB.
- Hardware and pin compatibility with a wide range of Arduino Uno R3 shields.
- Programmable through the Arduino integrated development environment (IDE) that is supported on Microsoft Windows*, Mac OS*, and Linux host operating systems.
- Support for Yocto 1.4 Poky* Linux release.

You can see Intel Galileo generation 1 device in the Figure below.



For Intel Galileo generation 2, you can see it in Figure below.



1.2 Getting Hardware

How to get Intel Galileo?

Officially you can buy it from the official distributor

- Sparkfun, https://www.sparkfun.com/products/12720
- Arduino store, http://store.arduino.cc/index.php?main_page=product_info&cPath=40&products_id=522
- Adafruit, https://www.adafruit.com/products/2188
- Seeedstudio, http://www.seeedstudio.com/depot/Intel-Galileo-Gen-2-p-2014.html?cPath=6_7
- Amazon, http://www.amazon.com/gp/product/B00GGM6KJQ/

You can buy Intel Galileo on your local electronics store. In addition, you also buy Intel Galileo peripheral devices for instance, keyboard, mouse, HDMI cable, SD card, USB hub, etc.

1.3 Development Tools

In this book, we use Visual Studio 2013 for development. For setupping development environment, you can read it on chapter 2.

1.4 Electronics Devices

We need electronic components to build our testing, for instance, Resistor, LED, sensor devices and etc. I recommend you can buy electronic component kit. You find them on your local electronics shops

You can buy Intel Galileo starter kit, for instance starter kit from Seeedstudio http://www.seeedstudio.com/depot/preorderGrove-starter-kit-plus-Intel-IoT-Edition-for-Intel-Galileo-Gen-2-p-1978.html .

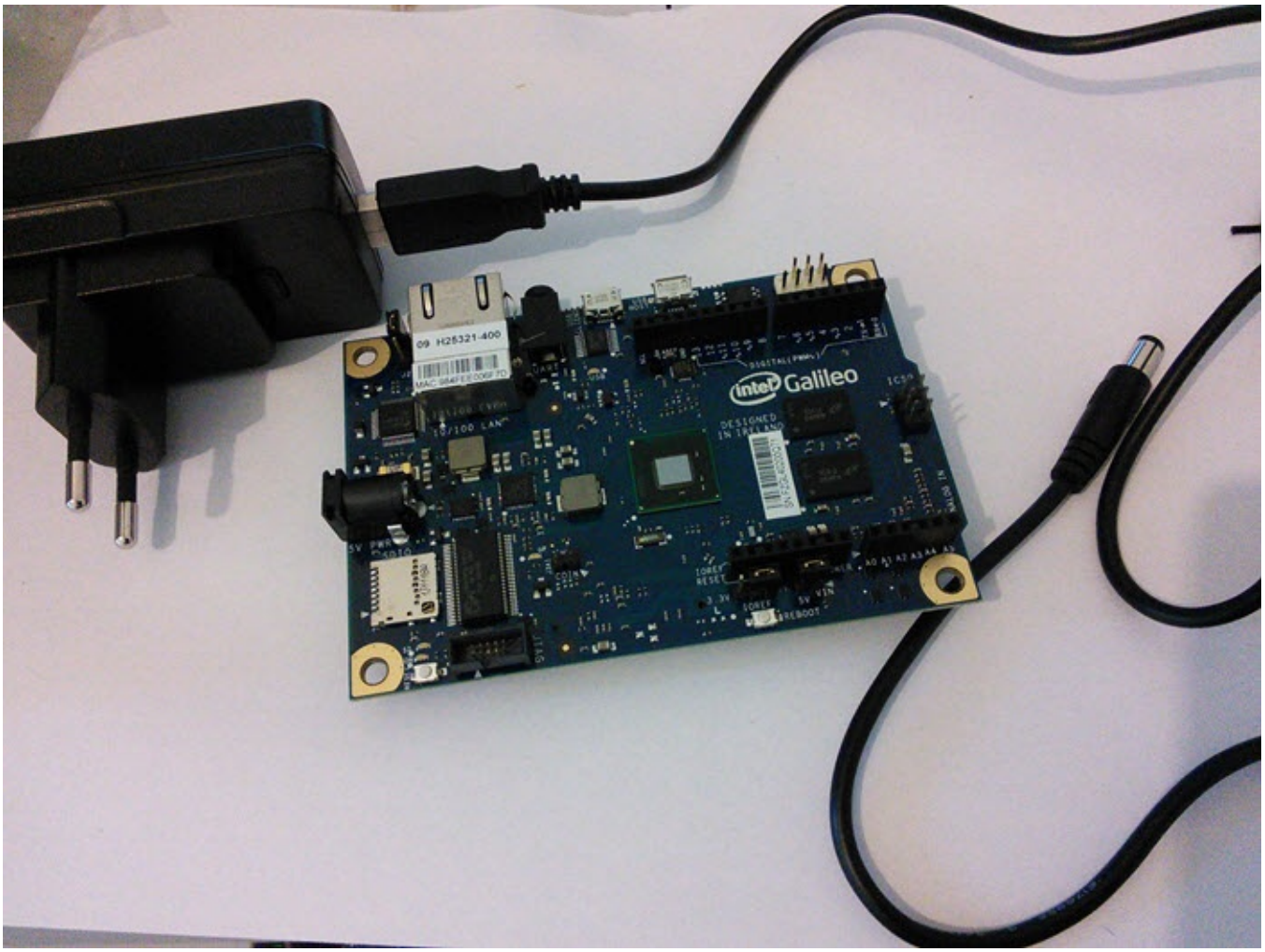

Grove-LCD RGB Backlight
Grove
Starter
Kit
FROM SEEEDSTUDIO
D4
D3
D2
UART
D8
D7
D6
D5
A0
A1
A2
A3
I2C
I2C
I2C
I2C
micro SD
Adaptor

1.5 Getting Started

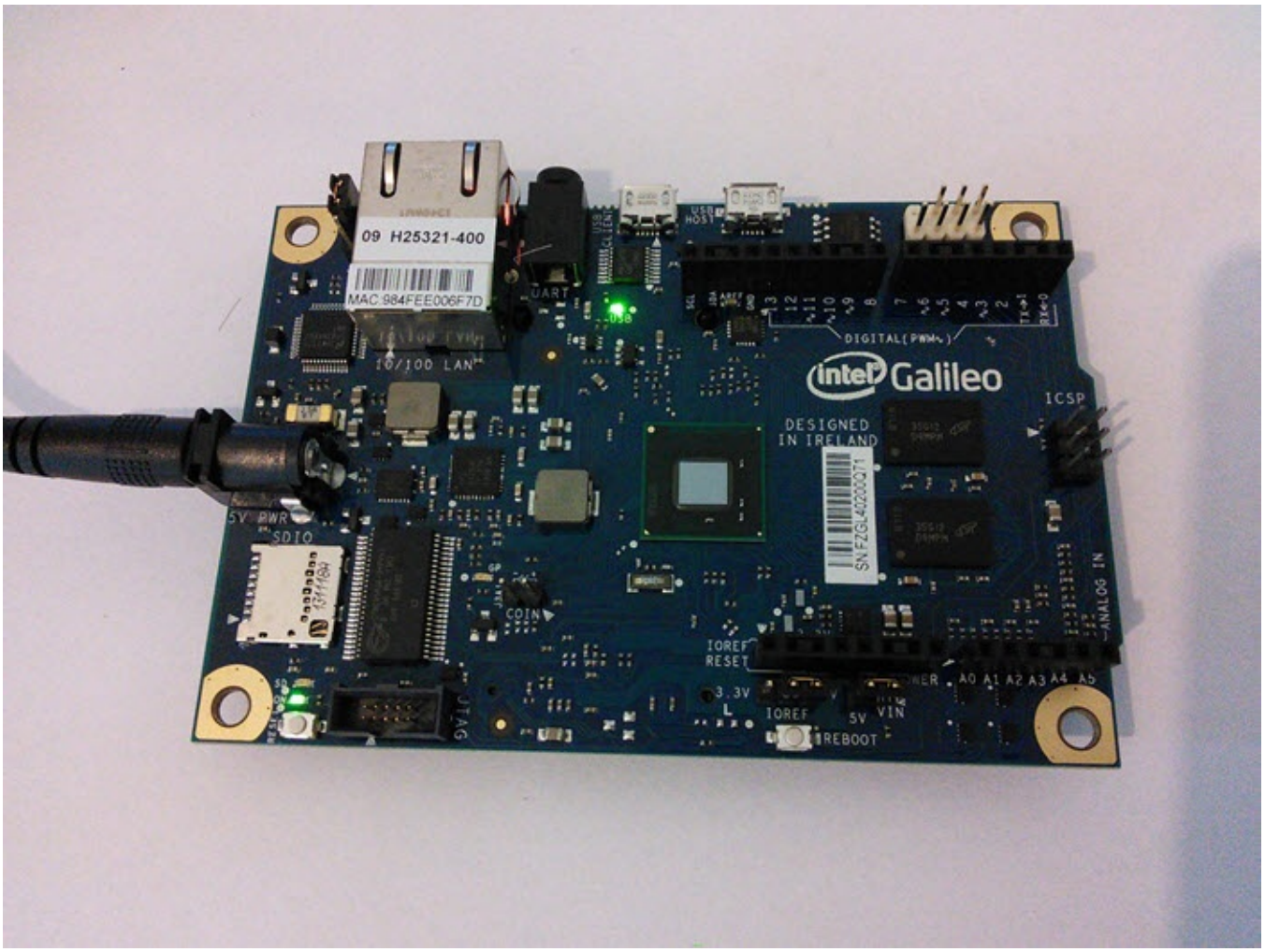
After you bought Intel Galileo, you got the following box.

Then, you open it. You will find Intel Galileo board and its power.


intel Galileo



Now you can plug the power on Intel Galileo. If success, you can see green light on the board, shown in Figure below.



Next chapter, we will explore how to deploy Windows Embedded on Intel Galileo board.

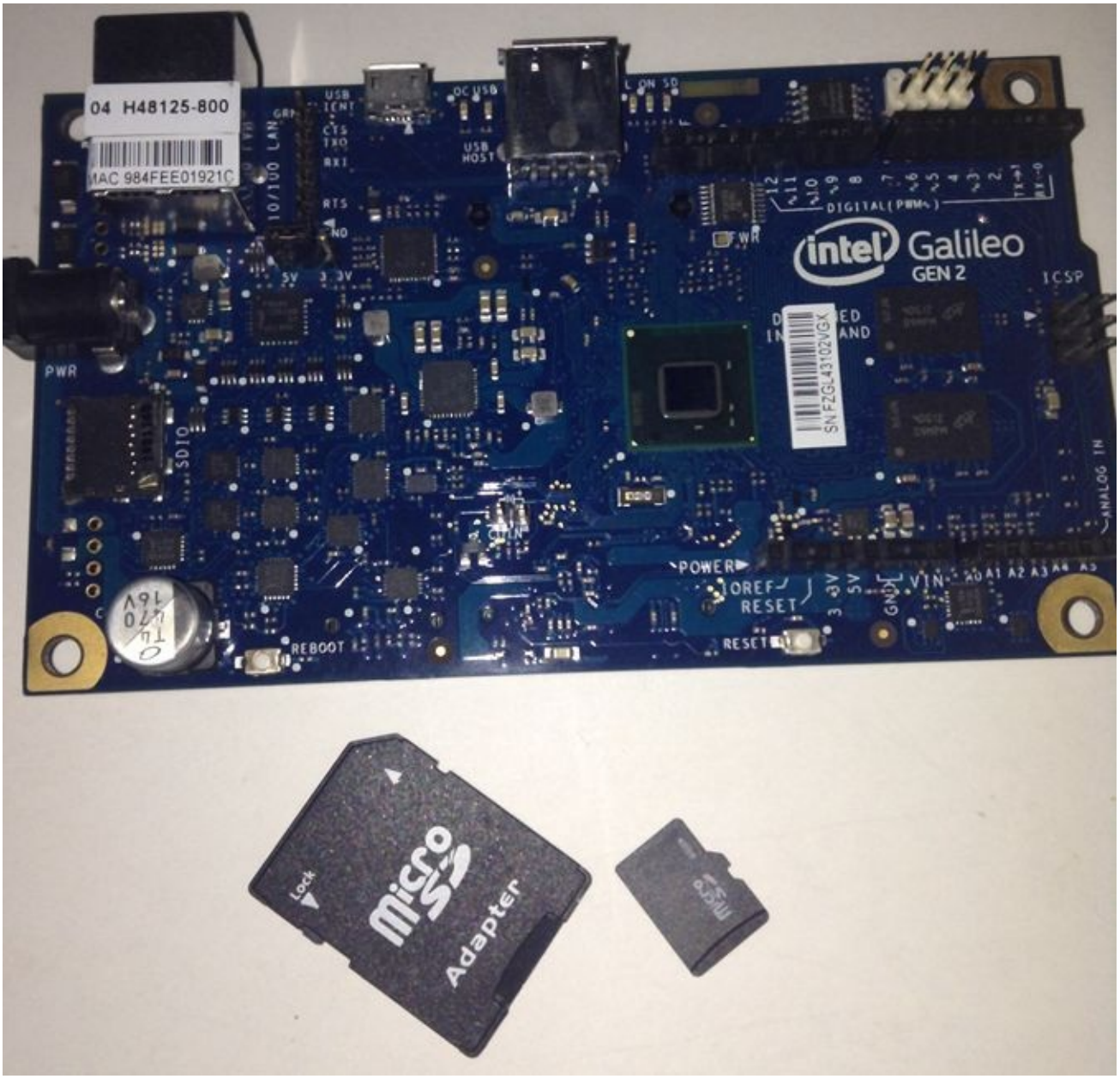
2. Deploying Windows IoT on Intel Galileo

This chapter explains how to deploy Windows IoT on Intel Galileo using Windows Embedded software.

2.1 Preparation

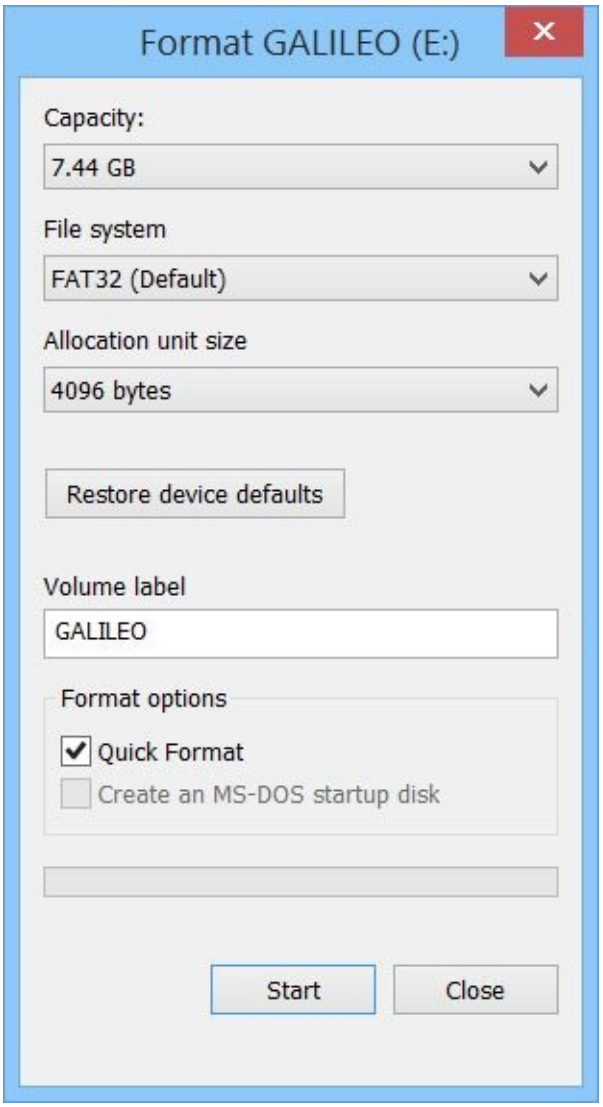
In this chapter, we explore Windows Embedded OS to develop application for Intel Galileo target. FYI, Intel Galileo has pin-compatible with a wide range of Arduino Uno R3 shields.

To deploy Windows Embedded on Intel Galileo, we need microSD card with 16Gb size. Format this microSD card with FAT32 format type.

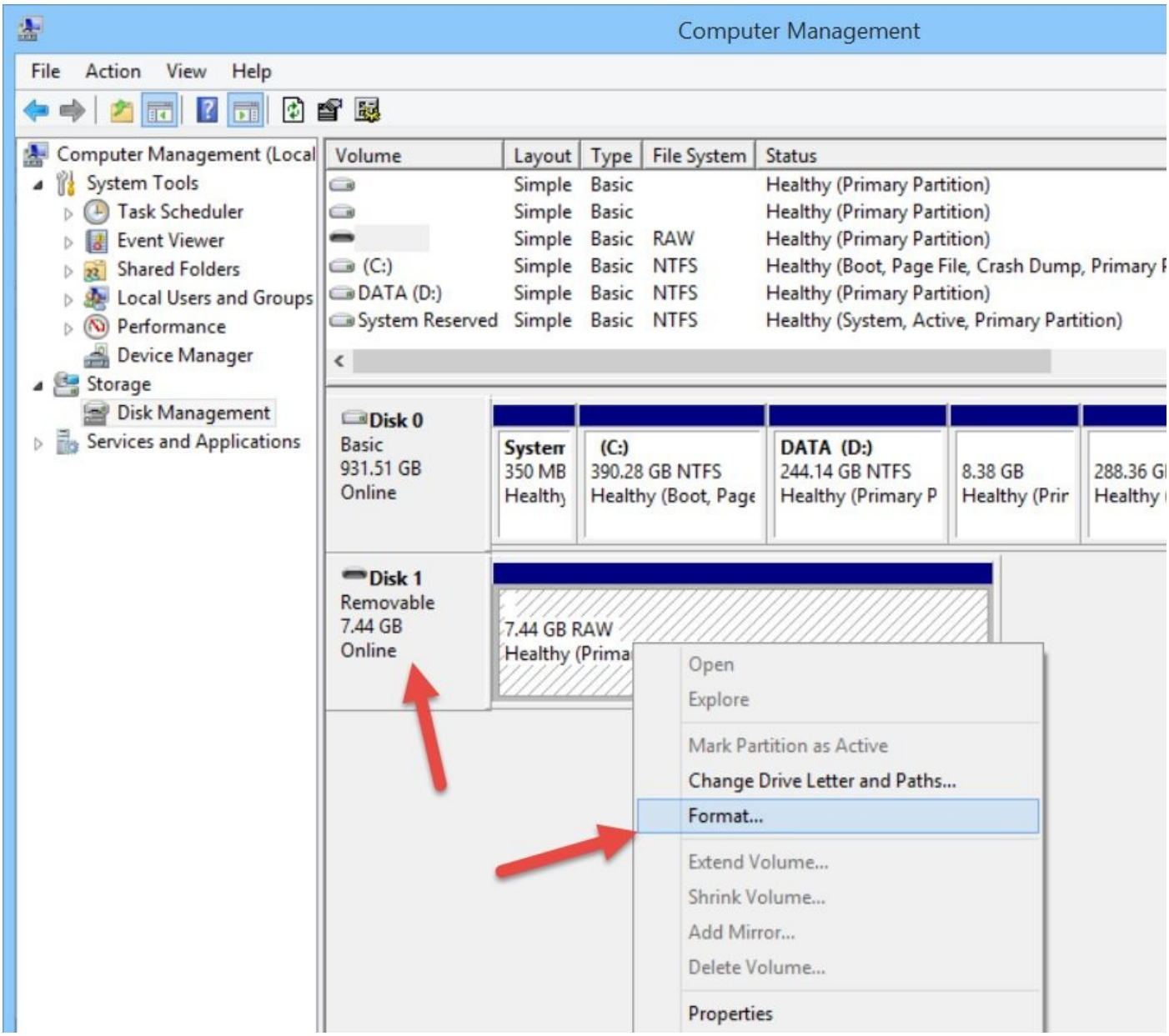


We use MicroSD adapter to attach microSD card. Then, we can format this microSD.

On Windows platform, you can do right-click on your microSD path.



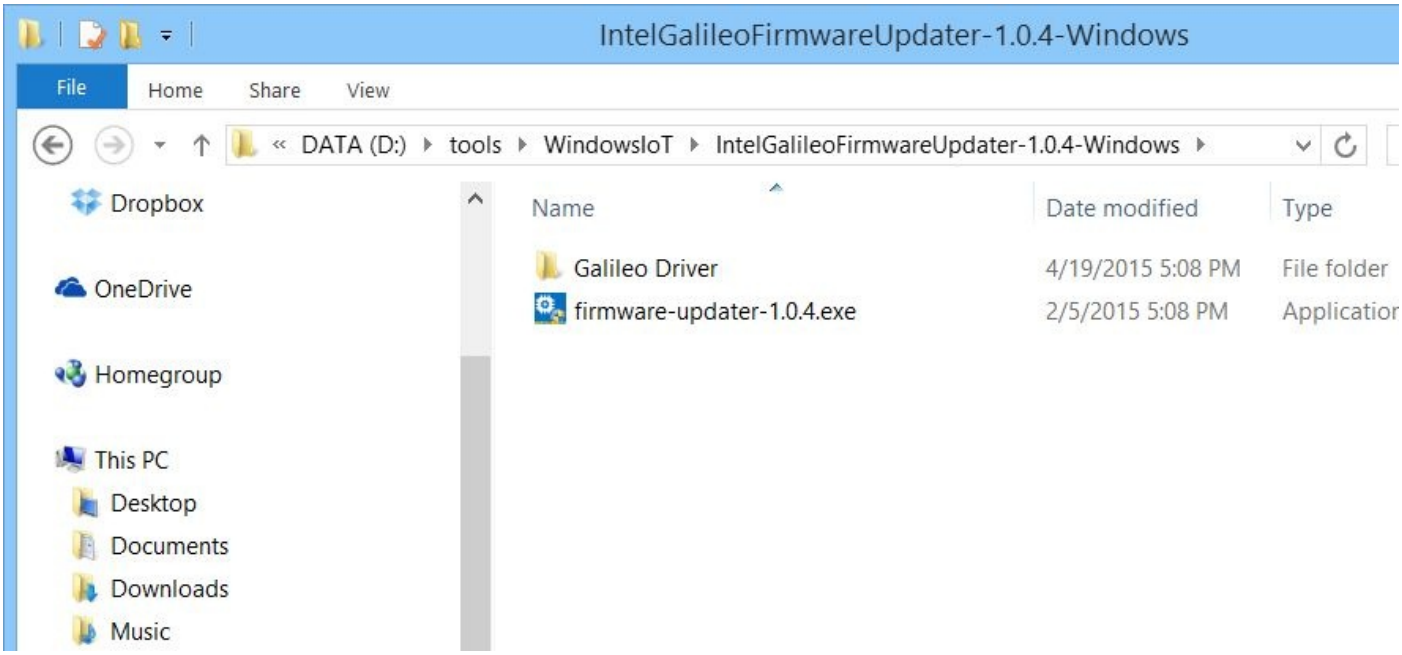
You also can use Disk Management on Computer Management tool.


Computer Management
File Action View Help
Computer Management (Local
System Tools
Task Scheduler
Event Viewer
Shared Folders
Local Users and Groups
Performance
Device Manager
Storage
Disk Management
Services and Applications
Volume Layout Type File System Status
Simple Basic Healthy (Primary Partition)
Simple Basic Healthy (Primary Partition)
Simple Basic RAW Healthy (Primary Partition)
(C:) Simple Basic NTFS Healthy (Boot, Page File, Crash Dump, Primary
DATA (D:) Simple Basic NTFS Healthy (Primary Partition)
System Reserved Simple Basic NTFS Healthy (System, Active, Primary Partition)
Disk 0
Basic
931.51 GB
Online
System
350 MB
Healthy
(C:)
390.28 GB NTFS
Healthy (Boot, Page
DATA (D:)
244.14 GB NTFS
Healthy (Primary P
8.38 GB
Healthy (Prir
Disk 1
Removable
7.44 GB
Online
7.44 GB RAW
Healthy (Prima
Open
Explore
Mark Partition as Active
Change Drive Letter and Paths...
Format...
Extend Volume...
Shrink Volume...
Add Mirror...
Delete Volume...
Properties

2.2 Updating Intel Galileo Firmware

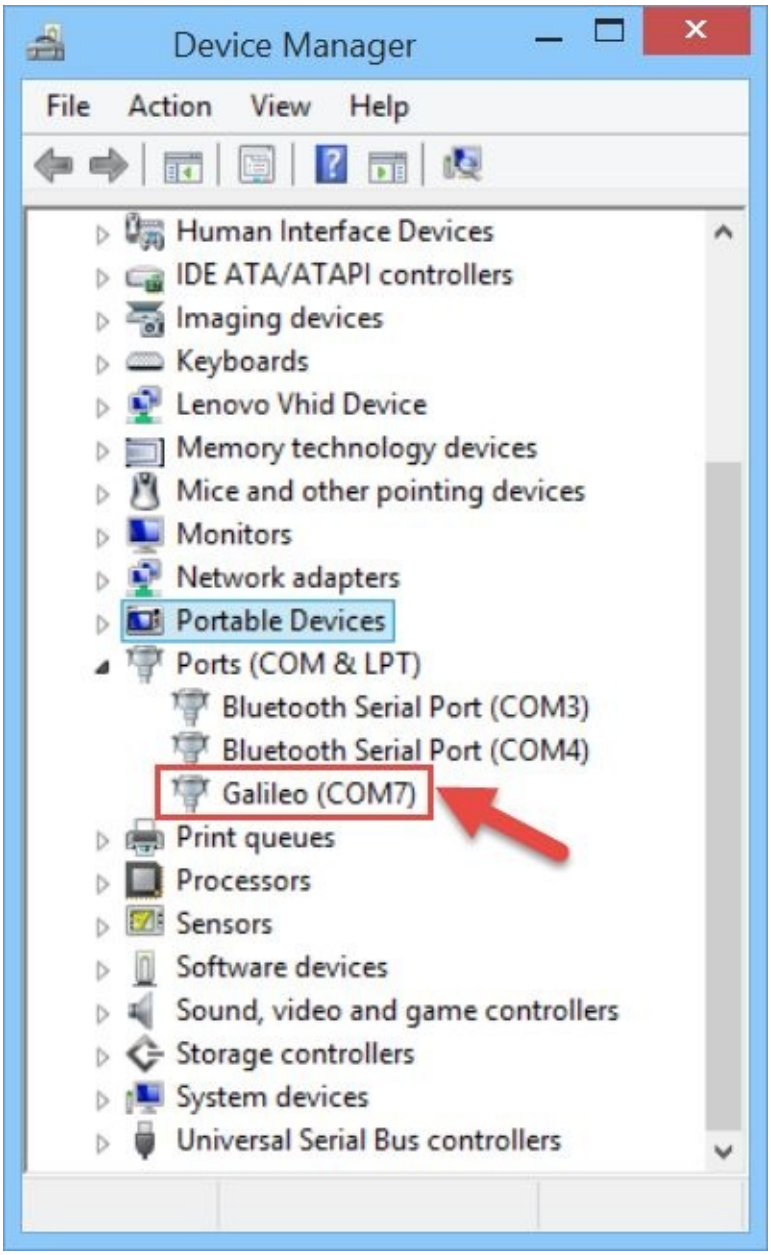
Before deploying, we must update board firmware. You can use Intel Galileo Firmware Updater Tool. You can download it on http://www.intel.com/support/galileo/sb/CS-035101.htm .

A sample of Intel Galileo Firmware Updater can be seen in Figure below.



Firstly, remove all microSD from Intel Galileo board. Now you can plug in the power adapter into Intel Galileo board. After that, plug in microUSB to Client USB on Intel Galileo board to computer.

On Device Manager, it will detect Intel Galileo board. For instance, it shows on COM7 on my computer.



Now you can run Intel Galileo Firmware Updater tool. Select Port for your Intel Galileo.

After that, click **Update Firmware** button. You will get a confirmation dialog. Click **Yes** to confirm.

It will take several minutes to complete the updating Firmware process.

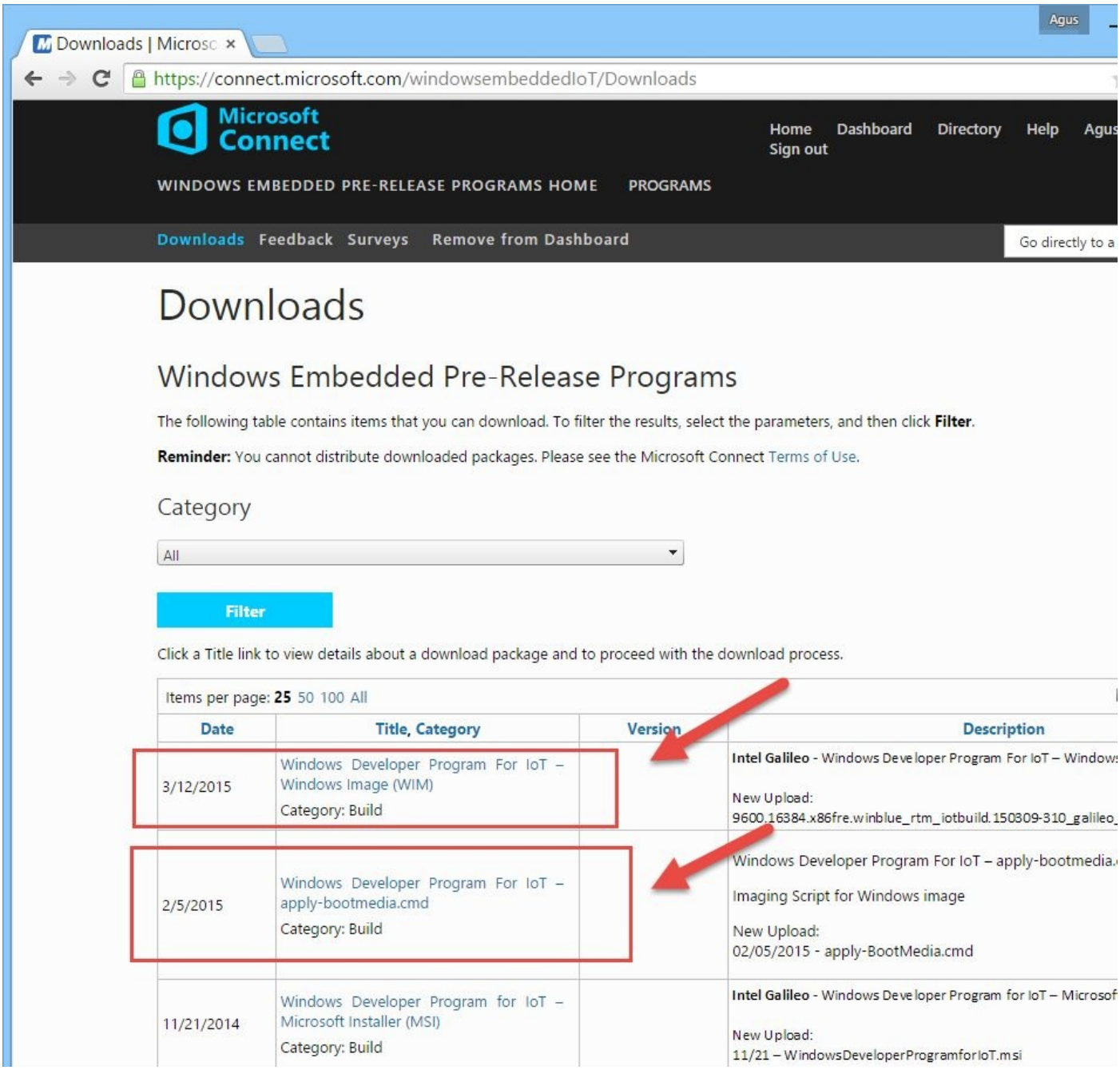

Intel® Galileo Firmware Updater 1.0.4
File About
Port: COM7
Firmware: 1.0.4
Browse for .cap file
Galileo Firmware Update
Target firmware successfully updated.
OK
ate Firmware

2.3 Deploying Windows IoT on Intel Galileo Board

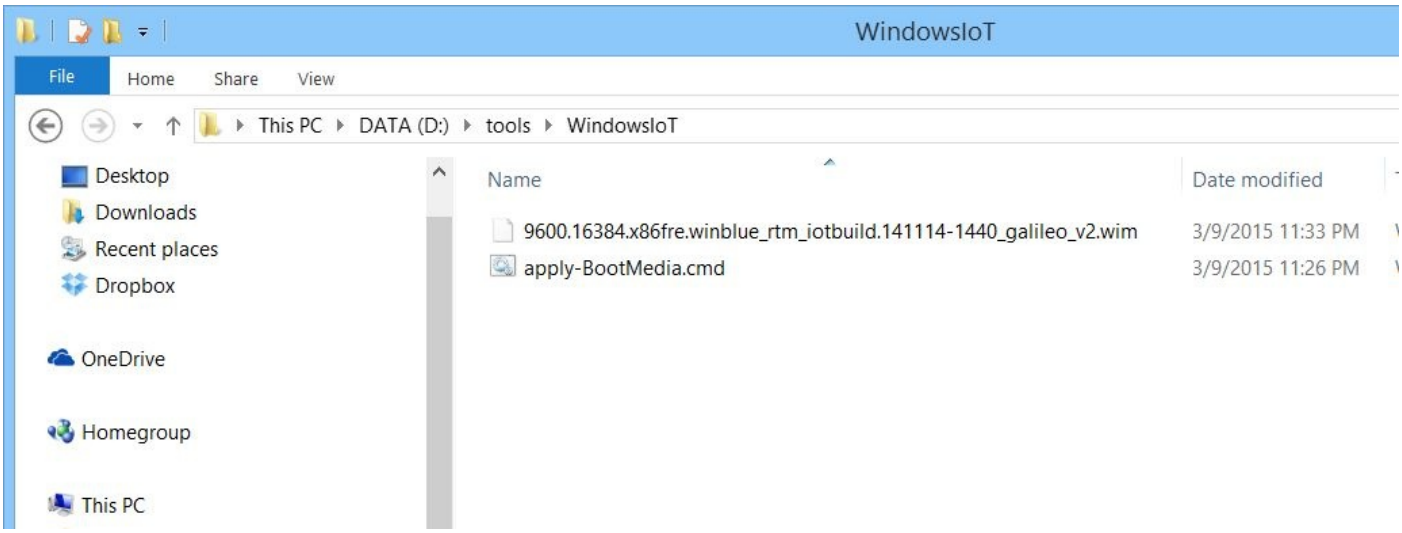
The next step is to download Windows Embedded from Windows Developer Program for the Internet of Things (IoT), http://connect.microsoft.com/windowsembeddediot/SelfNomination.aspx?ProgramID=8558 .

Sign in the portal with your passport ID. Register new account if you don't have passport ID. After signed, you can click **Downloads** menu. You download the following items:

- Windows Image (WIM)
- apply-bootmedia.cmd



A sample of my Windows Image and BootMedia.



For illustration, I have Windows Embedded image file, **9600.16384.x86fre.winblue_rtm_iotbuild.150309-0310_galileo_v2.wim** . Then, we can deploy it on microSD. In this case, I configure:

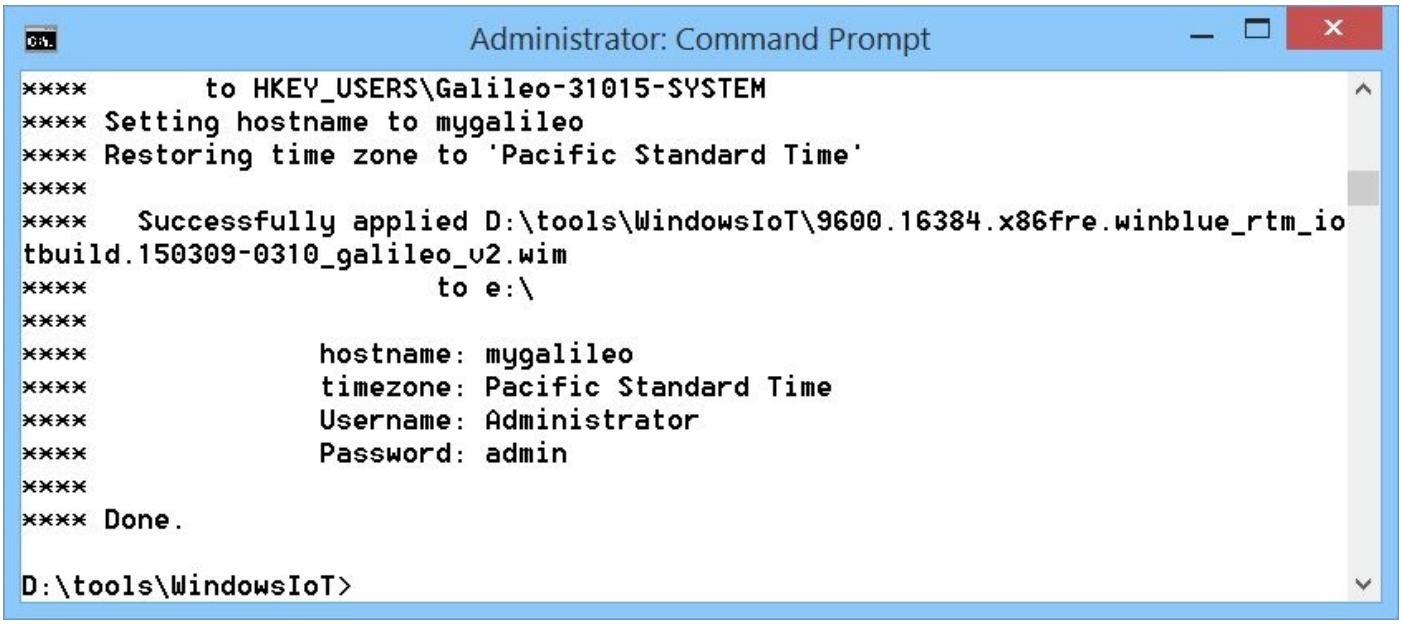
- hostname: mygalileo
- username: Administrator
- password: admin
- my microSD is located to path E. You can change it

This is default configuration which can be applied in our development.

Insert microSD into your computer. Open Command Prompt as Administrator. Navigate to a folder which **apply-BootMedia.cmd** and image file are located. Then, type this command.

```
apply-BootMedia.cmd -destination e:\ -image 9600.16384.x86fre.winblue_rt
```

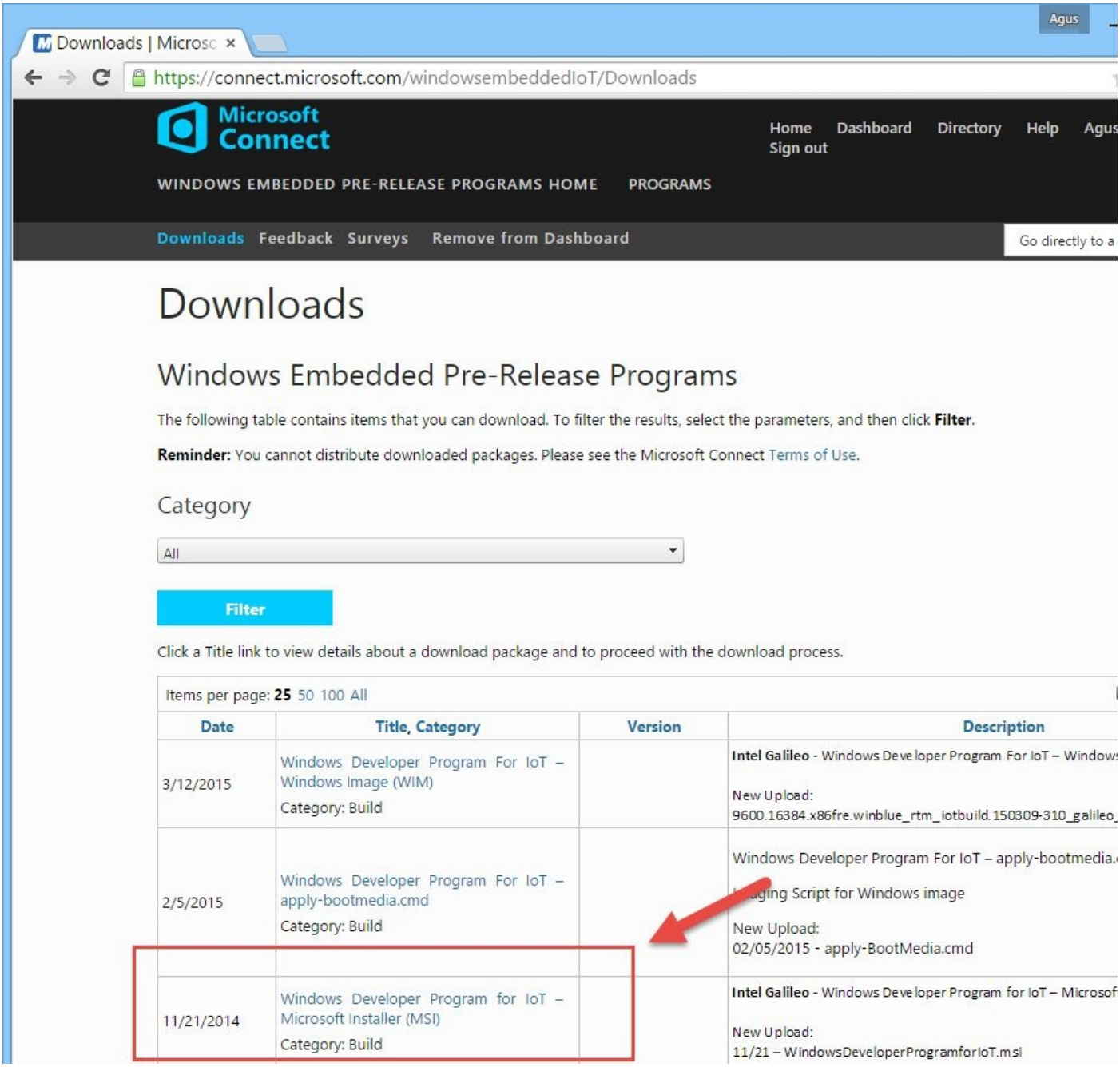
It will takes several minutes.

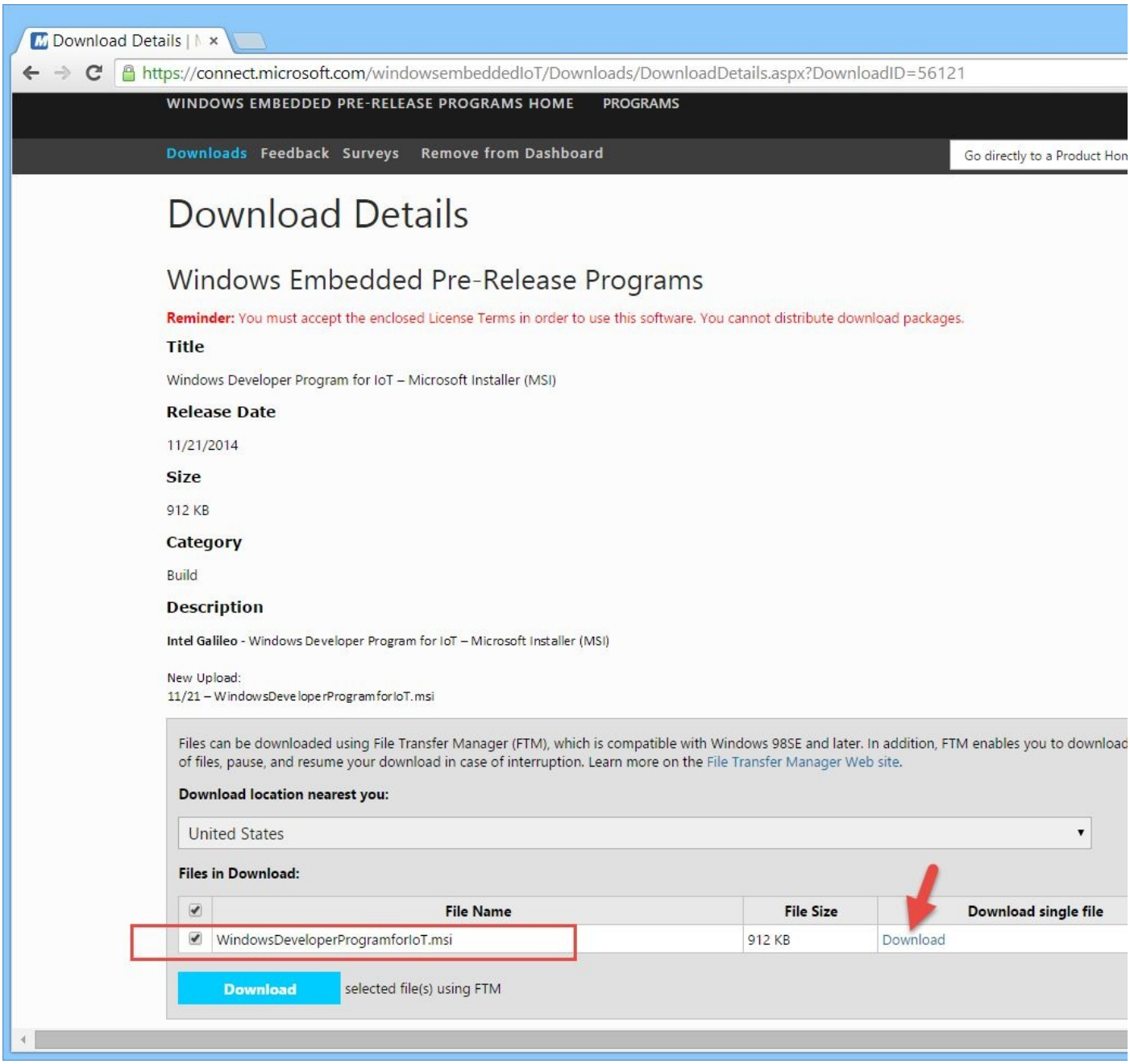


If done, you can see the image information, for instance, hostname, username and password.

2.4 Installing Development Tools

In this section, we setup our development tool. For illustration, I use Visual Studio 2013 Ultimate Edition. You can download Windows Developer Program For IoT - Microsoft Installer (MSI).





After downloaded, you can run *.msi file. You will get the installer dialog.

Follow the installation instructions.

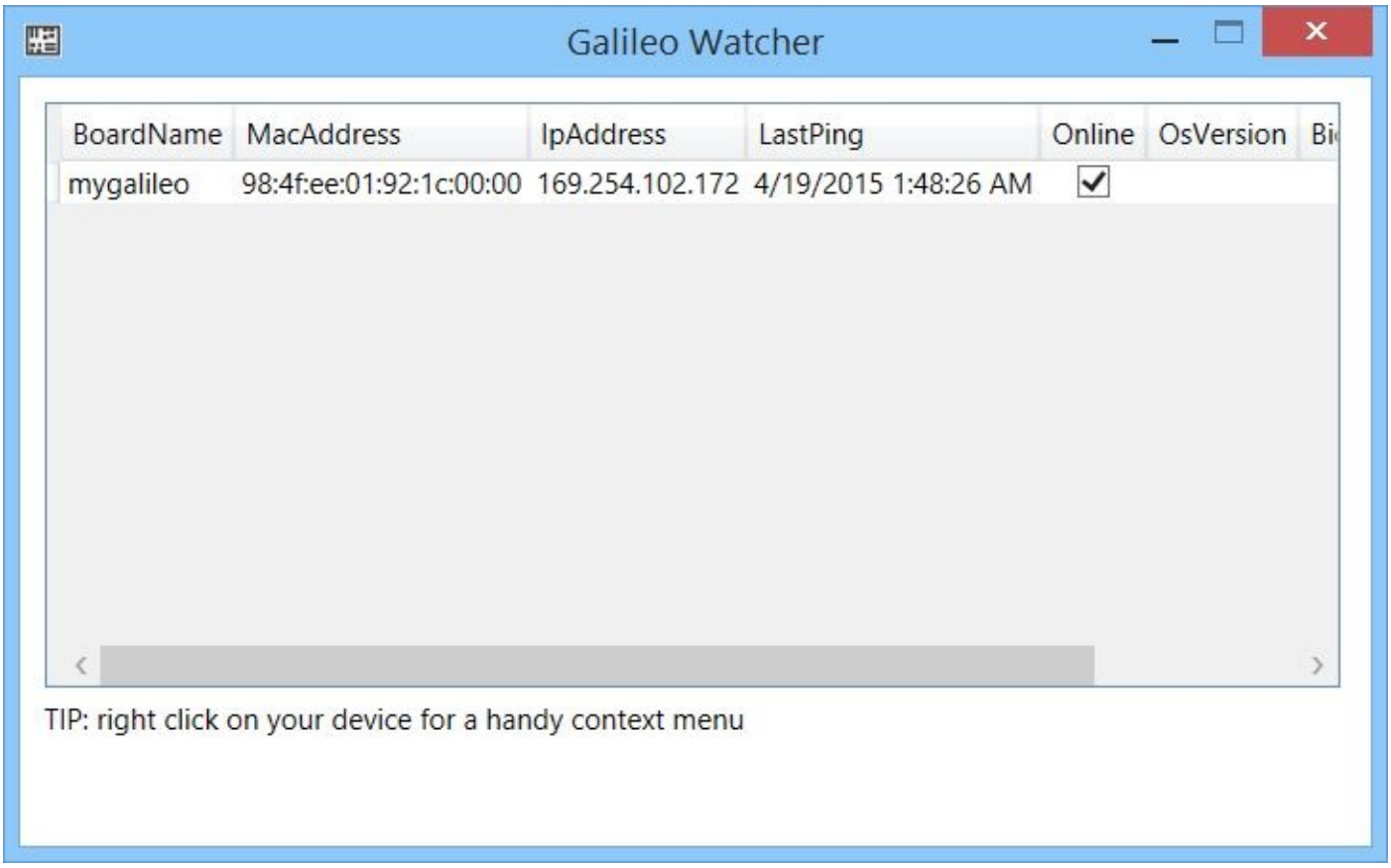
If finished, our Visual Studio 2013 will be added Windows For IoT project template. We will try it on section 2.9.

2.5 Connecting Intel Galileo and Computer

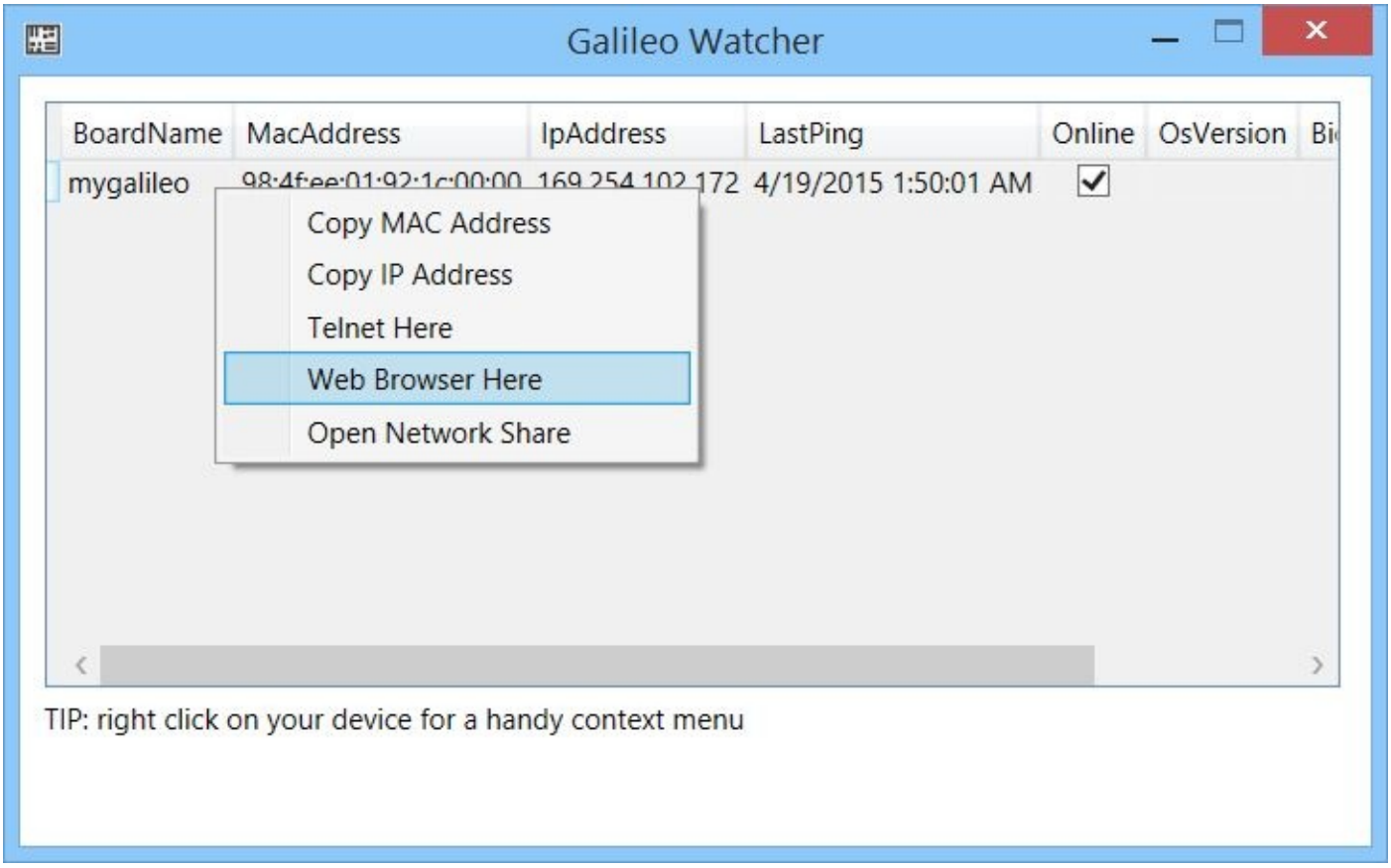
After we installed image on microSD and development tool on computer, you can put microSD into Intel Galileo board. We need UTP cable. We don’t need a cross cable. Connect Intel Galileo and computer via UTP cable.

Now you can plug in the power adapter into Intel Galileo. Wait it until Intel Galileo completes the booting process.

If finished, you can see your Intel Galileo on Galileo Watcher tool from computer.

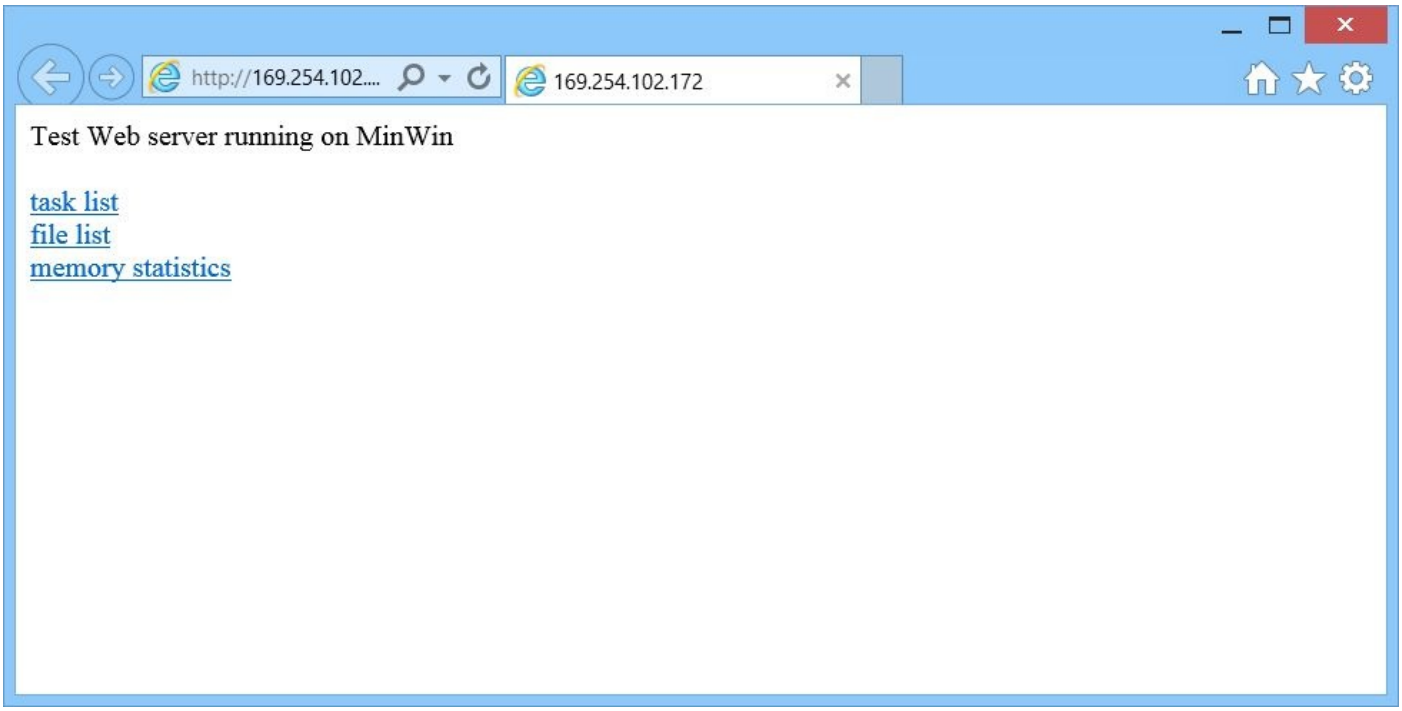


Try to do right-click on our board. Select **Web Browser Here** menu.



A browser will run and navigate to our web server of Intel Galileo board.

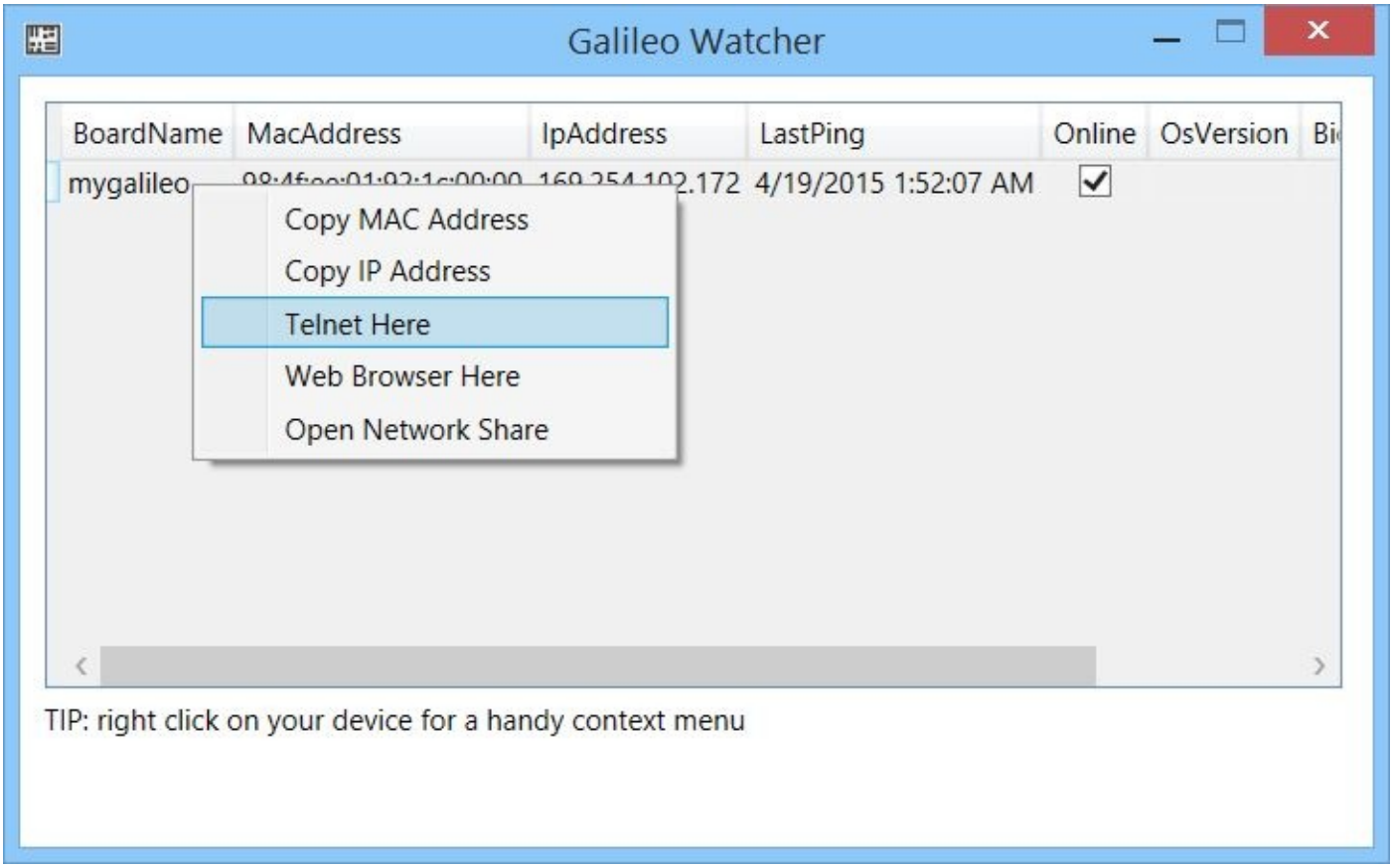
A sample output can be seen in Figure below.



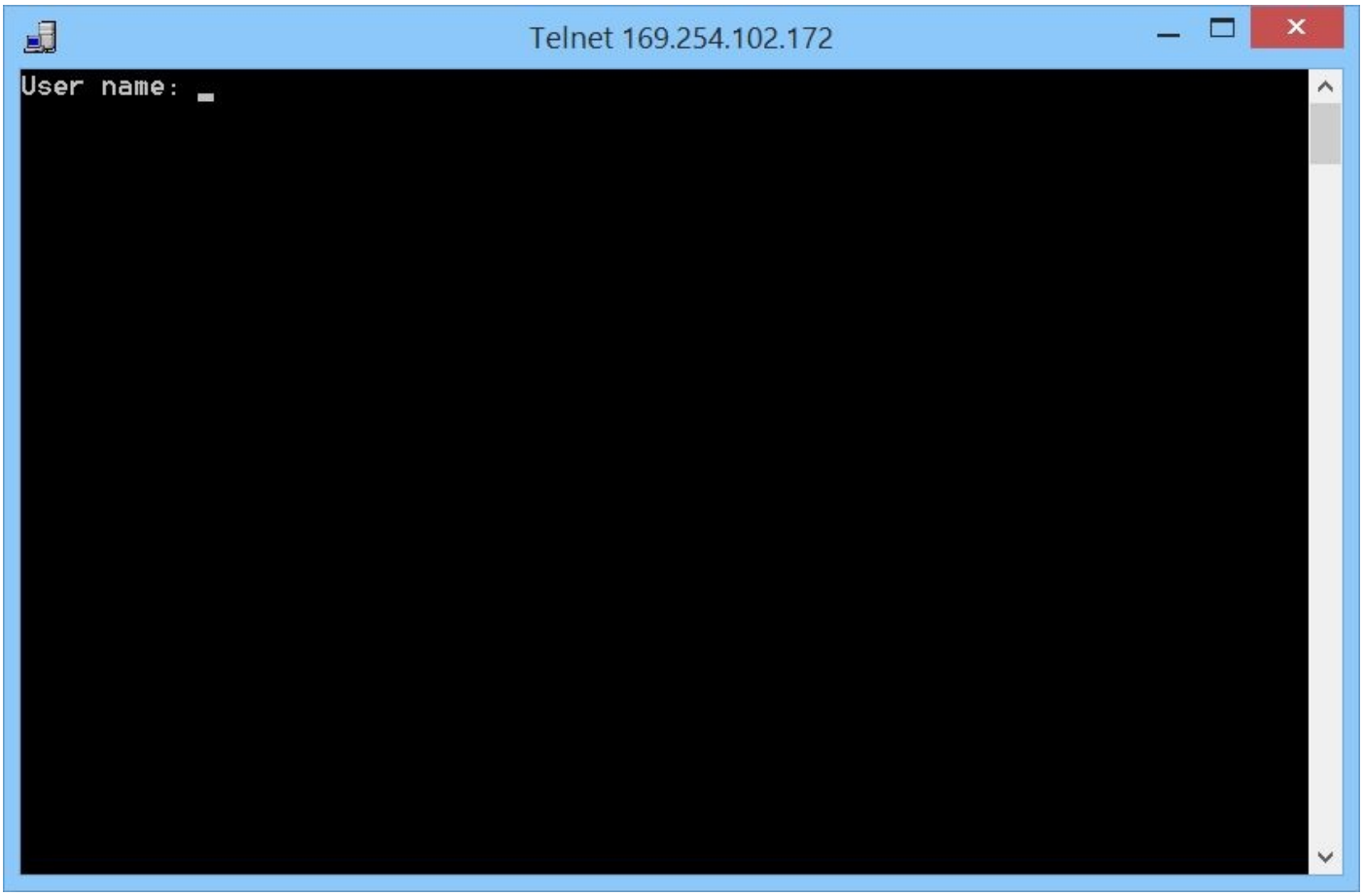
2.6 Telnet

We can connect to Intel Galileo via Telnet. Firstly, your computer must install Telnet client. You can get it from **Program and Features**.

From Galileo Watcher, right-click on Intel Galileo. Select **Telnet Here**.

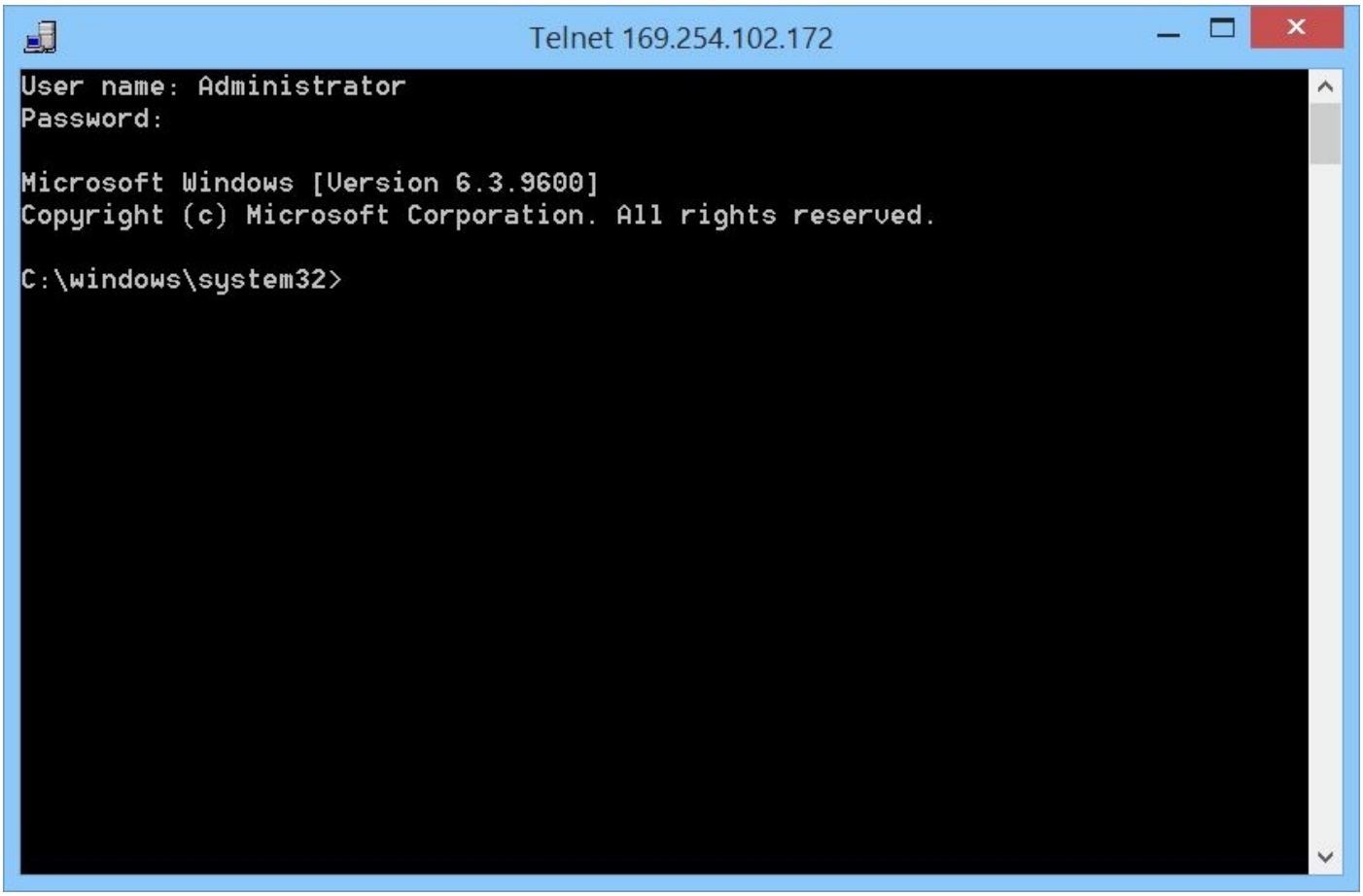


Then, you will get Command Prompt of Telnet.

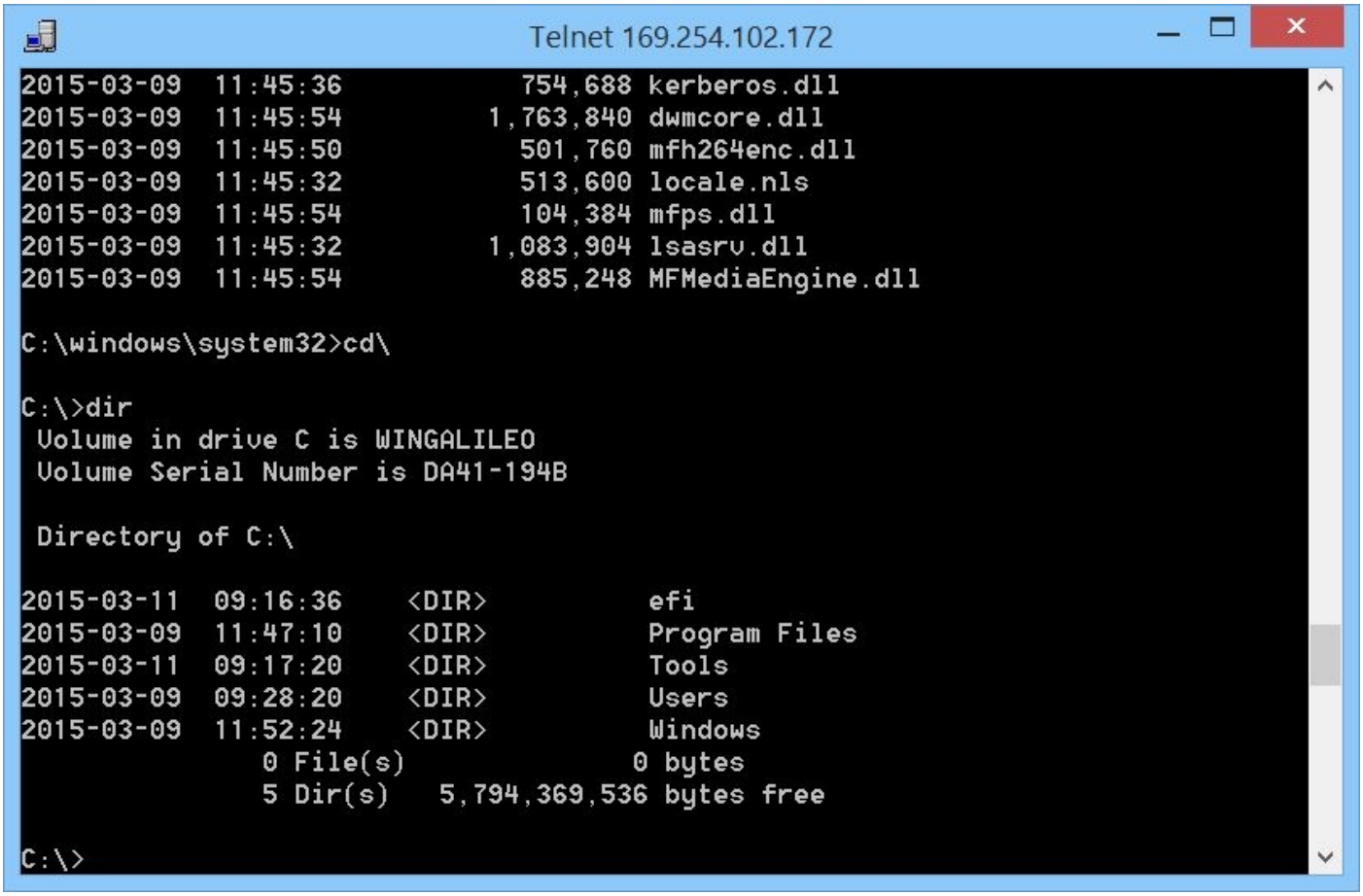


Entry your account. In this case, my Intel Galileo has configure with account

- Username: Administrator
- Password: admin
- Hostname: mygalileo



Now you can execute normal activities on Command Prompt.

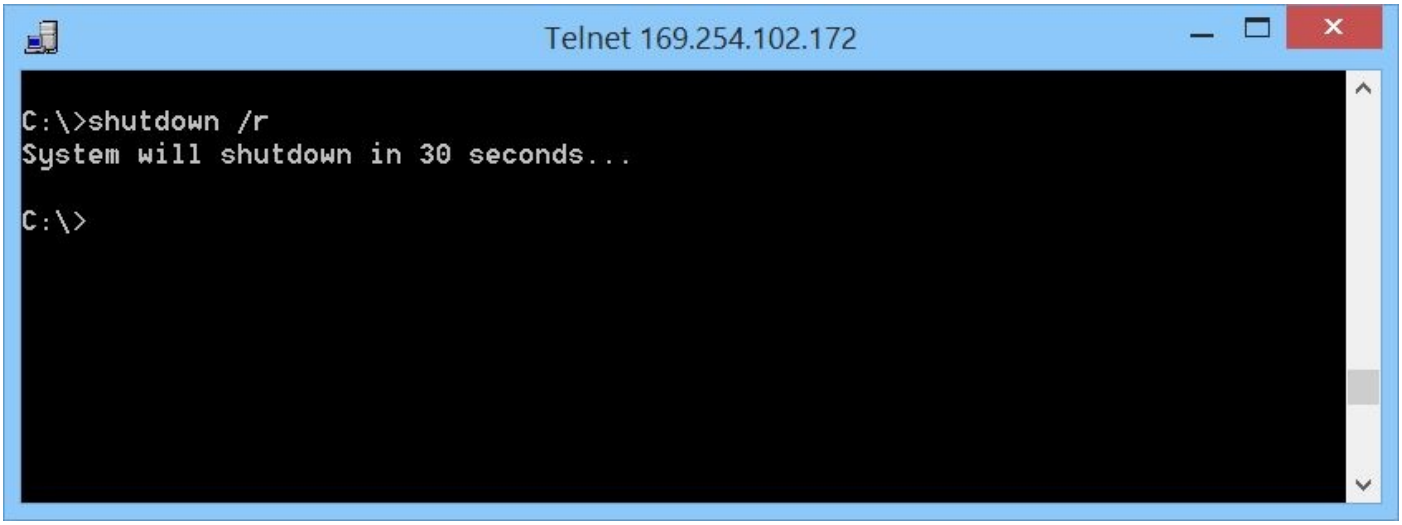


2.7 Reboot and Shutdown

We can restart and shutdown our Intel Galileo.

To restart, type this command on Command Prompt via Telnet.

```
shutdown /r
```

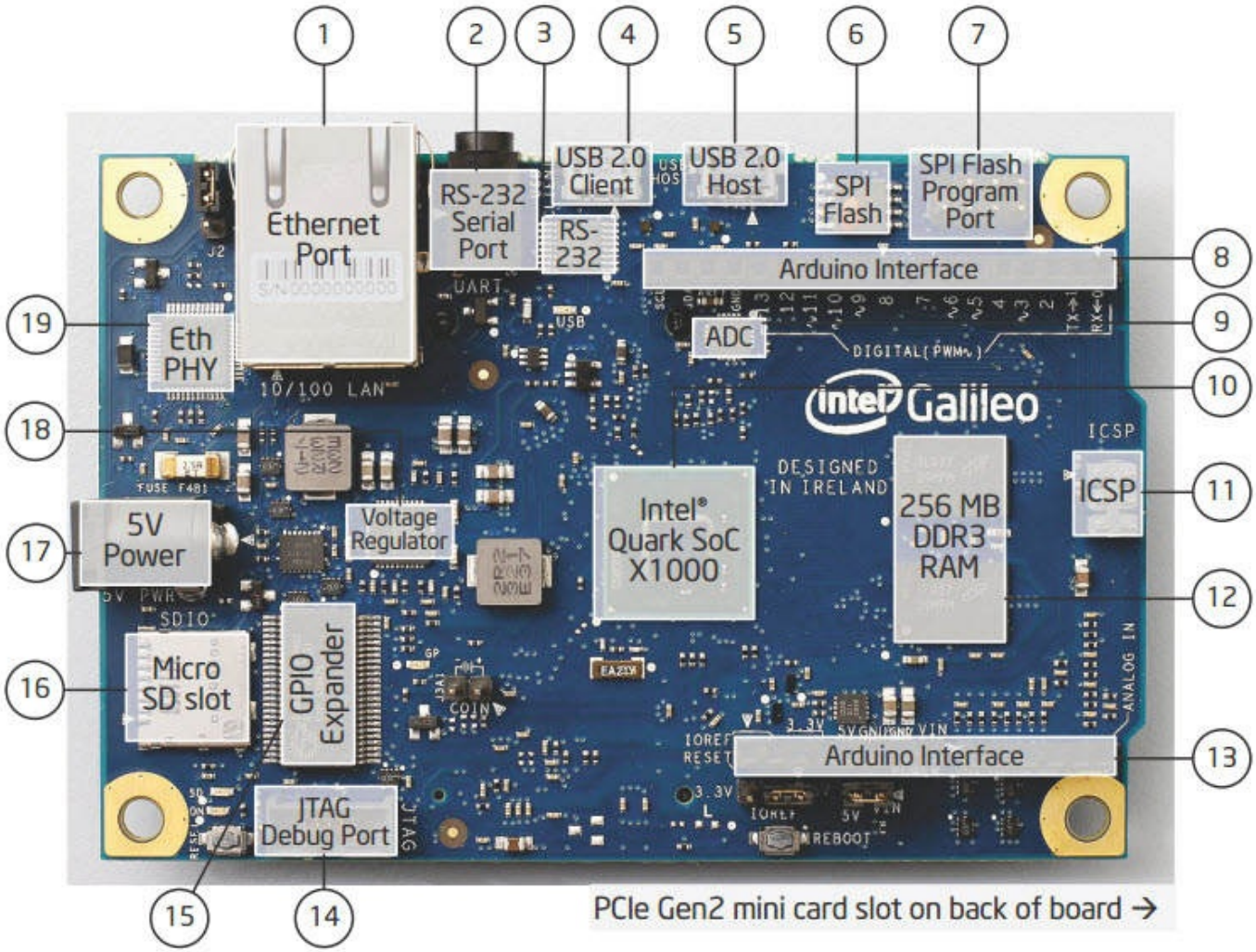


To shutdown, type this command on Command Prompt via Telnet.

```
shutdown /s
```

2.8 Intel Galileo Board

Firstly, we must understand Intel Galileo board schema. You can find information about the board on https://communities.intel.com/docs/DOC-22475 . The following is a board schema for Intel Galileo generation 1.

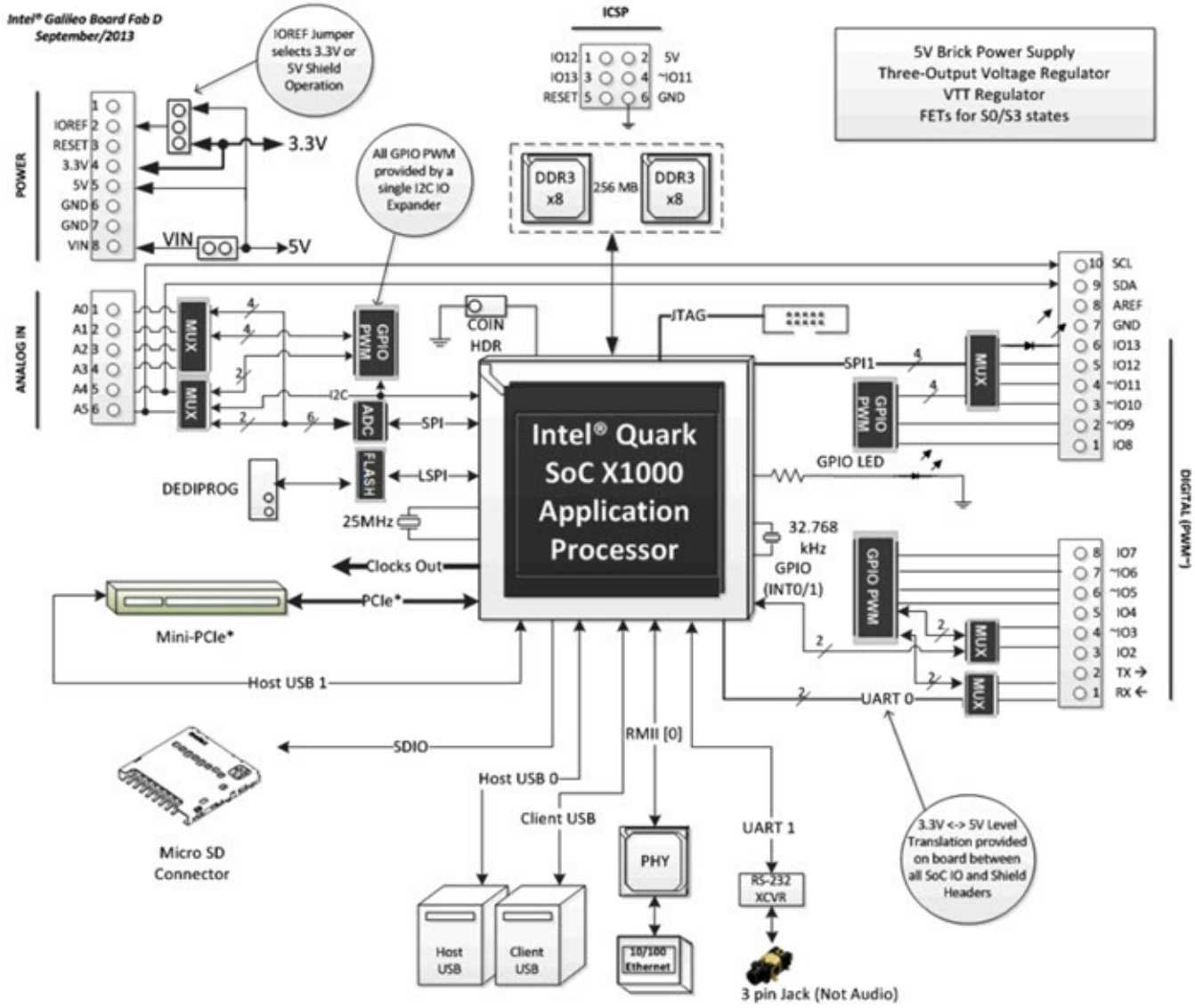


Note:

1. Ethernet Port
2. RS-232 Serial Port
3. RS-232
4. USB 2.0 Client
5. USB 2.0 Host
6. SPI Flash
7. SPI Flash Program Port
8. Shield Interface
9. ADC
10. Intel Quark SoC X1000
11. ICSP
12. 256 MB DDR3 RAM
13. Arduino Interface

14. JTAG Debug Port
15. GPIO Expander
16. Micro SD slot
17. 5V Power
18. Voltage Regulator
19. Eth PHY

The detail of board schema also can be found on https://communities.intel.com/docs/DOC-21822. I put it in Figure below.

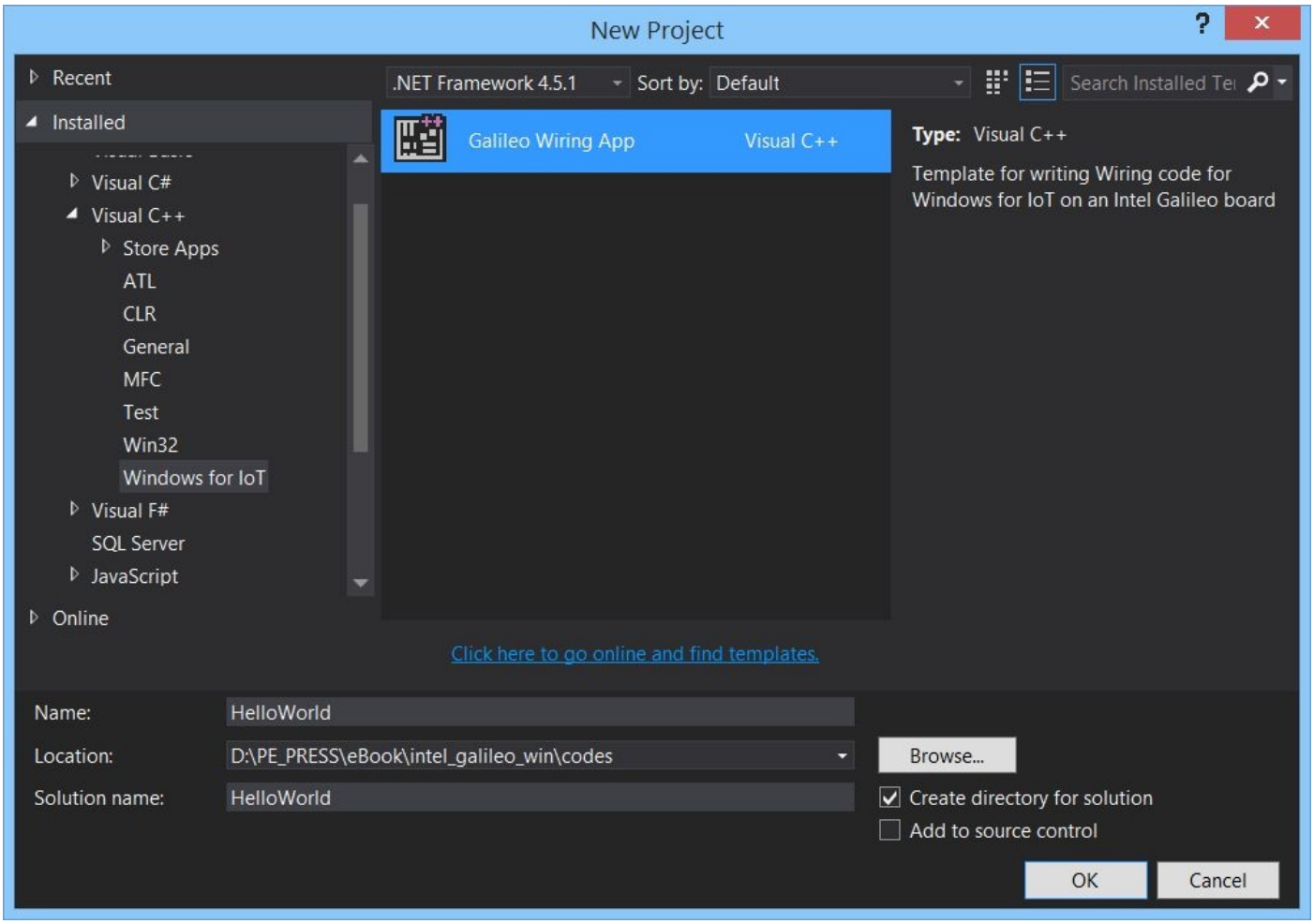


2.9 Hello World

In this scenario, we build a simple app, blinking LED.

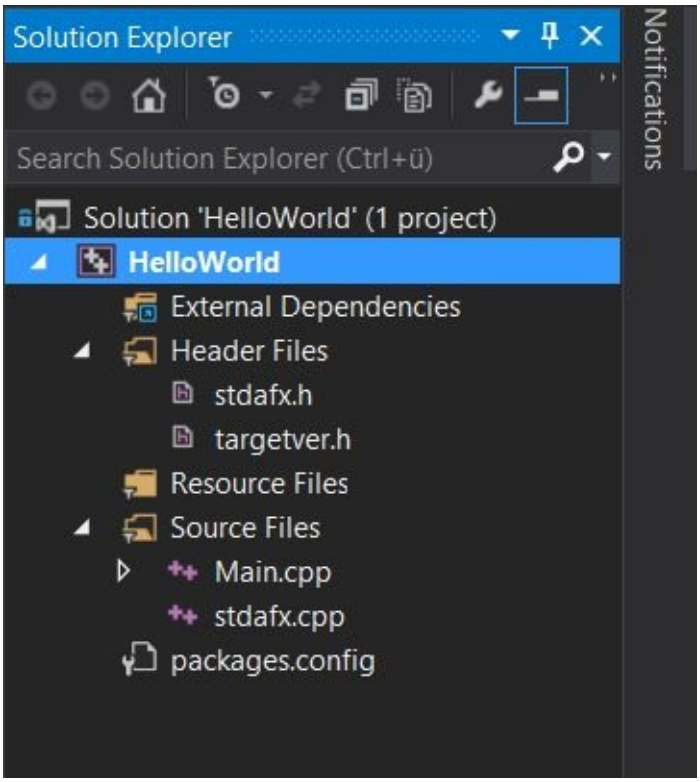
2.9.1 Creating A Project

Open Visual Studio 2013. Create Galileo Wiring App from Visual C++ —> Windows for IoT.

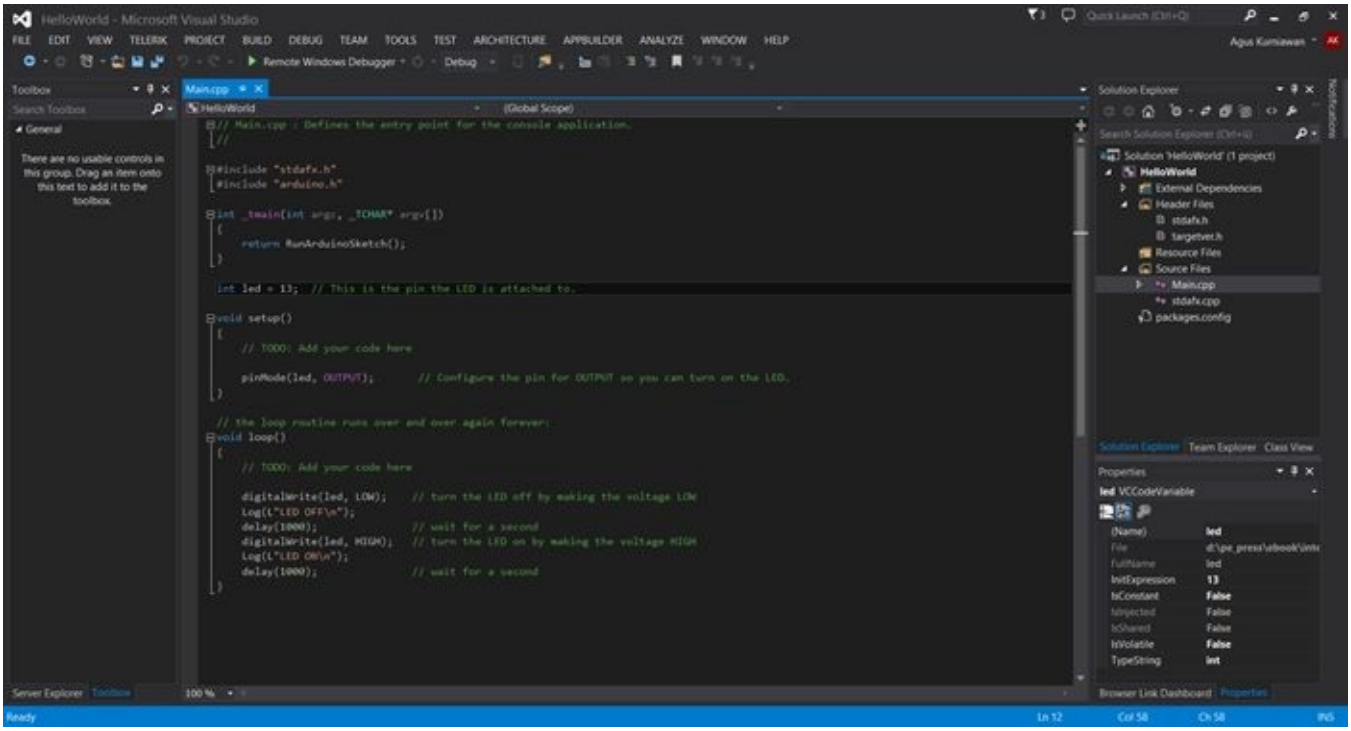


For instance, my project is **HelloWorld**.

After that, you can see our project structure, shown in Figure below.

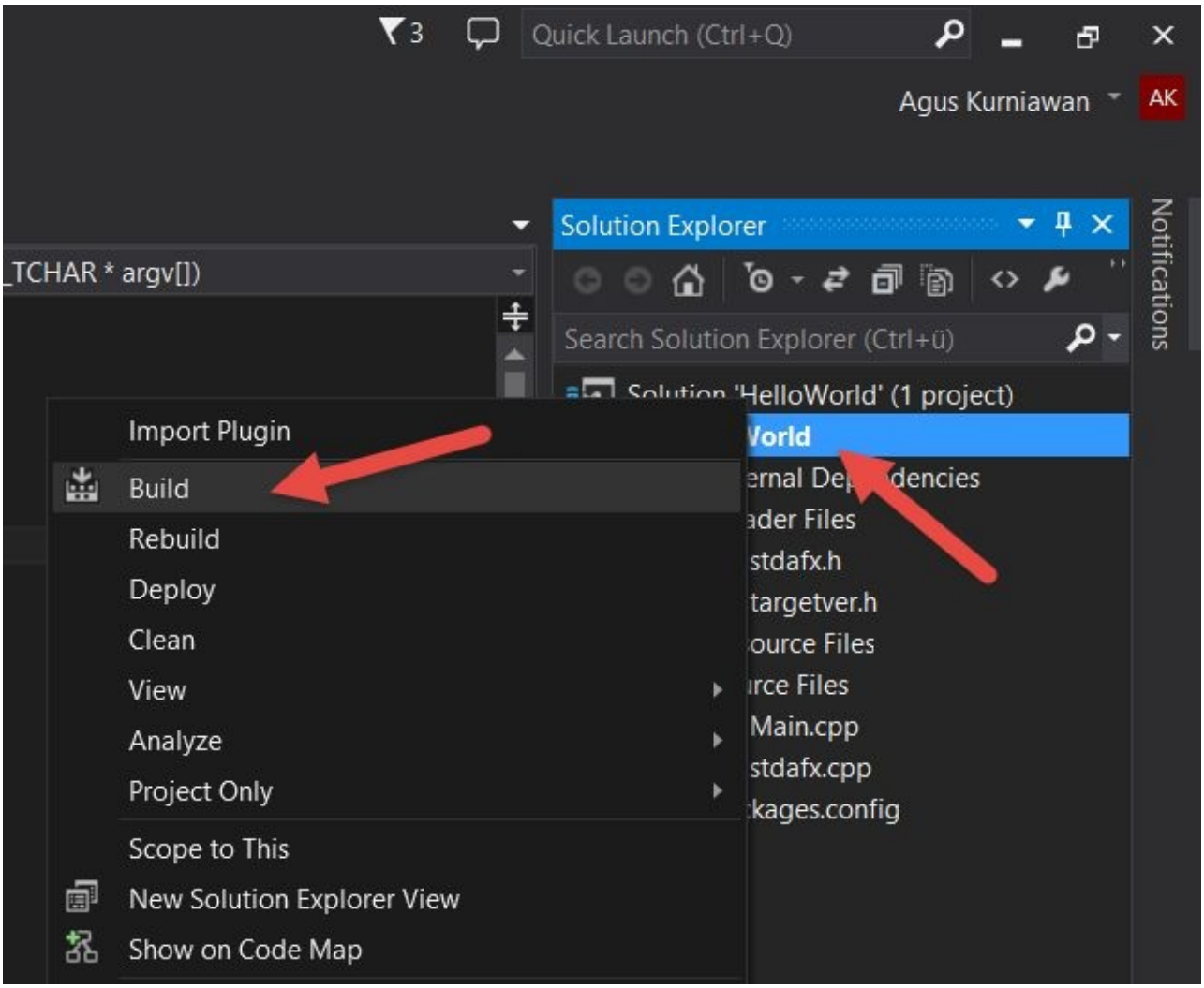


Open **Main.cpp** file, you can see our blinking LED.

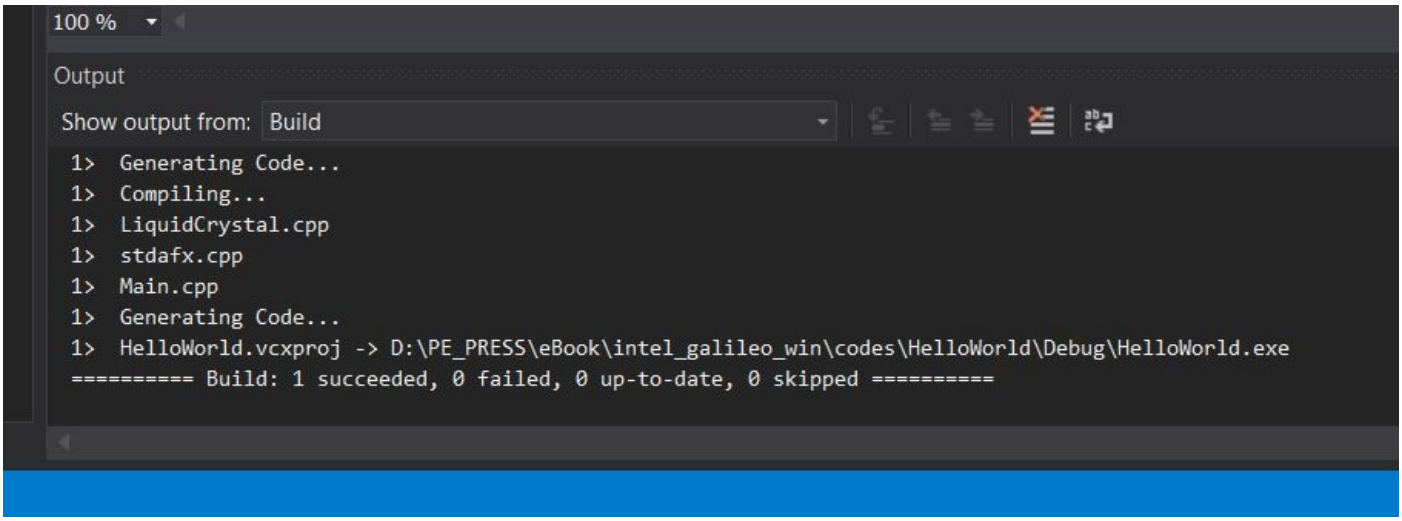


2.9.2 Compiling

To compile our program, you can right-click on the project. Select menu **Build**.



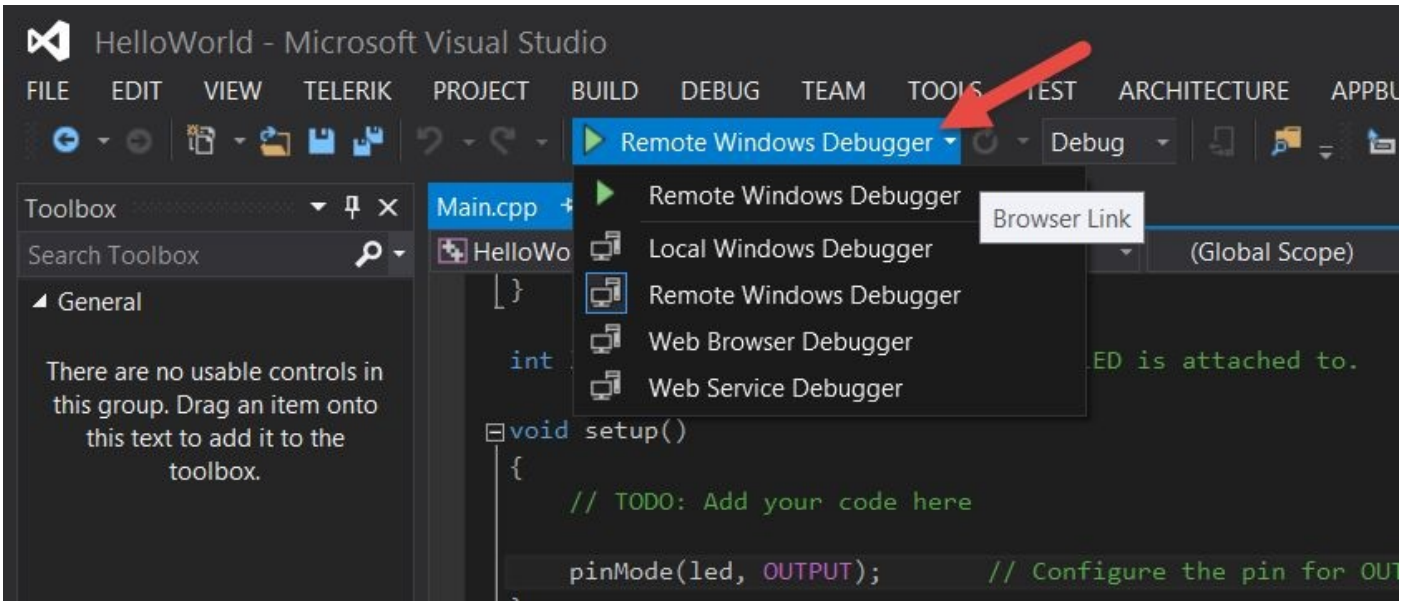
If success, you can see it on **Output** panel.



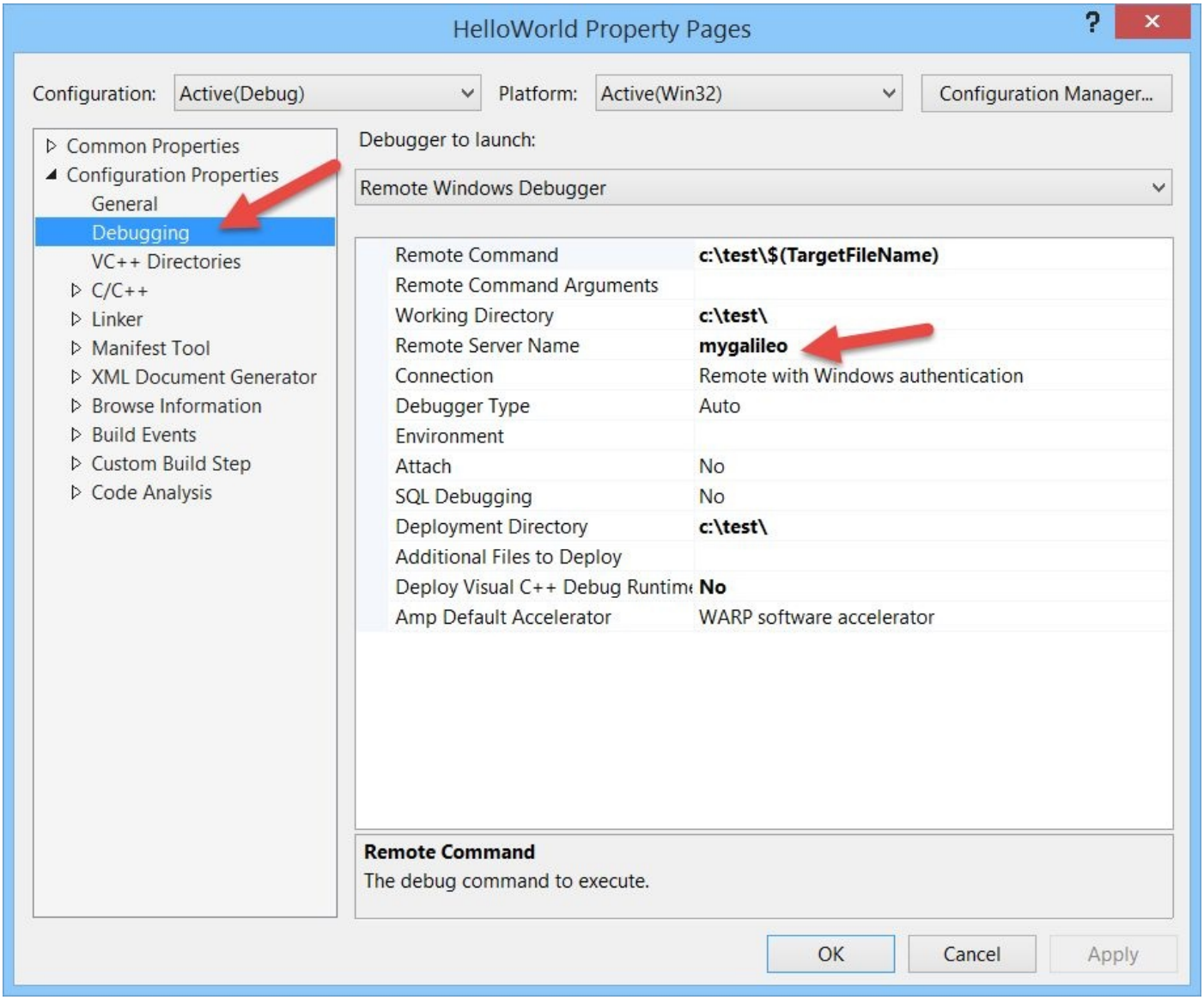
2.9.3 Running

Now you can deploy and debug the program into the board.

Click **Remote Windows Debugger**.



The default remote hostname is **mygalileo**. We have already configured it on section 2.3. If you change it, you must change the remote hostname. You can find it on Project Property.

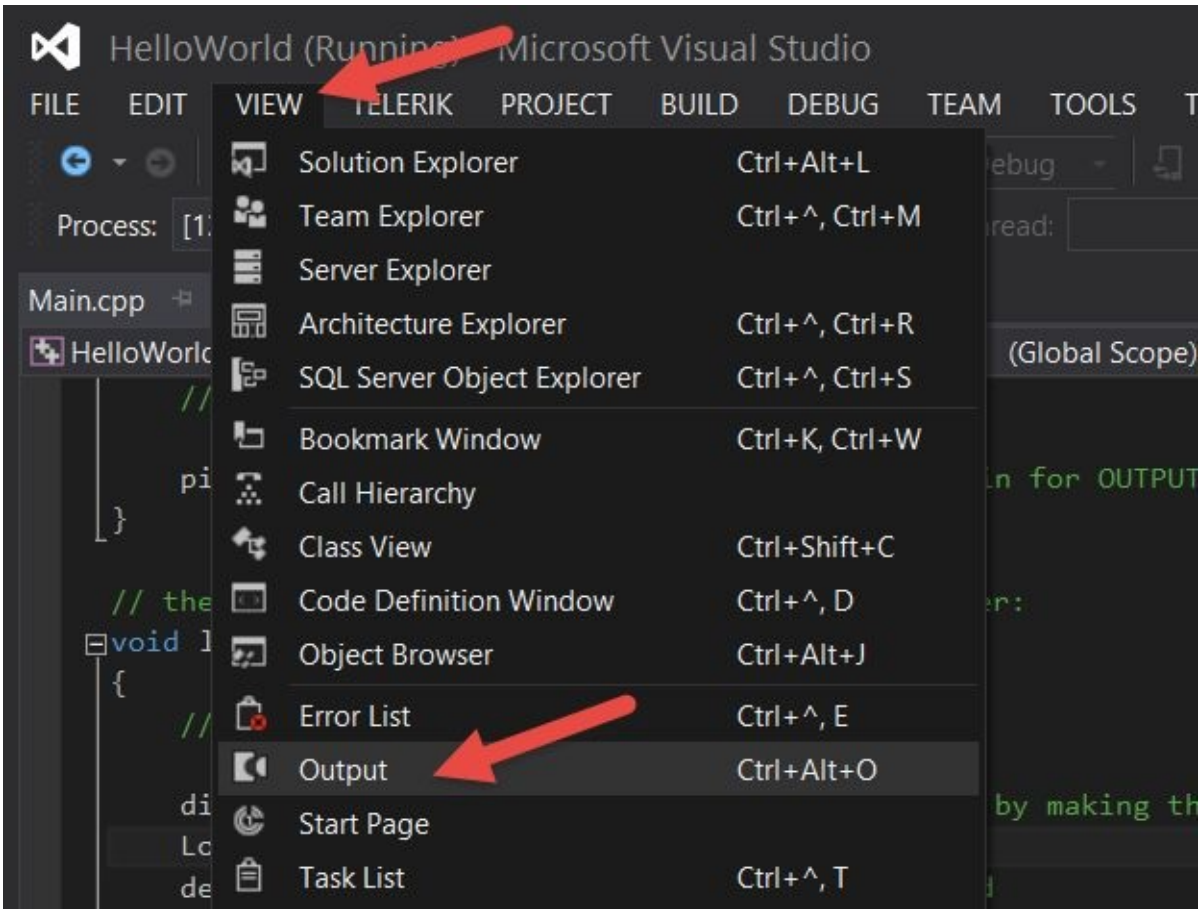


After executed, you will get a security dialog. Entry Intel Galileo account.

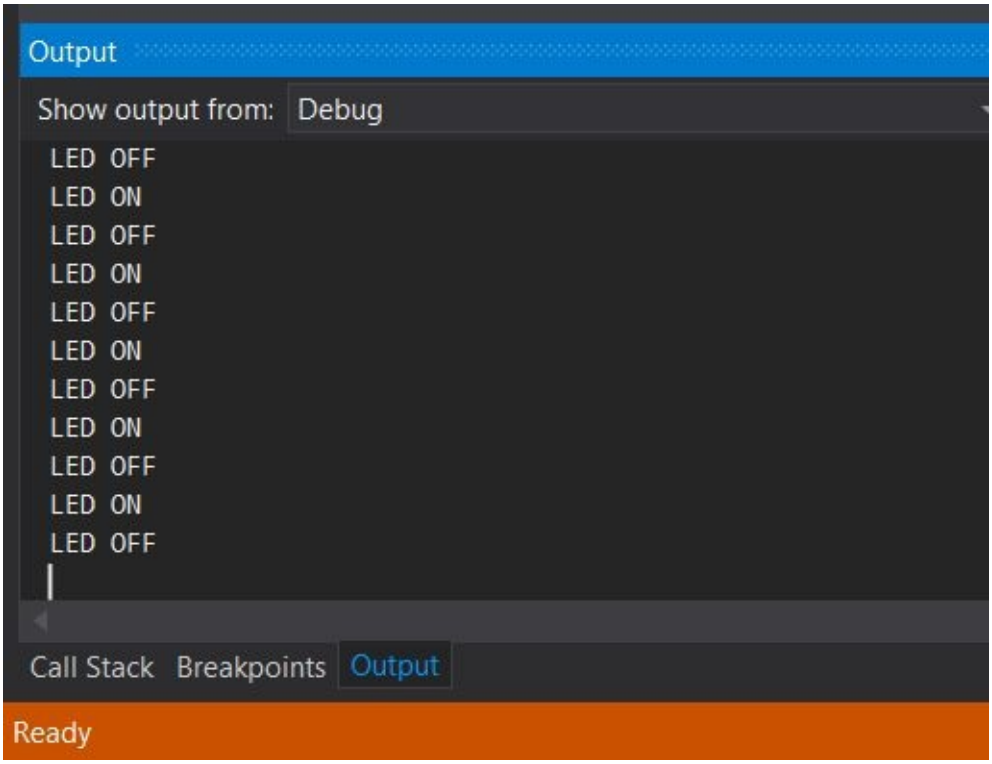
You can also do Telnet to Intel Galileo board. After that, you don’t need to authenticate for deploying.

If success, the program will run.

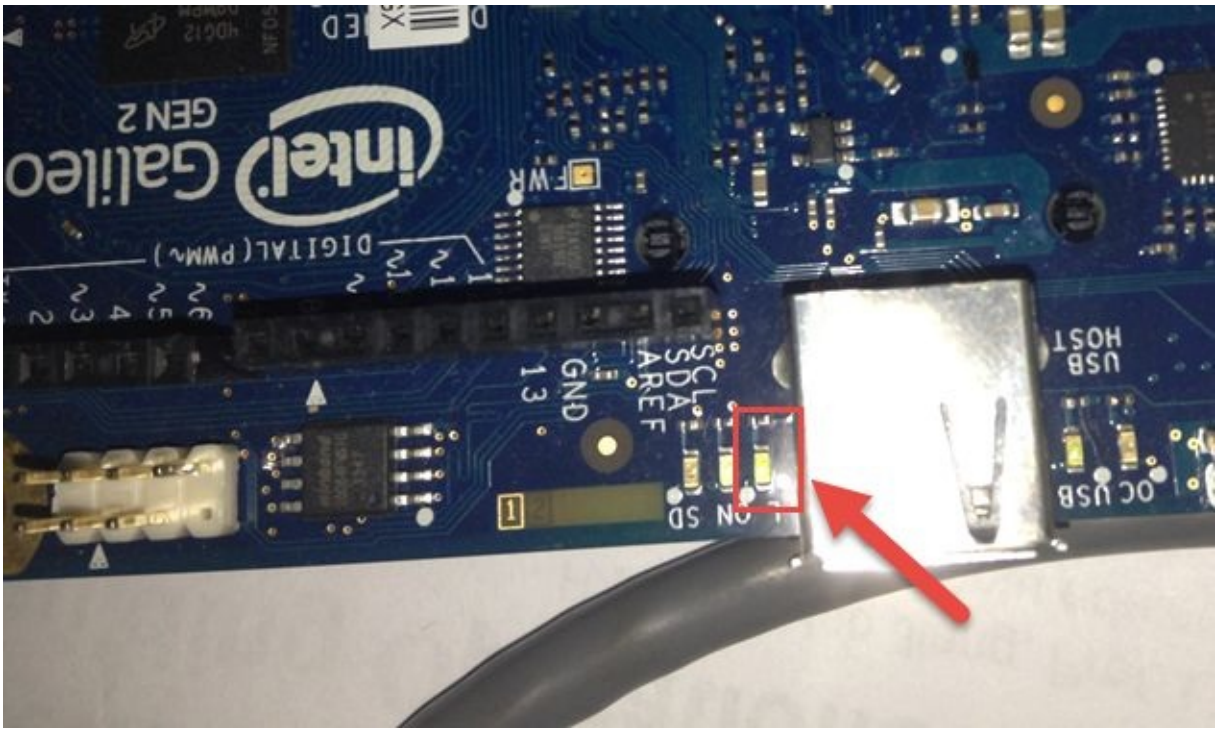
You can see the program output on **Output** panel. It can be shown by clicking View->Output menu.



You will see the program output on Output panel on Visual Studio.



You also see blinking LED on the board. See red sign in Figure below.


intel Galileo
GEN 2
DIGITAL(PWM~)
GND
SCL
SDA
AREF
USB HOST
OC USB

3. Digital I/O

This chapter explains how to work with Digital I/O using Windows for IoT on Intel Galileo board.

2.1 Getting Started

In this chapter, we are going to learn how to work with digital I/O using Windows for IoT on Intel Galileo. There are two demo we want to implement:

- Seven segment display
- Push button

Let’s start.

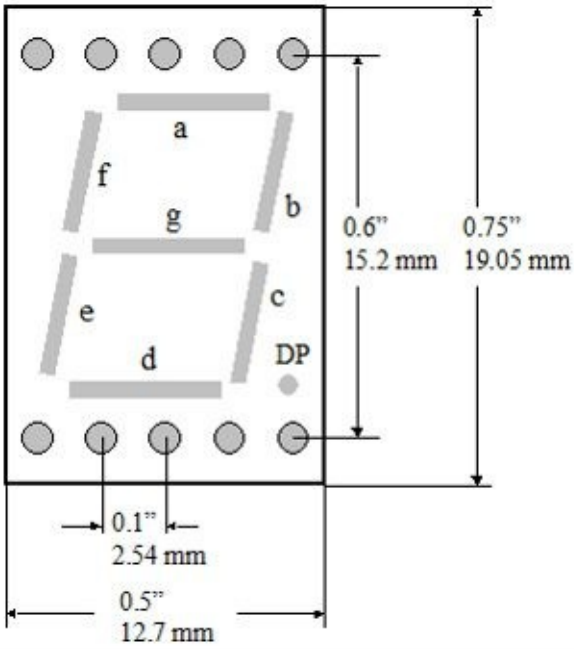
2.2 Seven-Segment Display

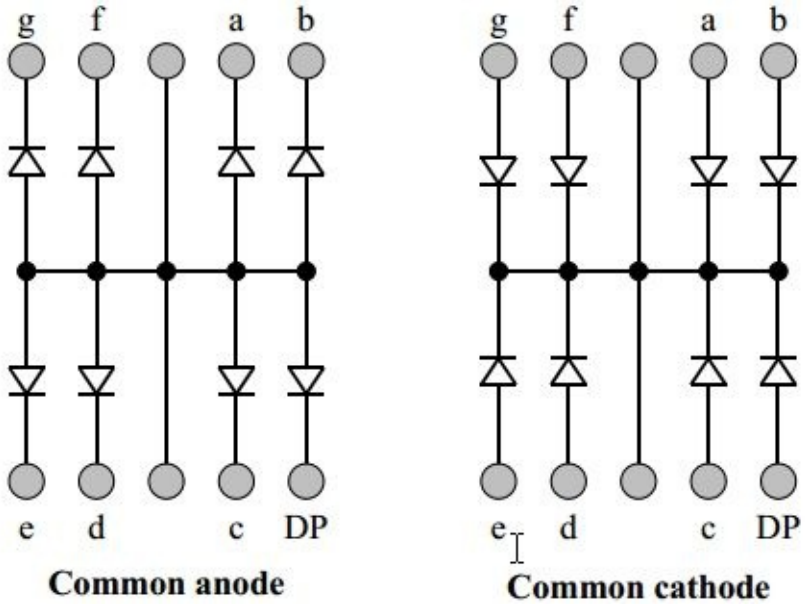
In this section, we're going to build an application to display number 0 to 9 on a 7 segment display device. This device will be connected to Intel Galileo via digital output pins and then we develop application to illustrate how to write data to digital output pins.

Let's start.

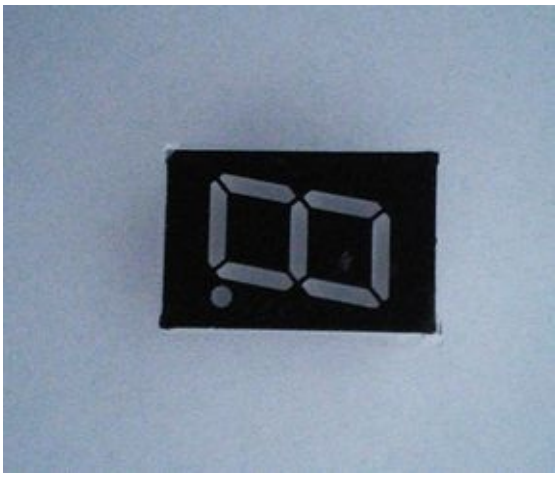
2.2.1 Getting Started with 7 Segment Display

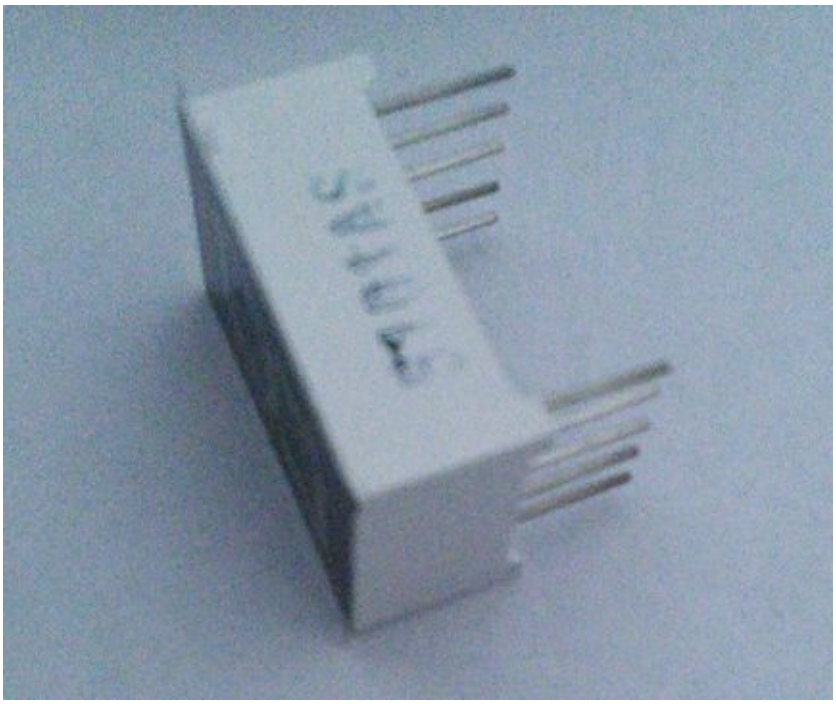
To understand 7 segment display, I refer to a datasheet from SA56-11GWA and SC56-11GWA , http://www.kitronik.co.uk/pdf/7_segment_display_datasheet.pdf . The following is device schema.





The following is a sample physical of 7 segment display.





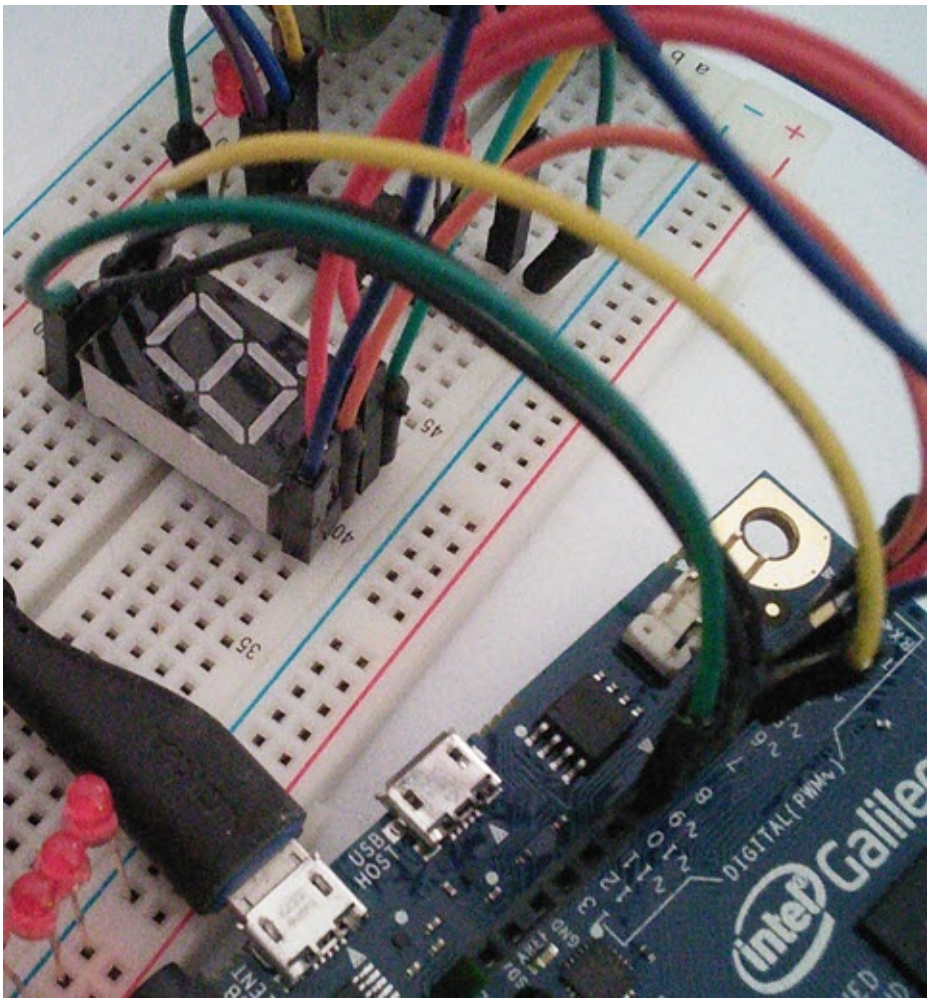
2.2.2 Wiring

The next step, we must connect our 7 segment to Intel Galileo.

Here is pin configuration from 7 segment display to Intel Galileo.

- Digital 2 —> pin DP
- Digital 3 —> pin c
- Digital 4 —> pin d
- Digital 5 —> pin e
- Digital 6 —> pin b
- Digital 7 —> pin a
- Digital 8 —> pin f
- Digital 9 —> pin g
- GND —> pin GND (center pin of 7 segment / Unlabeled pin)

The following is a hardware implementation for connecting 7 segment display to Intel Galileo.



2.2.3 Building Application

Algorithm to display number on 7 segment display is easy. We just combine value on 7 segment display pins. For instance, to display number 0, we set value LOW on the g pin. Otherwise, the pins are set to HIGH value.

Create a new project, Intel Galileo, with project name, **SevenSegment**. Read section 2.9.1.

The following is implementation to display value 0,1…9 on 7 segment display on **Main.cpp** file.

```
#include "stdafx.h"
#include "arduino.h"

int _tmain(int argc, _TCHAR* argv[])
{
    return RunArduinoSketch();
}
```

```
int pin_a = 7;
int pin_b = 6;
int pin_c = 3;
int pin_d = 4;
int pin_e = 5;
int pin_f = 8;
int pin_g = 9;
int pin_dp = 2;

void setup()
{
        pinMode(pin_a, OUTPUT);
        pinMode(pin_b, OUTPUT);
        pinMode(pin_c, OUTPUT);
        pinMode(pin_d, OUTPUT);
        pinMode(pin_e, OUTPUT);
        pinMode(pin_f, OUTPUT);
        pinMode(pin_g, OUTPUT);
        pinMode(pin_dp, OUTPUT);
}

void displaynull(){
        digitalWrite(pin_a, LOW);
        digitalWrite(pin_b, LOW);
        digitalWrite(pin_c, LOW);
        digitalWrite(pin_d, LOW);
        digitalWrite(pin_e, LOW);
        digitalWrite(pin_f, LOW);
        digitalWrite(pin_g, LOW);
        digitalWrite(pin_dp, LOW);
}

void display(int num){
        switch (num){
        case 0:
                digitalWrite(pin_a, HIGH);
                digitalWrite(pin_b, HIGH);
                digitalWrite(pin_c, HIGH);
                digitalWrite(pin_d, HIGH);
                digitalWrite(pin_e, HIGH);
                digitalWrite(pin_f, HIGH);
                digitalWrite(pin_g, LOW);
                digitalWrite(pin_dp, HIGH);
                break;
        case 1:
                digitalWrite(pin_a, LOW);
                digitalWrite(pin_b, HIGH);
                digitalWrite(pin_c, HIGH);
                digitalWrite(pin_d, LOW);
                digitalWrite(pin_e, LOW);
                digitalWrite(pin_f, LOW);
                digitalWrite(pin_g, LOW);
                digitalWrite(pin_dp, HIGH);
```

```
            break;
        case 2:
            digitalWrite(pin_a, HIGH);
            digitalWrite(pin_b, HIGH);
            digitalWrite(pin_c, LOW);
            digitalWrite(pin_d, HIGH);
            digitalWrite(pin_e, HIGH);
            digitalWrite(pin_f, LOW);
            digitalWrite(pin_g, HIGH);
            digitalWrite(pin_dp, HIGH);
            break;
        case 3:
            digitalWrite(pin_a, HIGH);
            digitalWrite(pin_b, HIGH);
            digitalWrite(pin_c, HIGH);
            digitalWrite(pin_d, HIGH);
            digitalWrite(pin_e, LOW);
            digitalWrite(pin_f, LOW);
            digitalWrite(pin_g, HIGH);
            digitalWrite(pin_dp, HIGH);
            break;
        case 4:
            digitalWrite(pin_a, LOW);
            digitalWrite(pin_b, HIGH);
            digitalWrite(pin_c, HIGH);
            digitalWrite(pin_d, LOW);
            digitalWrite(pin_e, LOW);
            digitalWrite(pin_f, HIGH);
            digitalWrite(pin_g, HIGH);
            digitalWrite(pin_dp, HIGH);
            break;
        case 5:
            digitalWrite(pin_a, HIGH);
            digitalWrite(pin_b, LOW);
            digitalWrite(pin_c, HIGH);
            digitalWrite(pin_d, HIGH);
            digitalWrite(pin_e, LOW);
            digitalWrite(pin_f, HIGH);
            digitalWrite(pin_g, HIGH);
            digitalWrite(pin_dp, HIGH);
            break;
        case 6:
            digitalWrite(pin_a, HIGH);
            digitalWrite(pin_b, LOW);
            digitalWrite(pin_c, HIGH);
            digitalWrite(pin_d, HIGH);
            digitalWrite(pin_e, HIGH);
            digitalWrite(pin_f, HIGH);
            digitalWrite(pin_g, HIGH);
            digitalWrite(pin_dp, HIGH);
            break;
        case 7:
            digitalWrite(pin_a, HIGH);
```

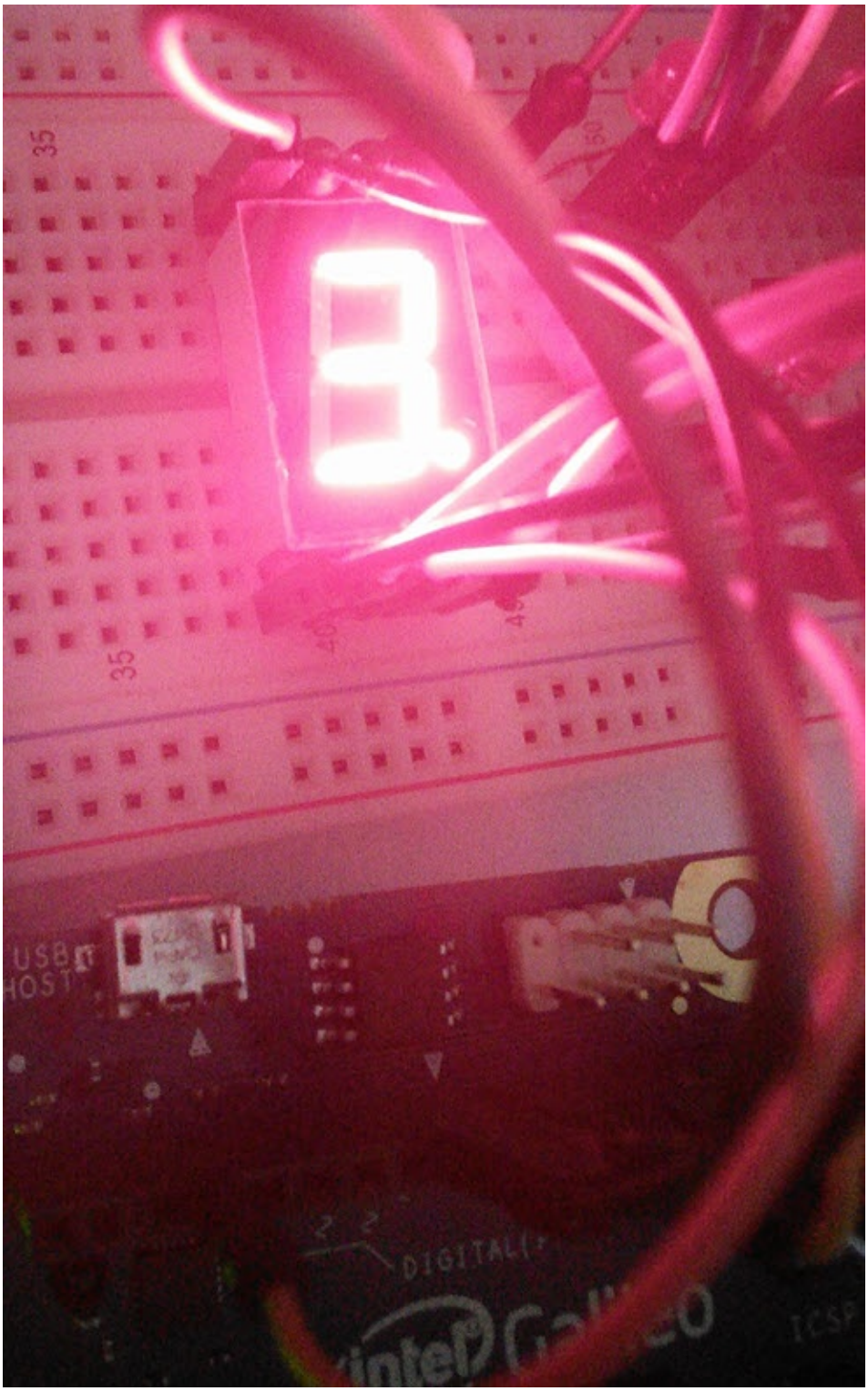
```
                digitalWrite(pin_b, HIGH);
                digitalWrite(pin_c, HIGH);
                digitalWrite(pin_d, LOW);
                digitalWrite(pin_e, LOW);
                digitalWrite(pin_f, LOW);
                digitalWrite(pin_g, LOW);
                digitalWrite(pin_dp, HIGH);
                break;
        case 8:
                digitalWrite(pin_a, HIGH);
                digitalWrite(pin_b, HIGH);
                digitalWrite(pin_c, HIGH);
                digitalWrite(pin_d, HIGH);
                digitalWrite(pin_e, HIGH);
                digitalWrite(pin_f, HIGH);
                digitalWrite(pin_g, HIGH);
                digitalWrite(pin_dp, HIGH);
                break;
        case 9:
                digitalWrite(pin_a, HIGH);
                digitalWrite(pin_b, HIGH);
                digitalWrite(pin_c, HIGH);
                digitalWrite(pin_d, HIGH);
                digitalWrite(pin_e, LOW);
                digitalWrite(pin_f, HIGH);
                digitalWrite(pin_g, HIGH);
                digitalWrite(pin_dp, HIGH);
                break;
        }

}

// the loop routine runs over and over again forever:
void loop()
{
        displaynull();
        delay(1000);
        Log(L"0\n");
        display(0);
        delay(1000);
        displaynull();
        delay(1000);
        Log(L"1\n");
        display(1);
        delay(1000);
        displaynull();
        delay(1000);
        Log(L"2\n");
        display(2);
        delay(1000);
        displaynull();
        delay(1000);
        Log(L"3\n");
```

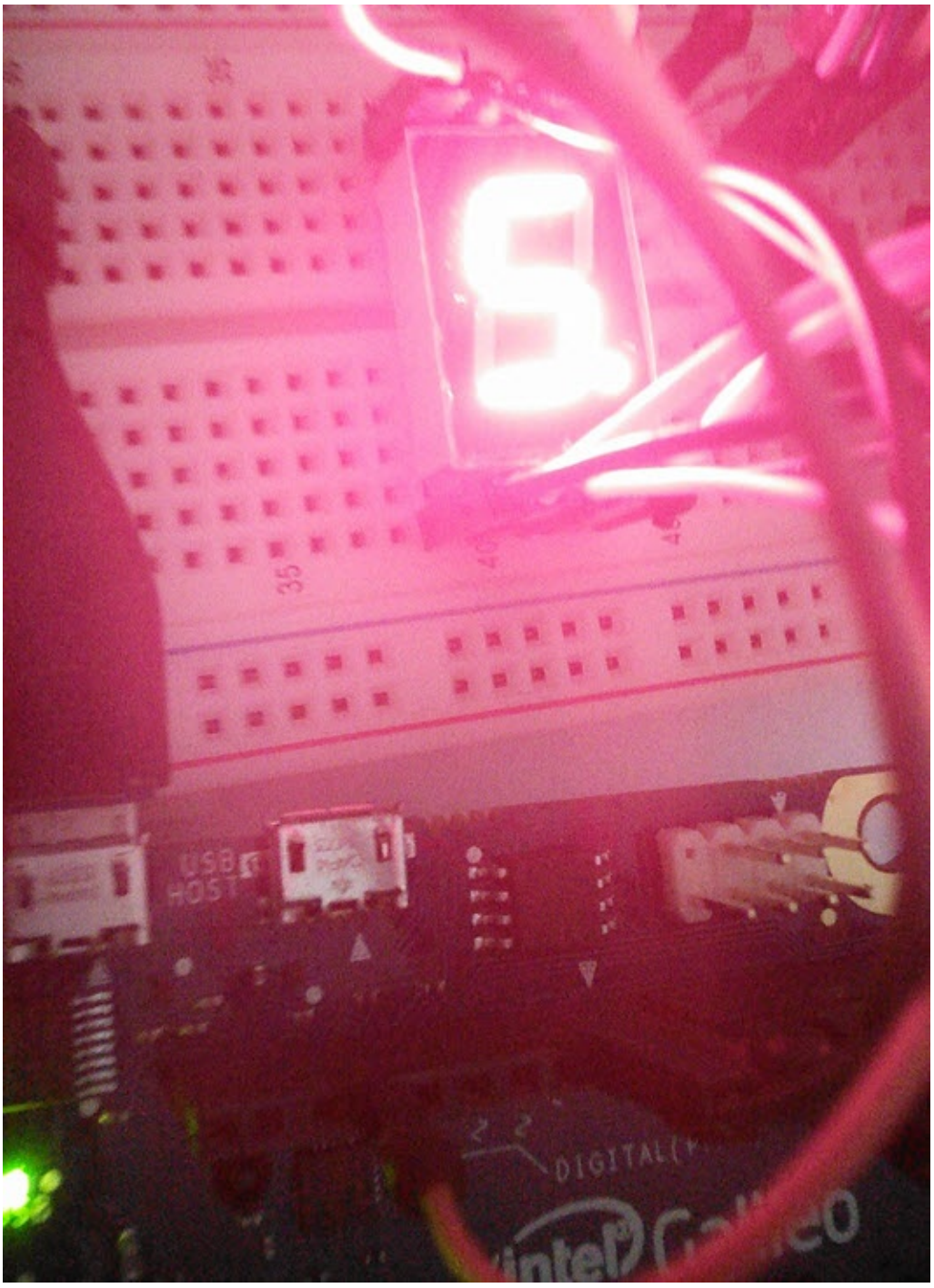
```
        display(3);
        delay(1000);
        displaynull();
        delay(1000);
        Log(L"4\n");
        display(4);
        delay(1000);
        displaynull();
        delay(1000);
        Log(L"5\n");
        display(5);
        delay(1000);
        displaynull();
        delay(1000);
        Log(L"6\n");
        display(6);
        delay(1000);
        displaynull();
        delay(1000);
        Log(L"7\n");
        display(7);
        delay(1000);
        displaynull();
        delay(1000);
        Log(L"8\n");
        display(8);
        delay(1000);
        displaynull();
        delay(1000);
        Log(L"9\n");
        display(9);
        delay(1000);
}
```

Save this code.

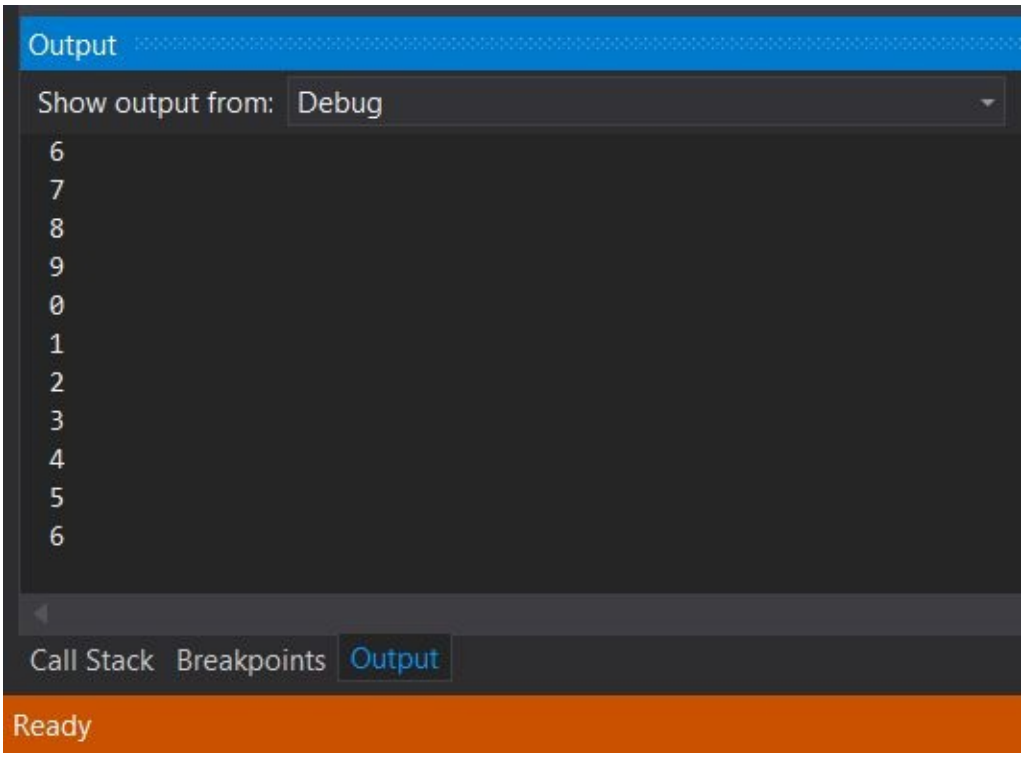
2.2.4 Testing

Now you can run this application by uploading program to Intel Galileo board. The following is sample outputs for 7 segment on Intel Galileo board.


35
USB
HOST
DIGITAL



You can see debug output on Visual Studio too.

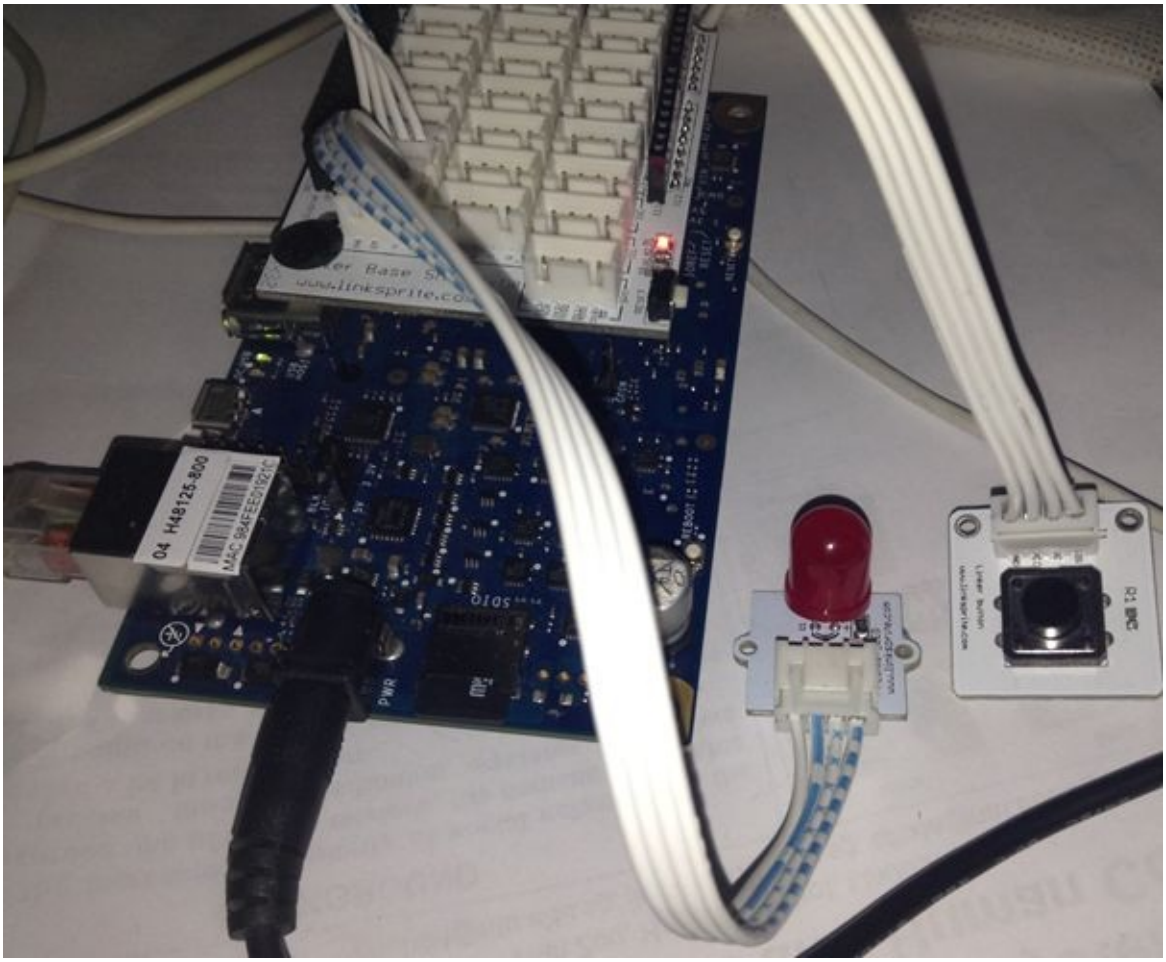

Output
Show output from: Debug
6
7
8
9
0
1
2
3
4
5
6
Call Stack
Breakpoints
Output
Ready

2.3 Getting Digital Input: Push Button Demo

The second demo is to build a simple app to use digital input on Intel Galileo board. In this demo, we need a button and a LED.

2.3.1 Wiring

In this case, you can configure a button to digital pin 5 and a LED on digital pin 6. The following is my wiring.



2.3.2 Writing Program

Now you can create Intel Galileo Wiring, called **PushButton**. On **Main.cpp**, write the following code.

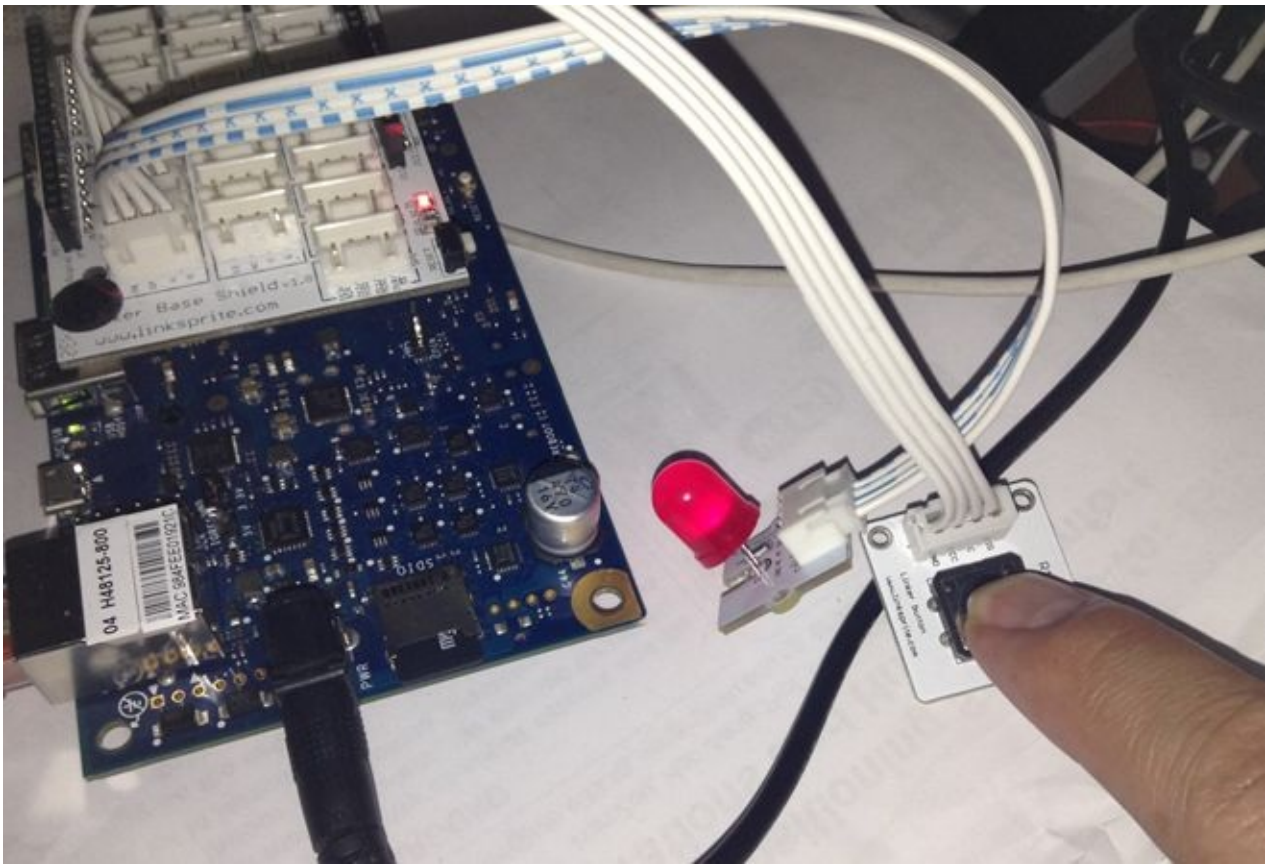
```
#include "stdafx.h"
#include "arduino.h"

int _tmain(int argc, _TCHAR* argv[])
{
```

```
    return RunArduinoSketch();
}

int led = 6;
int button = 5;

void setup()
{
    pinMode(led, OUTPUT);
    pinMode(button, INPUT);
}

void loop()
{
   if (digitalRead(button) == HIGH)
   {
        digitalWrite(led, HIGH);
   }
   else
   {
        digitalWrite(led, LOW);
   }
}
```

In this code, we check button state. If a button is pressed, we turn on our LED. Otherwise, we turn off a LED if button doesn’t be pressed.

2.3.3 Testing

Now you can compile this program and debug this program. Try to press a button. Then, you can see a LED is ON.



4. Analog I/O

This chapter explains how to work with Analog I/O using Windows for IoT on Intel Galileo board.

4.1 Getting Started

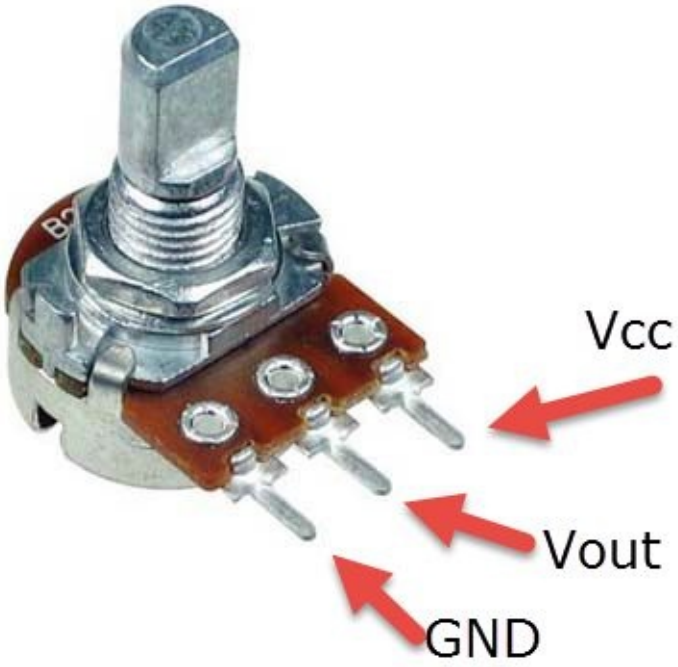
To illustrate how to work with analog I/O, we build two demo about analog input and analog output. You can follow the demo instructions on next section.

4.2 Reading Analog Input

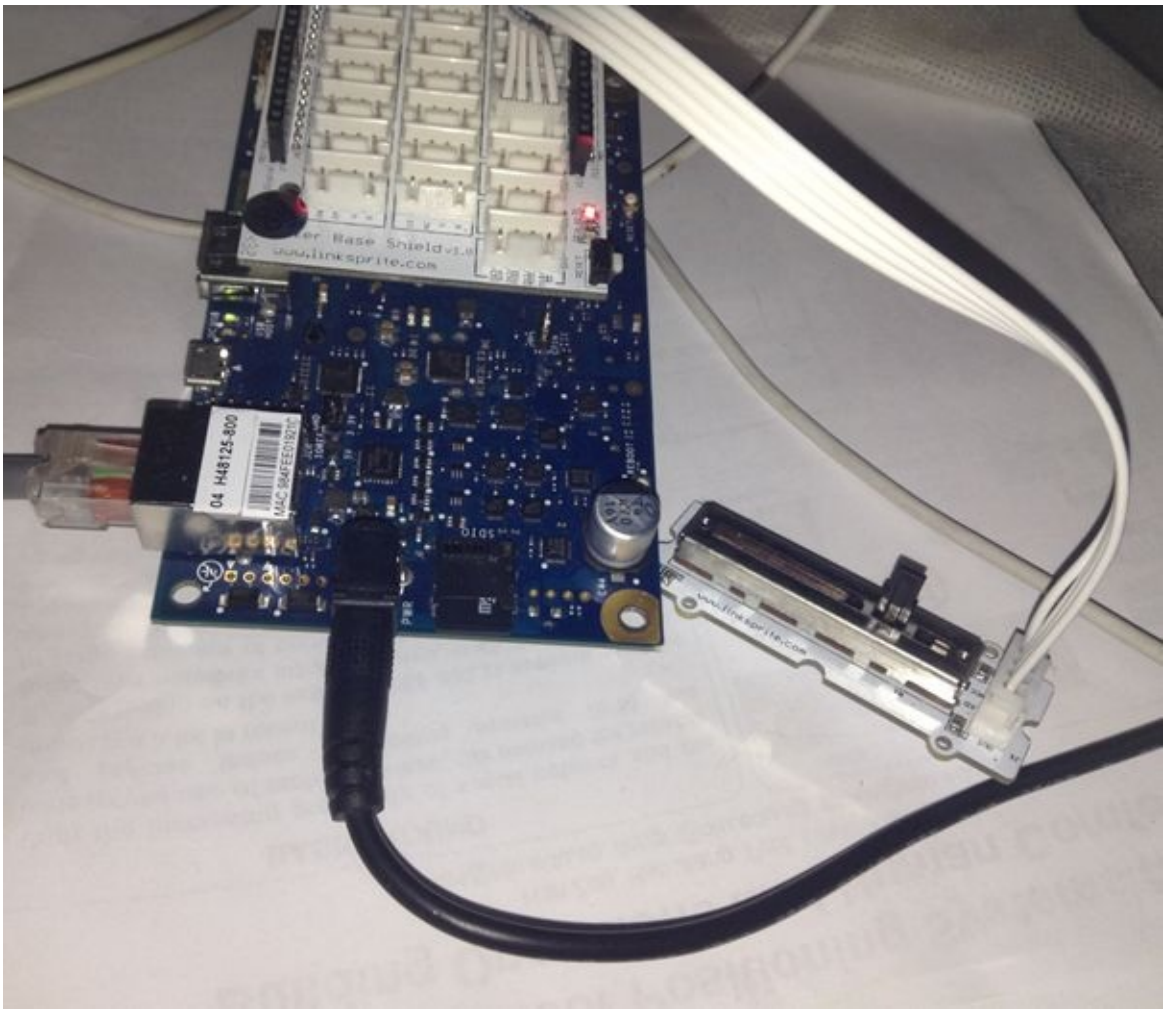
In this section, we learn how to read analog input on Intel Galileo. For illustration, I use Potentiometer as analog input source. Let’s start!.

4.2.1 Hardware Configuration

To understand Potentiometer, you see its scheme in Figure below.



You can connect VCC to Intel Galileo VCC 5V. Vout to Intel Galileo Analog input A0. In addition, GND to Intel Galileo GND. The following is hardware implementation. I use slide potentiometer.



4.2.2 Writing Application

We can use analogRead(), http://arduino.cc/en/Reference/AnalogWrite , to read analog input in Intel Galileo.

Now open Visual Studio 2013. Create Galileo Wiring, called AnalogInputDemo. Then, write the following code on **Main.cpp** file.

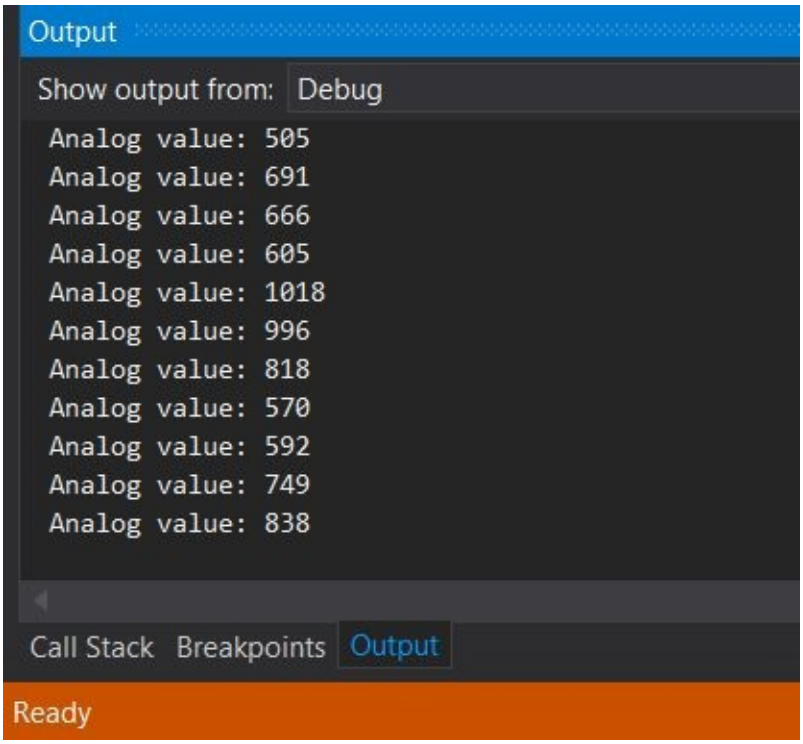
```
#include "stdafx.h"
#include "arduino.h"

int _tmain(int argc, _TCHAR* argv[])
{
    return RunArduinoSketch();
}

int potentiometer = 0;
int val = 0;

void setup()
{

}
```

```
// the loop routine runs over and over again forever:
void loop()
{
        val = analogRead(potentiometer);
        Log(L"Analog value: %d\n",val);
        delay(1000);
}
```

Save this code

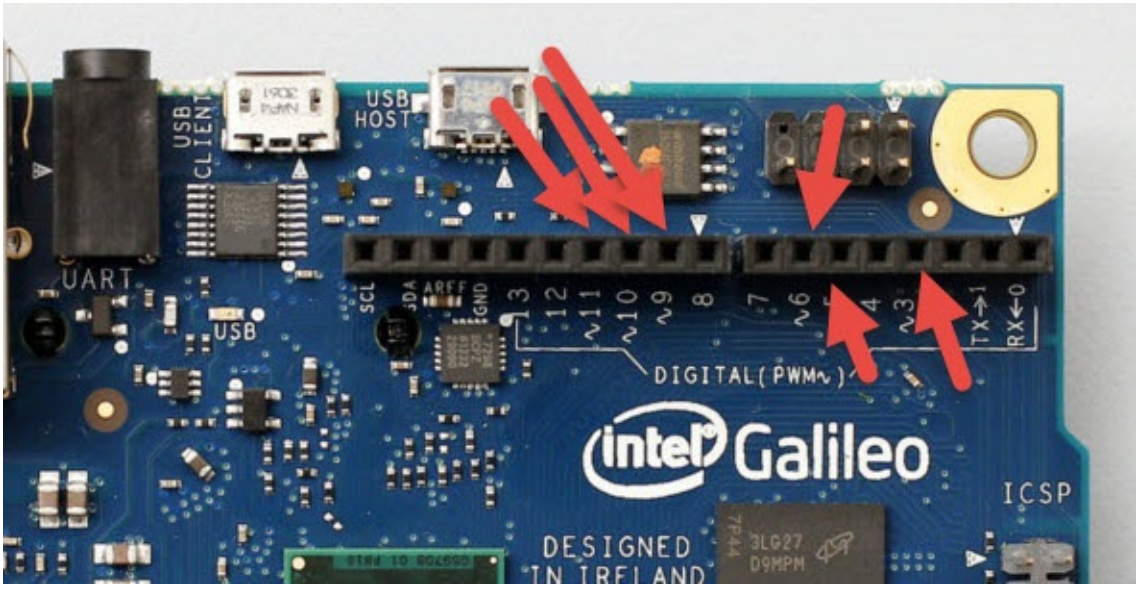
4.2.3 Testing

Compile and upload the program to Intel Galileo board. A sample output on Visual Studio can be seen in Figure below.



4.3 Working with Analog Output

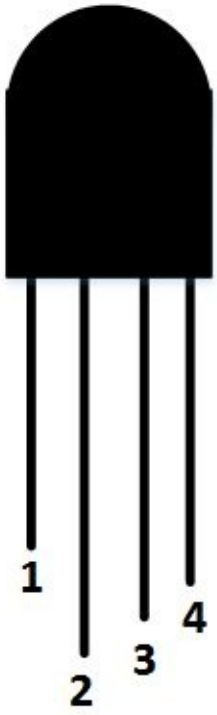
The last section, we learn how to work with analog output on Intel Galileo. We use RGB LED which sets colors by giving R, G and B values on its pins. We use Intel Galileo Analog output (PWM). If you see the board, PWM pins are identified by 2 sign. You can see PWM pins on the following Figure below.



4.3.1 RGB LED

In this scenario we build an application to control RGB LED color using Intel Galileo Analog output (PWM). RGB LED has 4 pins that you can see it in Figure below.

To understand these pins, you can see the following Figure.



Note:

- Pin 1: Red
- Pin 2: Common pin
- Pin 3: Green
- Pin 4: Blue

Now we can start to build an application and hardware implementation.

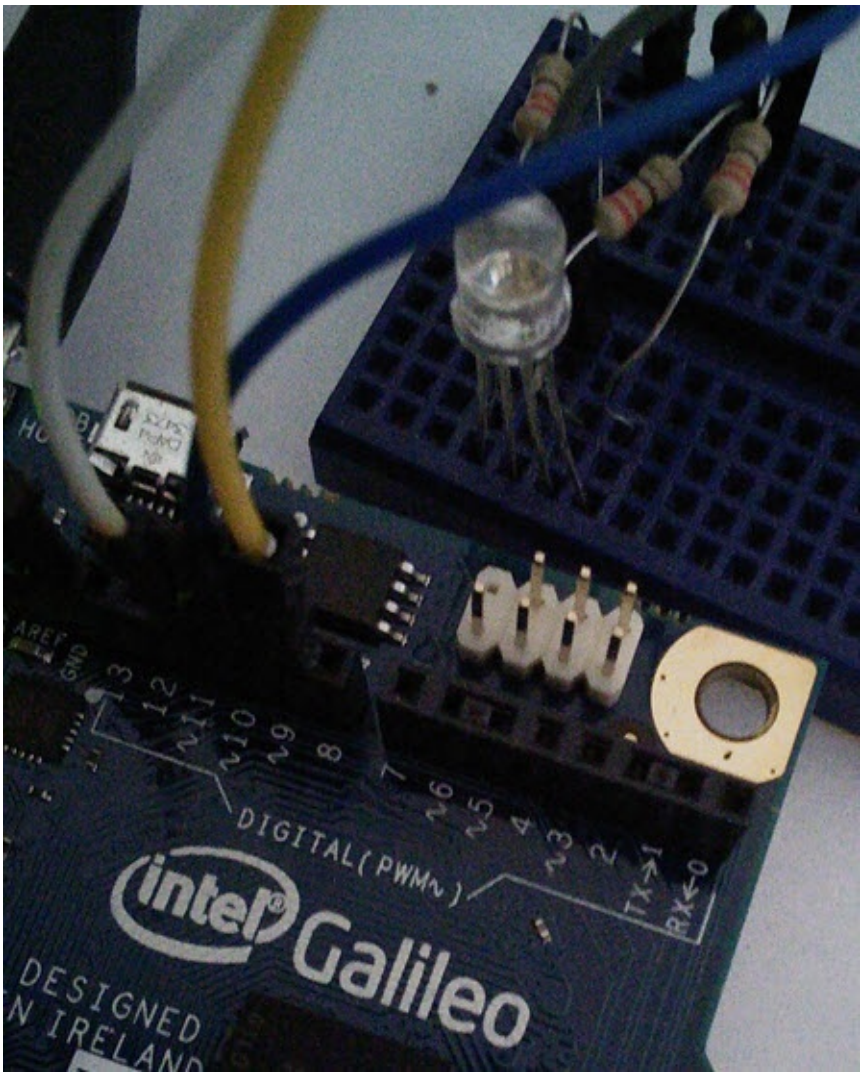
4.3.2 Hardware Configuration

To connect RGB LED pins to Intel Galileo, we can configure the following schema.

- RGB LED pin 1 (red) is connected to Intel Galileo PWM pin 11
- RGB LED pin 2 is connected to Intel Galileo VCC 5V
- RGB LED pin 3 (green) is connected to Intel Galileo PWM pin 10
- RGB LED pin 4 (blue) is connected to Intel Galileo PWM pin 9

Note: you can use resistor to handle voltage issues.

Hardware implementation can be seen in Figure below.



4.3.3 Writing Application

We can generate a color by combining R, G, and B values. This value can be 0 until 255. In this case, we will generate colors: red, green, blue, yellow, and purple.

Create Galileo Wiring on Visual Studio, called RGBDemo. Write this code on **Main.cpp** file.

```
#include "stdafx.h"
#include "arduino.h"

int _tmain(int argc, _TCHAR* argv[])
{
    return RunArduinoSketch();
}

int redPin = 11;
int greenPin = 10;
int bluePin = 9;

void setup()
```

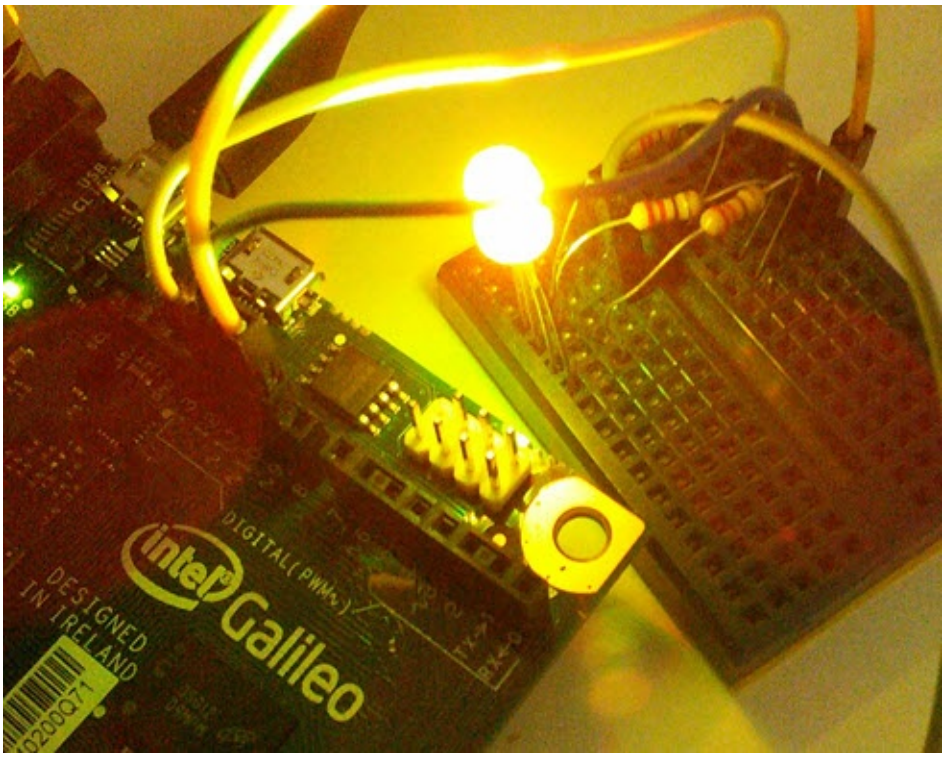
```
{
        pinMode(redPin, OUTPUT);
        pinMode(greenPin, OUTPUT);
        pinMode(bluePin, OUTPUT);
}

void setColor(int red, int green, int blue)
{
        analogWrite(redPin, red);
        analogWrite(greenPin, green);
        analogWrite(bluePin, blue);
}

// the loop routine runs over and over again forever:
void loop()
{
        setColor(255, 0, 0);  // red
        Log(L"red\n");
        delay(1000);
        setColor(0, 255, 0);  // green
        Log(L"green\n");
        delay(1000);
        setColor(0, 0, 255);  // blue
        Log(L"blue\n");
        delay(1000);
        setColor(255, 255, 0);  // yellow
        Log(L"yellow\n");
        delay(1000);
        setColor(80, 0, 80);  // purple
        Log(L"purple\n");
        delay(1000);
        setColor(0, 255, 255);  // aqua
        Log(L"aqua\n");
        delay(1000);
}
```

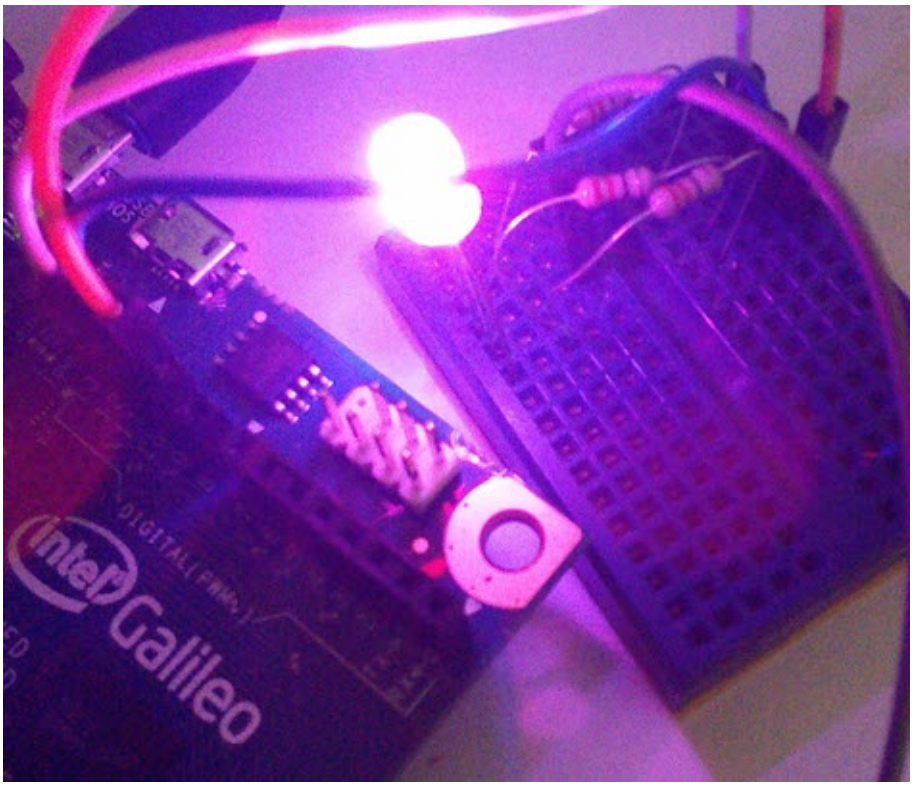
Save this code.

4.3.4 Testing

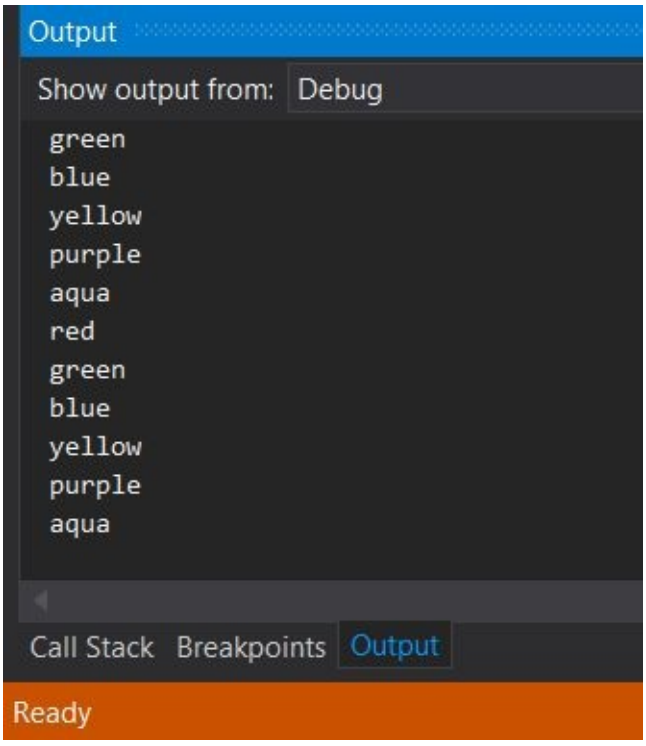
Now you can compile and upload program to Intel Galileo board. Several output for RGB LED can be seen in Figure below.


DIGITAL(PWM~)
intel
Galileo
DESIGNED
IN IRELAND


DIGITAL(PWM~)
intel
Galileo
DESIGNED
IN IRELAND



You also can see the debug output on Visual Studio.



5. Serial Communication

This chapter explains how to work with serial communication (UART) on Intel Galileo board with Windows for IoT.

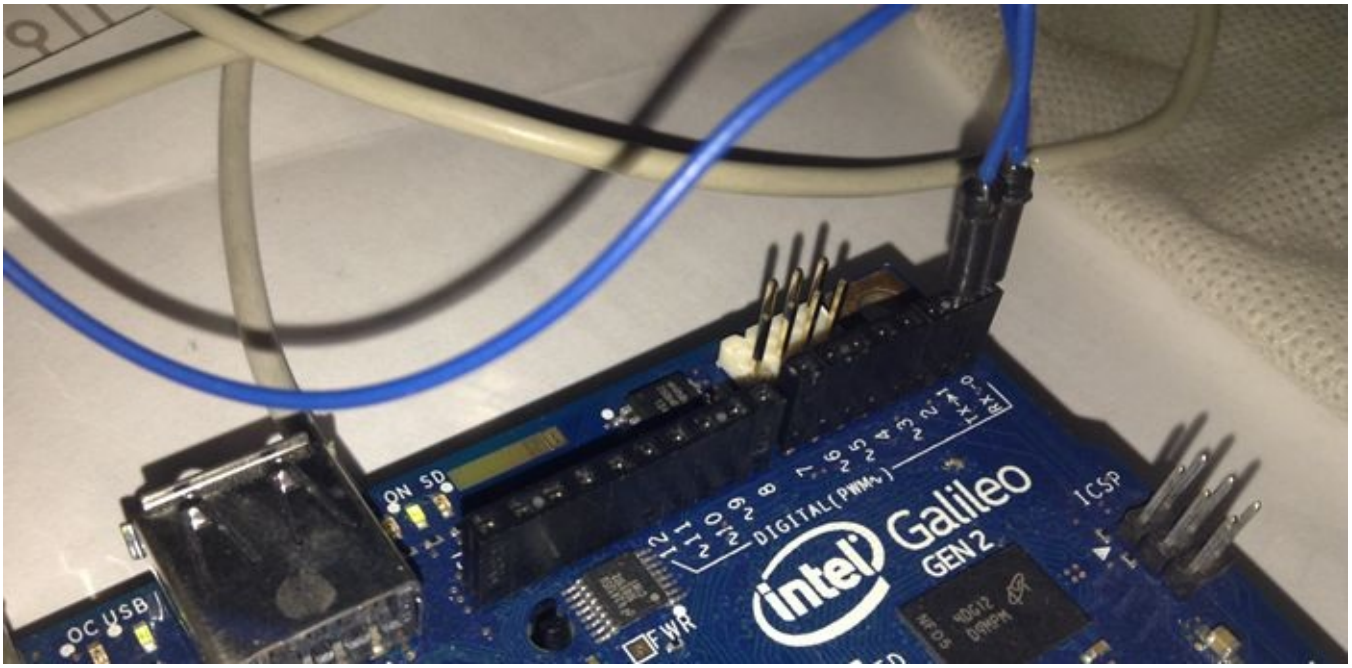
5.1 Getting Started

In this chapter, we explore how to work with UART on Intel Galileo board.

5.2 Building Serial Port Application

In this section we work with serial communication (UART) which we can use Serial object. To illustrate our case, we access Intel Galileo UART using Serial.write() to write and Serial.read() to read.

For illustration, we build the UART loopback. We connect Rx on digital pin 0 to Tx on digital pin 1. The following is our wiring.



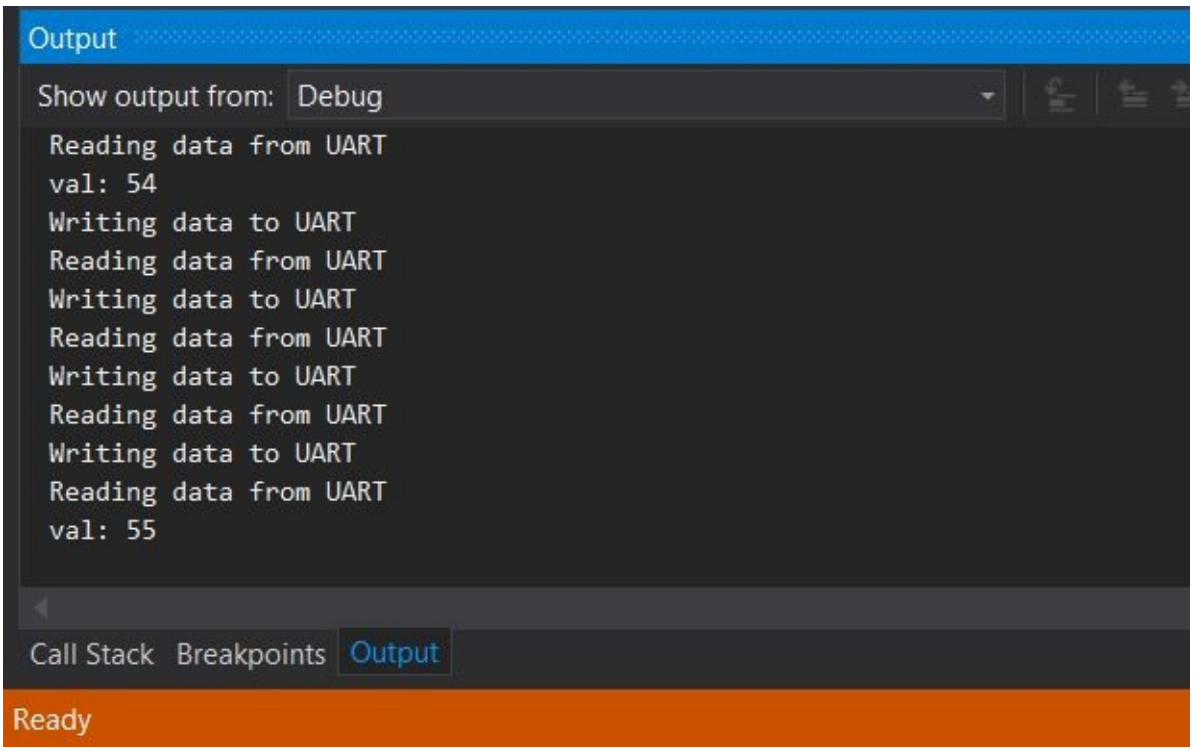
Now we can create Galileo Wiring on Visual Studio 2013, called **SerialDemo**. Then, write this code on **Main.cpp**.

```
#include "stdafx.h"
#include "arduino.h"

int _tmain(int argc, _TCHAR* argv[])
{
    return RunArduinoSketch();
}

int led = 13;  // This is the pin the LED is attached to.
int counter = 50;

void setup()
{
    pinMode(led, OUTPUT);
        Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop()
{
    digitalWrite(led, HIGH);
```

```
        Log(L"Writing data to UART\n");
        Serial.write(counter);

        if (Serial.available())
        {
                digitalWrite(led, LOW);
                Log(L"Reading data from UART\n");
                int val = Serial.read();
                if (val>0)
                        Log(L"val: %d\n", val);

        }

        counter++;
        if (counter > 70)
                counter = 50;

        delay(1000);
}
```

Save this code and then try to upload to the Intel Galileo board. If success, you can see LED on pin 13 is blinking.

You also can verify by opening Debug output on Visual Studio. A sample output can be seen in Figure below.



6. I2C/TWI

This chapter explains how to work and write program using Windows for IoT for accessing Intel Galileo I/O via I2C protocol

6.1 Getting Started

Firstly, we must know I/O mapping on Intel Galileo. We can read this mapping on Intel website, https://communities.intel.com/docs/DOC-21920 . You can see the following of I/O mapping on Intel Galileo.

Galileo I/O Mappings

Arduino IDE ID	GPIO			PWM	Int	Dir	Muxed with	Initial Setup
	Source	Pin	Linux	Linux				
IO0	Cypr	GPORT4_BIT6_PWM2	50	N/A	-	BI	UART0_RXD	I w/ pullup off
IO1	Cypr	GPORT4_BIT7_PWM0	51	N/A	-	BI	UART0_TXD	I w/ pullup off
IO2	SoC (Cypr)	GPIO<6> (GPORT2_BIT0_PWM6_A3)	14 (32*)	-	0	BI	-	I w/ pullup off
IO3	SoC (Cypr)	GPIO<7> (GPORT0_BIT2_PWM3)	15 (18*)	3	1	BI	(PWM)	I w/ pullup off
IO4	Cypr	GPORT1_BIT4_PWM6	28		-	BI	-	I w/ pullup off
IO5	Cypr	GPORT0_BIT1_PWM5	17	5	-	BI	(PWM)	I w/ pullup off
IO6	Cypr	GPORT1_BIT0_PWM6	24	6	-	BI	(PWM)	I w/ pullup off
IO7	Cypr	GPORT1_BIT3_PWM0	27		-	BI	-	I w/ pullup off
IO8	Cypr	GPORT1_BIT2_PWM2	26		-	BI	-	I w/ pullup off
IO9	Cypr	GPORT0_BIT3_PWM1	19	1	-	BI	(PWM)	I w/ pullup off
IO10	Cypr	GPORT0_BIT0_PWM7	16	7	-	BI	(PWM) SPI1_SS_B	I w/ pullup off
IO11	Cypr	GPORT1_BIT1_PWM4	25	4	-	BI	(PWM) SPI1_MOSI	I w/ pullup off
IO12	Cypr	GPORT3_BIT2_PWM3	38		-	BI	SPI1_MISO	I w/ pullup off
IO13	Cypr	GPORT3_BIT3_PWM1	39		-	BI	SPI1_SCK	I w/ pullup off
IO14	Cypr	GPORT4_BIT0_PWM6	44		-	BI	AD7298:VIN0	I w/ pullup off
IO15	Cypr	GPORT4_BIT1_PWM4	45		-	BI	AD7298:VIN1	I w/ pullup off
IO16	Cypr	GPORT4_BIT2_PWM2	46		-	BI	AD7298:VIN2	I w/ pullup off
IO17	Cypr	GPORT4_BIT3_PWM0	47		-	BI	AD7298:VIN3	I w/ pullup off
IO18	Cypr	GPORT4_BIT4_PWM6	48		-	BI	AD7298:VIN4	I w/ pullup off
IO19	Cypr	GPORT4_BIT5_PWM4	49		-	BI	AD7298:VIN5	I w/ pullup off

* See Galileo I/O Function Muxing table below.

Galileo I/O Function Muxing

Mux Selector		Cypress GPIO pin	Linux GPIO ID	Dir	Initial Setup
0	1				
UART0_RXD	IO0	GPORT3_BIT4_PWM7	40	O	unknown
UART0_TXD	IO1	GPORT3_BIT5_PWM5	41	O	unknown
SPI1_SS_B	IO10	GPORT3_BIT6_PWM3	42	O	unknown
SPI1_MOSI	IO11	GPORT3_BIT7_PWM1	43	O	unknown
SPI1_MISO	IO12	GPORT5_BIT2_PWM3	54	O	unknown
SPI1_SCK	IO13	GPORT5_BIT3_PWM1	55	O	unknown
AD7298:VIN0	IO14	GPORT3_BIT1_PWM5	37	O	0
AD7298:VIN1	IO15	GPORT3_BIT0_PWM7	36	O	0
AD7298:VIN2	IO16	GPORT0_BIT7_PWM1	23	O	0
AD7298:VIN3	IO17	GPORT0_BIT6_PWM3	22	O	0
AD7298:VIN4	IO18	GPORT0_BIT5_PWM5	21	O	0
AD7298:VIN5	IO19	GPORT0_BIT4_PWM7	20	O	0
IO2 via SoC GPIO<6>	IO2 via Cypress GPORT2_BIT0_PWM6	GPORT1_BIT7_PWM0	31	O	unknown
IO3 via SoC GPIO<7>	IO3 via Cypress GPORT0_BIT2_PWM3	GPORT1_BIT6_PWM2	30	O	unknown
I2C	(AD7298:VIN4 or IO18) and (AD7298:VIN5 or IO19)	GPORT1_BIT5_PWM4	29	O	1

6.2 I2C/TWI

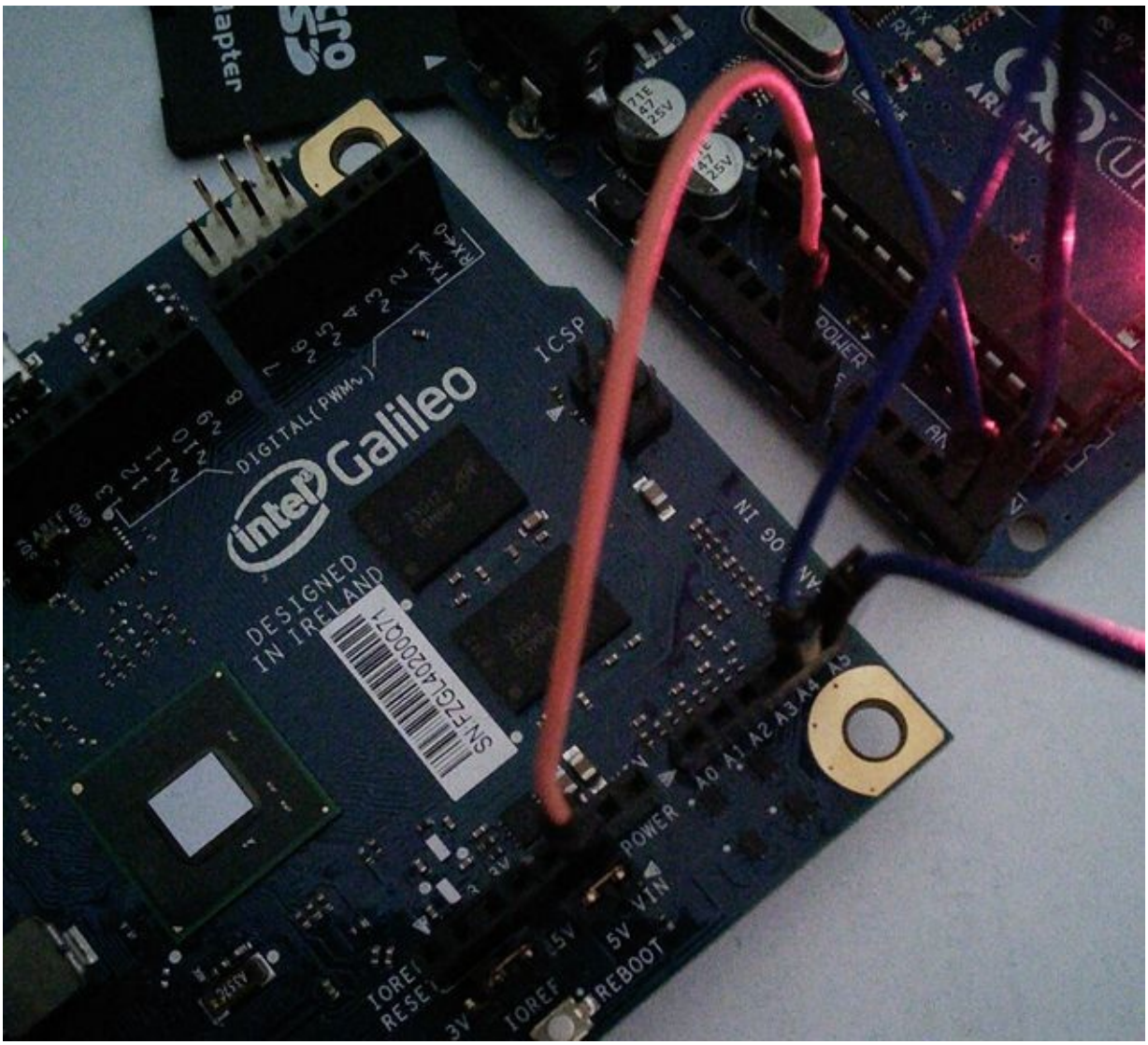
The I2C (Inter-Integrated Circuit) bus was designed by Philips in the early ’80s to allow easy communication between components which reside on the same circuit board. TWI stands for Two Wire Interface and for most marts this bus is identical to I²C. The name TWI was introduced by Atmel and other companies to avoid conflicts with trademark issues related to I²C.

Intel Galileo has I2C interface which are connected via Analog A4 (SDA) and A5 (SCL) pins. For illustration, we will build I2C application. We need an Arduino board as I2C source. I use Arduino Uno.

For wiring scenario, we implement the following hardware schema:

- Arduino A4 (SDA) to Intel Galileo A4 (SDA)
- Arduino A5 (SCL) to Intel Galileo A5 (SCL)
- Arduino GND to Intel Galileo GND

You can see the hardware implementation in Figure below.



The first step is to build a program for Arduino Uno. We define I2C address 0x15. Arduino will send a random number to requester on I2C channel.

We create a program for Arduino, called **arduino_i2c_write**, and write this code.

```
#include <Wire.h>

const byte SLAVE_ADDRESS = 0x15;
int led = 13;
byte x;

// source:
// http://forum.arduino.cc/index.php?topic=197633.0
byte randomDigit() {
  unsigned long t = micros();
  byte r = (t % 10) + 1;
  for (byte i = 1; i <= 4; i++) {
    t /= 10;
    r *= ((t % 10) + 1);
    r %= 11;
  }
  return (r - 1);
}

void sendData(){
  byte data = randomDigit();
  Wire.write(data);
  Serial.print("send: ");
  Serial.println(data,DEC);
}

void setup() {
   pinMode(led, OUTPUT);
   Serial.begin(9600);
   Wire.begin(SLAVE_ADDRESS);
   Wire.onRequest(sendData);
}

void loop() {
  delay(100);
}
```

In this program, we use Wire library, http://arduino.cc/en/reference/wire, to implement I2C on Arduino. Now you compile and deploy it to Arduino board. Then, you can connect your Arduino to Intel Galileo via I2C wiring.

The next step is to write a program for Intel Galileo using Windows for IoT. We will read data on I2C channel with address 0x15. The data will generated by Arduino.

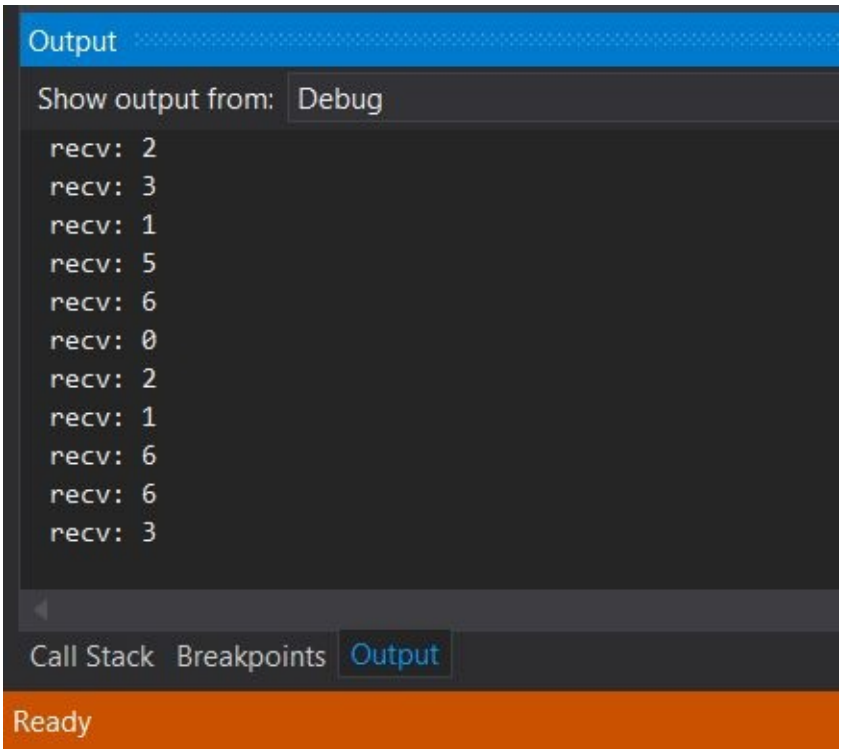
Create a new project of Galileo Wiring from Visual Studio, called **IICDemo**, and write the

following code on **Main.cpp** file.

```
#include "stdafx.h"
#include "arduino.h"
#include <Wire.h>

int _tmain(int argc, _TCHAR* argv[])
{
    return RunArduinoSketch();
}

const byte SLAVE_ADDRESS = 0x15;

void setup()
{
        Wire.begin();
}

// the loop routine runs over and over again forever:
void loop()
{
        // request a byte from slave device 0x15
        Wire.requestFrom(SLAVE_ADDRESS, 1);

        // slave may send less than requested
        while (Wire.available())
        {
                ULONG c = Wire.read(); // receive a byte
                Log(L"recv: %x\n",c);         // print
        }

        delay(1000);

}
```

Save this code and try to run on Intel Galileo board

A sample output from Visual Studio debugging can be seen in Figure below..


Output
Show output from: Debug
recv: 2
recv: 3
recv: 1
recv: 5
recv: 6
recv: 0
recv: 2
recv: 1
recv: 6
recv: 6
recv: 3
Call Stack Breakpoints Output
Ready

7. SPI

This chapter explains how to work and write program for accessing Intel Galileo I/O from Windows for IoT.

7.1 Getting Started

Firstly, we must know I/O mapping on Intel Galileo. We can read this mapping on Intel website, https://communities.intel.com/docs/DOC-21920 . You can see the following of I/O mapping on Intel Galileo.

Galileo I/O Mappings

Arduino IDE ID	GPIO			PWM	Int	Dir	Muxed with	Initial Setup
	Source	Pin	Linux	Linux				
IO0	Cypr	GPORT4_BIT6_PWM2	50	N/A	-	BI	UART0_RXD	I w/ pullup off
IO1	Cypr	GPORT4_BIT7_PWM0	51	N/A	-	BI	UART0_TXD	I w/ pullup off
IO2	SoC (Cypr)	GPIO<6> (GPORT2_BIT0_PWM6_A3)	14 (32*)	-	0	BI	-	I w/ pullup off
IO3	SoC (Cypr)	GPIO<7> (GPORT0_BIT2_PWM3)	15 (18*)	3	1	BI	(PWM)	I w/ pullup off
IO4	Cypr	GPORT1_BIT4_PWM6	28		-	BI	-	I w/ pullup off
IO5	Cypr	GPORT0_BIT1_PWM5	17	5	-	BI	(PWM)	I w/ pullup off
IO6	Cypr	GPORT1_BIT0_PWM6	24	6	-	BI	(PWM)	I w/ pullup off
IO7	Cypr	GPORT1_BIT3_PWM0	27		-	BI	-	I w/ pullup off
IO8	Cypr	GPORT1_BIT2_PWM2	26		-	BI	-	I w/ pullup off
IO9	Cypr	GPORT0_BIT3_PWM1	19	1	-	BI	(PWM)	I w/ pullup off
IO10	Cypr	GPORT0_BIT0_PWM7	16	7	-	BI	(PWM) SPI1_SS_B	I w/ pullup off
IO11	Cypr	GPORT1_BIT1_PWM4	25	4	-	BI	(PWM) SPI1_MOSI	I w/ pullup off
IO12	Cypr	GPORT3_BIT2_PWM3	38		-	BI	SPI1_MISO	I w/ pullup off
IO13	Cypr	GPORT3_BIT3_PWM1	39		-	BI	SPI1_SCK	I w/ pullup off
IO14	Cypr	GPORT4_BIT0_PWM6	44		-	BI	AD7298:VIN0	I w/ pullup off
IO15	Cypr	GPORT4_BIT1_PWM4	45		-	BI	AD7298:VIN1	I w/ pullup off
IO16	Cypr	GPORT4_BIT2_PWM2	46		-	BI	AD7298:VIN2	I w/ pullup off
IO17	Cypr	GPORT4_BIT3_PWM0	47		-	BI	AD7298:VIN3	I w/ pullup off
IO18	Cypr	GPORT4_BIT4_PWM6	48		-	BI	AD7298:VIN4	I w/ pullup off
IO19	Cypr	GPORT4_BIT5_PWM4	49		-	BI	AD7298:VIN5	I w/ pullup off

* See Galileo I/O Function Muxing table below.

Galileo I/O Function Muxing

Mux Selector		Cypress GPIO pin	Linux GPIO ID	Dir	Initial Setup
0	1				
UART0_RXD	IO0	GPORT3_BIT4_PWM7	40	O	unknown
UART0_TXD	IO1	GPORT3_BIT5_PWM5	41	O	unknown
SPI1_SS_B	IO10	GPORT3_BIT6_PWM3	42	O	unknown
SPI1_MOSI	IO11	GPORT3_BIT7_PWM1	43	O	unknown
SPI1_MISO	IO12	GPORT5_BIT2_PWM3	54	O	unknown
SPI1_SCK	IO13	GPORT5_BIT3_PWM1	55	O	unknown
AD7298:VIN0	IO14	GPORT3_BIT1_PWM5	37	O	0
AD7298:VIN1	IO15	GPORT3_BIT0_PWM7	36	O	0
AD7298:VIN2	IO16	GPORT0_BIT7_PWM1	23	O	0
AD7298:VIN3	IO17	GPORT0_BIT6_PWM3	22	O	0
AD7298:VIN4	IO18	GPORT0_BIT5_PWM5	21	O	0
AD7298:VIN5	IO19	GPORT0_BIT4_PWM7	20	O	0
IO2 via SoC GPIO<6>	IO2 via Cypress GPORT2_BIT0_PWM6	GPORT1_BIT7_PWM0	31	O	unknown
IO3 via SoC GPIO<7>	IO3 via Cypress GPORT0_BIT2_PWM3	GPORT1_BIT6_PWM2	30	O	unknown
I2C	(AD7298:VIN4 or IO18) and (AD7298:VIN5 or IO19)	GPORT1_BIT5_PWM4	29	O	1

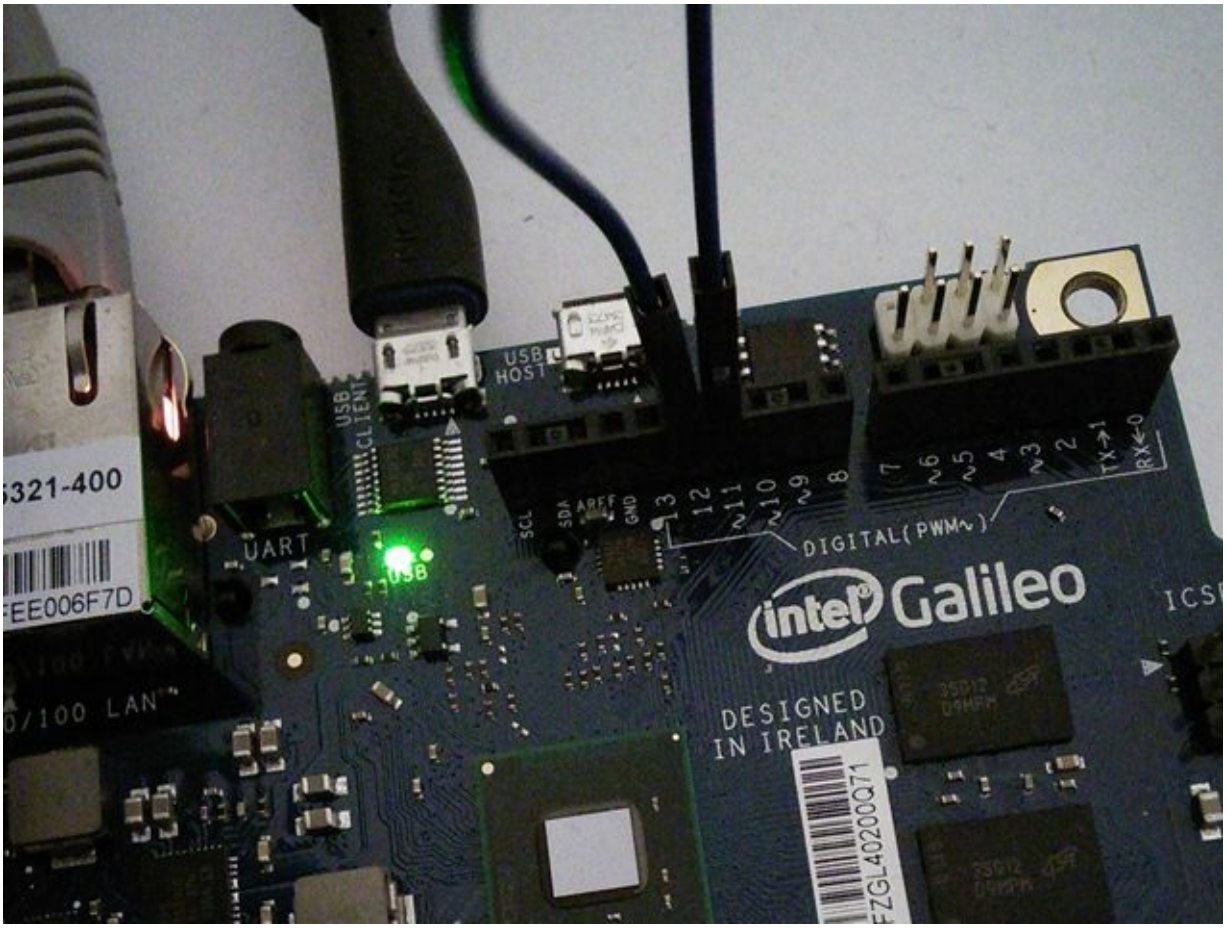
7.2 SPI

The Serial Peripheral Interface (SPI) is a communication bus that is used to interface one or more slave peripheral integrated circuits (ICs) to a single master SPI device; usually a microcontroller or microprocessor of some sort.

By defaults, Intel Galileo has SPI with 4MHz to support Arduino Uno shields. Programmable to 25 MHz. While Galileo has a native SPI controller, it will act as a master and not as an SPI slave. We can use Intel Galileo SPI pins on

- Pin 10 (SS)
- Pin 11 (MOSI)
- Pin 12 (MISO)
- Pin 13 (SCK).

For illustration, we build a program to write and read data to/from Intel Galileo SPI. We build a SPI loopback which digital pin 11 (MOSI) is be connected to digial pin 12 (MISO). Hardware implementation can be seen in Figure below.



We will use mraa library to access SPI. To write and read data to SPI, we can use Spi.Transfer(). Then, we receive incoming data from returning Spi. Transfer().

Create Galileo Wiring from Visual Studio, called **SPIDemo**, and write this code on

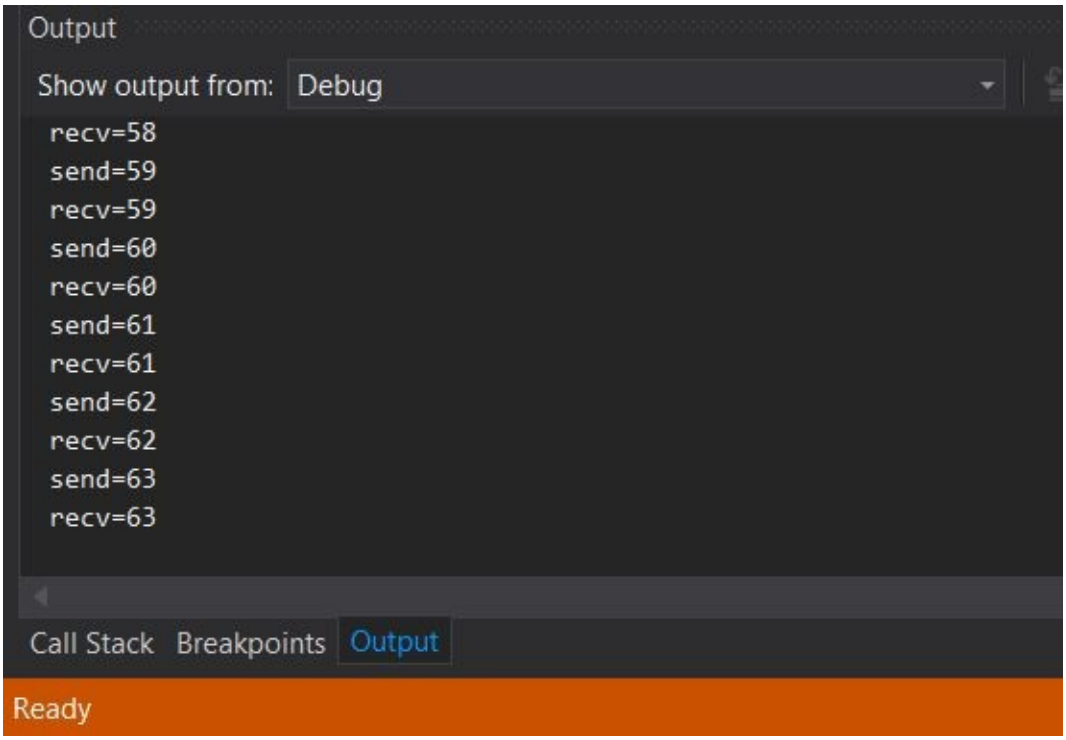
Main.cpp file.

```
#include "stdafx.h"
#include "arduino.h"
#include "SPI.h"

int _tmain(int argc, _TCHAR* argv[])
{
    return RunArduinoSketch();
}

ULONG val = 40;

void setup()
{
        SPI.begin();
}

// the loop routine runs over and over again forever:
void loop()
{
        ULONG ret = SPI.transfer(val);
        Log(L"send=%d\n",val);
        Log(L"recv=%d\n", ret);

        delay(1000);
        val++;
        if (val >= 70)
                val = 40;

}
```

This code will send counter data which is started from 40. If it reaches counter 70+, it will be reset to 40 again.

Now you can compile and debug to Intel Galileo board.

The following is a sample output from Visual Studio debugging.


Output
Show output from: Debug
recv=58
send=59
recv=59
send=60
recv=60
send=61
recv=61
send=62
recv=62
send=63
recv=63
Call Stack
Breakpoints
Output
Ready

Source Code

Source code can be downloaded on
http://www.aguskurniawan.net/book/galileo_win23042015.zip.

Contact

If you have question related to this book, please contact me at aguskur@hotmail.com . My blog: http://blog.aguskurniawan.net.