Nanyang Polytechnic

School of Information Technology

Course : Diploma in Infocomm & Security (ITDF12)

Module : Sensor Technologies and Project (ITP272)

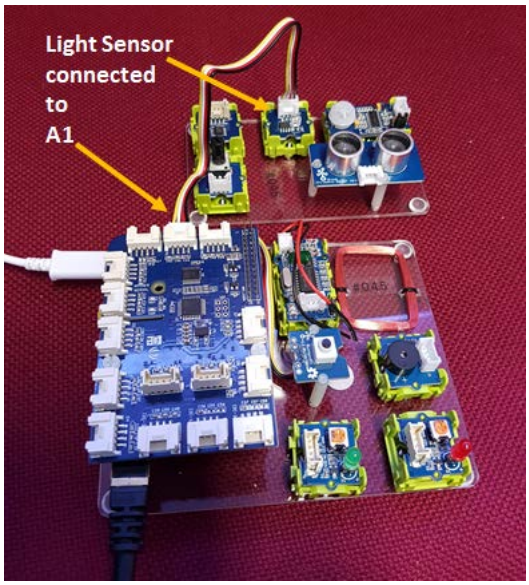
Raspberry Pi Practical : Programming Raspberry Pi with Grove Light Sensor

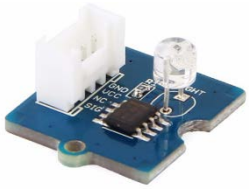
Objectives:

- Learn how to interface with Analog Input port
- Learn how to create program to read in Light sensor input
- Learn how to create program with state machines to work with Light Sensor, LEDs and button

Exercise 1: LightDetect

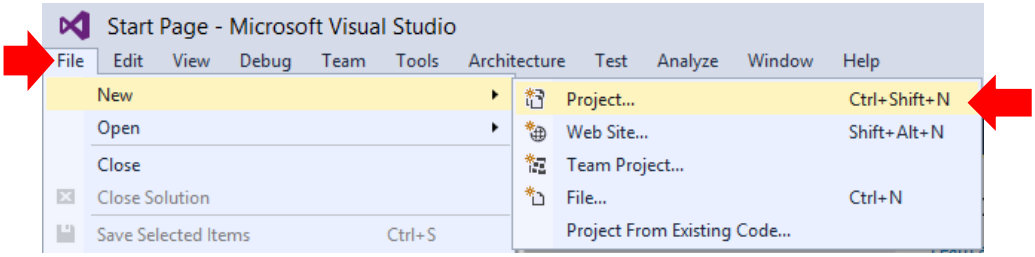
1. Today we will be creating a program that will detect the amount of light in the surrounding.
2. We shall use Digital Input port **A1** of the GrovePi for this sensor. Ensure that you have port **A1** connected to **Light Sensor**



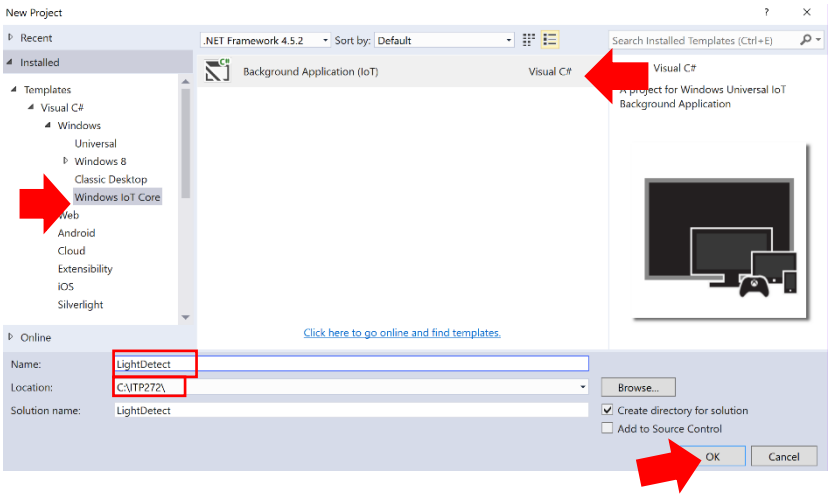


Light Sensor

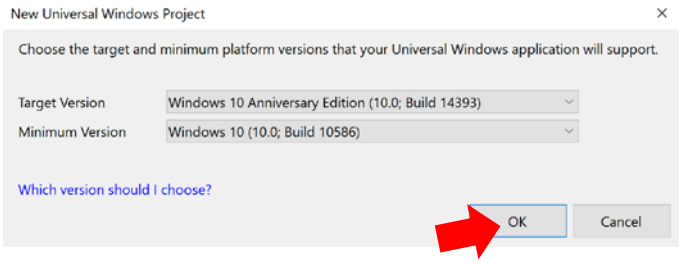
3. Start Visual Studio 2015, Click on File – New - Project.



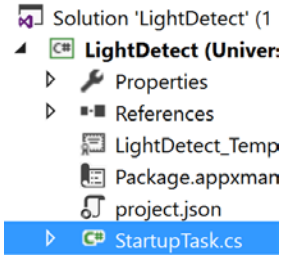
4. From the Installed Templates, select Visual C# – Windows IoT Core – Background Application. Name your project as **LightDetect**, save it in you ITP272 folder and Click OK.



5. You should see this pop-up. Click OK.



6. At the solution explorer on the right side of the screen, check that you have StartupTask.cs. This is the startup program file. Double Click on Startup Task.cs.

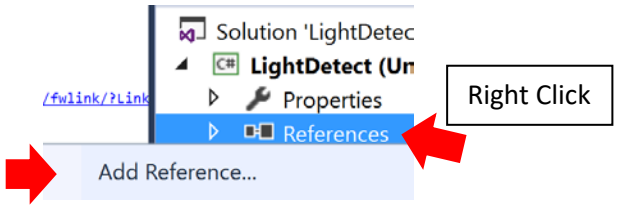


7. You should see the method

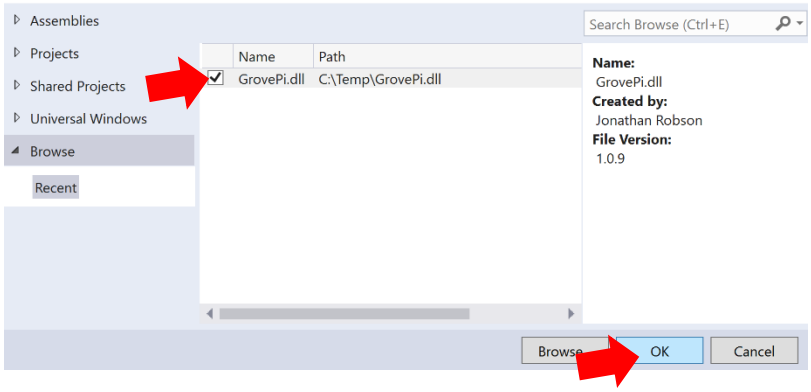
```
public void Run(IBackgroundTaskInstance taskInstance)
```

Your program starts running from this **Run** method.

8. But first, you need to add Reference. Right Click on References on Solution Explorer and Click on Add Reference.



9. Download GrovePi.dll and select it. it should appear in your **Recent** if you have used it previously before. Check on it and Click OK.



10. Go to your "**StartupTask.cs**" and type in the following codes above your namesapce to include required libraries. It is fine to be in gray initially since you have not used them in the codes

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net.Http;
using Windows.ApplicationModel.Background;

// The Background Application template is documented at

using System.Diagnostics;
using System.Threading.Tasks;
using GrovePi;
using GrovePi.Sensors;
```

11. Add in these codes in the StartupTask.cs

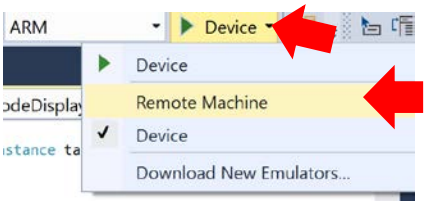
```
public sealed class StartupTask : IBackgroundTask
{
    //Use port A1 for light sensor
    Pin lightPin = Pin.AnalogPin1;

    private void Sleep(int NoOfMs)
    {
        Task.Delay(NoOfMs).Wait();
    }
```

12. Then, in the **Run()** method.

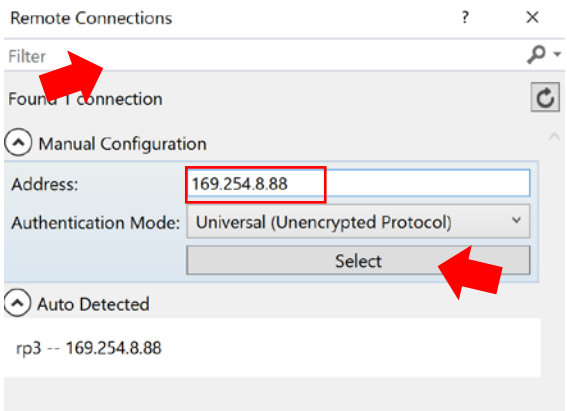
```
public void Run(IBackgroundTaskInstance taskInstance)
{
    //
    // TODO: Insert code to perform background work
    //
    // If you start any asynchronous methods here, prevent the task
    // from closing prematurely by using BackgroundTaskDeferral as
    // described in http://aka.ms/backgroundtaskdeferral
    //

    int adcValue = 0;

    while (true)
    {
        Sleep(300);
        adcValue = DeviceFactory.Build.GrovePi().AnalogRead(lightPin);
        Debug.WriteLine("Light ADC = " + adcValue);
    }
}
```

DeviceFactory.Build.GrovePi().AnalogRead() method is used to read in values from the GrovePi Analog Input. We need pass in the lightPin object so that the Raspberry Pi knows it needs to read in from A1. The value read in is a digital value. A bigger value indicates a higher brightness. As we're not going to display the exact measurements of light, we did not include the formula to convert the digital value into actual stimulus values.

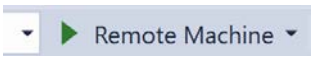
13. In order to deploy the program to your hardware, click on the Drop Down button beside the Device and Select Remote Machine.



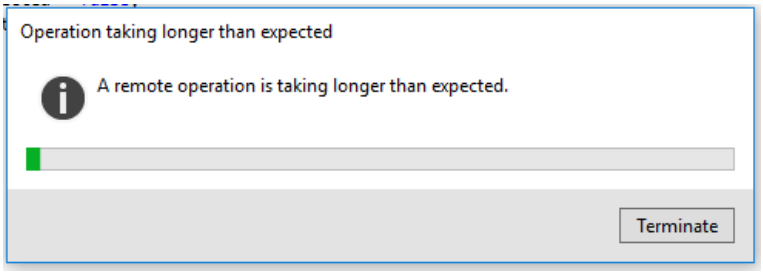
14. Key in the address manually like shown below. Click on Select after that.



15. Click on Remote Machine to Run the Program.



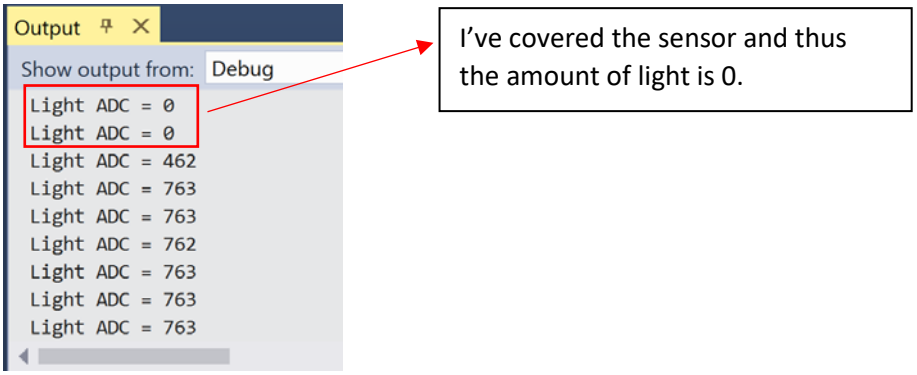
16. You may see the following warning but it is fine. Every time when you deploy a project for the first time on the hardware, it will take a long time to deploy. Just wait for a while and it should be deployed successfully.



17. Once deployed successfully, you can see that in the output, it will show some values.

    A higher value indicates that the surrounding is brighter and a lower value indicates the surround light is dim.

    Try cover the sensor with your hand/finger. You should see the value decreases.



    You can use this value to check for brightness of surrounding to determine how you software will behave.

Exercise 2: LightButtonLedSmartLight

Implement a Smart lighting system can function in 3 modes as below

- Turn on the light
- Detect environment brightness and turns on only when it is dim
- Turn off the light

User can decide to toggle between the 3 modes by a button. So user can choose to always leave the light On or to have the lights off or to let system control the light automatically to only turn on the light when it is dark.

To design a system that meets the above criteria, we can design a state machine to function as required.

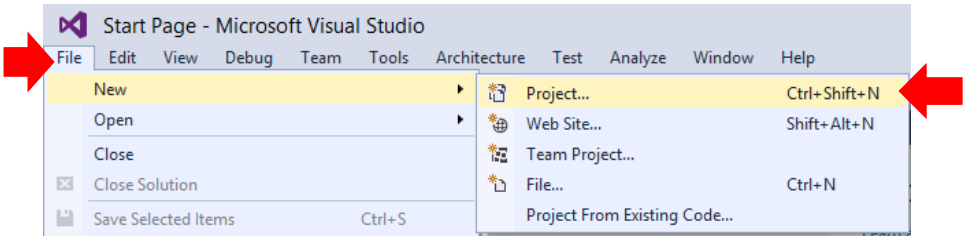
We've going to write the codes such that your program will behave as below. Show your tutor when you're done coding and verifying.

Your program starts in **MODE_ON**

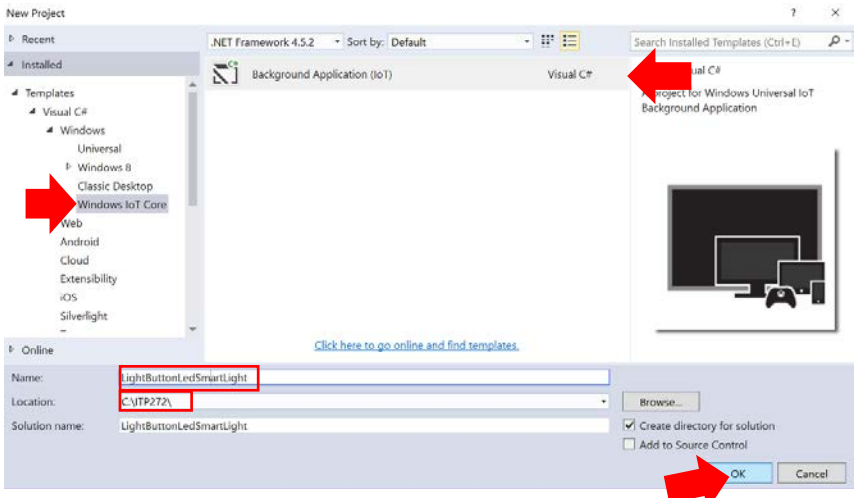
- **MODE_ON**
  - Red LED Stays on regardless of the surrounding light condition
  - If push button is pressed, change mode to **MODE_SMART**
- **MODE_SMART**
  - Red LED turns off under normal surrounding light condition
  - Red LED turns on when surrounding is dark (Simulate by covering the light sensor)
  - If push button is pressed, change mode to **MODE_OFF**
- **MODE_OFF**
  - Red LED light is off
  - If push button is pressed, change mode to **MODE_ON**

You can try to do this on your own to test your own understanding so that you can write you own program for your project. The next pages shows the solution to this and you could use it to verify and see where you go wrong. Check with your tutor when in doubt.

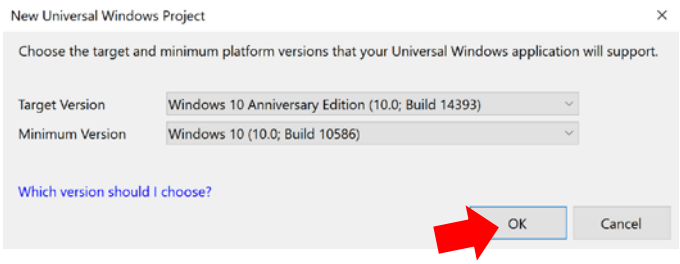
1. In this exercise, we will be creating the program **LightButtonLedSmartLight** as stated in the previous page.
2. Add in Red LED and push button. Ensure that you have the following connections
    - A1 : Light Sensor
    - D5 : Red LED
    - D4 : Push Button
3. Start Visual Studio 2015, Click on File – New - Project.



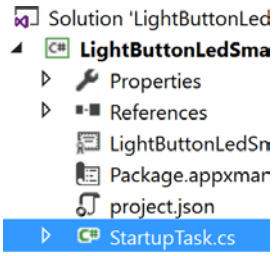
4. From the Installed Templates, select Visual C# – Windows IoT Core – Background Application. Name your project as **LightButtonLedSmartLight**, save it in you ITP272 folder and Click OK.



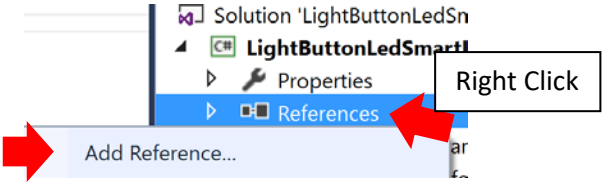
5. You should see this pop-up. Click OK.



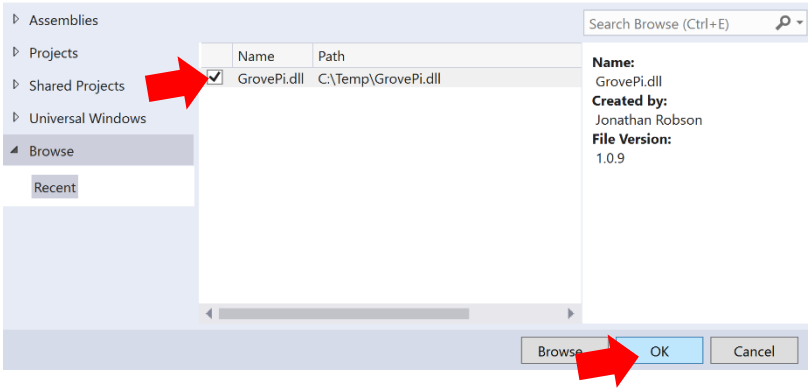
6. At the solution explorer on the right side of the screen, check that you have StartupTask.cs. This is the startup program file. Double Click on Startup Task.cs.



7. Add Reference. Right Click on References on Solution Explorer and Click on Add Reference.



8. Select GrovePi.dll as usual.



9. Go to your "**StartupTask.cs**" and type in the following codes above your namesapce to include required libraries. It is fine to be in gray initially since you have not used them in the codes

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net.Http;
using Windows.ApplicationModel.Background;

// The Background Application template is documented at

using System.Diagnostics;
using System.Threading.Tasks;
using GrovePi;
using GrovePi.Sensors;
```

10. Add in these codes in the **StartupTask.cs**

```
public sealed class StartupTask : IBackgroundTask
{
    //State Machine variables to control different mode of operation
    const int MODE_ON = 1;
    const int MODE_SMART = 2;
    const int MODE_OFF = 3;
    static int curMode;     //stores the current mode the program is at

    //Use port A1 for light sensor, D5 for Red LED and D4 for Push button
    Pin lightPin = Pin.AnalogPin1;
    ILed ledRed = DeviceFactory.Build.Led(Pin.DigitalPin5);
    IButtonSensor button = DeviceFactory.Build.ButtonSensor(Pin.DigitalPin4);

    //This is for main logic controller to know that a button is pressed
    private bool buttonPressed = false;

    private void Sleep(int NoOfMs)
    {
        Task.Delay(NoOfMs).Wait();
    }
```

11. Create a self-monitoring method **startButtonMonitoring()** to monitor the button and update status.

```
//This is created to self monitor button status.
//When button is pressed, buttonPressed will be true.
//The main program can check for this buttonPressed to know whether button has been pressed
1 reference
private async void startButtonMonitoring()
{
    await Task.Delay(100);
    while (true)
    {
        Sleep(100);
        string buttonState = button.CurrentState.ToString();
        if (buttonState.Equals("On"))
        {
            Sleep(100);
            buttonState = button.CurrentState.ToString();
            if (buttonState.Equals("On"))
            {
                buttonPressed = true;
            }
        }
    }
}//End of startButtonMonitoring()
```

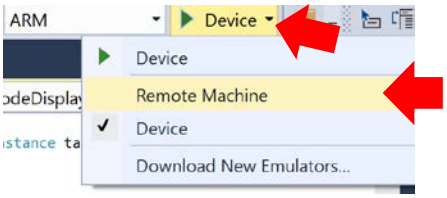
12. Create a handler methods to for the 3 different modes of operation. Add the codes between **startButtonMonitoring()** and **Run()** method

```
}//End of startButtonMonitoring()

//This method handles all logic when curMode is MODE_ON
private void handleModeOn()
{
    //  1.  Define Behaviour in this mode
        ledRed.ChangeState(SensorStatus.On);


    // 2.   Must write the condition to move on to other modes
        if (buttonPressed == true)
        {
            //must always clear back to false
            buttonPressed = false;

            //Move on to Mode Smart when button is pressed
            curMode = MODE_SMART;
            Debug.WriteLine("===Entering  MODE_SMART===");
        }
}

//This method handles all logic when curMode is MODE_SMART
private void handleModeSmart()
{
    //  1.  Define Behaviour in this mode
        //Check environment light. Turn on Red LED only when environment is dim
        int adcValue = DeviceFactory.Build.GrovePi().AnalogRead(lightPin);
        if(adcValue < 50)
            ledRed.ChangeState(SensorStatus.On);
        else
            ledRed.ChangeState(SensorStatus.Off);

    // 2.   Must write the condition to move on to other modes
        if (buttonPressed == true)
        {
            //must always clear back to false
            buttonPressed = false;

            //Move on to Mode Off when button is pressed
            curMode = MODE_OFF;
            Debug.WriteLine("===Entering  MODE_OFF===");
        }
}

//This method handles all logic when curMode is MODE_OFF
private void handleModeOff()
{
    //  1.  Define Behaviour in this mode
        ledRed.ChangeState(SensorStatus.Off);

    // 2.   Must write the condition to move on to other modes
        if (buttonPressed == true)
        {
            //must always clear back to false
            buttonPressed = false;

            //Move on to Mode ON when button is pressed
            curMode = MODE_ON;
            Debug.WriteLine("===Entering  MODE_ON===");
        }
}

0 references
public void Run(IBackgroundTaskInstance taskInstance)
```

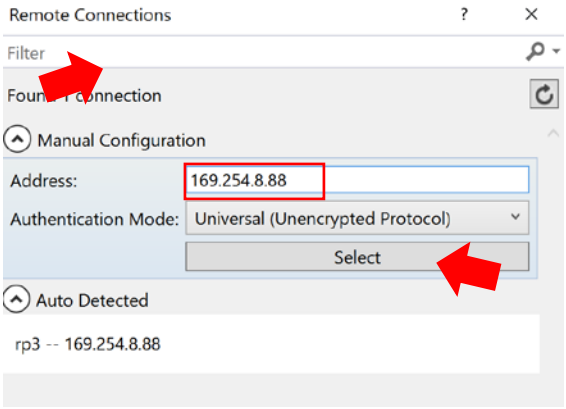
13. Lastly, add the codes in the Run() method to setup the state machine transitions.

```
public void Run(IBackgroundTaskInstance taskInstance)
{
    //
    // TODO: Insert code to perform background work
    //
    // If you start any asynchronous methods here, prevent the task
    // from closing prematurely by using BackgroundTaskDeferral as
    // described in http://aka.ms/backgroundtaskdeferral
    //

    //Start button self monitoring
    startButtonMonitoring();

    //Initialize Mode
    curMode = MODE_ON;
    ledRed.ChangeState(SensorStatus.Off);
    Debug.WriteLine("=== Entering  MODE_ON ===");

    //This makes sure the main program runs indefinitely
    while (true)
    {
        Sleep(300);

        //state machine
        if (curMode == MODE_ON)
            handleModeOn();
        else if (curMode == MODE_SMART)
            handleModeSmart();
        else if (curMode == MODE_OFF)
            handleModeOff();
        else
            Debug.WriteLine("ERROR: Invalid Mode. Please check your logic");
    }
}
```

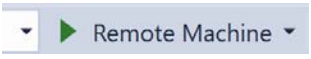
14. Deploy the program to your hardware, click on the Drop Down button beside the Device and Select Remote Machine.



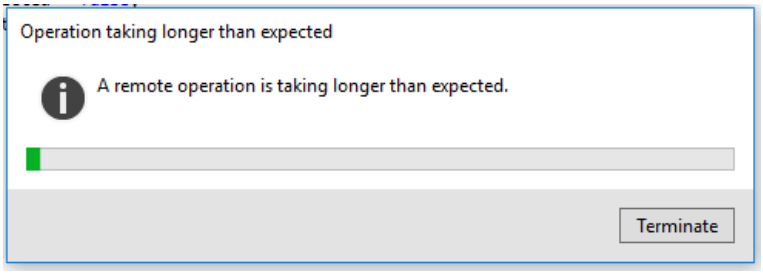
15. Key in the address manually like shown below. Click on Select after that.



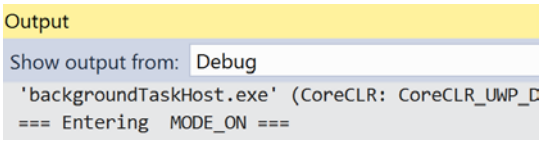
16. Click on Remote Machine to Run the Program.



17. You may see the following warning but it is fine. Every time when you deploy a project for the first time on the hardware, it will take a long time to deploy. Just wait for a while and it should be deployed successfully.



18. After the program is deployed, your program should start in MODE_ON as see from the Output Window.



You should see the Red LED lights up in this mode. Try covering the light sensor to simulate environment going dim. You should notice that the Red LED remains on. In this mode, it doesn't checks on the environment setting.

19. We shall now test to verify out program is working as required

Press the button. The program should switch to MODE_SMART. In this mode you should see that the Red LED is off. In this mode, red LED only turns on when environment is VERY dark. Try cover the light sensor with your hand to simulate a dark environment. You should observe that the Red LED turns on now.

Take a look at the codes to see how this is done. In this mode, the program is checking for the adc value and it only turns on the Red LED when adc is < 50. So under normal environment light, you should verify the adc value to be greater than 50. You can change this value 50 to a suitable value so that you can fine tune how dark it gets before your system will turn on the light.

```
if (adcValue < 50)
    ledRed.ChangeState(SensorStatus.On);
else
    ledRed.ChangeState(SensorStatus.Off);
```

Press the button again. The program should switch to MODE_OFF. In this mode, the Red LED will be off regardless of the environment light. You can cover the light sensor but the Red LED will not light up.

If you pressed the button again now, it will change to MODE_ON again with Red LED turning back on.

Check the code to make sure you understand what is going on so that you know how to design state machine to fulfil your project needs.

==End of Practical==