



School of Information Technology

Course : Diploma in Infocomm & Security (ITDF12)

Module : Sensor Technologies and Project (ITP272)

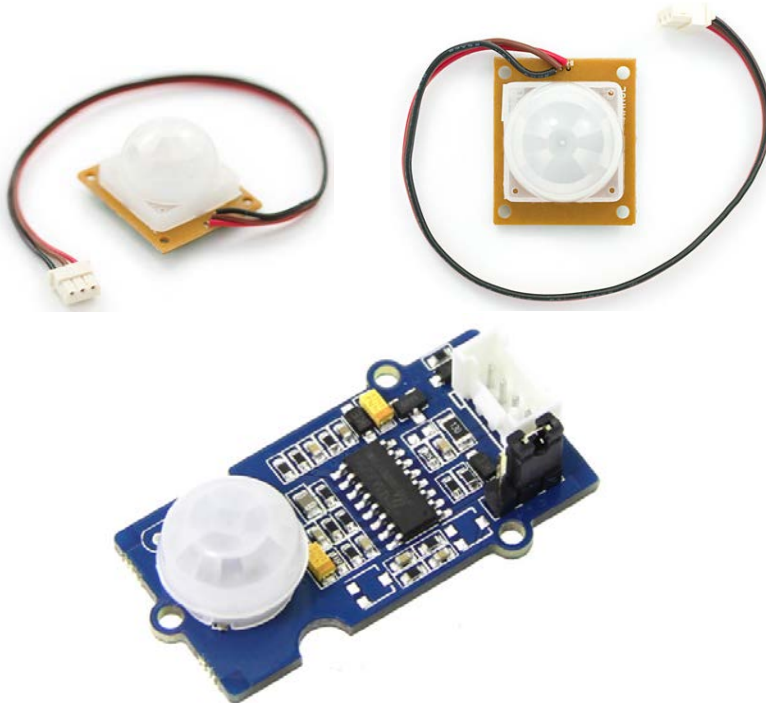
Raspberry Pi Practical : Programming Raspberry Pi with Grove Motion Sensor

Objectives:

- Learn how to interface with Digital I/O port
- Learn how to create program to monitor PIR motion sensor.
- Learn how to create program with state machines to work with Motion Sensor and LEDs

Practical

Introduction to PIR Motion Sensor

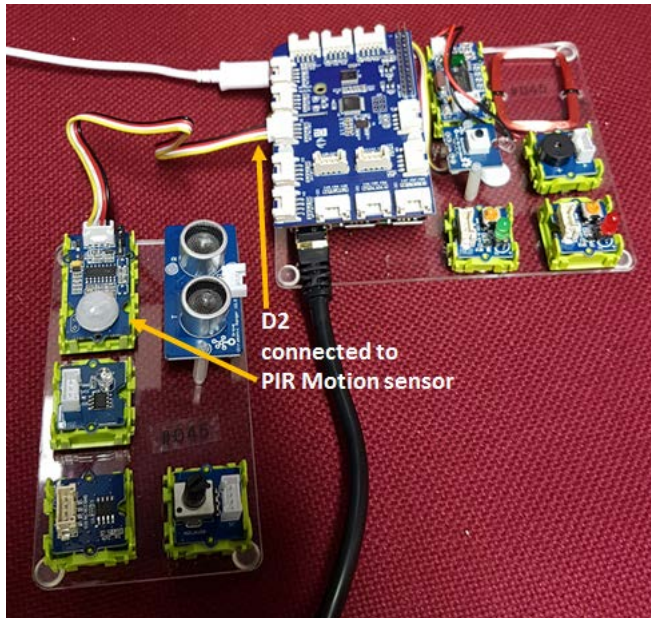


Passive InfraRed (PIR) Sensor is an electronic sensor that measures infrared (IR) light radiating from objects in its field of view. They are often referred to as PIR, "*Passive Infrared*", "*Pyroelectric*" or "*IR motion*" sensors. The term passive in this instance refers to the fact that PIR devices do not generate or radiate any energy for detection purposes. They work entirely by detecting the energy given off by other objects. It has a wide lens range and easy to interface with.

PIR Sensors allow you to sense motion, almost always used to detect whether a human, animals or other objects has moved in or out of the sensors range. Output digital pulse high (3.3V or 5V) when triggered (motion detected), digital low when idle (no motion detected). The sensor in a motion detector is split in two halves. They are wired up so that they can cancel each other out. If one half sees more or less IR radiation than the other, the output will swing high or low. Sensitivity Range can be up to 6 metres depending on the model of the sensor. The Grove sensor is our Lab Kit is has the specification of 3 metres range detection.

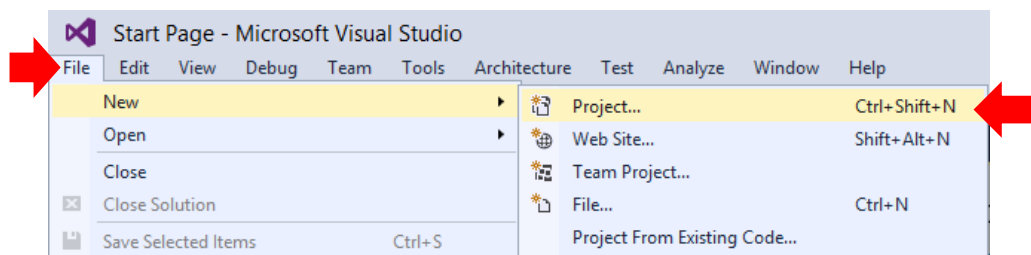
Exercise 1: MotionDetect

1. Today we will be creating a program that uses Motion Sensor to detect movement.
2. We shall use Digital Input port **D2** of the GrovePi for this sensor. Ensure that you have port **D2** connected to **Motion sensor**. See below for clearer illustration.

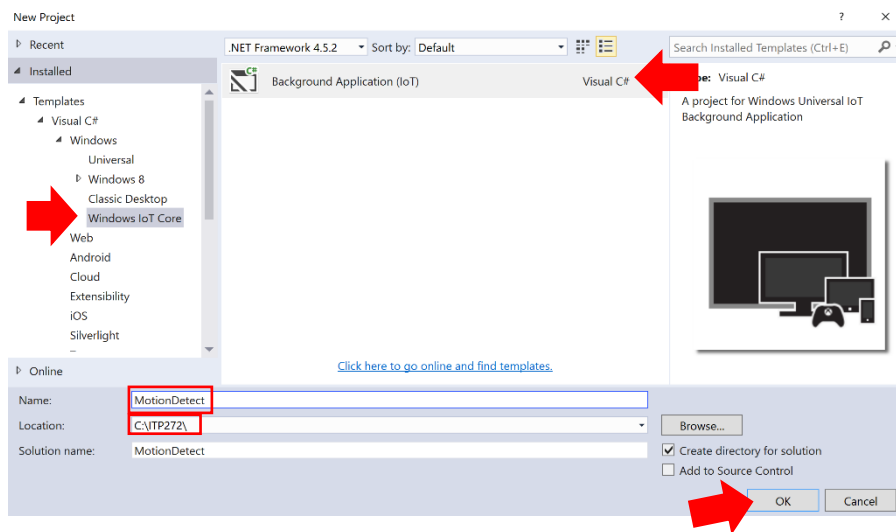


PIR Motion Sensor

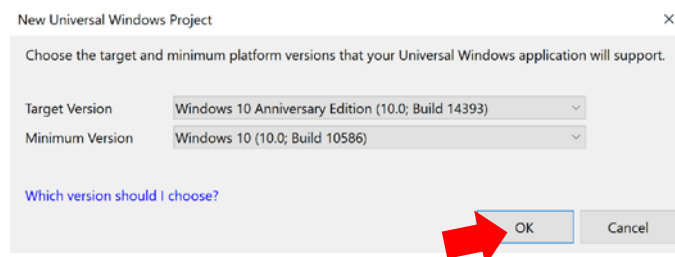
3. Firstly, start Visual Studio 2015, Click on File – New - Project.



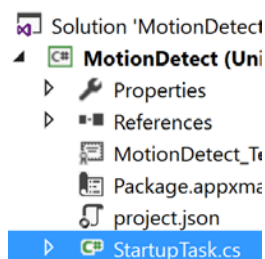
- From the Installed Templates, select Visual C# – Windows IoT Core – Background Application. Name your project as **MotionDetect**, save it in you ITP272 folder and Click OK.



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



- 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.

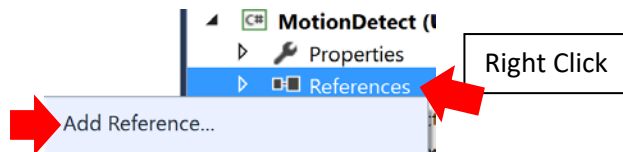


- You should see the method

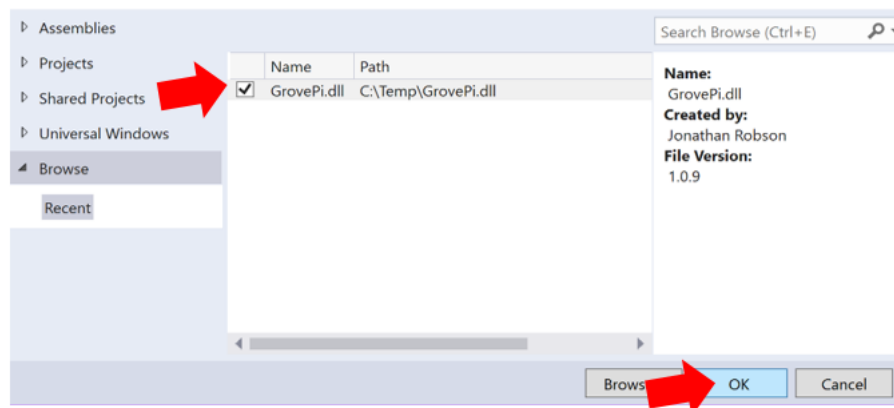
```
public void Run(IBackgroundTaskInstance taskInstance)
```

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

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



- Download the GrovePi.dll from blackboard and follow steps in **Practical 1 Exercise 2** to Browse and select it. If you've previously already downloaded this file and selected it, it should appear in your Recent. Check on it and Click OK.



- Go to your "StartupTask.cs" and type in the following codes above your namespace 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 D2 for Motion
    Pin PirMotionSensorPin = Pin.DigitalPin2;

    //This is for main logic controller to know that motion is detected
    private bool motionDetected = false;

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

12. Create a method **startMotionMonitoring()** to monitor the motion sensor status.
Add the codes above the **Run()** method

```
//This is created to self monitor PIR sensor status.
//When movement is detected, motionDetected will be true.
//The main program can check for this motionDetected to know whether there is movement
private async void startMotionMonitoring()
{
    await Task.Delay(100);
    while (true)
    {
        Sleep(100);
        string motionState = DeviceFactory.Build.GrovePi().DigitalRead(PirMotionSensorPin).ToString();
        if (motionState.Equals("1"))
        {
            motionDetected = true;
            Task.Delay(3000).Wait();
        }
    }
}
```

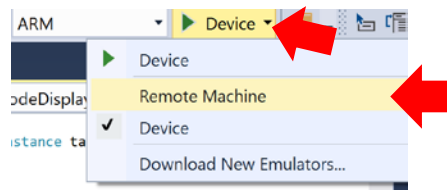
13. Then, in the **Run()** method. Add the codes below

```
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 PIR sensor self monitoring
    startMotionMonitoring();

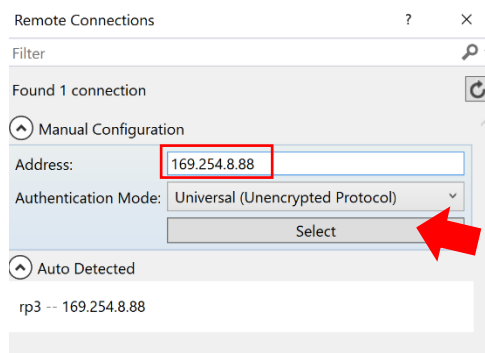
    Debug.WriteLine("Please Wave!");

    //This makes sure the main program runs indefinitely
    while (true)
    {
        Sleep(300);
        if (motionDetected == true)
        {
            //must always clear back to false
            motionDetected = false;
            Debug.WriteLine("Found Movement!");
        }
    }
}
```

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



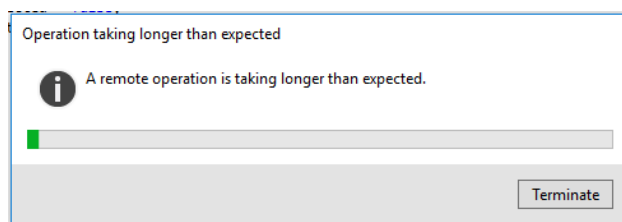
15. If it is auto detected, double click on it. If not, key in the address manually like shown below. Click on Select after that.



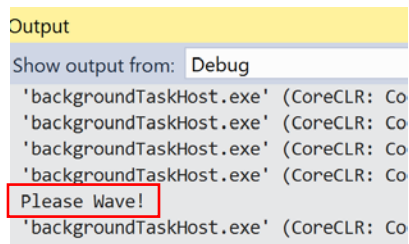
16. Click on Remote Machine to Run the Program.



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.



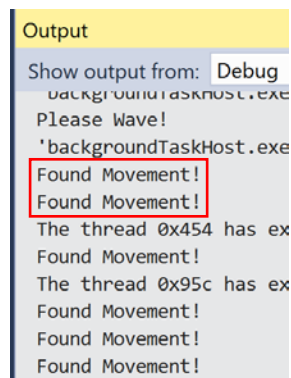
18. Once deployed successfully, you will see that in your Output, it will show “Please Wave!” like shown below.



```
Output
Show output from: Debug
'backgroundTaskHost.exe' (CoreCLR: Co
'backgroundTaskHost.exe' (CoreCLR: Co
'backgroundTaskHost.exe' (CoreCLR: Co
'backgroundTaskHost.exe' (CoreCLR: Co
Please Wave!
'backgroundTaskHost.exe' (CoreCLR: Co
```

The sensor is now monitoring for movement and updating the program. As long there is movement near the motion sensor, the program will detect it and display “Found Movement” at the Output window.

Now wave at the motion sensor to create a movement. You should see “Found Movement!” shown in your Output.



```
Output
Show output from: Debug
'backgroundTaskHost.exe'
Please Wave!
'backgroundTaskHost.exe'
Found Movement!
Found Movement!
The thread 0x454 has ex
Found Movement!
The thread 0x95c has ex
Found Movement!
Found Movement!
Found Movement!
```

You’ve learnt in this exercise how to read in status from the motion sensor and use the variable **motionDetected** to check for movement detected. Please understand the codes and clarify with tutor if required before moving on to the next exercise.

Exercise 2: MotionLedDisco

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

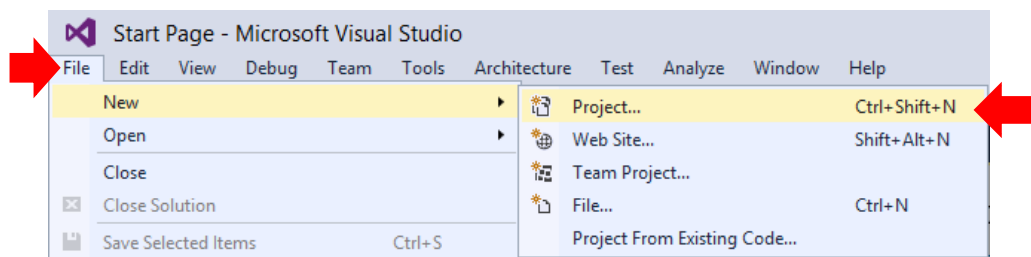
Your program starts in **MODE_NORMAL**

- **MODE_NORMAL**
 - Green LED Stays on
 - Red LED is off
 - If PIR sensor detects motion, change mode to **MODE_DISCO**
- **MODE_DISCO**
 - Red LED blink followed by Green LED blink and repeat this
 - If PIR sensor detects motion, change mode to **MODE_CLOSE**
- **MODE_CLOSE**
 - Turn off both Red and Green LED
 - If PIR sensor detects motion, change mode to **MODE_NORMAL**

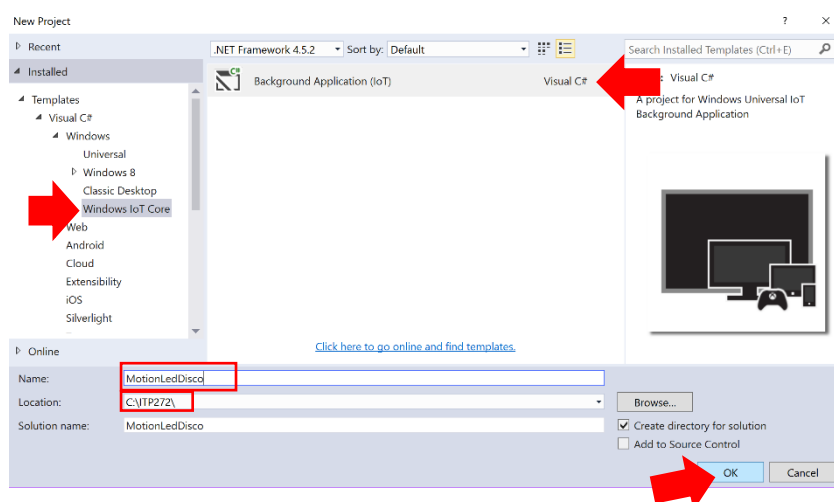
1. Add in Red LED and Green LED. Ensure that you have the following connections

- D2 : Motion Sensor
- D5 : Red LED
- D6 : Green LED

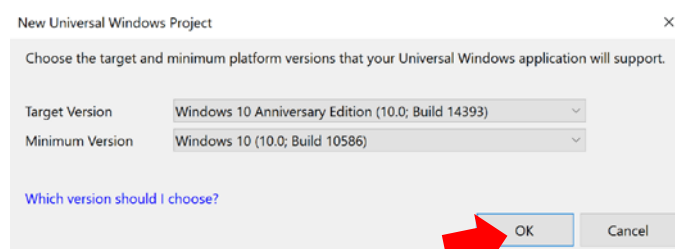
2. Start Visual Studio 2015, Click on File – New - Project.



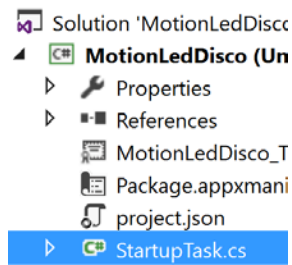
3. From the Installed Templates, select Visual C# – Windows IoT Core – Background Application. Name your project as **MotionLedDisco**, save it in your ITP272 folder and Click OK.



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



- 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.

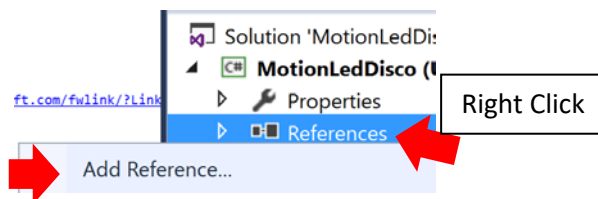


- You should see the method

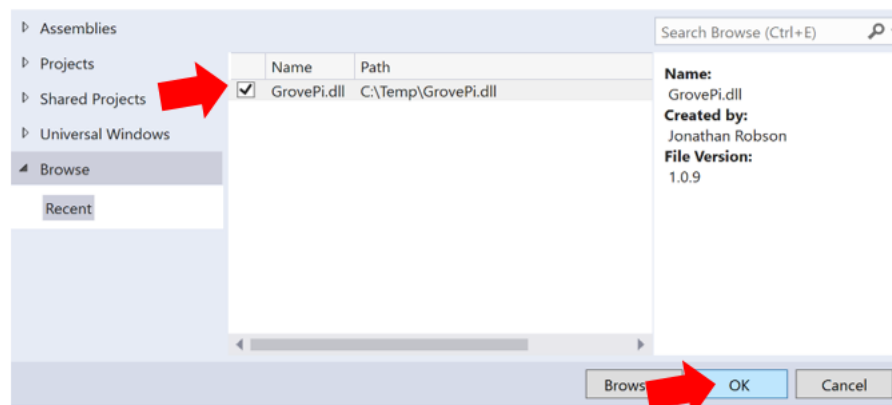
```
public void Run(IBackgroundTaskInstance taskInstance)
```

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

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



- Download the GrovePi.dll from blackboard and follow steps in **Practical 1 Exercise 2** to Browse and select it. If you've previously already downloaded this file and selected it, it should appear in your Recent. Check on it and Click OK.



9. Go to your "StartupTask.cs" and type in the following codes above your namespace 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 modes of operation
    const int MODE_NORMAL = 1;
    const int MODE_DISCO = 2;
    const int MODE_CLOSE = 3;
    static int curMode;

    //Use port D2 for Motion, D5 for Red LED, D6 for Green LED
    Pin PirMotionSensorPin = Pin.DigitalPin2;
    ILed ledRed = DeviceFactory.Build.Led(Pin.DigitalPin5);
    ILed ledGreen = DeviceFactory.Build.Led(Pin.DigitalPin6);

    //This is for main logic controller to know that motion is detected
    private bool motionDetected = false;

    //So that the program has some time to pause before next execution
    //(you should have remembered this by now)
    private void Sleep(int NoOfMs)
    {
        Task.Delay(NoOfMs).Wait();
    }
}
```

11. Create a self-monitoring method for the motion sensor status. Add the codes below the **Sleep()** method

```
//This is created to self monitor PIR sensor status.
//When movement is detected, motionDetected will be true.
//The main program can check for this motionDetected to know whether there is movement
private async void startMotionMonitoring()
{
    await Task.Delay(100);
    while (true)
    {
        Sleep(100);
        string motionState = DeviceFactory.Build.GrovePi().DigitalRead(PirMotionSensorPin).ToString();
        if (motionState.Equals("1"))
        {
            motionDetected = true;
            Task.Delay(3000).Wait();
        }
    }
}
```

12. Create a handler methods to handle the 3 different modes.

```

} //End of startMotionMonitoring()

private void handleModeNormal()
{
    // 1. Define Behaviour in this mode
    // Only Green LED is on during normal mode
    ledGreen.ChangeState(SensorStatus.On);
    ledRed.ChangeState(SensorStatus.Off);

    // 2. Must write the condition to move on to other modes
    // motion detected to move to Mode DISCO
    if (motionDetected == true)
    {
        motionDetected = false;

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

private void handleModeDisco()
{
    // 1. Define Behaviour in this mode
    // Toggle between Red and Green LED
    ledGreen.ChangeState(SensorStatus.Off);
    ledRed.ChangeState(SensorStatus.On);
    Sleep(1000);
    ledGreen.ChangeState(SensorStatus.On);
    ledRed.ChangeState(SensorStatus.Off);
    Sleep(1000);

    // 2. Must write the condition to move on to other modes
    // motion detected to move to Mode CLOSE
    if (motionDetected == true)
    {
        motionDetected = false;

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

private void handleModeClose()
{
    // 1. Define Behaviour in this mode
    // All LED Off
    ledGreen.ChangeState(SensorStatus.Off);
    ledRed.ChangeState(SensorStatus.Off);

    // 2. Must write the condition to move on to other modes
    // motion detected to move to Mode NORMAL
    if (motionDetected == true)
    {
        motionDetected = false;

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

public void Run(IBackgroundTaskInstance taskInstance)
{

```

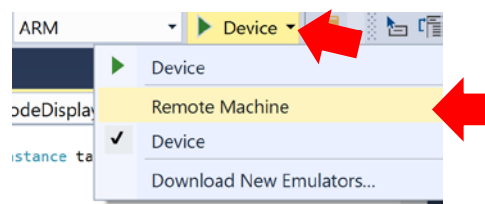
13. Lastly, Add the following codes in the **Run()** method to setup the state machine transition.

```
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
    //
    startMotionMonitoring();

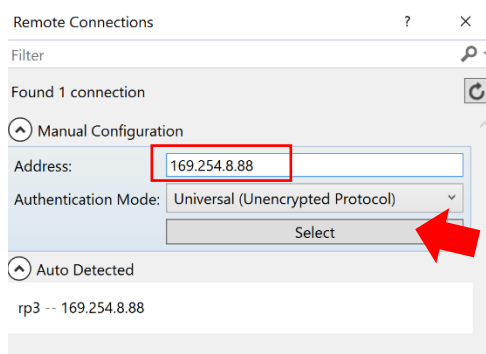
    //Init Mode
    curMode = MODE_NORMAL;
    ledRed.ChangeState(SensorStatus.Off);
    ledGreen.ChangeState(SensorStatus.Off);
    Debug.WriteLine("===Entering MODE_NORMAL===");

    while (true)
    {
        Sleep(300);
        if (curMode == MODE_NORMAL)
            handleModeNormal();
        else if (curMode == MODE_DISCO)
            handleModeDisco();
        else if (curMode == MODE_CLOSE)
            handleModeClose();
        else
            Debug.WriteLine("ERROR: Invalid Mode. Please check your logic");
    }
}
```

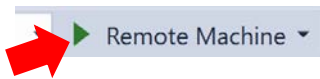
14. In order to deploy the program to your hardware, click on the Drop Down button beside the Device and Select Remote Machine.



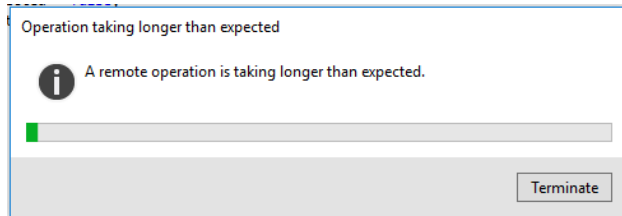
15. If it is auto detected, double click on it. If not, key in the address manually like shown below. Click on Select after that.



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.



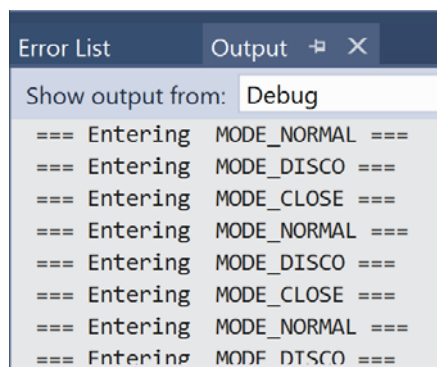
18. Once deployed successfully the device will enter into **MODE_NORMAL**, you will see that the Green LED is On while Red LED is off.

Wave your hand at the sensor to create movement. The motion sensor detects the movement and it change mode to **MODE_DISCO**. In this mode, you should see the Red LED blink followed by the Green LED and repeats alternately as long as there is no movement detected.

Wave your hand at the sensor again to create movement. The motion sensor detects the movement and it change mode to **MODE_CLOSE**. In this mode, you should see both Red and Green LED are off. Both LEDs remain off as long as there is no movement detected.

If you wave your hand again, the program will change mode to **MODE_NORMAL** again.

Open your Output window. You can see the Mode that the device is in from the debug messages.



Look through the code and understand how the codes are creating this behaviour.

==End of Practical==