



School of Information Technology

Course	:	Diploma in Infocomm & Security (ITDF12)
Module	:	Sensor Technologies and Project (ITP272)

Raspberry Pi Practical : Programming Raspberry Pi with Grove LED and Grove button

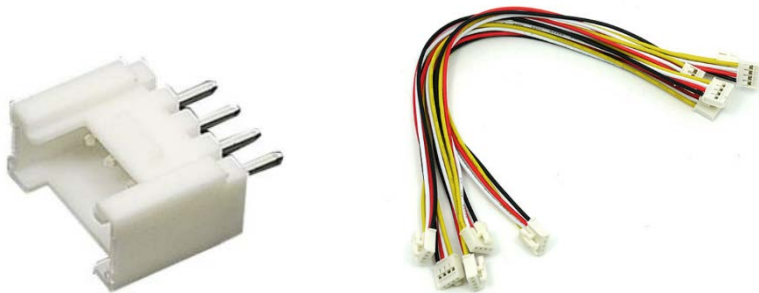
Objectives:

- Understand more on GrovePi Communication interfaces especially Digital I/O and Analog Input
- Learn how to interface with Digital I/O port
- Learn how to create program to control LED.
- Learn how to create program to read push button status.
- Learn how to create program with state machines to design multi-mode display.

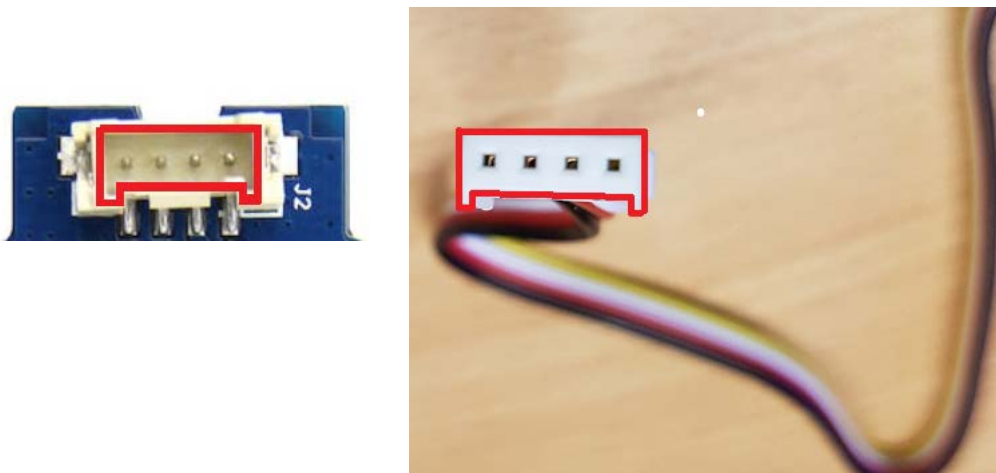
In this Lab practical, we're going to develop program that is going to interface with a Grove Light Emitting Diode (LED) and a Grove push button.

Grove

The sensors and electronic components that we're using in our Lab Kitset are all using a Grove connectors. These 4 pin connectors and cable provide connectivity for the power, ground (GND) and sensor signals. These connector provide ease of connection between the Raspberry Pi and the sensors.



The Grove connector and cable is designed to ensure cable is plug into connector in the correct orientation. This safeguard against wrong connection to prevent accidental shorting of signals. Any sensors or electronic component using this type of connectors/cables will have a Grove prefix to their description.



Grove LED

LED is an acronym which stands for Light Emitting Diode. The picture below shows a LED that uses the Grove connector and thus it's name Grove LED.



An LED has 2 terminals namely Anode (+) and Cathode (-). If you connect a positive voltage to it's Anode (+) and GND to it's Cathode (-), the LED will light up. The electrical symbol of the LED is shown below.



The brightness of the LED depends on the current that is flowing through it. The LED will not light up if you connect the power in reverse polarity (positive to Cathode (-) and negative to Anode (+)) and you may damage the LED. Thus check that your LED is connected correctly as follows

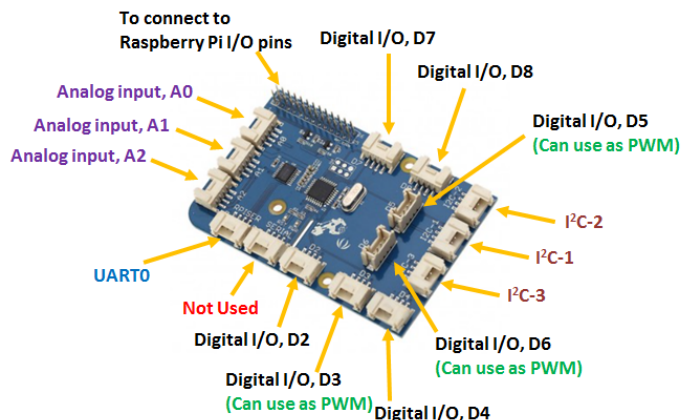


Grove Button

The Grove button is a push button. The push button has 2 states. The “On” state and “Off” state. When you press on the button, it is at the “On” state and when you release the button, it returns back to the “Off” state.



Grove Pi+ Input Output interfaces



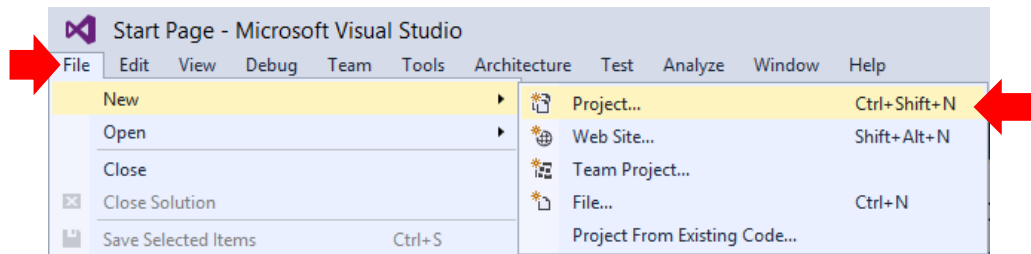
The Grove Pi+ is used to connect to various Grove sensor and electronic devices. The Grove Pi+ provide various different types of communication interfaces with the sensors and devices as follows

- Digital Input / Output (I/O)
- Analog Input
- Pulse Width Modulation (PWM)
- Inter-Integrated Circuit (I2C)
- Universal Asynchronous Receiver Transmitter (UART)

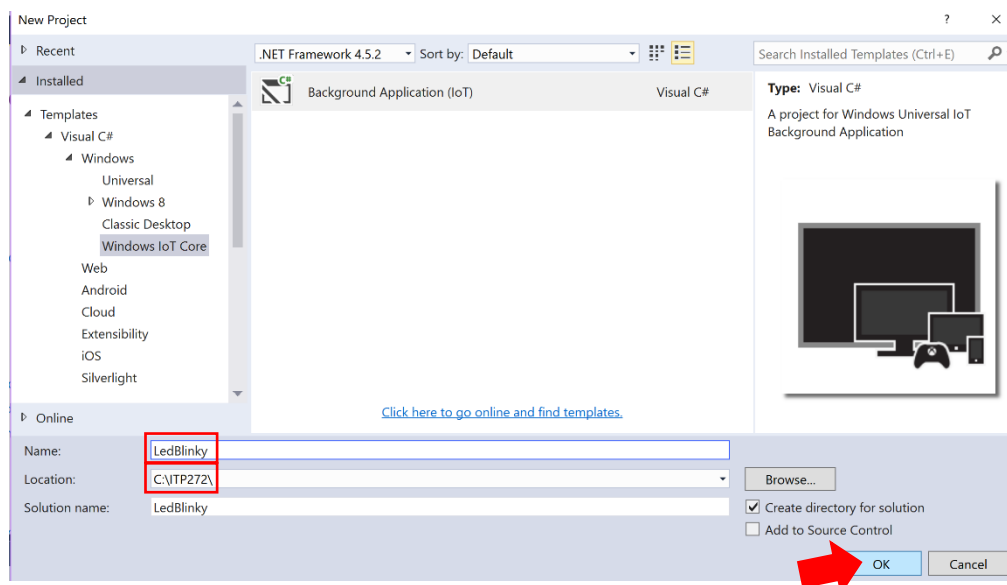
We'll talk more about the differences of these interfaces in detail in later chapter but we shall now first take a look at the more commonly used interface. The most common interface used are the Digital I/O and Analog Input. Digital I/O are used for sensor/devices that operates in only 2 states like LED which is either On state or Off state and Push button which is either Pressed or Not Pressed. Analog Input are used for sensors like temperature sensor whereby the readings coming in contain a range of possible values (more than 2 states). In this Lab Practical, we shall look at how to program the Raspberry Pi to interface with the LED and Push Button using the Digital I/O through the Grove Pi+.

Exercise 1: Create ledBlinky Program on Raspberry Pi

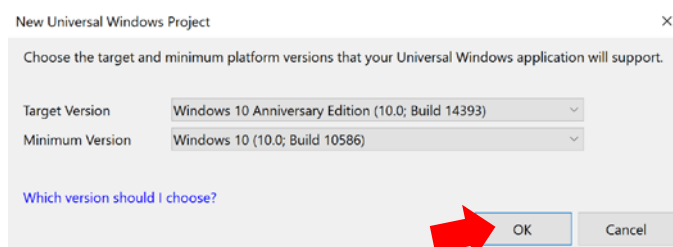
1. Today we will be creating a program that will cause the LED to blink continuously.
2. Please refer to **Practical 1 Exercise 1** to verify Raspberry Pi has stable connection before you proceed.
3. Start Visual Studio 2015, Click on File – New - Project.



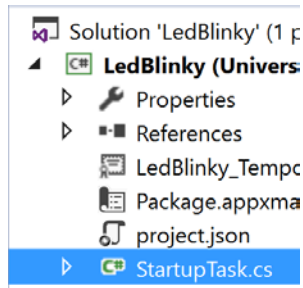
4. From the Installed Templates, select Visual C# – Windows IoT Core – Background Application. Name your project as LedBlinky, save it in your ITP272 folder and Click OK.



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.

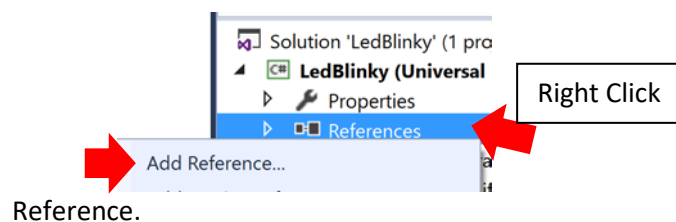


7. You should see the method

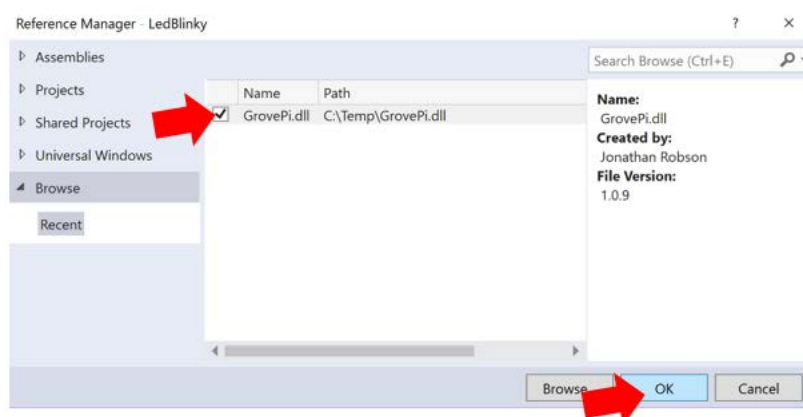
```
public void Run(IBackgroundTaskInstance taskInstance)
```

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



9. 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.



10. 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 these codes to the “StartupTask.cs” file.

```
public sealed class StartupTask : IBackgroundTask
{
    // Use port D5 for Red LED
    ILed ledRed = DeviceFactory.Build.Led(Pin.DigitalPin5);

    //This is created to let the program wait for the specified number in millisecond
    private void Sleep(int NoOfMs)
    {
        Task.Delay(NoOfMs).Wait();
    }
}
```

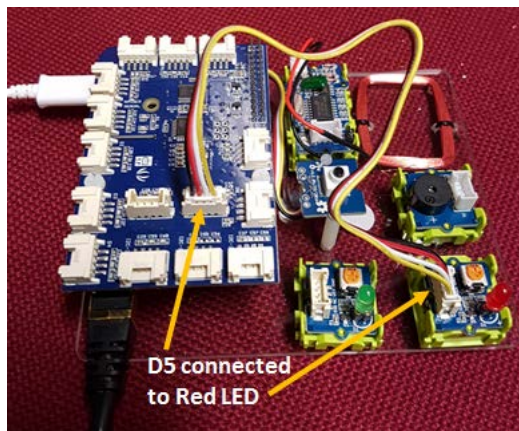
`ILed` is an object which is used to control the LED. You need the `DeviceFactory.Build.led()` to indicate which pin of the GrovePi is the LED connected to. In our case, it is D5, so we indicate it as `Pin.DigitalPin5`

12. Next, add in the while loop condition in the “Run” method

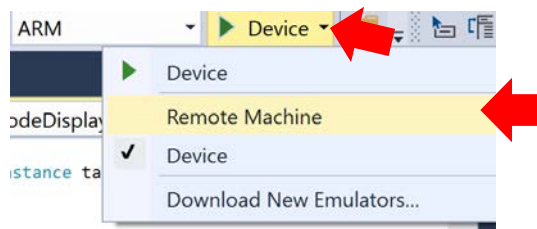
```
public void Run(IBackgroundTaskInstance taskInstance)
{
    //
    // TODO: Insert code to perform background work
    //
    // If you start any asynchronous methods here, prevent the task
    // from closing prematurely by using BackgroundTaskDeferral as
    // described in http://aka.ms/backgroundtaskdeferral
    //
    //The keyword 'while' makes sure that the program runs indefinitely
    while (true)
    {
        Sleep(300);
        if(ledRed.CurrentState == SensorStatus.Off)
        {
            ledRed.ChangeState(SensorStatus.On);
            Debug.WriteLine("Turning ON LED");
        }
        else
        {
            ledRed.ChangeState(SensorStatus.Off);
            Debug.WriteLine("Turning OFF LED");
        }
    }
}
```

In the code above, the `.CurrentState` is used to check whether LED is On or Off state. The `.ChangeState ()` method is used turn the LED to On or Off state.

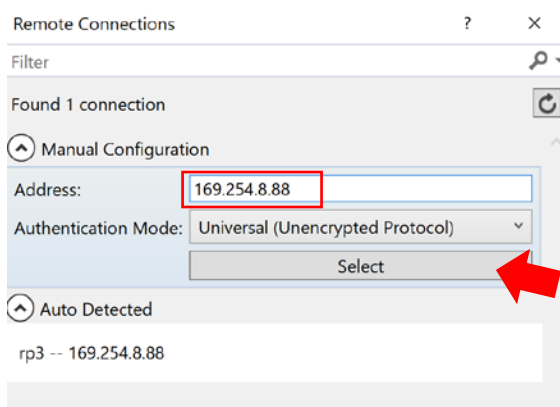
13. We shall use port **D5** of the GrovePi for this sensor. Ensure that you have connected Grove Red LED correctly into the port **D5** of the GrovePi



14. Click on the Drop Down button beside the Device and Select “**Remote Machine**” to configure the deployment settings.



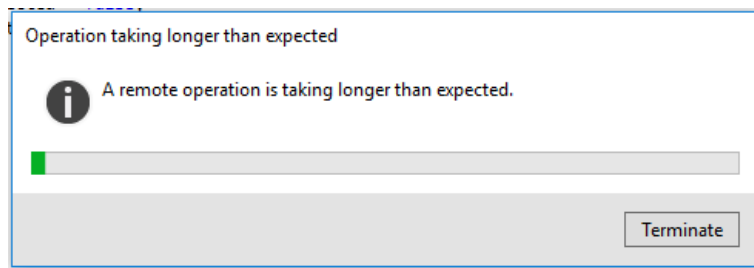
15. Key in the address “**169.254.8.88**” manually as shown below and click on “**Select**” after that. (if you didn't see this screen, refer to **Practical 1 Exercise 2** on steps to display this screen).



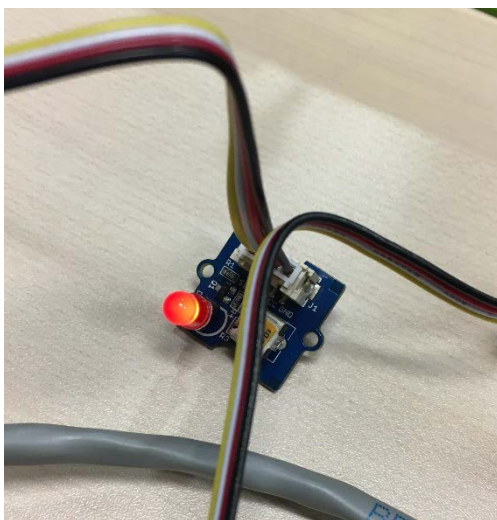
16. Click on Remote Machine to deploy and run the program on Raspberry Pi.



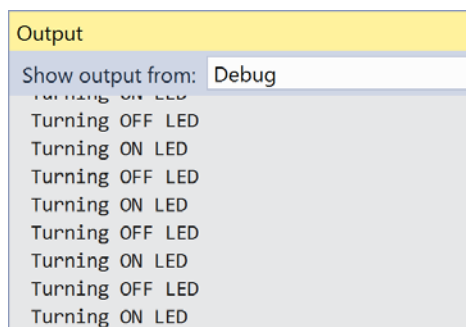
17. You may see the following warning but it is fine. Every time when you deploy a project for the first time on the hardware, it will take a long time to deploy. Just wait for a while and it should be deployed successfully.



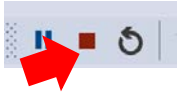
18. After about 10 seconds, notice there is a Red LED blinking. Examine the codes and make sure you understand what is going on. Clarify with your tutor if you do not understand the codes.



19. Another way to know if your codes are working well is to look at the Output. Sometimes the LED lights might not be as bright but when you look at the Output, you can use the debugging messages to verify your program is running as intended. (if the lights is showing correctly, Maybe you can check again if you put in the cable at port D5 as the codes are designed so that it only work on port D5)



20. Be careful when you would like to stop the program. Click on the Stop icon (below) only when the LED is OFF. If you stop the program while the LED is ON, the LED will remain ON. Haha, so beware!



How to know if program is still deploying?

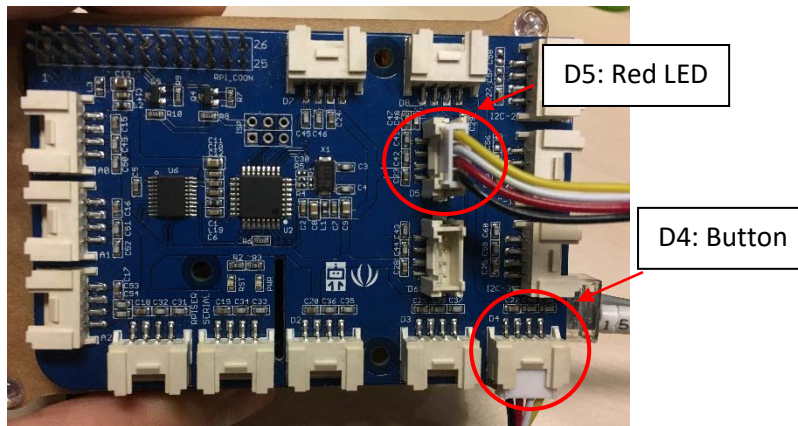
If this icon is still showing, that means the program is still deploying and you have to wait.



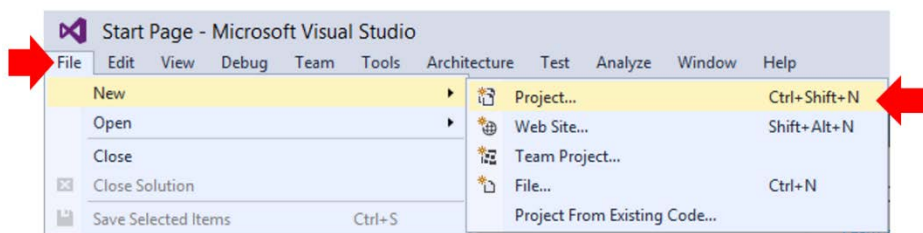
If the program is taking too long to deploy, terminate and do it again.

Exercise 2: Create LedButtonToggle Program

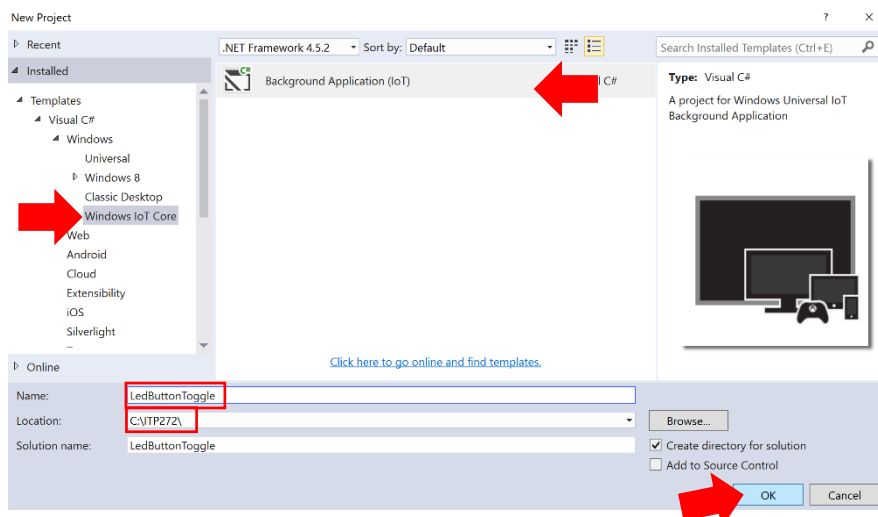
1. Ensure that you have connected the **Button** at **port D4** and **Red LED** to **port D5** at GrovePi device. See image below for clearer vision.



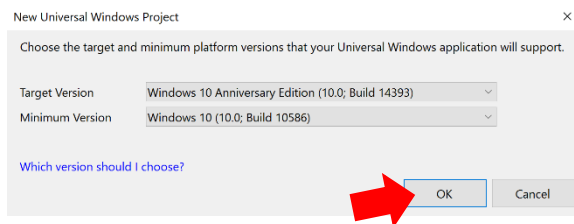
2. Start Visual Studio 2015, Click on File menu > Select New Project.



3. From the Installed Templates, select Visual C# - Expand Windows and Select Windows IoT Core. Select Background Application (IoT). Name your project as **“LedButtonToggle”** and Save it at your ITP272 folder then Click OK.



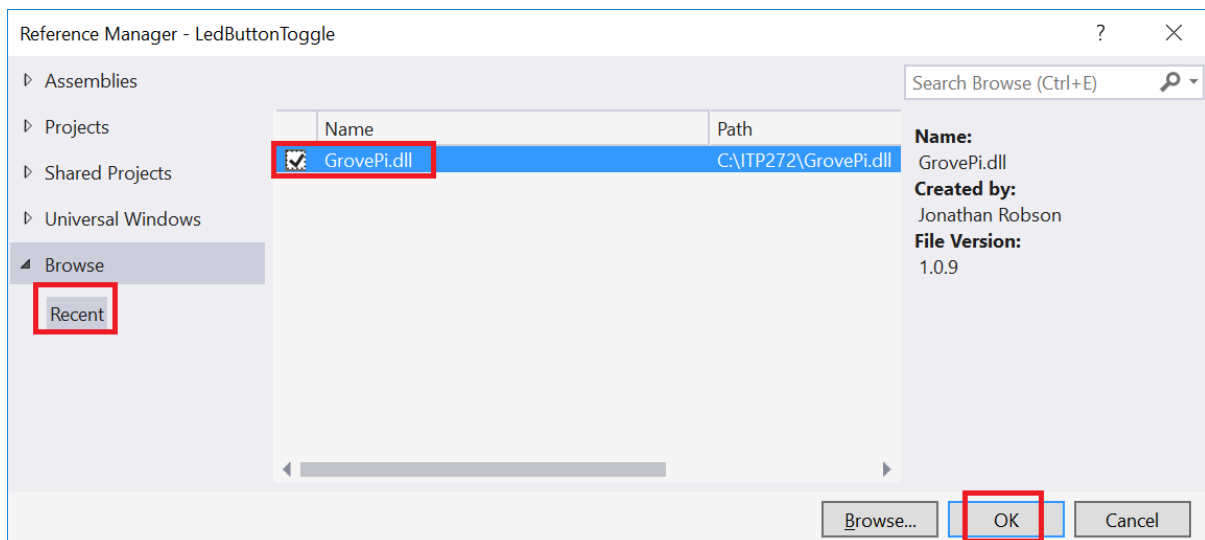
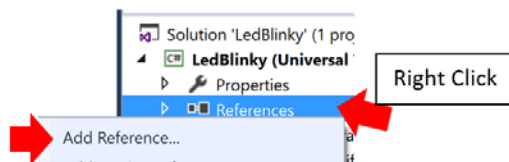
4. Click OK when you see this.



5. A “StartupTask.cs” should be opened once it is created. The main method is

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

6. Right Click on References and Click Add Reference. Select GrovePi.dll (since you have used it earlier, it will be saved in “Recent” so that it is easier for future usage)



7. Type in the codes to be included in the required library. It is okay for it to be in grey now since it is not used yet.

```
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;
```

8. Add the codes to the **StartupTask.cs** file above the “**Run()**” method. The code for **IButtonSensor** is similar to **ILed** which is to assign the Button to D4. The **Sleep** method is similar to **Lab Practical 1**.

```
public sealed class StartupTask : IBackgroundTask
{
    //State Machine variables to control different mode of operation
    //const keyword is to declare a constant field or local and in this case, the constant is the integer
    const int MODE_OFF = 1;
    const int MODE_ON = 2;
    //to store the current mode the program is at
    static int curMode;

    //use D5 for Red LED and D4 for Push Button
    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 being pressed
    private bool buttonPressed = false;

    //A method to make the device to wait awhile before executing the next command
    private void Sleep(int NoOfMs)
    {
        Task.Delay(NoOfMs).Wait();
    }
}
```

9. Next, you're going to read the state of the pushbutton. Add the following codes.

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

```
//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()
```

The **.CurrentState.ToString()** is used to check the status of the Grove push button. When it is pressed, it will return “On”. This **startButtonMonitoring()** method is created such that once it is called, it will run by itself own forever in its **while()** loop. Its sole purpose is to monitor the button and update the variable **buttonPressed** to true when the button is pressed. This **buttonPressed** variable is used to inform the “**Run()**” method that the button is pressed. The “**Run()**” method will then decide what to do and it needs to clear the **buttonPressed** back to false after processing.

10. In the application, we're going to create a state machine that has 2 different states (MODE_OFF and MODE_ON). We'll have more explanation of the state machine in the later part of the Lab. Meanwhile, we're going to create the handler methods of these 2 different states. Add the codes in between the **StartButtonMonitoring()** method and the **Run()** method.

```
//End of startButtonMonitoring()

//This method handles all logic when curMode is MODE_OFF
1 reference
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===");
    }
}
//End of HandleModeOff()

//This method handles all logic when curMode is MODE_ON
1 reference
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 Off when button is pressed
        curMode = MODE_OFF;
        Debug.WriteLine("===Entering MODE_OFF===");
    }
}
//End of HandleModeOff

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

11. We're now going to write the logic for the "Run()" method. Add in the codes below

```
    } //End of HandleModeOff

0 references
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();

    //Init 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);

        //Usually when program has complex handling, we use a state machine
        //to handle the behaviour.
        //We shall now practise how to implement a toggle LED using states
        //when LED is Off, we assign it to state MODE_OFF
        //when LED is On, we assign it to state MODE_ON
        //Their different handling is written in different methods
        //This logic here ensure the correct method is run in the different states
        if (curMode == MODE_ON)
            handleModeOn();
        else if (curMode == MODE_OFF)
            handleModeOff();
        else
            Debug.WriteLine("ERROR: Invalid Mode. Please check your logic");
    }
}
```

12. Here's an explanation on what the code does. It is important to understand the logic of the codes here as subsequent Labs will be based on the same fundamental logics. You can also read the comments added in the codes to understand the codes. Please clarify with tutor if you have doubts.

- The Program starts from the **Run()** method
- Calls **startButtonMonitoring()** method that will run on it's own to monitor the button and update the variable **buttonPressed** to true whenever the button is pressed
- The application is designed with a state machine. A state machine is be used to create complex handling and

behaviour. The application can be broken down to different states/modes and the behaviour can be different for different states/modes. The state machine uses **const int** to create different modes. In this application, it creates 2 different modes which are **MODE_OFF** and **MODE_ON**. It then keep tracks of it's current mode of the application in the variable **curMode**. So the code **cur_mode = MODE_ON** is initialising the application to start from the **MODE_ON** mode.

- **ledRed.ChangeState(SensorStatus.Off)** is to turn off the LED initially
- **Debug.WriteLine()** is a command to print debug message in the Output Window. This helps us to keep track of what the application is doing.
- As mentioned before the Raspberry Pi program needs to be running indefinitely (forever) checking on the sensors and deciding on what to do. So we need to have a while loop that is always running non-stop. This is done by **while(true)** {...}
- We mentioned just now that we've designed a state machine to handle the different states. This is done inside the body of the while loop. The following if-else statements check the variable **curMode** to see what the current mode of the application is and then call the corresponding Handler method which contain the behaviour of that particular mode.

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

As we set the initial mode to **MODE_ON**, this will cause the while loop to always call **handleModeOn()**.

- In a state machine handler method, you need to also define 2 things
 - Behaviour in current mode
 - Condition to move on to other mode(s)

```
//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 Off when button is pressed
        curMode = MODE_OFF;
        Debug.WriteLine("===Entering MODE_OFF===");
    }
}
```

Behaviour in current mode (MODE_ON)

It defines to turn on the LED in this mode.

Condition to move to other mode(s)

If button is pressed

- Reset the variable **buttonPressed** back to false
- Change mode to **MODE_OFF**
- Display debug message to show current mode of software

- When mode is changed to **MODE_OFF**, the while loop in **Run()** will now always call **handleModeOff()** handler which has a different behaviour from **MODE_ON**

```
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===");
    }
}
//End of HandleModeOff()
```

Behaviour in current mode (MODE_OFF)

It defines to turn off the LED in this mode.

Condition to move to other mode(s)

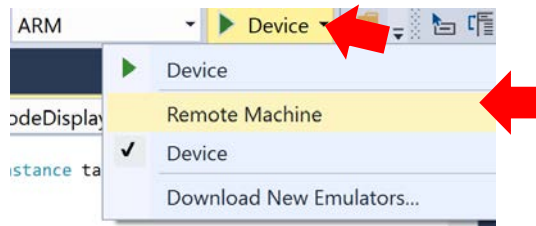
If button is pressed

- Reset the variable **buttonPressed** back to false
- Change mode to **MODE_ON**
- Display debug message to show current mode of software

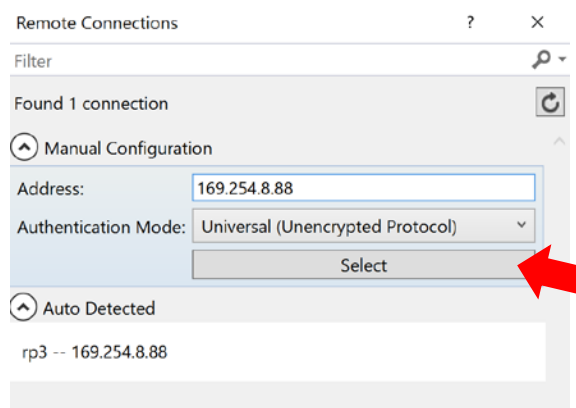
- This state machine will keep running and changing modes. You can add in many modes for complex handling.

13. The current application has only 2 modes and actually it does not really need a state machine. The handling can be easily be done with simple if-else logic. The state machine is still used here for this simple 2 modes application because it is simpler for you to understand it and once you know how to use it, you can create many modes to design more complex handling.

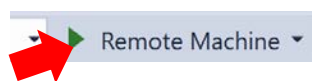
14. You can now deploy the program to your hardware. Click on the Drop Down button beside the Device and Select **Remote Machine**.



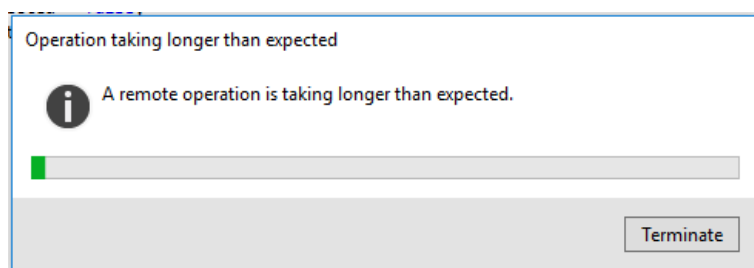
15. Key in the address “169.254.8.88” manually like shown below. Click on **Select** after that.



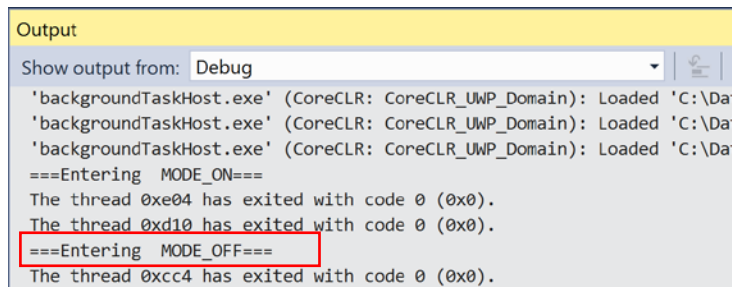
16. Click on Remote Machine to Run the Program.



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 deployed successfully, it will enter MODE_ON. So, you should be able to see the red LED lights up. Now, press the Button. You should see that the LED turns off. Open your Output window and you should see something like the following.

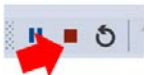


```
Output
Show output from: Debug
'backgroundTaskHost.exe' (CoreCLR: CoreCLR_UWP_Domain): Loaded 'C:\Da
'backgroundTaskHost.exe' (CoreCLR: CoreCLR_UWP_Domain): Loaded 'C:\Da
'backgroundTaskHost.exe' (CoreCLR: CoreCLR_UWP_Domain): Loaded 'C:\Da
===Entering  MODE_ON===
The thread 0xe04 has exited with code 0 (0x0).
The thread 0xd10 has exited with code 0 (0x0).
===Entering  MODE_OFF===
The thread 0xcc4 has exited with code 0 (0x0).
```

The application is designed to start with MODE_ON which turns on LED. It will change to MODE_OFF when the button is pressed. At MODE_OFF, it turns off the LED. When button is pressed again, it will change back to MODE_ON and turns on the LED. This continues and thus the LED toggles on and off for every button pressed.

You need to be able to understand the code and explain the behaviour so that you know how to design the behaviour of the application.

19. You can now stop the program or maybe you can play around with the button for a while but MAKE SURE to stop the program when it is at MODE_OFF.



Exercise 3: Creating LedButtonMultiModeDisplay

The exercise illustrates how you could program a system with different modes as below.

This system consists of 3 modes.

- MODE_ON
- MODE_BLINK
- MODE_OFF

MODE_ON

The system will start up with “MODE_ON”.

In this mode, the system will behave as follows

- Always on both Red and Green LED
- If button is pressed, it will go to “MODE_BLINK”

MODE_BLINK

In this mode, the system will behave as follows

- Start blinking Red LED every second for 3 times
- Turn off Red LED and start blinking Green LED every second for 3 times
- Turn off Green LED and change to “MODE_OFF”

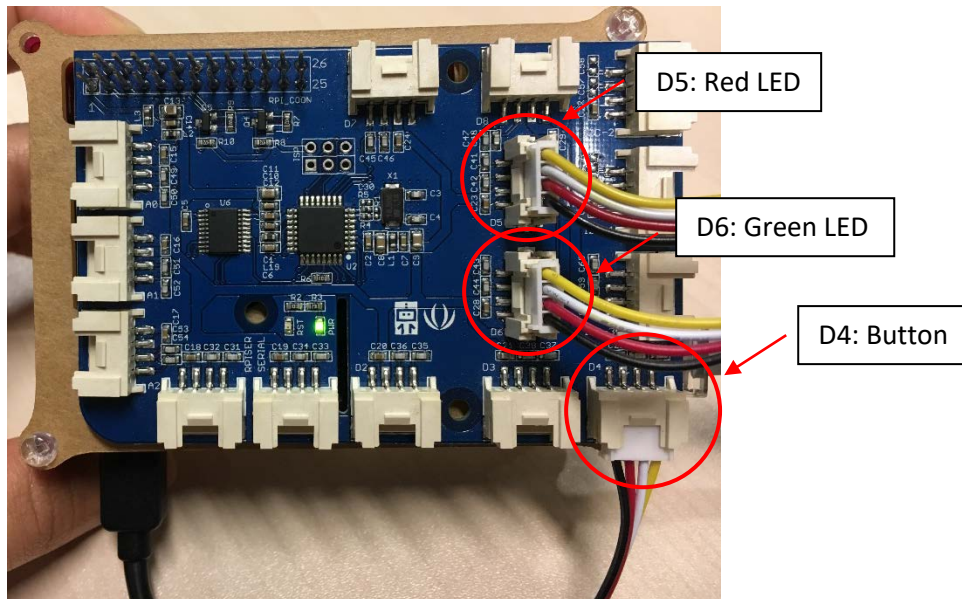
MODE_OFF

In this mode, the system will behave as follows

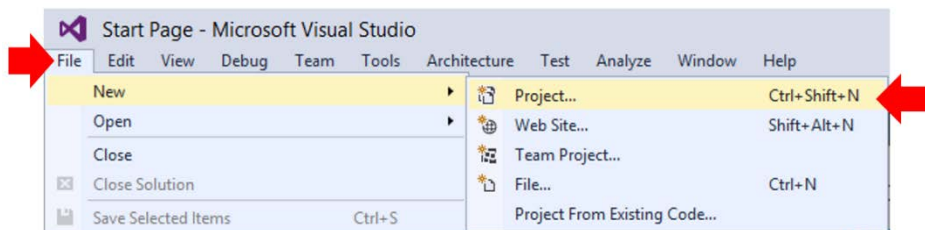
- Turn Off both Red and Green LED
- if button is pressed, it will go back to “MODE_ON”

ITP272 Sensor Technologies and Project

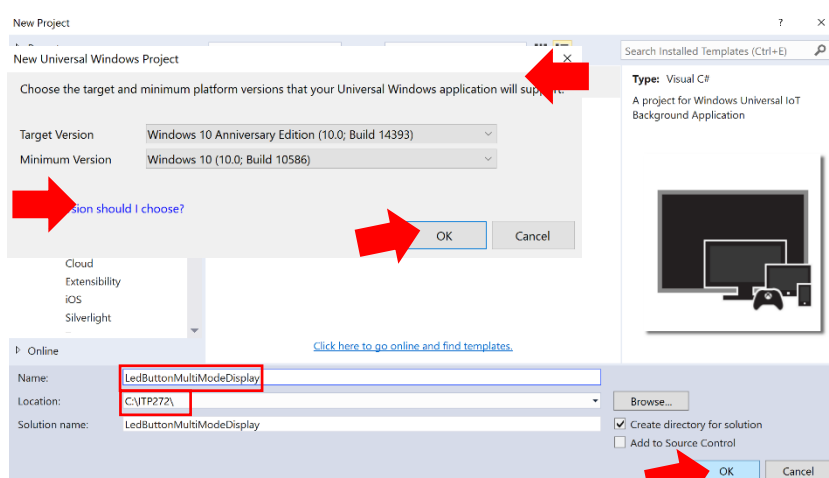
1. Add in **Green LED** connection to port **D6**. You should now have **Button** connected to port **D4**, **Red LED** connected to port **D5** and **Green LED** at port **D6**. See image below for clearer vision.



2. Start Visual Studio 2015, Click on File menu > Select New Project.



3. From the Installed Templates, select Visual C# - Expand Windows and Select Windows IoT Core. Select Background Application (IoT). Name your project as **LedButtonMultiModeDisplay** and Save it at your ITP272 folder then Click OK.

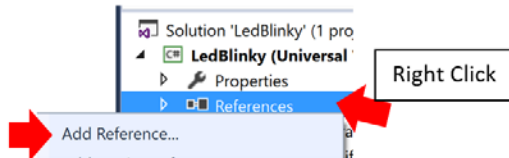


4. Click OK when you see this.

5. A 'StartupTask.cs' should be opened once it is created. The main method is

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

6. Right Click on References and Click Add Reference. Select GrovePi.dll (since you have used it earlier, it will be saved in Recents so that it is easier for future usage)



7. Type in the codes to be included in the required library. It is okay for it to be in grey now since it is not used yet.

```
// The Background Application t
using System.Diagnostics;
using System.Threading.Tasks;
using GrovePi;
using GrovePi.Sensors;
```

8. Add the codes to the **StartupTask.cs** file above the **Run()** method

```
public sealed class StartupTask : IBackgroundTask
{
    //State Machine variables to control different mode of operation
    //const keyword is to declare a constant field or local and in this case, the constant is the integer
    const int MODE_ON = 1;
    const int MODE_OFF = 2;
    const int MODE_BLINK = 3;
    static int curMode; //stores the current mode the program is at

    //Use D5 for Red LED, D6 for Green LED and D4 for Button
    ILed ledRed = DeviceFactory.Build.Led(Pin.DigitalPin5);
    ILed ledGreen = DeviceFactory.Build.Led(Pin.DigitalPin6);
    IButtonSensor button = DeviceFactory.Build.ButtonSensor(Pin.DigitalPin4);

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

    //This method is to make the program wait awhile before the next execution
    private void Sleep(int NoOfMs)
    {
        Task.Delay(NoOfMs).Wait();
    }
}
```

9. Next, you're going to read the state of the pushbutton. Add in the following codes

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

//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()
```

10. Similar to Exercise 2, we're now going to add in all the state machine handler methods. In this exercise, we have 3 handler methods. Add the codes for the handler **handleModeOn()**

```
} //End of startButtonMonitoring()

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

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

        //Move on to Mode Blink when button is pressed
        curMode = MODE_BLINK;
        Debug.WriteLine("===Entering MODE_BLINK===");
    }
}
} //End of handleModeOn()
```

11. Add in the below codes for the handler **handleModeBlink()** and **handleModeOff()**

```

} //End of handleModeOn()

1 reference
private void handleModeBlink()
{
    // 1. Define Behaviour in this mode

    //Off both LED
    ledRed.ChangeState(SensorStatus.Off);
    ledGreen.ChangeState(SensorStatus.Off);

    //Blink Red LED for 3 secs
    for (int i = 0; i < 3; i++)
    {
        ledRed.ChangeState(SensorStatus.On);
        Sleep(1000);
        ledRed.ChangeState(SensorStatus.Off);
        Sleep(1000);
    }

    //Blink Green LED for 3 secs
    for (int i = 0; i < 3; i++)
    {
        ledGreen.ChangeState(SensorStatus.On);
        Sleep(1000);
        ledGreen.ChangeState(SensorStatus.Off);
        Sleep(1000);
    }

    // 2. Must write the condition to move on to other modes
    //Move on to Mode OFF right after the blinking
    curMode = MODE_OFF;
    Debug.WriteLine("===Entering MODE_OFF===");
}

1 reference
private void handleModeOff()
{
    // 1. Define Behaviour in this mode

    //Off both LED
    ledRed.ChangeState(SensorStatus.Off);
    ledGreen.ChangeState(SensorStatus.Off);

    // 2. Must write the condition to move on to other modes
    if (buttonPressed == true)
    {
        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)

```


20. We're now going to write the logic for the "Run()" method. Add in 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
    //

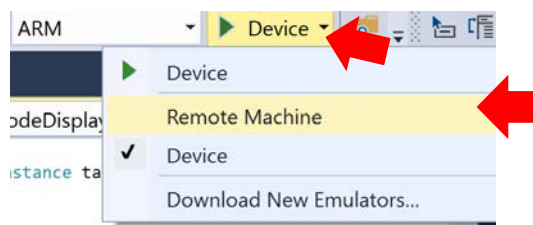
    //Start button self monitoring
    startButtonMonitoring();

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

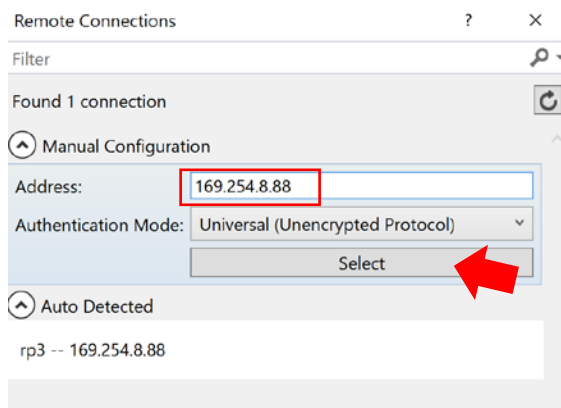
    //This makes sure the main program runs indefinitely
    while (true)
    {
        Sleep(300);

        //State machine handling
        if (curMode == MODE_ON)
            handleModeOn();
        else if (curMode == MODE_BLINK)
            handleModeBlink();
        else if (curMode == MODE_OFF)
            handleModeOff();
        else
            Debug.WriteLine("ERROR: Invalid Mode. Please check your logic");
    }
}
```

12. Deploy and run the program on Raspberry Pi. Click on the Drop Down button beside the Device and select Remote Machine.



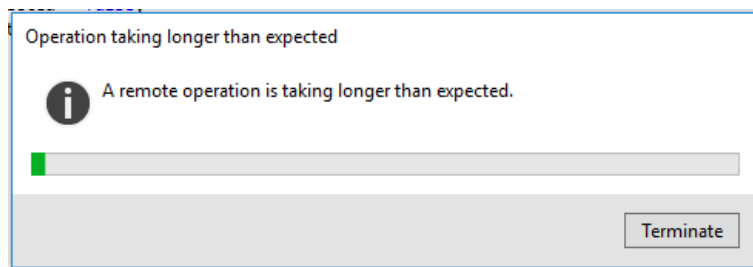
13. Key in the address “**169.254.8.88**” manually like shown below. Click on Select after that.



14. Click on Remote Machine to Run the Program.



15. 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.



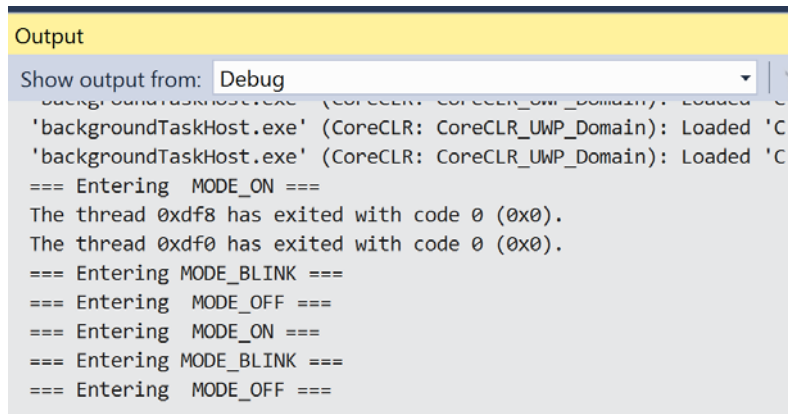
16. After program deployed successfully, it will enter MODE_ON. You should be able to see the red and green LED turn on.

Click on the button and it will go into MODE_BLINK. In MODE_BLINK, the red LED will blink 3 times followed by the Green LED blinking 3 times. After the blinking, it will go into MODE_OFF.

In the MODE_OFF, both Red and Green LED remains off until the button is pressed.

Click on the button and the program goes back to MODE_ON.

17. Play around with the button and notice how the LED lights changes at the different modes. You can go to Output and to see the mode that the device is at. An example is shown below.



```
Output
Show output from: Debug
'backgroundTaskHost.exe' (CoreCLR: CoreCLR_UWP_Domain): Loaded 'C:
'backgroundTaskHost.exe' (CoreCLR: CoreCLR_UWP_Domain): Loaded 'C:
=== Entering MODE_ON ===
The thread 0xdf8 has exited with code 0 (0x0).
The thread 0xdf0 has exited with code 0 (0x0).
=== Entering MODE_BLINK ===
=== Entering MODE_OFF ===
=== Entering MODE_ON ===
=== Entering MODE_BLINK ===
=== Entering MODE_OFF ===
```

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

==End of Practical==