Nanyang Polytechnic

School of Information Technology

Course : Diploma in Infocomm & Security (ITDF12)

Module : Sensor Technologies and Project (ITP272)

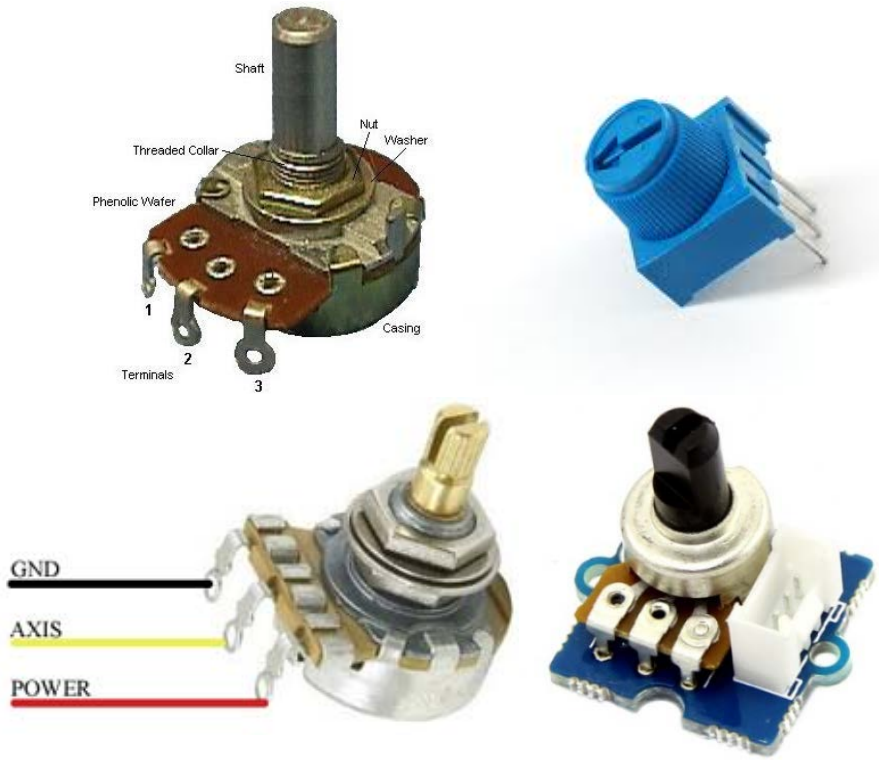
Raspberry Pi Practical : Potentiometer (Analog Input)

Objectives:

- Learn what a potentiometer is
- Learn how to interface with Analog Input
- Learn how to read in Potentiometer value
- Learn how to program with state machines to use potentiometer as well as LED

Practical

Understanding Potentiometer or Grove - Rotary Angle Sensor



The Grove Rotary Angle Sensor is actually a potentiometer. A potentiometer has three pins:

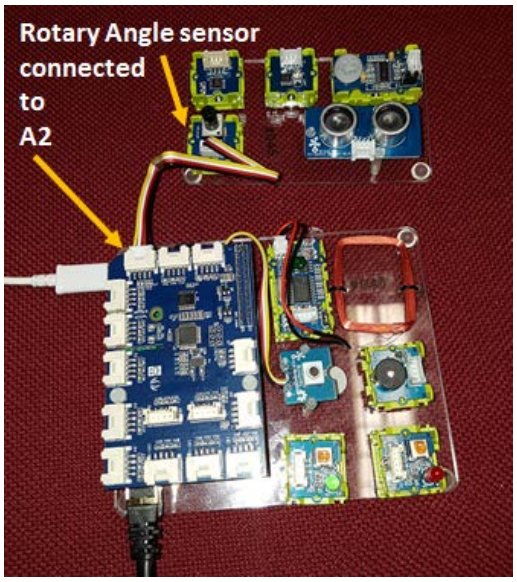
- one for an input voltage (such as 3.3V or 5V)
- one for ground or GND (0V)
- one that is connected to the wiper controlled by a Knob

When the potentiometer's knob is turned, the output voltage varies (from 0V at one side, 1.65V in the middle, and 3.3V turned all the way). The changes are linear; as a result, you can measure the position of the knob based on its output voltage. As the output voltage can have various a range of different states (more than 2), it is read in to the Raspberry Pi using Analog Input of the Grove Pi.

Exercise 1: Create PotVoltage Program

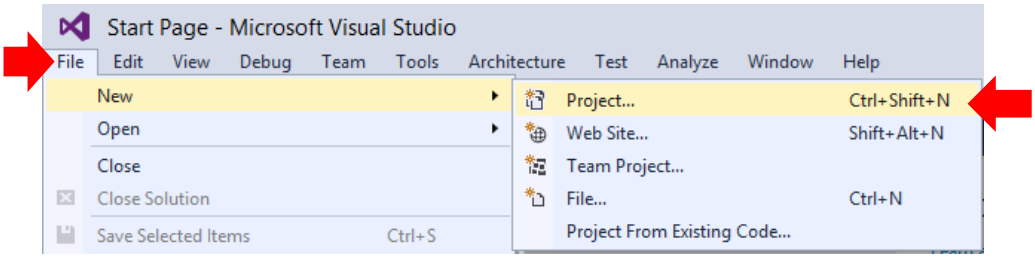
1. Today we will be creating a program that allows you monitor the input from Rotary Angle Sensor or commonly referred to as the Pot (potentiometer). You can see the potentiometer used as volumn or Frequency tuning knob.

2. We shall use Analog Input port **A2** of the GrovePi for this sensor. Ensure that you have the port **A2** connected at **Potentiometer (Rotary Angle Sensor)**.

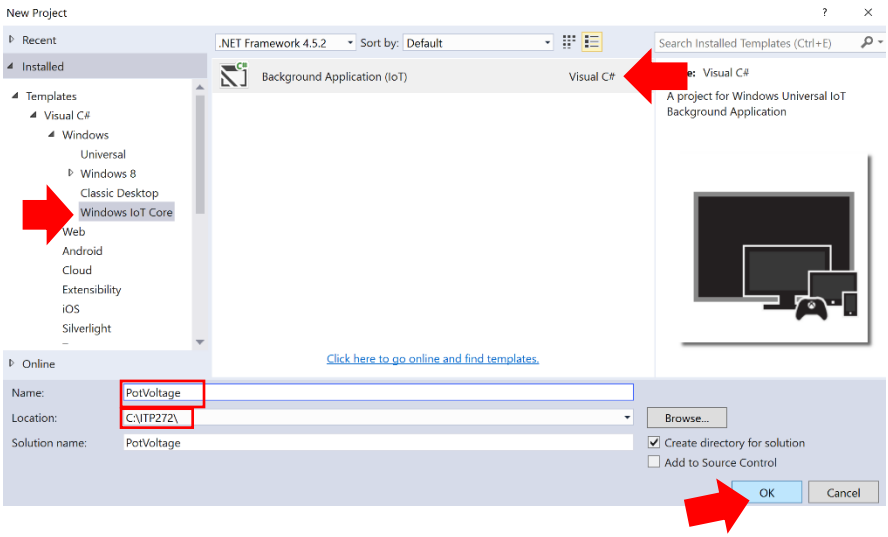


Rotary Angle Sensor

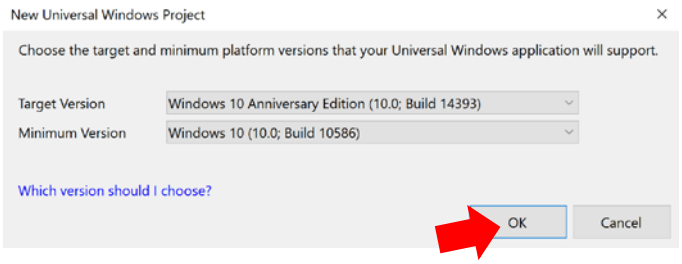
3. Firstly, start Visual Studio 2015, Click on File – New - Project.



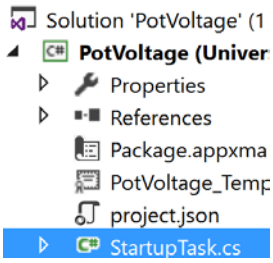
4. From the Installed Templates, select Visual C# – Windows IoT Core – Background Application. Name your project as **PotVoltage**, save it in you ITP272 folder and Click OK.



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.

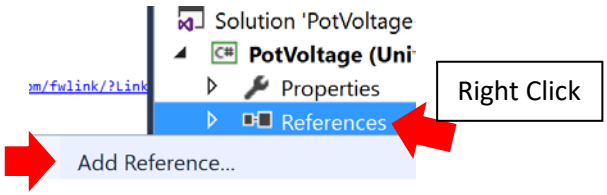


7. You should see the method

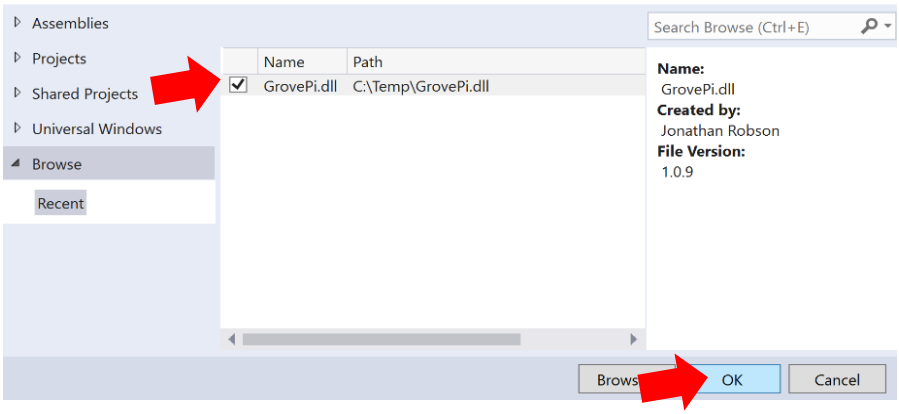
```
public void Run(IBackgroundTaskInstance taskInstance)
```

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 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 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**.

We've created a Pin object (potPin) and assigned it to the Analog Input 2 (A2) as we've connected our Sensor to A2

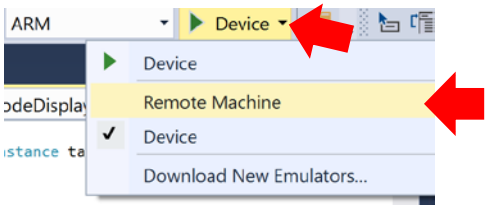
```
public sealed class StartupTask : IBackgroundTask
{
    //Use A2 for potentiometer
    Pin potPin = Pin.AnalogPin2;

    //See Signal conditioning Chapter - ADC for definition and details
    const double EFSR = 5;  //ADC Full scale voltage Range, E(FSR) for Grove Sensor is 5 V input
    const int N = 1023;   //(10-bit resolution ADC, quantization level N = 1023)

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

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

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

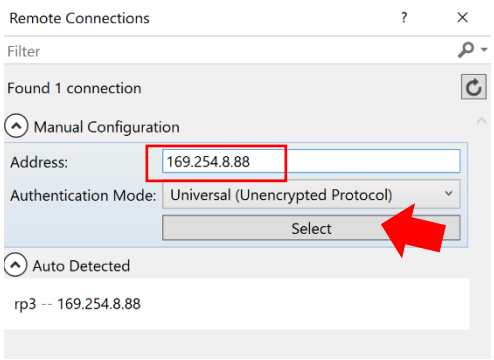
```
public void Run(IBackgroundTaskInstance taskInstance)
{
    //
    // TODO: Insert code to perform background work
    //
    // If you start any asynchronous methods here, prevent the task
    // from closing prematurely by using BackgroundTaskDeferral as
    // described in http://aka.ms/backgroundtaskdeferral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

    int adcValue = 0;
    double Q = 0.0; //Voltage resolution
    double sensorVoltage = 0.0;

    while (true)
    {
        Sleep(300);
        adcValue = DeviceFactory.Build.GrovePi().AnalogRead(potPin);
        Debug.WriteLine("Pot ADC = " + adcValue);

        Q = EFSR / N;   //See Signal conditioning Chapter
        sensorVoltage = adcValue * Q;
        Debug.WriteLine("Convert to Voltage: " + sensorVoltage.ToString("n2") + "V");
    }
}
```

DeviceFactory.Build.GrovePi().AnalogRead() method is used to read in values from the GrovePi Analog Input. We need pass in the potPin object so that the Raspberry Pi knows it needs to read in from A2. The value read in is a digital value. We use a formula to convert the digital value into the actual electrical voltage coming into the raspberry Pi from the sensor output. We'll talk about the formula later

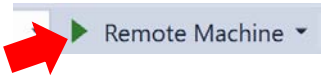
13. Click on the Drop Down button beside the Device and Select "**Remote Machine**" to configure the deployment settings.



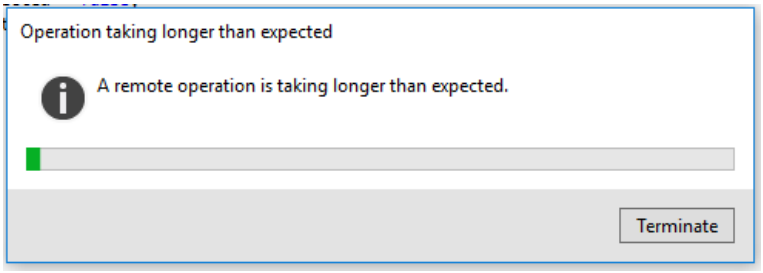
14. Key in the address "**169.254.8.88**" manually as shown below and click on "**Select**" after that. (if you didn't see this screen, refer to **Practical 1 Exercise 2** on steps to display this screen).



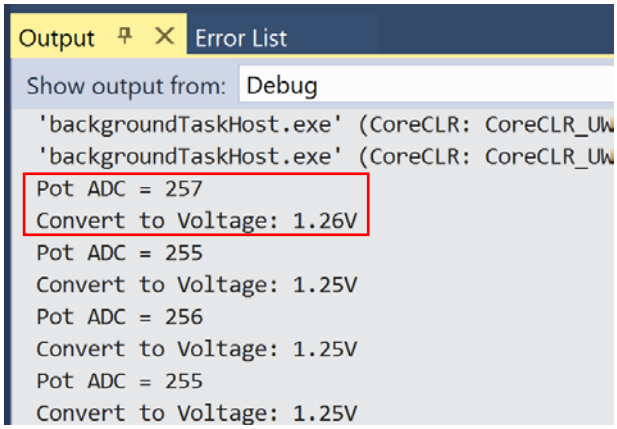
15. Click on Remote Machine to Run the Program.



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.



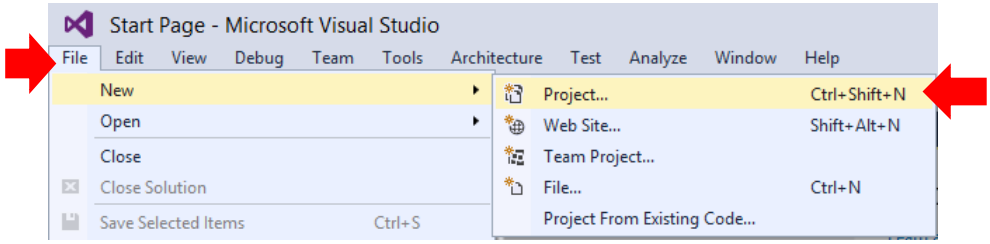
17. Once deployed successfully, Turn the potentiometer's knob from left, center, and to the right. The Output windows will display different ADC Value with the voltage value for every turn. You can use the value read in to know where and how much the user turned.



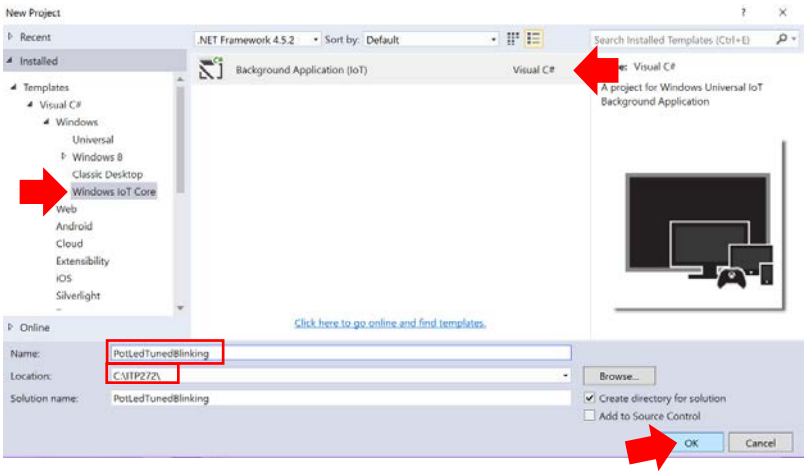
Exercise 2: Create PotLedTunedBlinking

1. In this exercise, we will be creating a program that will allow you to adjust the frequency of the blinking Red LED.
2. Add in Red LED and Push Button. Ensure that you have the following connections
    - A2 : Potentiometer
    - D5 : Red LED
    - D4 : Push Button.

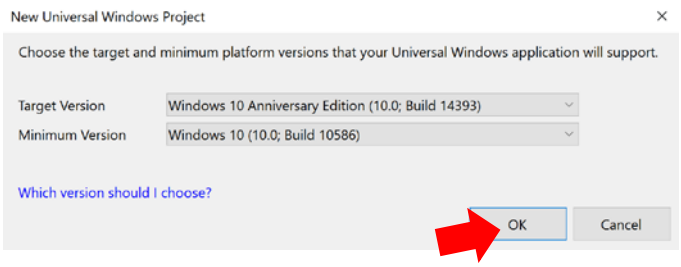
3. Firstly, start Visual Studio 2015, Click on File – New - Project.



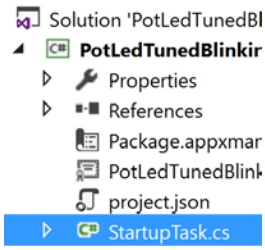
4. From the Installed Templates, select Visual C# – Windows IoT Core – Background Application. Name your project as **PotLedTunedBlinking**, save it in you ITP272 folder and Click OK.



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.

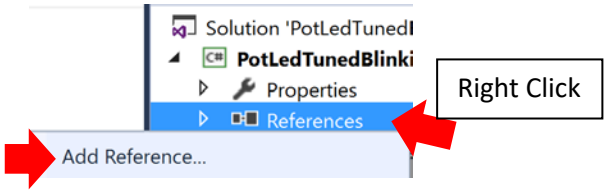


7. You should see the method

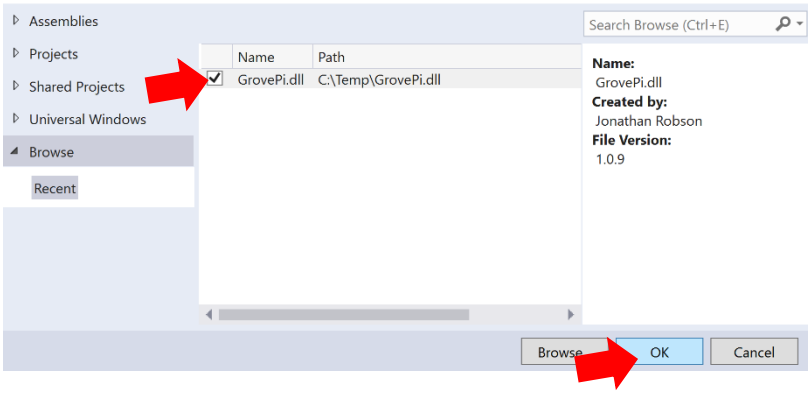
```
public void Run(IBackgroundTaskInstance taskInstance)
```

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 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. Type in the codes to include required libraries. It is fine to be in gray initially since you have not use it.

```
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
{
    //State Machine vairables to control different mode of operation
    const int MODE_BLINK = 1;
    const int MODE_OFF = 2;
    static int curMode;     //stores the current mode the program is at

    //Use port A2 for potentiometer, D5 for Red Led and D4 for button
    Pin potPin = Pin.AnalogPin2;
    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;

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

12. Create a self-monitoring method to monitor and update button status.

```
    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()
```

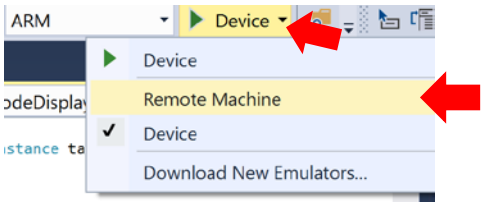
13. Create a handler methods for the operation of the various modes. Add the codes between the **startButtonMonitoring()** and the **Run()** method.

```
}//End of startButtonMonitoring()

1 reference
private void handleModeBlink()
{
    //  1.  Define Behaviour in this mode
         //Blink Red LED
        //speed of blinking follows Potentiometer adc value
        int adcValue = 0;
        adcValue = DeviceFactory.Build.GrovePi().AnalogRead(potPin);
        ledRed.ChangeState(SensorStatus.On);
        Sleep(adcValue);
        ledRed.ChangeState(SensorStatus.Off);
        Sleep(adcValue);

   // 2.   Must write the condition to move on to other modes
        //Move on to Mode OFF right after the blinking
        if (buttonPressed == true)
        {
            buttonPressed = false;

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

1 reference
private void handleModeOff()
{
    //  1.  Define Behaviour in this mode
        //Off  lED
        ledRed.ChangeState(SensorStatus.Off);

    // 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===");
        }
}

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

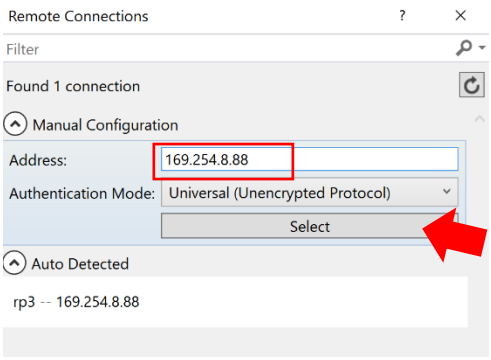
14. Lastly, add the codes for the **Run()** method.

```
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
    //

    //Start button self monitoring
    startButtonMonitoring();

    //Initialize Mode
    curMode = MODE_BLINK;
    Debug.WriteLine("=== Entering  MODE_BLINK ===");

    while (true)
    {
        Sleep(300);
        //State machine handling
        if (curMode == MODE_BLINK)
            handleModeBlink();
        else if (curMode == MODE_OFF)
            handleModeOff();
        else
            Debug.WriteLine("ERROR: Invalid Mode. Please check your logic");
    }
}
```

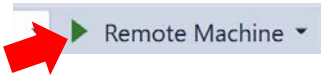
15. Click on the Drop Down button beside the Device and Select "**Remote Machine**" to configure the deployment settings



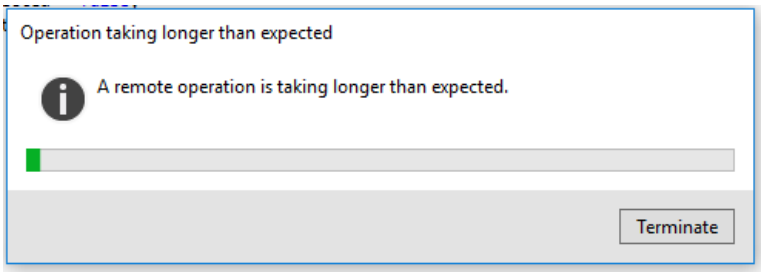
16. Key in the address "**169.254.8.88**" manually as shown below and click on "**Select**" after that. (if you didn't see this screen, refer to **Practical 1 Exercise 2** on steps to display this screen).



17. Click on Remote Machine to Run the Program.



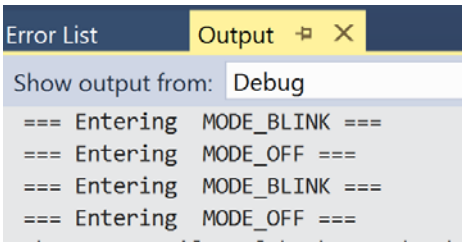
18. You may see the following warning but it is fine. Every time when you deploy a project for the first time on the hardware, it will take a long time to deploy. Just wait for a while and it should be deployed successfully.



19. After being deployed successfully, the application will start from MODE_BLINK. You will notice that the Red LED will blink.

    Try turning the Rotary Angle Sensor to adjust the duration of the LED blinking. The application is able to check the rotation angle of the sensor to determine how long to turn the LED On and Off.

    Press the button and the device will enter into MODE_OFF. You should see that the Red LED turns off.
    Look at your Output Window and you should see something like this.



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

==End of Practical==