Nanyang Polytechnic

School of Information Technology

Course : Diploma in Infocomm & Security (ITDF12)

Module : Sensor Technologies and Project (ITP272)

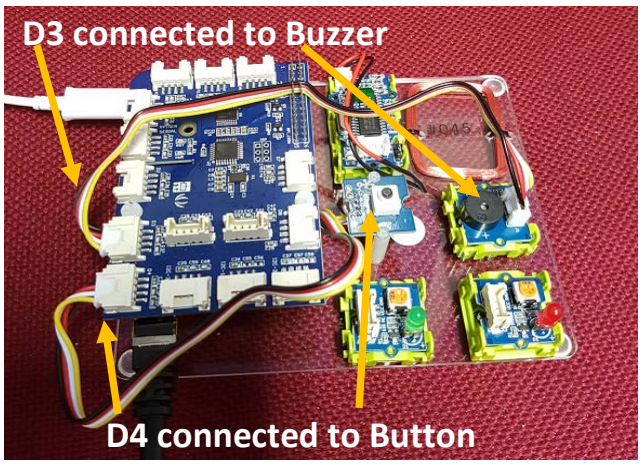
Raspberry Pi Practical : Programming Raspberry Pi with Grove Buzzer

Objectives:

- Learn how to interface with Digital I/O port
- Learn how to create program to control Grove Buzzer.
- Learn how to create program with state machines to work with buzzer and LEDs

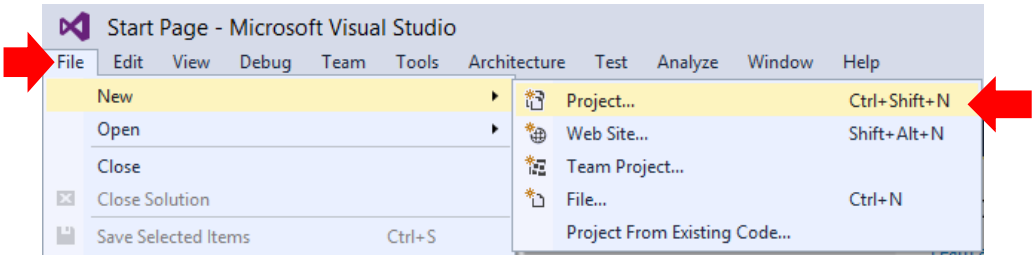
Exercise 1: Creating BuzzerButton

1. Today we will be creating a program that will sound a buzzer when the button is pressed.
2. We shall use Digital Input port **D3** of the GrovePi for the buzzer. Ensure that you have port **D3** connected to the **Buzzer** and port **D4** connected to the **Button**.

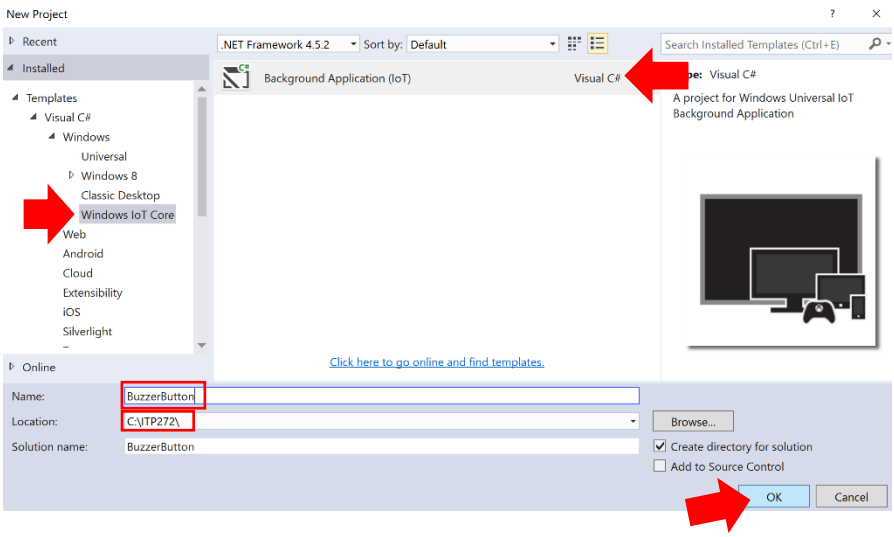


Buzzer

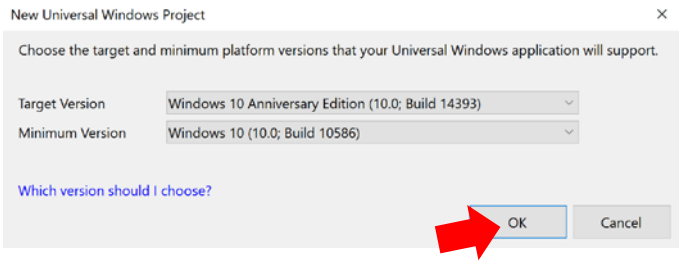
3. Start Visual Studio 2015, Click on File – New - Project.



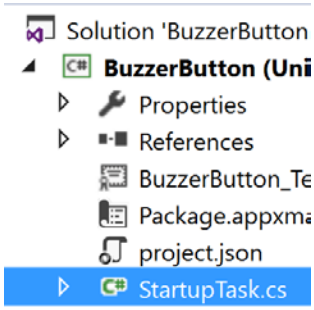
4. From the Installed Templates, select Visual C# – Windows IoT Core – Background Application. Name your project as **BuzzerButton**, save it in you ITP272 folder and Click OK.



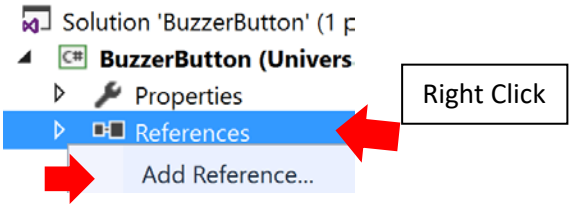
5. You should see this pop-up. Click OK.



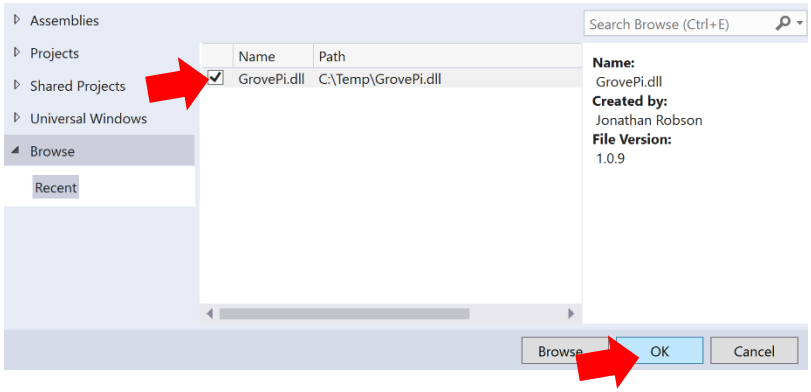
6. At the solution explorer on the right side of the screen, check that you have StartupTask.cs. This is the startup program file. Double Click on Startup Task.cs.



7. Add Reference for the GrovePi. Right Click on References on Solution Explorer and Click on Add Reference.



8. Download and select GrovePi.dll as usual. It should appear in your Recent if you've used it before. Check on it and Click OK.



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
{
    //We will use port D3 for Buzzer and D4 for Button
    Pin buzzerPin = Pin.DigitalPin3;
    IButtonSensor button = DeviceFactory.Build.ButtonSensor(Pin.DigitalPin4);

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

    //Creating a method to allow a pause before the next execution
    private void Sleep(int NoOfMs)
    {
        Task.Delay(NoOfMs).Wait();
    }
```

11. Next, create a **soundBuzzer()** method below Sleep() to sound the Buzzer.

```
private void soundBuzzer()
{
    DeviceFactory.Build.GrovePi().AnalogWrite(buzzerPin, 60);
    Sleep(80);
    DeviceFactory.Build.GrovePi().AnalogWrite(buzzerPin, 120);
    Sleep(80);
    DeviceFactory.Build.GrovePi().AnalogWrite(buzzerPin, 60);
    Sleep(80);
    DeviceFactory.Build.GrovePi().AnalogWrite(buzzerPin, 120);
    Sleep(80);
    DeviceFactory.Build.GrovePi().AnalogWrite(buzzerPin, 0);
    Sleep(2000);
}
```

You use **AnalogWrite()** to control the frequency of the buzzer. The codes here make the buzzer alternate between 2 different frequencies.

12. Create a self-monitoring method above the **Run()** method to monitor the button.

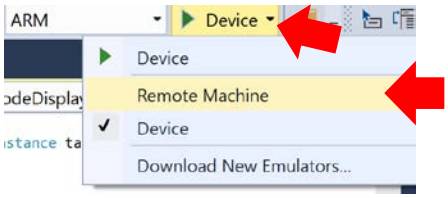
```
private async void startButtonMonitoring()
{
    await Task.Delay(100);
    while (true)
    {
        Sleep(100);
        string buttonState = button.CurrentState.ToString();
        if (buttonState.Equals("On"))
        {
            Sleep(100);
            buttonState = button.CurrentState.ToString();
            if (buttonState.Equals("On"))
            {
                buttonPressed = true;
            }
        }
    }
}//End of startButtonMonitoring()
```

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

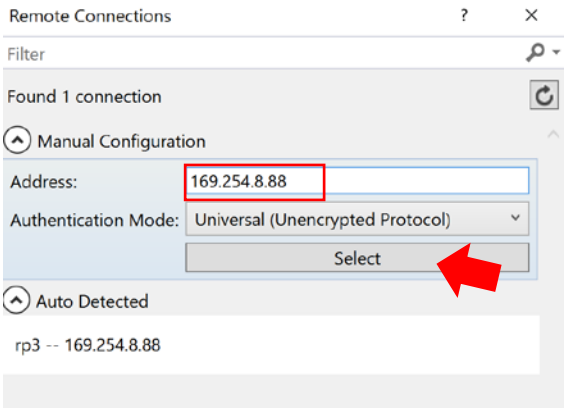
13. Finally, add the codes in the **Run()** method.

```
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
    //
    startButtonMonitoring();

    Debug.WriteLine("Program Started!");

    while (true)
    {
        Sleep(300);
        if (buttonPressed == true)
        {
            buttonPressed = false;
            Debug.WriteLine("Sounding Buzzer.");
            soundBuzzer();
        }
    }
}
```

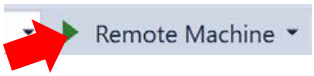
14. In order to deploy the program to your hardware, click on the Drop Down button beside the Device and Select Remote Machine.



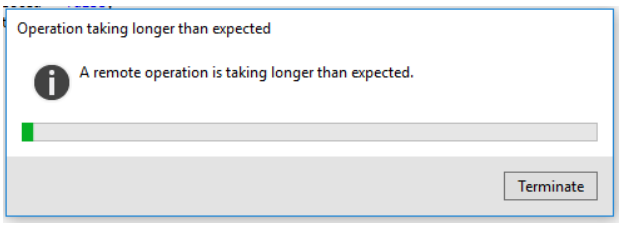
15. Key in the address manually like shown below. Click on Select after that.



16. Click on Remote Machine to Run the Program.

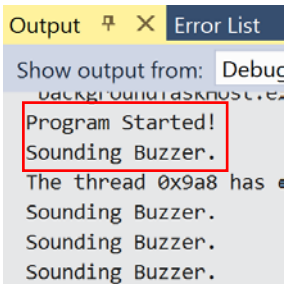


17. You may see the following warning but it is fine. Every time when you deploy a project for the first time on the hardware, it will take a long time to deploy. Just wait for a while and it should be deployed successfully.



18. Upon successful deployment, you will see a “Program Started!” in your Output window.

    Press the button and you should hear Buzzer sounded. In the Output window, it will show, “Sounding Buzzer.” Once done, you can end the program



    You’ve just learnt how to control the grove buzzer.

Exercise 2: Creating a BuzzerAlarmSystem

Implement a Home Alarm system. The Alarm system has a motion sensor that could detect movement in the house. Under normal mode, any movement in the house is ignored. When user leave the house, they could use a push button to set the system to armed mode where my any movement detected will sound the alarm.

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.

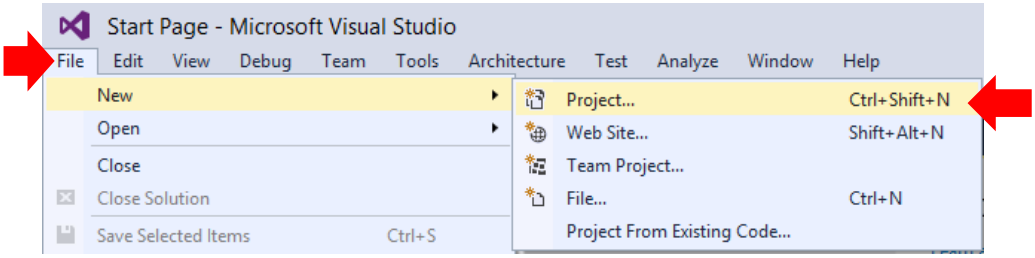
Your program starts in **MODE_Normal**

- **MODE_Normal**
  - Any movement picked up by the motion sensor is ignored
  - If push button is pressed, change mode to **MODE_ARM**

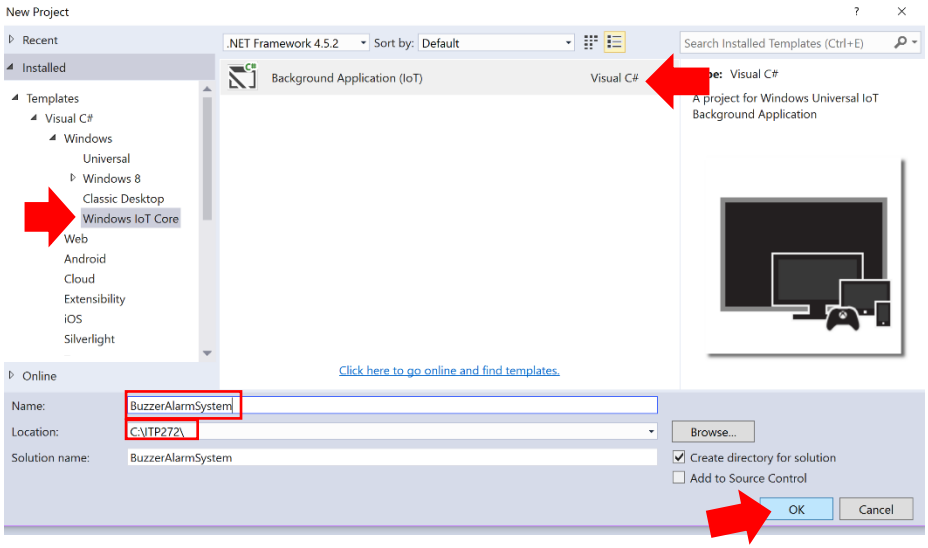
- **MODE_ARM**
  - Any movement picked up by the motion sensor will trigger the alarm indicating an intrusion in the house.
  - If push button is pressed, change mode to back **MODE_Normal**

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 later. 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.

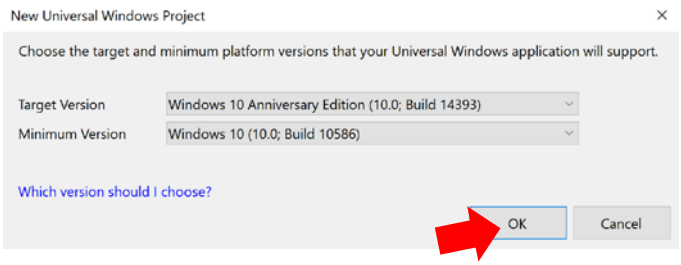
1. In this exercise, we will be creating a program **BuzzerAlarmSystem** as stated in the previous page.
2. Add in the PIR Motion Sensor. Ensure that you have the following connections
    - D2 : PIR Motion Sensor
    - D3 : Buzzer
    - D4 : Push Button
3. Start Visual Studio 2015, Click on File – New - Project.



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

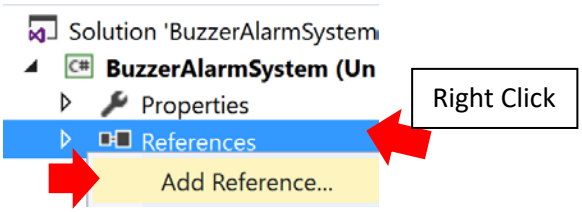


5. You should see this pop-up. Click OK.

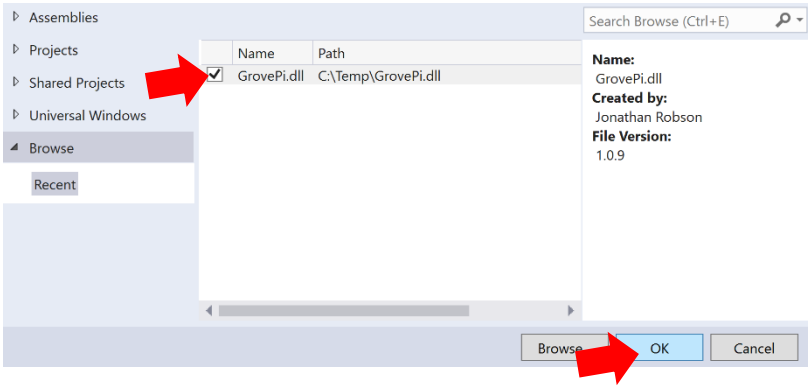


6. At the solution explorer on the right side of the screen, check that you have StartupTask.cs. This is the startup program file. Double Click on Startup Task.cs.

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



8. Select GroovePi.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 vairables to control different mode of operation
    const int MODE_NORMAL = 1;
    const int MODE_ARM = 2;
    static int curMode;

    //We shall use D2 for Motion, D3 for Buzzer, D4 for button
    Pin buzzerPin = Pin.DigitalPin3;
    IButtonSensor button = DeviceFactory.Build.ButtonSensor(Pin.DigitalPin4);
    Pin PirMotionSensorPin = Pin.DigitalPin2;

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

    private int count = 0;

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

11. Create a method for the **soundBuzzer**() as in previous exercise.

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

private void soundBuzzer()
{
    DeviceFactory.Build.GrovePi().AnalogWrite(buzzerPin, 60);
    Sleep(80);
    DeviceFactory.Build.GrovePi().AnalogWrite(buzzerPin, 120);
    Sleep(80);
    DeviceFactory.Build.GrovePi().AnalogWrite(buzzerPin, 60);
    Sleep(80);
    DeviceFactory.Build.GrovePi().AnalogWrite(buzzerPin, 120);
    Sleep(80);
    DeviceFactory.Build.GrovePi().AnalogWrite(buzzerPin, 0);
    Sleep(2000);
}//End of soundBuzzer
```

12. Create a two self-monitoring methods for the button and motion sensor as below

```
}//End of soundBuzzer
private async void startButtonMonitoring()
{
    await Task.Delay(100);
    while (true)
    {
        Sleep(100);
        string buttonState;
        buttonState = button.CurrentState.ToString();
        if (buttonState.Equals("On"))
        {
            Sleep(100);
            buttonState = button.CurrentState.ToString();
            if (buttonState.Equals("On"))
            {
                buttonPressed = true;
                Sleep(1000);
            }
        }
    }
}//End of startButtonMonitoring

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;
                Sleep(3000);
        }
    }
}//End of startMotionMonitoring
```

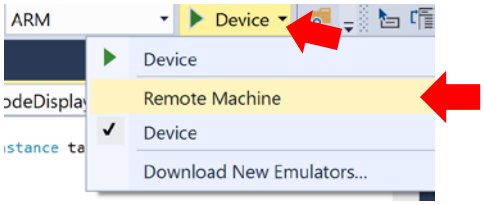
13. Create a handler methods for the operation of the various modes. Add the codes between the **startMotionMonitoring()** and the **Run()** method.

```
}//End of startMotionMonitoring

private void handleModeNormal()
{
    //  1.  Define Behaviour in this mode
    if (motionDetected == true)
    {
        motionDetected = false;
        Debug.WriteLine("Normal movement detected "+ count++);
    }

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

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

private void handleModeArm()
{
    //  1.  Define Behaviour in this mode
    if (motionDetected == true)
    {
        motionDetected = false;
        Debug.WriteLine("Intrusion Detected");
        soundBuzzer();
    }

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

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

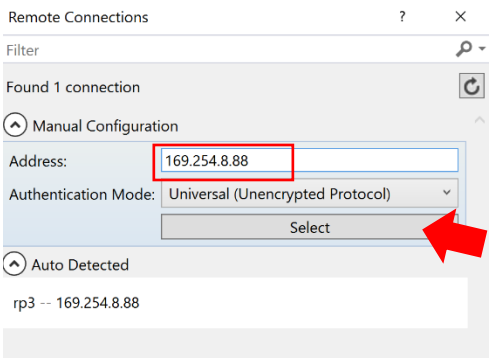
14. Lastly, add the codes for the **Run()** method.

```
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
    //

    startButtonMonitoring();
    startMotionMonitoring();

    //Init Mode
    curMode = MODE_NORMAL;
    Debug.WriteLine("===Entering  MODE_NORMAL===");

    while (true)
    {
        Sleep(300);

        if (curMode == MODE_NORMAL)
            handleModeNormal();
        else if (curMode == MODE_ARM)
            handleModeArm();
        else
            Debug.WriteLine("ERROR: Invalid Mode. Please check your logic");
    }
}
```

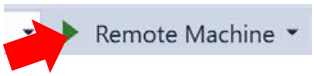
15. Click on the Drop Down button beside the Device and Select "**Remote Machine**" to configure the deployment settings



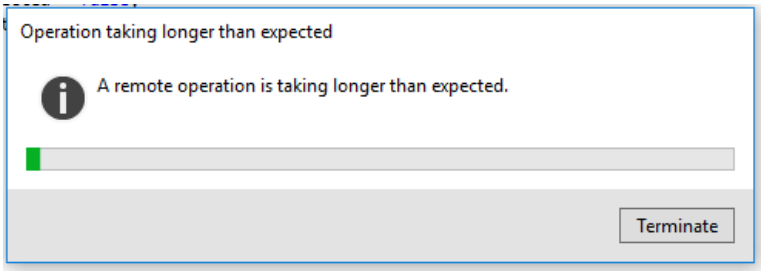
16. Key in the address "**169.254.8.88**" manually as shown below and click on "**Select**" after that. (if you didn't see this screen, refer to **Practical 1 Exercise 2** on steps to display this screen).



17. Click on Remote Machine to Run the Program.

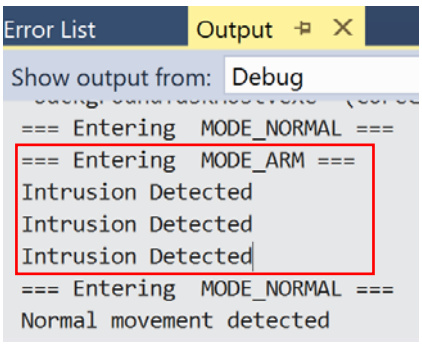


18. You may see the following warning but it is fine. Every time when you deploy a project for the first time on the hardware, it will take a long time to deploy. Just wait for a while and it should be deployed.



19. Once deployed successfully, the program will start at MODE_NORMAL. Wave your hand at the sensor. You should see the message “Normal movement detected” and no buzzer will sound.

    Press the button and device will enter into MODE_ARM. This mode is simulating no one at home and monitoring for intrusion. At this mode, any motion will trigger the buzzer. Now try waving at the sensor, you will hear the buzzer sound. In the Output, you should see the message “Intrusion Detected” as below.



    Press the button again and the device will go back to MODE_NORMAL. Waving your hand at the sensor, you should see message “Normal movement detected” and no buzzer will sound. You can stop the program at the MODE_NORMAL.

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

==End of Practical==