Nanyang Polytechnic

School of Information Technology

Course : Diploma in Infocomm & Security (ITDF12)

Module : Sensor Technologies and Project (ITP272)

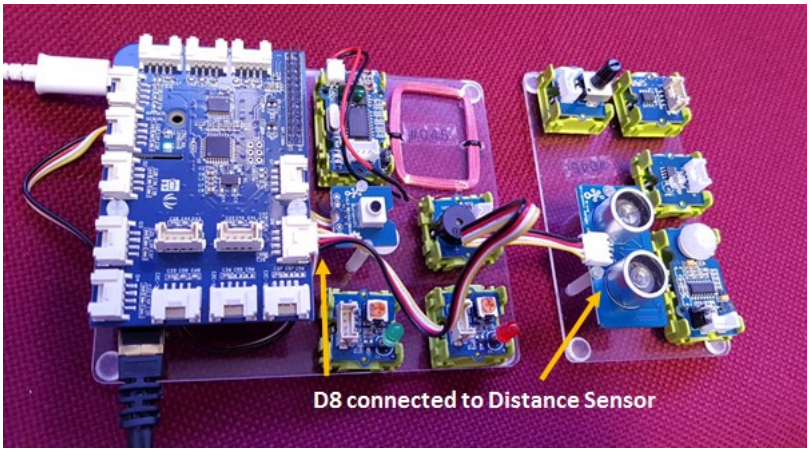
Raspberry Pi Practical : Programming Raspberry Pi with Grove Buzzer

Objectives:

- Learn how to interface with Digital I/O port
- Learn how to create program to control Grove Ultrasonic Ranger (Distance).
- Learn how to create program with state machines to work with Ultrasonic, buzzer and push button

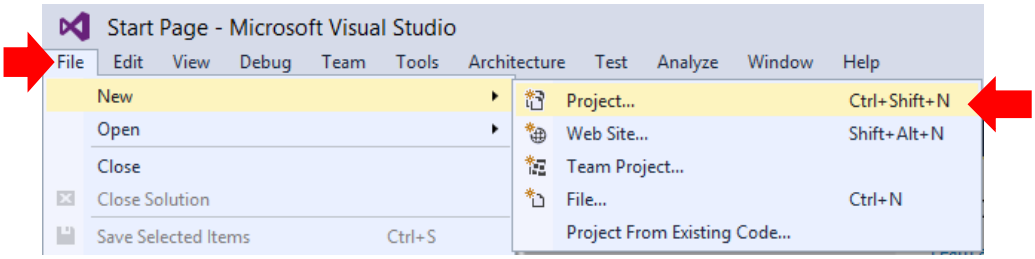
Exercise 1: Creating DistanceDetect

1. Today we will be creating a program that will measure the distance an object is from the sensor.
2. We shall use Digital Input port **D8** of the GrovePi for the Ultrasonic Distance Sensor. Ensure that you have port **D8** connected to the **Ultrasonic Sensor**.

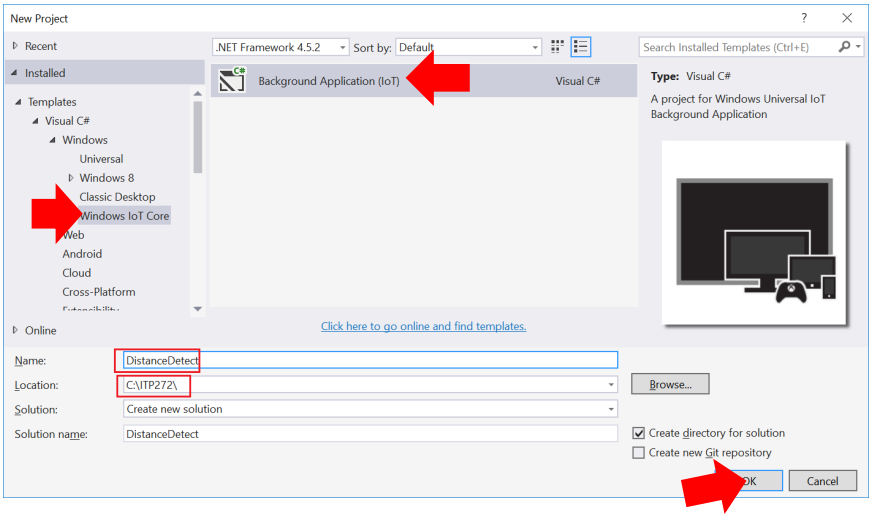


Ultrasonic Distance Sensor

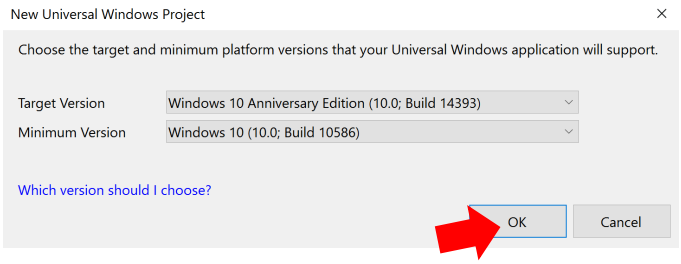
3. Start Visual Studio 2015, Click on File – New - Project.



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

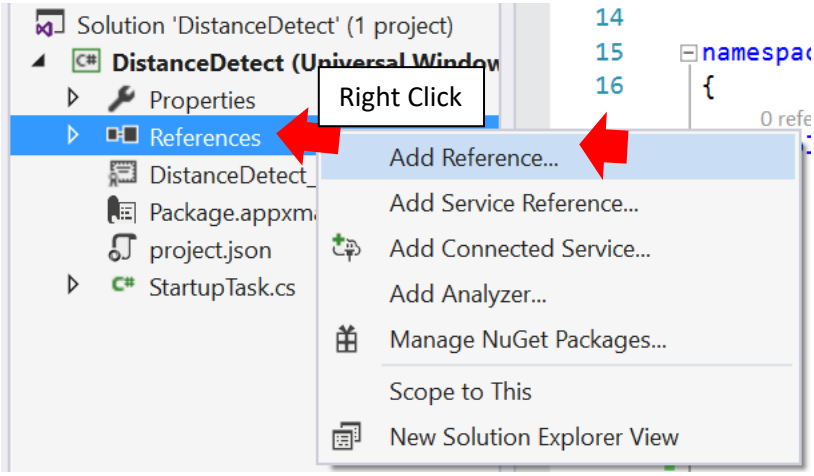


5. You should see this pop-up. Click OK.

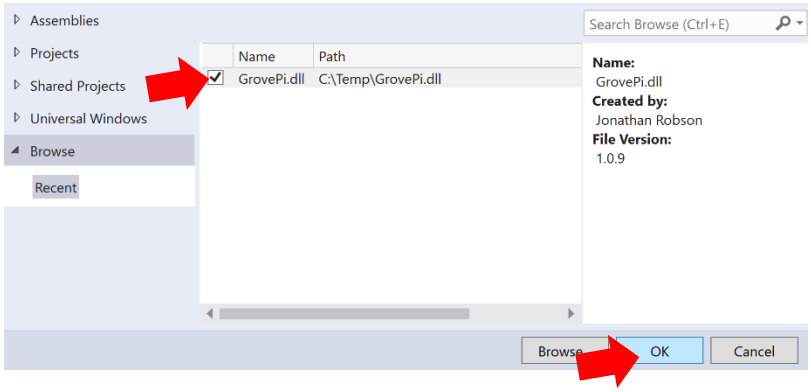


6. At the solution explorer on the right side of the screen, check that you have StartupTask.cs. This is the startup program file. Double Click on Startup Task.cs.

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

```
0 references
public sealed class StartupTask : IBackgroundTask
{
    //Use D8 for Ultrasonic distance sensor
    IUltrasonicRangerSensor sensor = DeviceFactory.Build.UltraSonicSensor(Pin.DigitalPin8);

    //used by sensor for internal processing
    private int distance = 400;

    //This is for main logic controller to check for distance
    int sensorDistance;

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

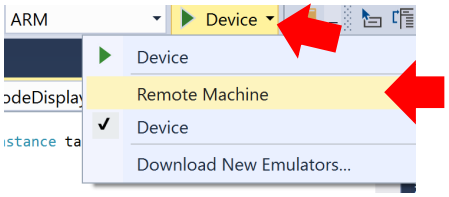
11. Create the **getDistance()** method below Sleep(). This method reads the distance from sensor and returns the value to the main program.

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

private int getDistance()
{
    // need to ensure you cover the correct FOV before distance is is reported correctly
    // better to cover with a big object like a file
    // will take some time to init before scanning
    int distanceRead = sensor.MeasureInCentimeters();
    if (distanceRead < 400 && distanceRead > 0)
        distance = distanceRead;
    return distance;
}//End of getDistance()
```

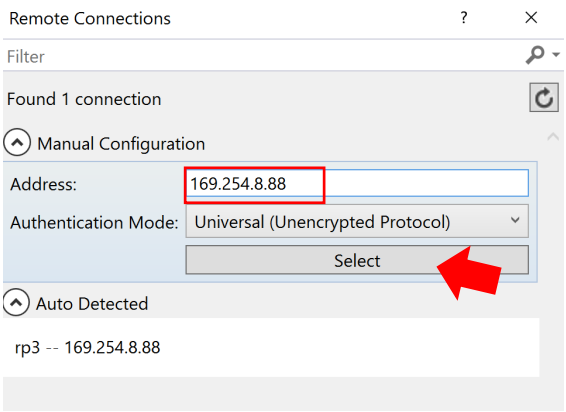
12. Finally, add the codes in the **Run()** method.

```
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
    //

    while (true)
    {
        Sleep(300);
        sensorDistance = getDistance();
        Debug.WriteLine("Sensor distance = " + sensorDistance);
        if (sensorDistance < 30)
            Debug.WriteLine("There is object nearby");
    }
}
```

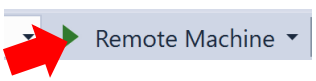
13. In order to deploy the program to your hardware, click on the Drop Down button beside the Device and Select Remote Machine.



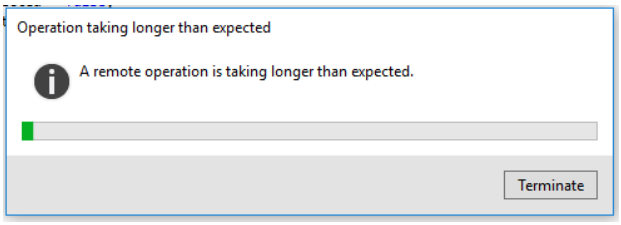
14. Key in the address manually like shown below. Click on **Select** after that.



15. Click on Remote Machine to Run the Program.

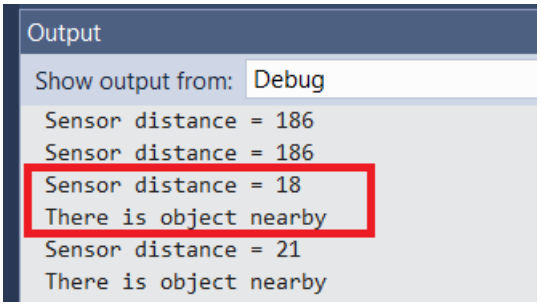


16. You may see the following warning but it is fine. Every time when you deploy a project for the first time on the hardware, it will take a long time to deploy. Just wait for a while and it should be deployed successfully.



17. Upon successful deployment, you should see the distance reported at the Output windows.

    A distance of 400 means the sensor has not been initialised yet. It takes some time before the sensor start reporting the distance. Put an object (like a file) around 20 cm from the sensor, you should see the distance being reported.



    You've just learnt how to get the distance from the Ultrasonic Distance sensor. In the **Run**() method, once you call the **getDistance**() method to read in the distance from the sensor and save the distance in the **sensorDistance** variable. The **Run**() method can check this **sensorDistance** variable to see if there is any object nearby the sensor.

Exercise 2: Creating a DistanceWarningAlarm

Implement a Home Gaming monitoring system. The system has a distance sensor placed near the Gaming Machine that could detect whether a player is sitting too close to the TV. The system is configured with 2 safety distance levels. A Warning level and an Alarm level. When the player is coming close to the TV reaching the Warning level, the system will emit a warning tone to warn the gamer. The warning tone will continue to sound until the guardian of the house hit the reset button. If the user continues further to close in to the TV to hit the Alarm level, it will emit an Alarm which also need to be cleared by the guardian. This system helps the guardian to monitor and prevent kids from staying too close to the TV.

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**

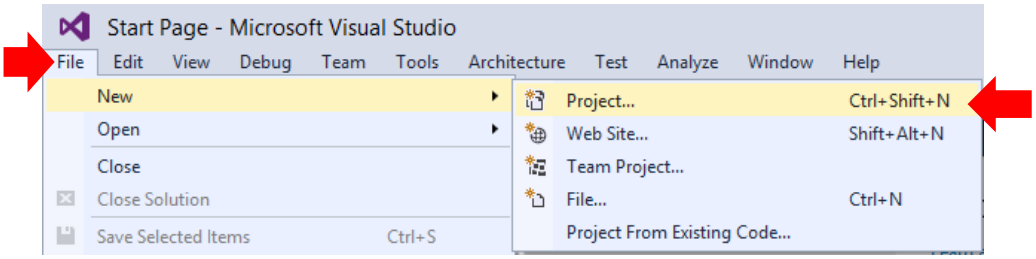
- **MODE_Normal**
  - Display any object's sensed and the distance from the sensor
  - If an object is within 15-25m away, change mode to **MODE_WARNING**
  - If an object is less than 15m away, change mode to **MODE_ALARM**

- **MODE_WARNING**
  - Emits a warning tone continuously to warn the player and as well inform the guardian until reset button is pressed
  - If the object is less than 15m away, change mode to **MODE_ALARM**
  - If reset button is pressed, change mode to back **MODE_Normal**

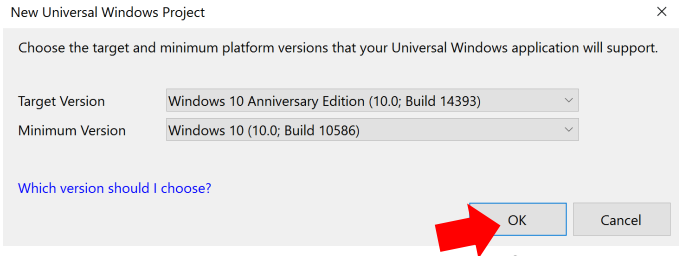
- **MODE_ALARM**
  - Emits an alarm continuously to warn the player and as well inform the guardian until reset button is pressed
  - If reset button is pressed, change mode to back **MODE_Normal**

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.

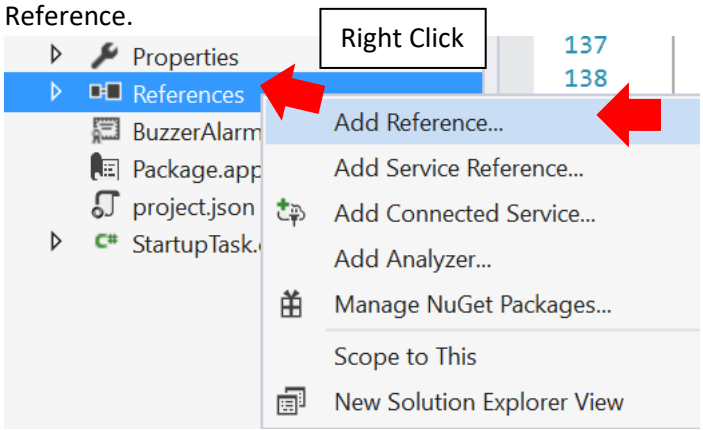
1. In this exercise, we will be creating a program **DistanceWarningAlarm** as stated in the previous page.
2. Add in a buzzer and Push button. Ensure that you have the following connections
    - D8 : Ultrasonic Distance Sensor
    - D3 : Buzzer
    - D4 : Push Button
3. Start Visual Studio 2015, Click on File – New - Project.



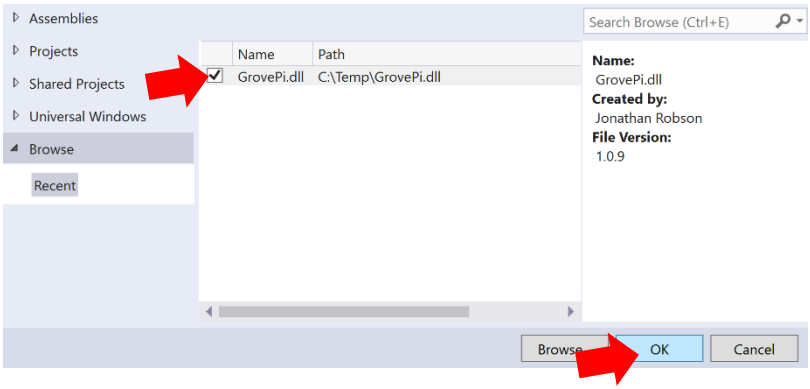
4. From the Installed Templates, select Visual C# – Windows IoT Core – Background Application. Name your project as **DistanceWarningAlarm**, save it in you ITP272 folder and Click OK.
5. You should see this pop-up. Click OK.



6. At the solution explorer on the right side of the screen, check that you have StartupTask.cs. This is the startup program file. Double Click on Startup Task.cs.
7. Add Reference. Right Click on References on Solution Explorer and Click on Add Reference.



8. Select GrovePi.dll as usual



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

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

// The Background Application template is documented at

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

10. Add in these codes in the StartupTask.cs

```
public sealed class StartupTask : IBackgroundTask
{
    //State Machine vairables to control different mode of operation
    const int MODE_NORMAL = 1;
    const int MODE_WARNING = 2;
    const int MODE_ALARM = 3;
    static int curMode;

    //Use D3 for buzzer, D4 for button and D8 for Ultrasonic distance sensor
    Pin buzzerPin = Pin.DigitalPin3;
    IButtonSensor button = DeviceFactory.Build.ButtonSensor(Pin.DigitalPin4);
    IUltrasonicRangerSensor sensor = DeviceFactory.Build.UltraSonicSensor(Pin.DigitalPin8);

    //used by sensor for internal processing
    private int distance = 400;

    //This is for main logic controller to check for distance or whether button is pressed
    private bool buttonPressed = false;
    private int sensorDistance;

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

11. Create a method for the **soundWarning**() and **soundAlarm**() below the Sleep().

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

```
private void soundWarning()
{
    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 soundWarning

private void soundAlarm()
{
    DeviceFactory.Build.GrovePi().AnalogWrite(buzzerPin, 200);
    Sleep(80);
    DeviceFactory.Build.GrovePi().AnalogWrite(buzzerPin, 240);
    Sleep(80);
    DeviceFactory.Build.GrovePi().AnalogWrite(buzzerPin, 190);
    Sleep(200);
    DeviceFactory.Build.GrovePi().AnalogWrite(buzzerPin, 0);
    Sleep(500);
}//End of soundAlarm
```

The **soundWarning**() is called to emit the warning tone and the **soundAlarm**() is called to emit the alarm.

12. Create the **startButtonMonitoring()** and **getDistance()** methods for the button and distance sensor below the **soundAlarm**()

```
}//End of soundAlarm
```

```
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 int getDistance()
{
    // need to ensure you cover the correct FOV before distance is is reported correctly
    // better to cover with a big object like a file
    // will take some time to init before scanning
    int distanceRead = sensor.MeasureInCentimeters();
    if (distanceRead < 400 && distanceRead > 0)
        distance = distanceRead;
    return distance;
}//End of getDistance()
```

The self-monitoring method, **startButtonMonitoring()**, will run by itself once it is called to monitor the button. It will update the button through the variable **buttonPressed**. Other parts of the program shall check on this **buttonPressed** to determine whether button was pressed. They need to always clear this variable **buttonPressed** to false after using it.

The **getDistance()** method needs to be called in the while loop in the main program to read distance from sensor. It then updates the variable **sensorDistance**. This variable **sensorDistance** will be used in other parts of the program to check for the distance.

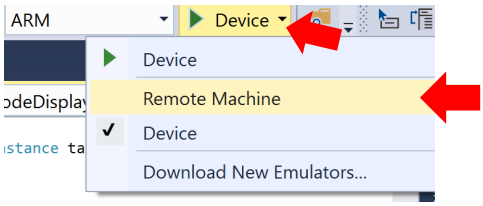
13. Create a handler methods for the operation of the various modes. Add the codes between the **getDistance()** and the **Run()** method.

```
}//End of getDistance()
private void handleModeNormal()
{
    //  1.  Define any Behaviour in this mode
    //Debug.WriteLine("Distance in safe range = "+distance);
    buttonPressed = false; //ignore any button pressed

    // 2.   Must write the condition to move on to other modes
    if (sensorDistance < 15)
    {
        //Move on to Mode Alarm when button
        curMode = MODE_ALARM;
        Debug.WriteLine("===Entering  MODE_ALARM===");
    }
    else if(sensorDistance < 25)
    {
        //Move on to Mode Warning when button
        curMode = MODE_WARNING;
        Debug.WriteLine("===Entering  MODE_WARNING===");
    }
}
private void handleModeWarning()
{
    //  1.  Define any Behaviour in this mode
    soundWarning();

    // 2.   Must write the condition to move on to other modes
    if (sensorDistance < 15)
    {
        //Move on to Mode Alarm object is too close for comfort
        curMode = MODE_ALARM;
        Debug.WriteLine("===Entering  MODE_ALARM===");
    }
    if(buttonPressed == true)
    {
        //reset on to Mode normal when button pressed
        curMode = MODE_NORMAL;
        Debug.WriteLine("===Entering  MODE_NORMAL===");
    }
}
private void handleModeAlarm()
{
    //  1.  Define any Behaviour in this mode
    soundAlarm();

    // 2.   Must write the condition to move on to other modes
    if (buttonPressed == true)
    {
        //reset on to Mode normal when button pressed
        curMode = MODE_NORMAL;
        Debug.WriteLine("===Entering  MODE_NORMAL===");
    }
}
```

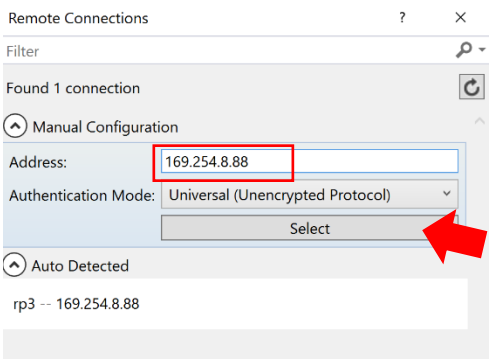
14. Lastly, add the codes for the **Run()** method.

```
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();

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

    while (true)
    {
        Sleep(300);
        sensorDistance = getDistance(); //get distance from sensor
        Debug.WriteLine("Sensor distance = " + sensorDistance);

        if (curMode == MODE_NORMAL)
            handleModeNormal();
        else if (curMode == MODE_WARNING)
            handleModeWarning();
        else if (curMode == MODE_ALARM)
            handleModeAlarm();
        else
            Debug.WriteLine("ERROR: Invalid Mode. Please check your logic");
    }
}
```

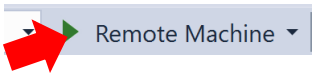
15. Click on the Drop Down button beside the Device and Select "**Remote Machine**" to configure the deployment settings



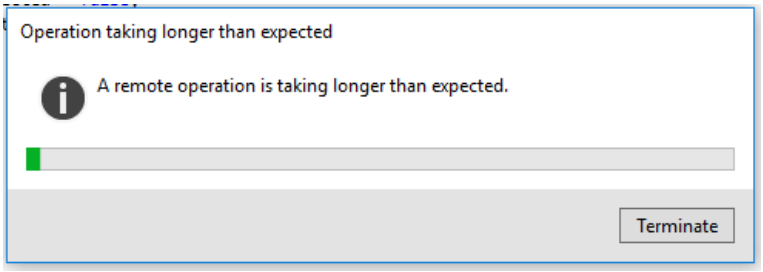
16. Key in the address "**169.254.8.88**" manually as shown below and click on "**Select**" after that. (if you didn't see this screen, refer to **Practical 1 Exercise 2** on steps to display this screen).



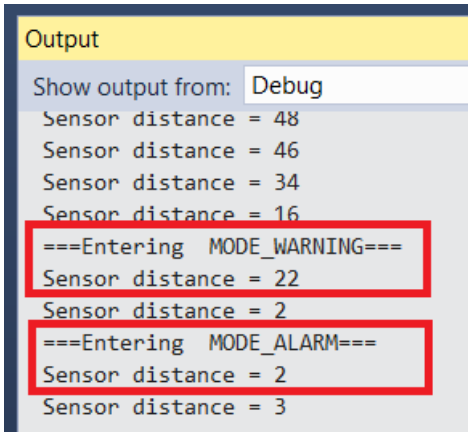
17. Click on Remote Machine to Run the Program.



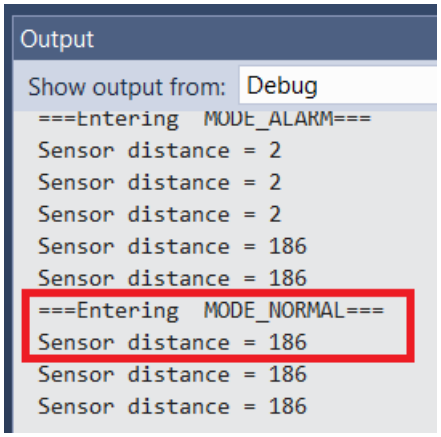
18. You may see the following warning but it is fine. Every time when you deploy a project for the first time on the hardware, it will take a long time to deploy. Just wait for a while and it should be deployed.



19. Once deployed successfully, the program will start at MODE_NORMAL. Put an object (like a book or file) around 20cm from the sensor, you should see the device goes to MODE_WARNING and emits a warning tone. If you move the object further closer to the sensor, it will goes into MODE_ALARM. Move the object away from the sensor now. The alarm continues to sound.



It will sound until the reset button (push button) is pressed. Now press the button and device will go back into MODE_NORMAL. You see that the alarm stop sounding.



Unguided Exercise: Creating a DistanceWarningCumHomeSecureSystem

Implement a monitoring system that has a distance sensor placed near the Gaming Machine that could detect whether a player is sitting too close to the TV. Under the normal mode, when the player is coming close to the TV (< 15cm), the system will emit a warning tone to warn the gamer. The warning tone will stop once the play move further away. Under this normal mode, the system will ignore any movement detected within the house.

There is a button to secure the house when everyone leave the house. When the button is pressed, the system will go to the Arm mode. In Arm mode, it ignores the distance sensor but it monitors for any movement. If there is movement, it will sound the alarm until no movement is detected. You can switch the Arm mode back to normal mode when you press the button again.

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**
  - Display any object's sensed and the distance from the sensor
  - Ignores any movement detected.
  - If an object is less than 15 cm away, change mode to **MODE_WARNING**
  - If button is pressed, change mode to **MODE_ARM**

- **MODE_WARNING**
  - Emits a warning tone to warn the player
  - If the object is more than or equal to15 cm away, change mode to **MODE_Normal**
  - If button is pressed, change mode to **MODE_ARM**

- **MODE_ARM**
  - Ignore any distance reported
  - If there is movement detected, change mode to **MODE_ALARM**
  - If button is pressed, change mode to **MODE_Normal**

- **MODE_ALARM**
  - Emits an alarm to warn the player.
  - If no movement is detected, change mode to **MODE_ARM**
  - If button is pressed, change mode to back **MODE_Normal**

==End of Practical==