Nanyang Polytechnic

School of Information Technology

Course : Diploma in Infocomm & Security (ITDF12)

Module : Sensor Technologies and Project (ITP272)

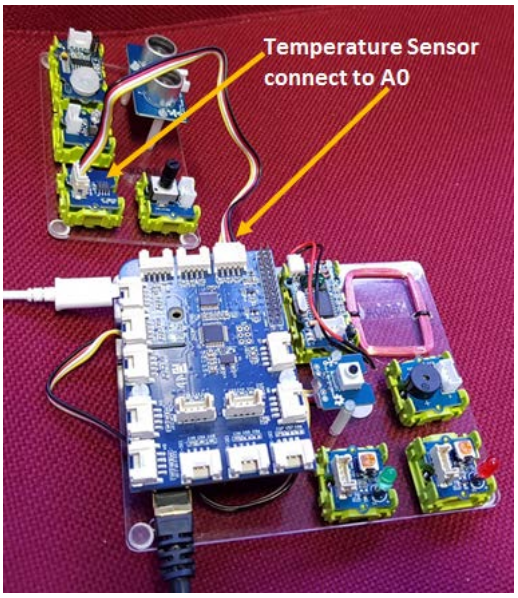
Raspberry Pi Practical : Programming Raspberry Pi with Grove Temperature

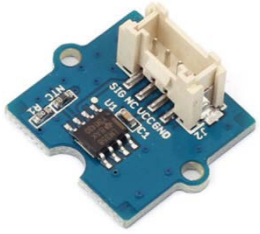
Objectives:

- Learn how to interface with Analog Input and use transfer function to derive temperature data
- Learn how to create program to interface with the Temperature Sensor.
- Learn how to create program with state machines to work with temperature, buzzer and push button

Exercise 1: Creating TempDetect

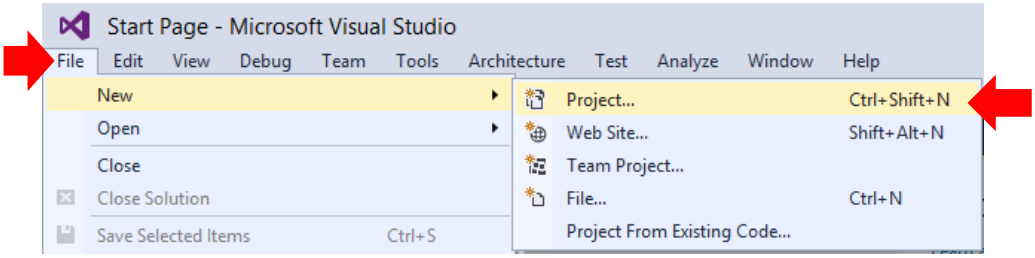
1. Today we will be creating a program that will read data from temperature sensor.
2. We shall use Analog port **A0** of the GrovePi for the Temperature. Ensure that you have port **A0** connected to the **Temperature Sensor**.





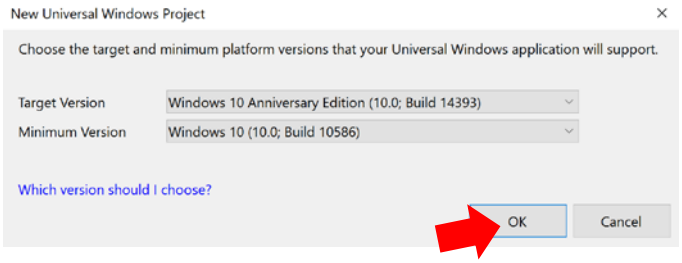
Temperature Sensor

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



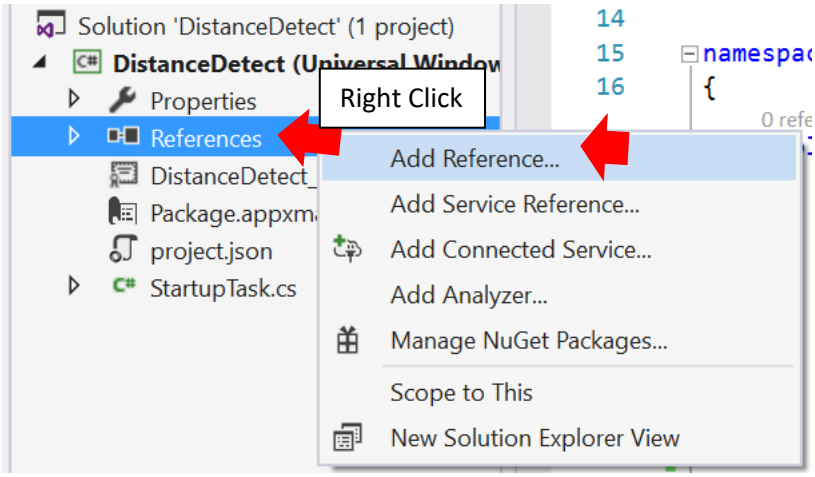
4. From the Installed Templates, select Visual C# – Windows IoT Core – Background Application. Name your project as **TempDetect**, save it in you ITP272 folder and Click OK.

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

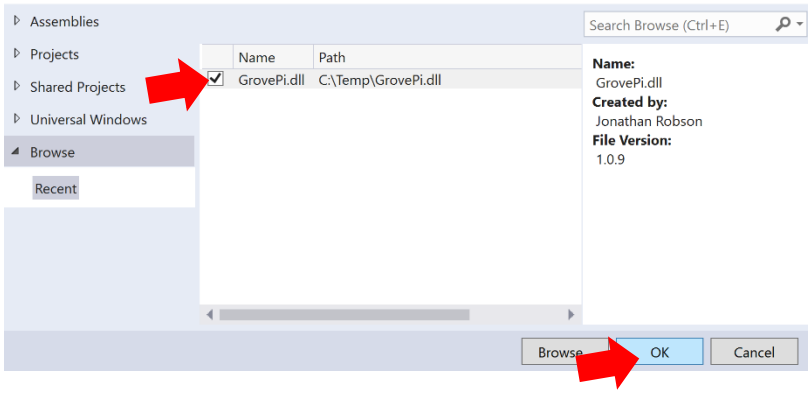


6. At the solution explorer on the right side of the screen, check that you have StartupTask.cs. This is the startup program file. Double Click on Startup Task.cs.

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. Add in the following namespace 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 do
using System.Diagnostics;
using System.Threading.Tasks;
using GrovePi;
using GrovePi.Sensors;

namespace TempDetect
```

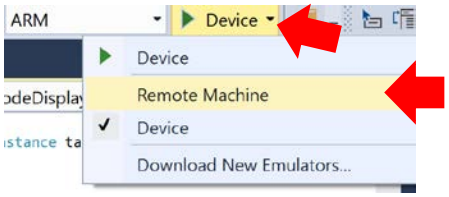
10. Add in these codes in the StartupTask.cs

```
public sealed class StartupTask : IBackgroundTask
{
    //Use A0 for Temp sensor
    Pin tempPin = Pin.AnalogPin0;

    double temp=23; ////This is for main logic controller to check for temp

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

    private async void startTempMonitoring()
    {
        await Task.Delay(100);
        int adcValue; double tempCalcuated = 0, R;

        while (true)
        {
            Sleep(1000);
            adcValue = DeviceFactory.Build.GrovePi().AnalogRead(tempPin);

            //Calculation explained in Lecture notes
            int B = 4250, R0 = 100000;
            R = 100000 * (1023.0 - adcValue) / adcValue;
            tempCalcuated = 1 / ( Math.Log(R/R0) / B + 1/298.15 ) - 273.15;
            if (!Double.IsNaN(tempCalcuated))
                temp = tempCalcuated;
        }
    }//End of startTempMonitoring()
```

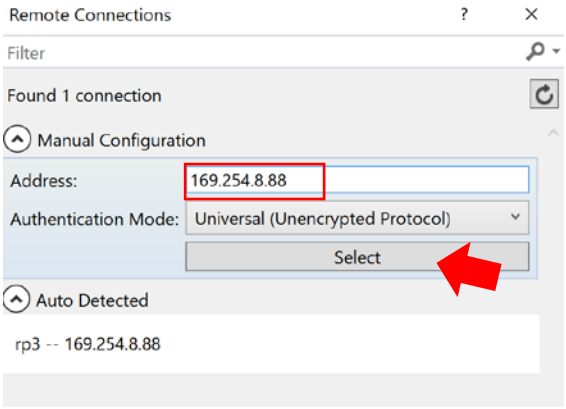
11. Finally, add the codes in the **Run()** method.

```
    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
        //

        //call the self monitoring method(s)
        startTempMonitoring();

        while (true)
        {
            Sleep(300);
            Debug.WriteLine("Temp in degrees Celsius is " +temp.ToString("N2"));
        }
    }
```

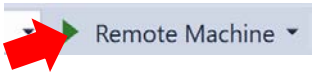
12. In order to deploy the program to your hardware, click on the Drop Down button beside the Device and Select Remote Machine.



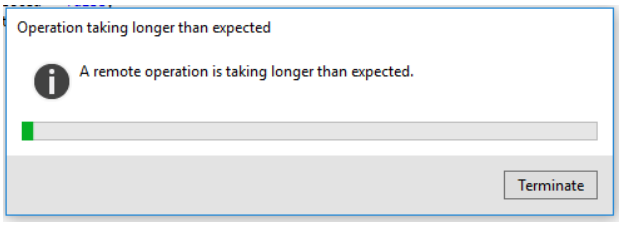
13. Key in the address manually like shown below. Click on **Select** after that.



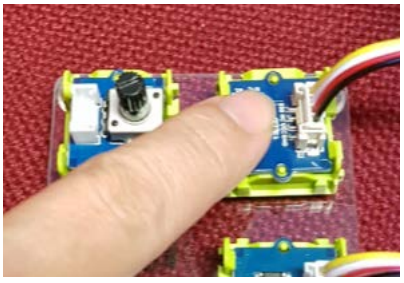
14. Click on Remote Machine to Run the Program.

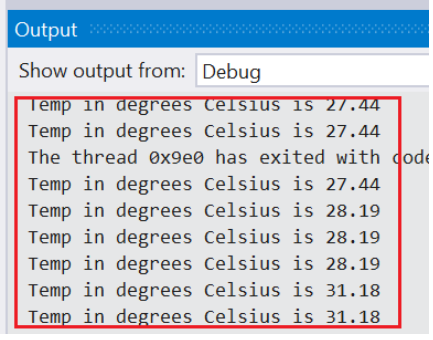


15. You may see the following warning but it is fine. Every time when you deploy a project for the first time on the hardware, it will take a long time to deploy. Just wait for a while and it should be deployed successfully.



16. Upon successful deployment, try rubbing your finger to raise it's temperature and then put it on top of your temperature sensor. You should see the temperature rising from the Output window.





Exercise 2: Creating a TempAlarm

Implement a simple alarm system. The system has a mode button used to determine whether you wish to monitor the temperature. Under normal mode, there will be no alarm regardless whether temperature is above set point or not. You can toggle the mode to monitor mode where the alarm will sound when temperature is above the set point.

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.

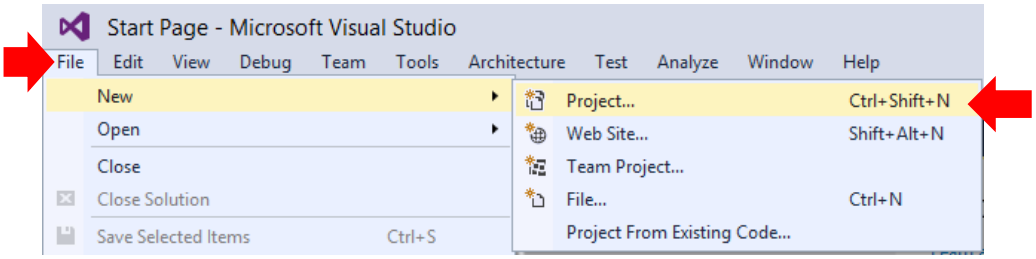
Your program starts in **MODE_Normal**

- **MODE_Normal**
  - Do not do anything.
  - If mode push button is pressed, change mode to **MODE_MONITOR**

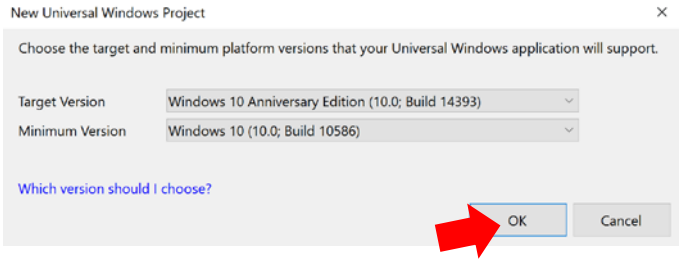
- **MODE_MONITOR**
  - Sound the Alarm temperature is above the set point
  - If mode push button is pressed, change mode to back **MODE_Normal**

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.

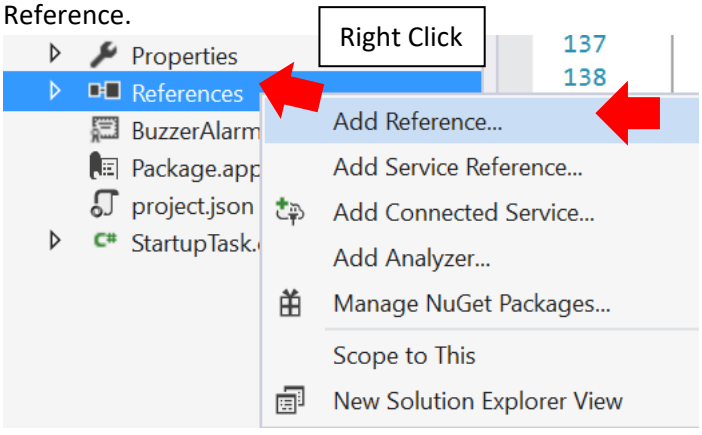
1. In this exercise, we will be creating a program **TempAlarm** as stated in the previous page.
2. Add in Push button and Buzzer. Ensure that you have the following connections
    - A0 : Temperature Sensor
    - D3 : Buzzer
    - D4 : Push Button
3. Start Visual Studio 2015, Click on File – New - Project.



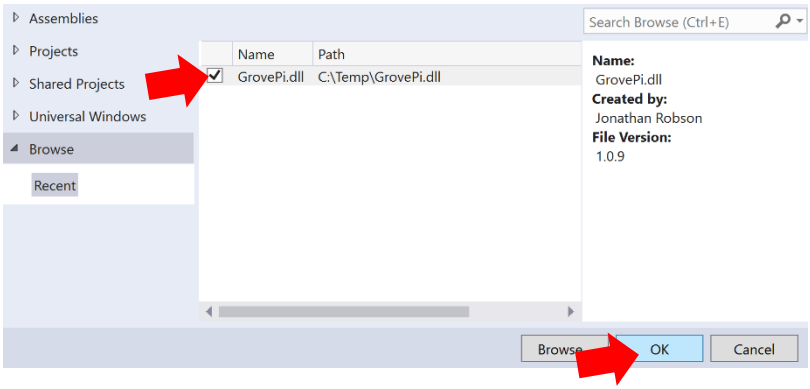
4. From the Installed Templates, select Visual C# – Windows IoT Core – Background Application. Name your project as **TempAlarm**, save it in you ITP272 folder and Click OK.
5. You should see this pop-up. Click OK.



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_MONITOR = 2;
    static int curMode;     //stores the current mode the program is at

    //We shall ase A0 for Temp sensor, D3 for Buzzer, D4 for button
    Pin tempPin = Pin.AnalogPin0;
    Pin buzzerPin = Pin.DigitalPin3;
    IButtonSensor button = DeviceFactory.Build.ButtonSensor(Pin.DigitalPin4);

    //This is for main logic controller to check status of sensor
    private bool buttonPressed = false;
    double temp = 23;

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

11. Create the soundBuzzer() and the Self-monitoring methods below the Sleep().

```
}//End Sleep()
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()

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()

private async void startTempMonitoring()
{
    await Task.Delay(100);
    int adcValue; double tempCalcuated = 0, R;

    while (true)
    {
        Sleep(1000);
        adcValue = DeviceFactory.Build.GrovePi().AnalogRead(tempPin);

        //Calculation explained in Lecture notes
        int B = 4250, R0 = 100000;
        R = 100000 * (1023.0 - adcValue) / adcValue;
        tempCalcuated = 1 / (Math.Log(R / R0) / B + 1 / 298.15) - 273.15;
        if (!Double.IsNaN(tempCalcuated))
            temp = tempCalcuated;
    }
}//End of startTempMonitoring()
```

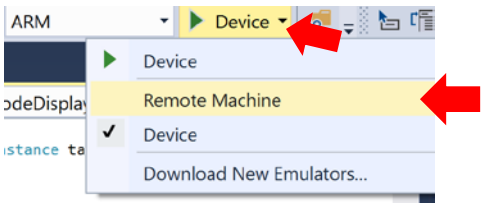
12. Create the handler methods for the various states

```
}//End of startTempMonitoring()
private void handleModeNormal()
{
    //  1.  Define Behaviour in this mode
    // do nothing

    // 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 Alarm when button is pressed
        curMode = MODE_MONITOR;
        Debug.WriteLine("===Entering  MODE_MONITOR===");
    }
}

private void handleModeMonitor()
{
    //  1.  Define Behaviour in this mode
    if (temp > 25)
    {
        Debug.WriteLine("High Temperature Alert!.");
        soundBuzzer();
    }

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

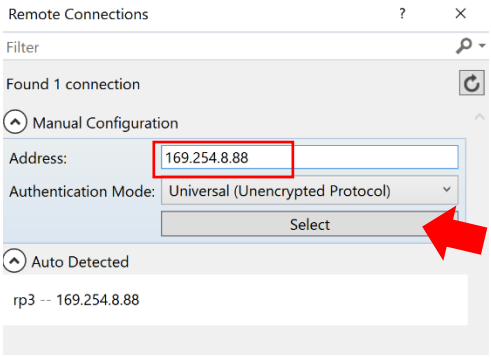
13. Lastly, add the codes for the **Run()** method.

```
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
    //

    //call the self monitoring method(s)
    startButtonMonitoring();
    startTempMonitoring();

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

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

        Debug.WriteLine("temp = " + temp.ToString("N2"));

        //state machine
        if (curMode == MODE_NORMAL)
            handleModeNormal();
        else if (curMode == MODE_MONITOR)
            handleModeMonitor();
        else
            Debug.WriteLine("ERROR: Invalid Mode. Please check your logic");
    }
}
```

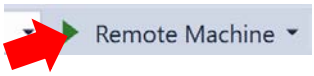
14. Click on the Drop Down button beside the Device and Select "**Remote Machine**" to configure the deployment settings



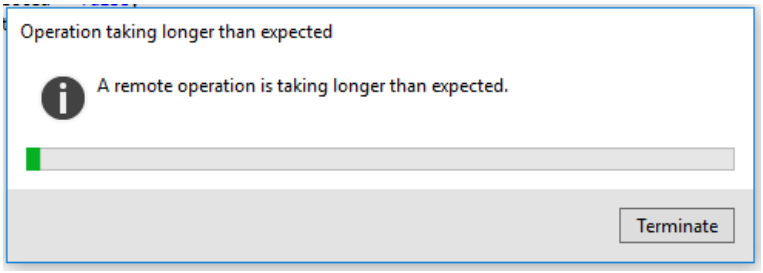
15. Key in the address "**169.254.8.88**" manually as shown below and click on "**Select**" after that. (if you didn't see this screen, refer to **Practical 1 Exercise 2** on steps to display this screen).



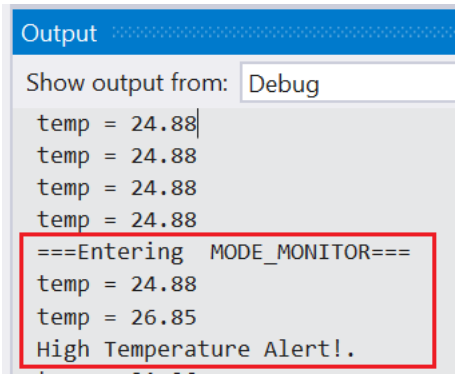
16. Click on Remote Machine to Run the Program.



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.



18. Once deployed successfully, the program will start in MODE_NORMAL. Try increasing the temperature and verify that no alarm is sounded. Press on the push button and check that the application goes to MODE. Now increase the temperature of the sensor above the set point, your alarm should sound and you should see the message "High Temperature Alert!" at the Output window.



==End of Practical==