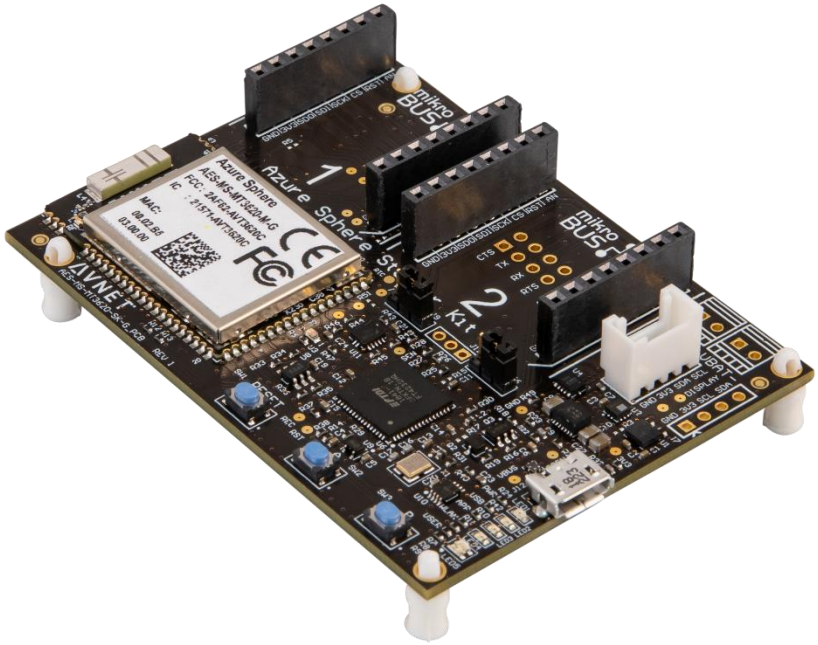
Avnet Technical Training Course

Azure Sphere: Running our First Application

Lab 2



Azure Sphere SDK:	19.05
Training Version:	v1
Date:	24 June 2019

Introduction

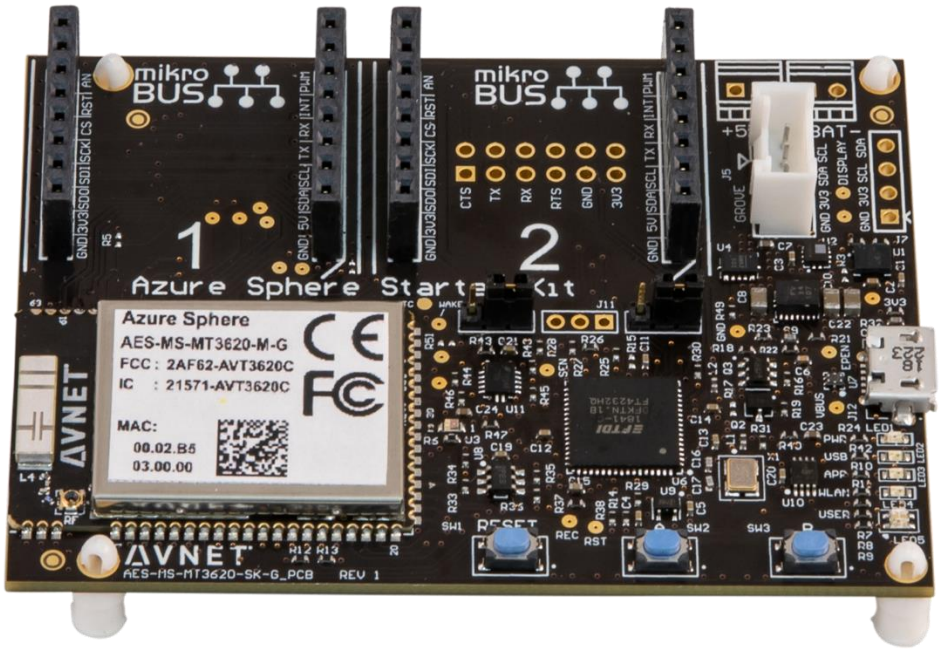
Lab 2 is where we start to run some Azure Sphere code! We'll pull a project from GitHub, build it, load it onto our Starter Kit then run it.

Avnet Azure Sphere Starter Kit Overview

The Avnet Azure Sphere Starter Kit from Avnet Electronics Marketing provides engineers with a complete system for prototyping and evaluating systems based on the MT3620 Azure Sphere device.

The Avnet Azure Sphere MT3620 Starter Kit supports rapid prototyping of highly secure, end-to-end IoT implementations using Microsoft's Azure Sphere. The small form-factor carrier board includes a production-ready MT3620 Sphere module with Wi-Fi connectivity, along with multiple expansion interfaces for easy integration of off-the-shelf sensors, displays, motors, relays, and more.

The Starter Kit includes Avnet's MT3620 Module. Having the module on the Starter Kit means that you can do all your development work for your IoT project on the Starter Kit and then easily migrate your Azure Sphere Application to your custom hardware design using Avnet's MT3620 Module.



Avnet Azure Sphere Starter Kit

Lab 2: Objectives

The objectives of Lab-2 are to pull code down from GitHub and run the application.

- Pull an Azure Sphere project down from GitHub
- Review the different build options in the project
- Build and run the project

Requirements

Hardware

- A PC running Windows 10 Anniversary Update or later (Version 1607 or greater)
- An unused USB port on the PC
- An Avnet Azure Sphere Starter Kit
- A micro USB cable to connect the Starter Kit to your PC

Software

- Visual Studio 2019 Enterprise, Professional, or Community version 16.04 or later; or Visual Studio 2017 version 15.9 or later **installed**
- Azure Sphere SDK 19.05 or the current SDK release **installed**

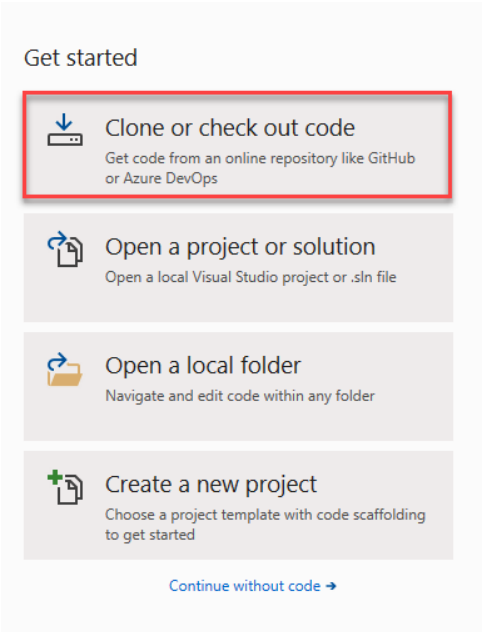
We've created an example project that reads the on-board ST sensors on the Starter Kit. The application implements the following features.

- Determine the BSSID address, Wi-Fi AP SSID and Wi-Fi Frequency
- Reads X, Y, Z accelerometer data from the onboard LSM6DSO device using the I2C Interface
- Reads X, Y, Z angular rate data from the onboard LSM6DSO device using the I2C Interface
- Reads the temperature from the onboard LSM6DSO device using the I2C Interface
- Reads the barometric pressure from the onboard LPS22HH device using the I2C Interface
- Reads the temperature from the onboard LPS22HH device using the I2C Interface
- Reads the state of the A and B buttons
- Send BSSID address, Wi-Fi AP SSID, Wi-Fi Frequency data, and application version to an IoT Hub or IoT Central Application
- Sends X, Y, Z accelerometer data to an IoT Hub or IoT Central Application
- Sends button state data to an IoT Hub or IoT Central Application
- Control user RGB LEDs from the cloud using device twin properties
- Control optional Relay Click relays from the cloud using device twin properties

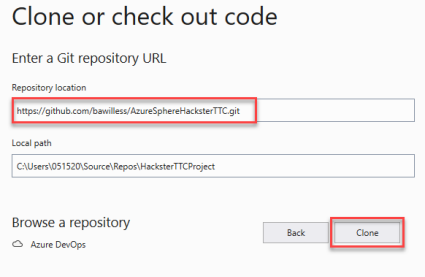
Pull Azure Sphere project down from GitHub

Let's get started by pulling the source from GitHub.

1. Open Visual Studio, I'll be using Visual Studio 2019. If you're using VS2017 you may see something different, but you should be able to find the same items in your application.



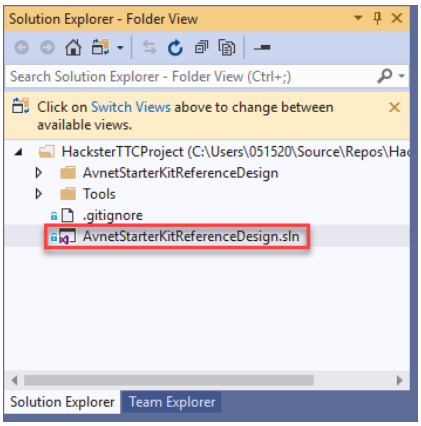
a. When the application opens you should see the "Get started" options. Select the "Clone or checkout code" option.



b. For the Repository location enter https://github.com/bawilless/AzureSphereHacksterTTC.git

c. Confirm the local path, or change it if you wish

d. Hit the Clone Button

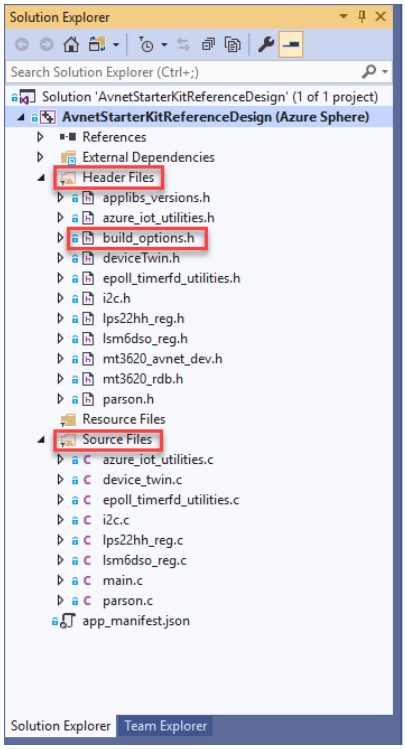


e. The project will download and the Solution Explorer will display the contents of the project.

f. Double click on the solution file AvnetStarterKitReferenceDesign.sln to open the VS solution.

g. Expand the Header Files and Source Files folders in the Solution Explorer window to see all the source files in the project.

Review the different build options in the project



The project has three different build options that are configured using #define pre-processor directives.

1. **No cloud connectivity**: This is the simplest configuration. The application runs and you can monitor sensor data from the Visual Studio debug window. This configuration is covered in this Lab.

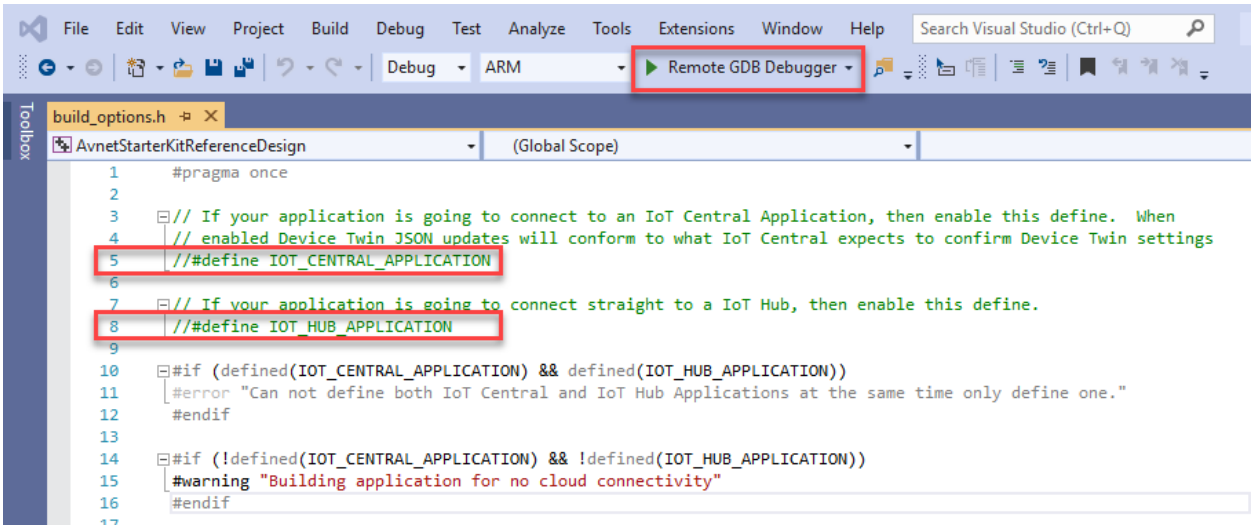
2. **IoT Hub connectivity**: This configuration requires an Azure IoT Hub and a Device Provisioning Service (DPS) in Azure. Once we have the IoT Hub and DPS configured, the application will connect to the IoT Hub and send telemetry data. We'll explore device twins and learn how we can use device twins to make application changes from the cloud. Last, we'll walk through how to provision a Time Series Insights (TSI) application to visualize the sensor data. This configuration is covered in Lab-4.

3. **IoT Central connectivity**: This configuration leverages Microsoft's IoT Central SaaS product. IoT Central provides a simple interface that allows users to visualize telemetry and to control connected devices. This configuration is covered in Lab-5.

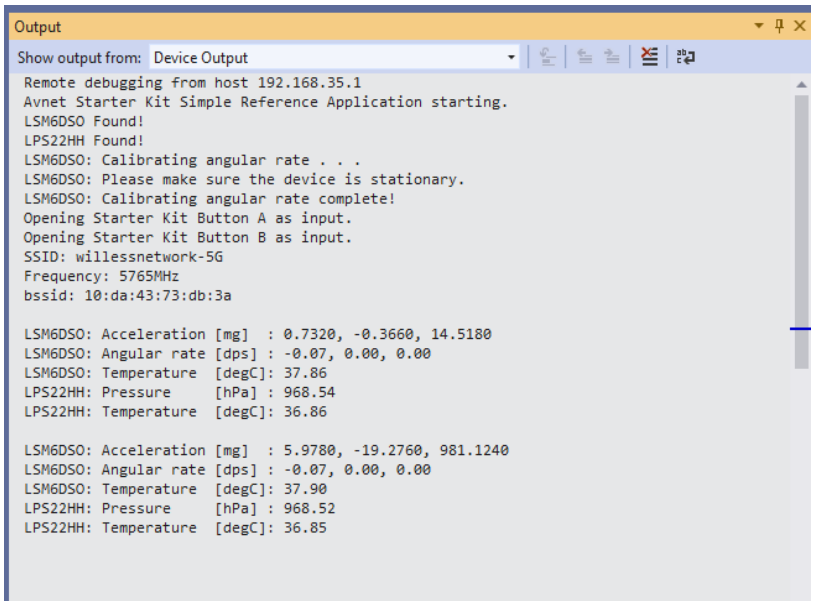
For Lab 2, we're going to look at the non-connected configuration. The build configurations are defined in the build_options.h file. When we look at the other configurations I'll describe the differences between the two "connected" build configurations. The non-connected configuration excludes code that connects to Azure and sends data to the cloud.

- Open the build_options.h file and confirm that both the #defines for IOT_CENTRAL_APPLICATION and IOT_HUB_APPLICATION are both commented out. Look around lines 5-8 for the #define pre-processor directives. This will build the no cloud connectivity configuration.

Build and run the project



Now we can run the application. Confirm that your Avnet Starter Kit is connected to your PC and click on the "Remote GDB Debugger" button at the top of the VS application. If prompted to save your files click "OK."



a. VS will build, link, side load, and run your application on the Sphere Starter Kit, all with just one click!

b. You should see your Output window open and debug will start to scroll in the window.

c. Pick you Starter Kit up and rotate it along different axis's and observe the Acceleration and Angular rates change as you move the device.

d. You can also press and release the A and B user buttons on the Starter Kit. The application will detect the button events and output a debug message. Be careful not the press the reset button closest to the Avnet Azure Sphere Module!

That's all there is to running an Azure Sphere application from Visual Studio.

Wrap Up

In this Lab we learned a little more about the Azure Sphere development environment

- How to pull a project from GitHub
- How to configure the example application for the “non-connected” build
- How to build, load and run the application

Revision History

Date	Version	Revision
25 June 19	01	Preliminary release