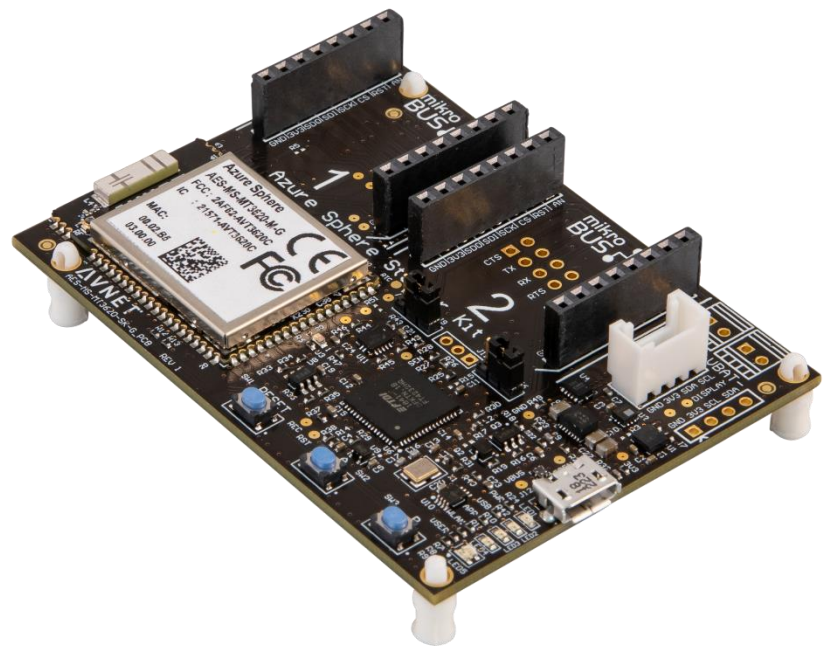


# Avnet Technical Training Course

## Azure Sphere: Running our First Application Lab 2



Azure Sphere SDK:	19.11
Training Version:	v5
Date:	18 January 2020

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

## Introduction

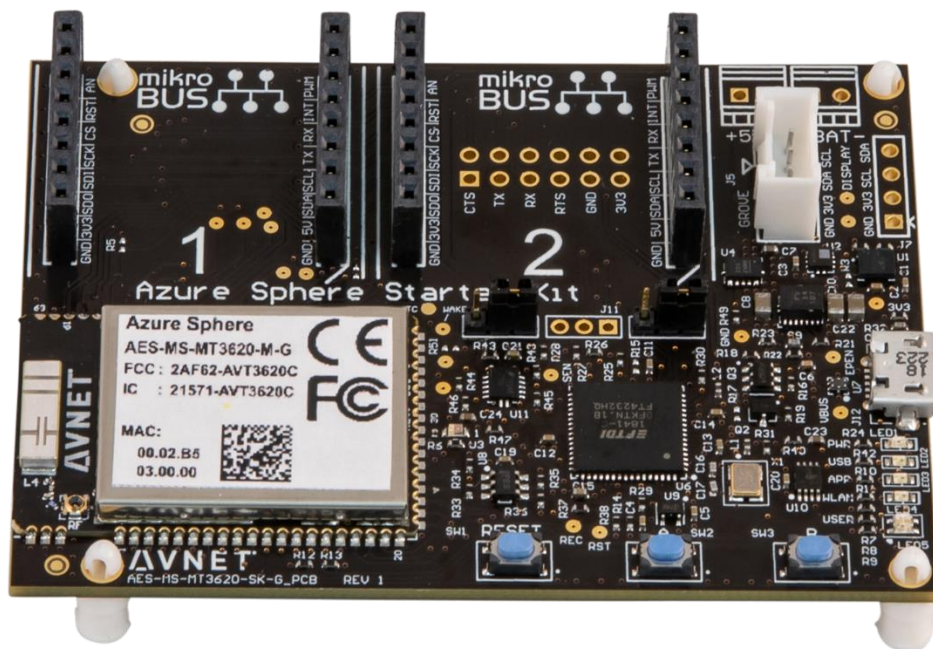
Lab 2 is where we start to run some Azure Sphere code! We'll pull a project from GitHub, build it, load it onto our Starter Kit then run it.

## 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 version 16.04 or later (Enterprise, Professional, or Community version); **or** Visual Studio 2017 version 15.9 or later, **installed** on PC
- Azure Sphere SDK 19.05 or the current SDK release **installed**

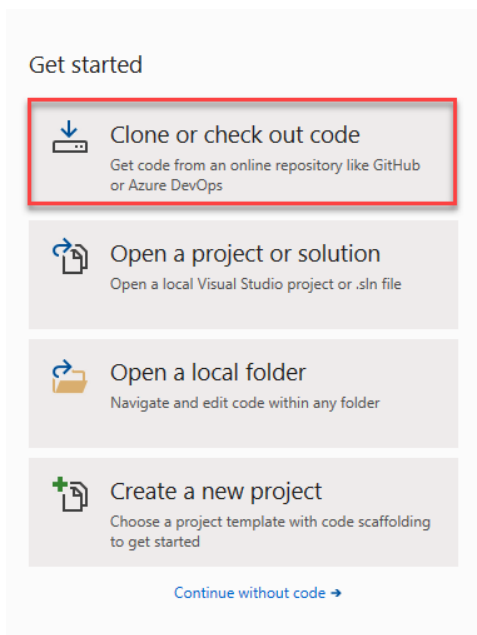
We've created an example project that reads the on-board ST sensors on the Starter Kit. The application implements the following features.

- 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 “Get started” options. Select “**Clone or checkout code**”

## Clone or check out code

Enter a Git repository URL

Repository location

Local path

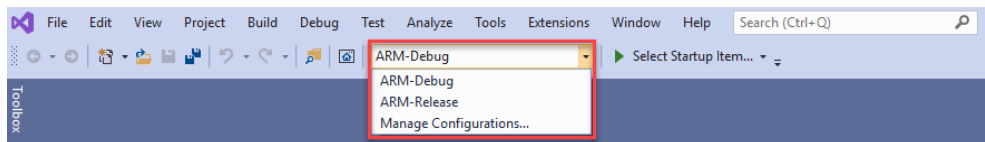
Browse a repository

Azure DevOps

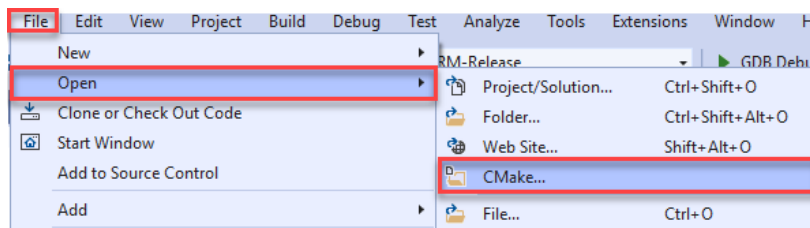
- For the **Repository location** enter  
<https://github.com/Avnet/AzureSphereHacksterTTC.git>
- Confirm the **Local path**, or change this if you wish
- Hit the **Clone** button
- The project will download and the Solution Explorer will display the contents of the project.

## Open the CMAKE Project

After the clone completes, the CMAKE project should open automatically. You'll know this is the case if you see the two different build targets listed in the target selection control.



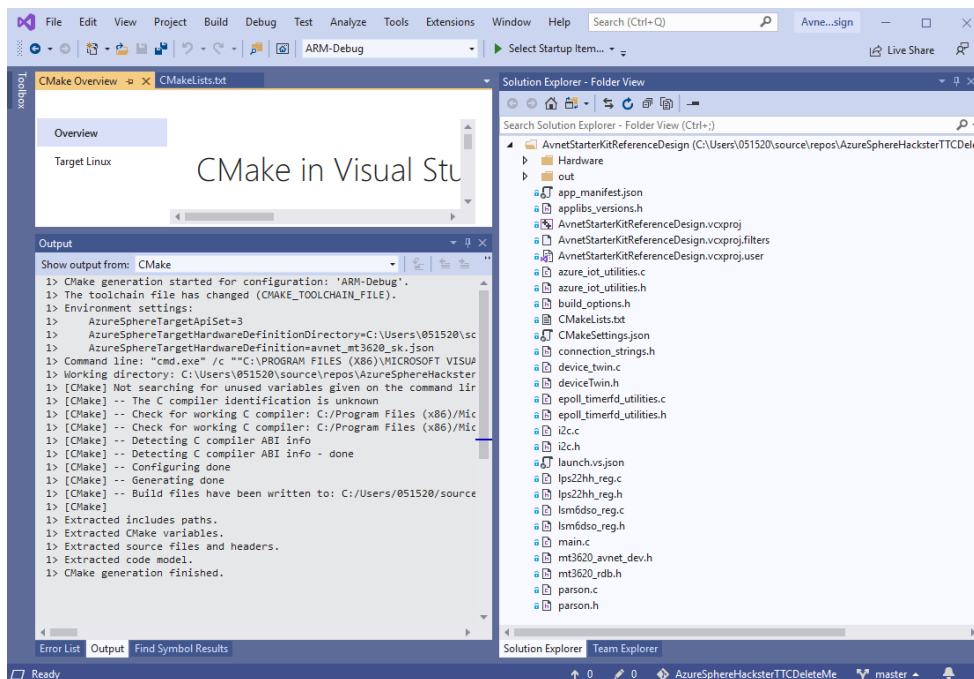
- If the project did not open, then just open a CMake project.  
 ie. Select **File** → **Open** → **CMake...**



- Navigate to the project folder and open the **CMakeLists.txt** file.

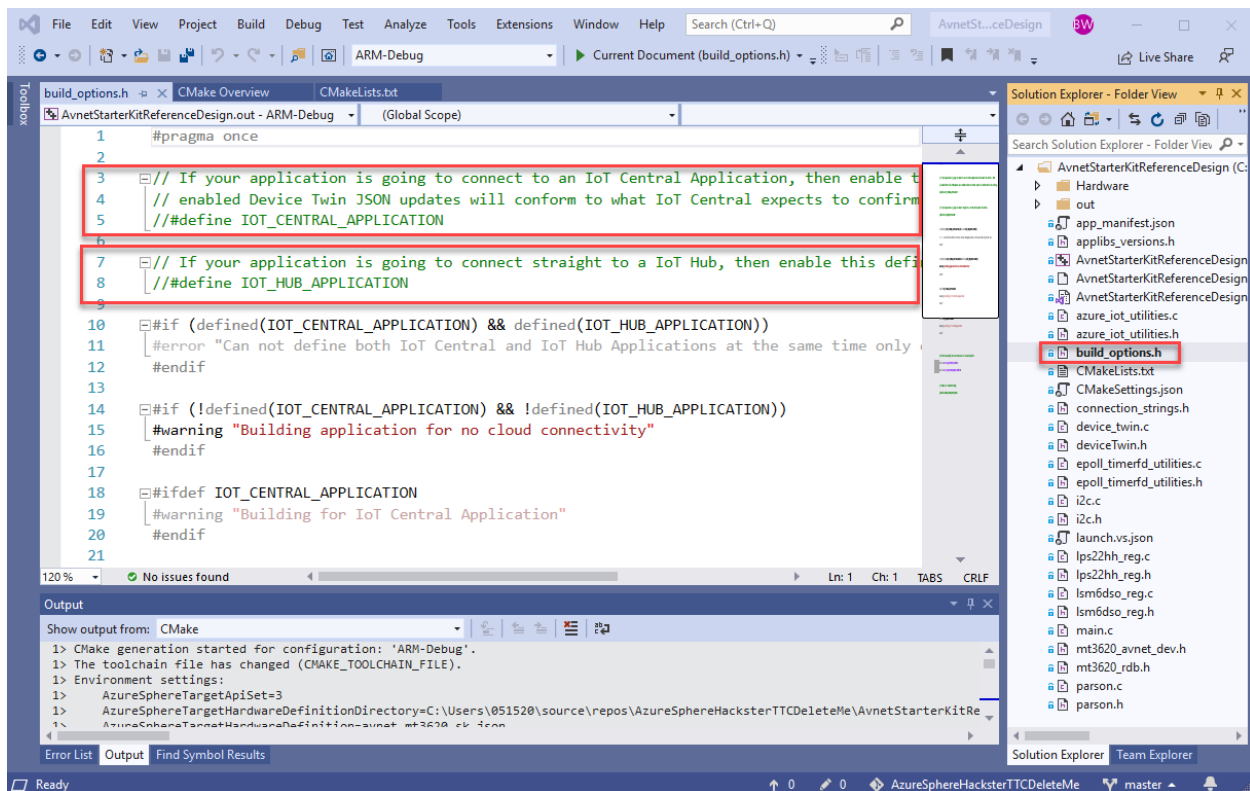
Name	Date modified	Type	Size
.vs	12/23/2019 2:42 PM	File folder	
Hardware	12/23/2019 2:08 PM	File folder	
CMakeLists	12/23/2019 10:39 ...	Text Document	1 KB

The open project will list all the source files in the Solution Explorer window



## Review the different build options in the project

The project has three different build options that are configured using #define pre-processor directives. The build is configured in the **build\_options.h** header file. Open **build\_options.h** to see the different options.

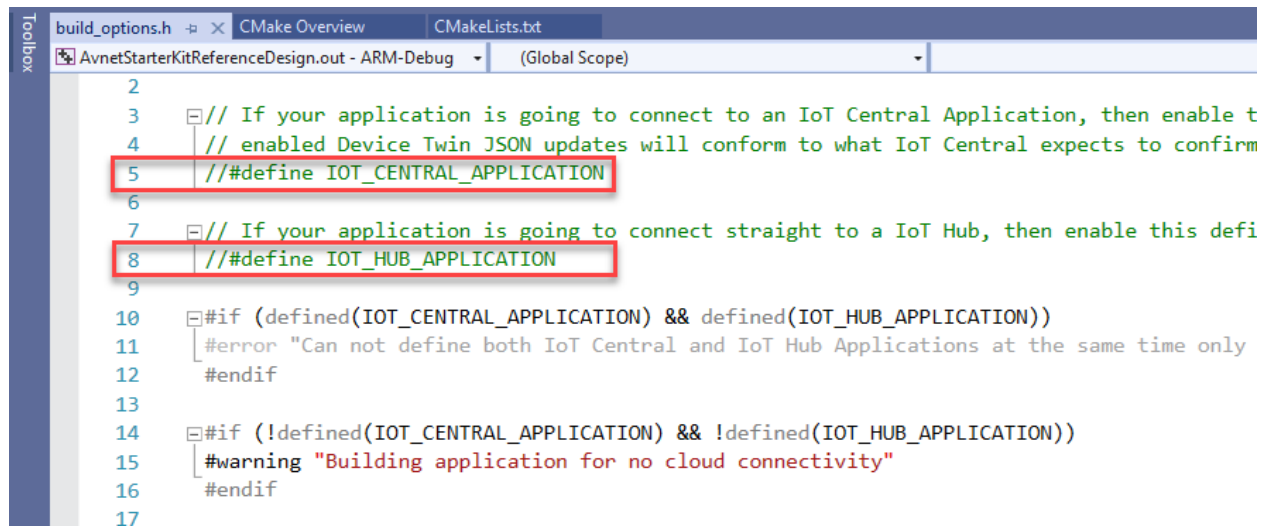


1. **No cloud connectivity:** This is the simplest configuration. The application runs and you can monitor sensor data from the Visual Studio debug window. This configuration is covered in this Lab, Lab-2.
  1. `IOT_CENTRAL_APPLICATION` (not defined)
  2. `IOT_HUB_APPLICATION` (not defined)
2. **IoT 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.
  1. `IOT_CENTRAL_APPLICATION` (not defined)
  2. `IOT_HUB_APPLICATION` (**defined**)
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.
  1. `IOT_CENTRAL_APPLICATION` (**defined**)
  2. `IOT_HUB_APPLICATION` (not defined)

For Lab 2, we're going to work with the non-connected configuration. The non-connected configuration excludes code that connects to Azure and sends data to the cloud.

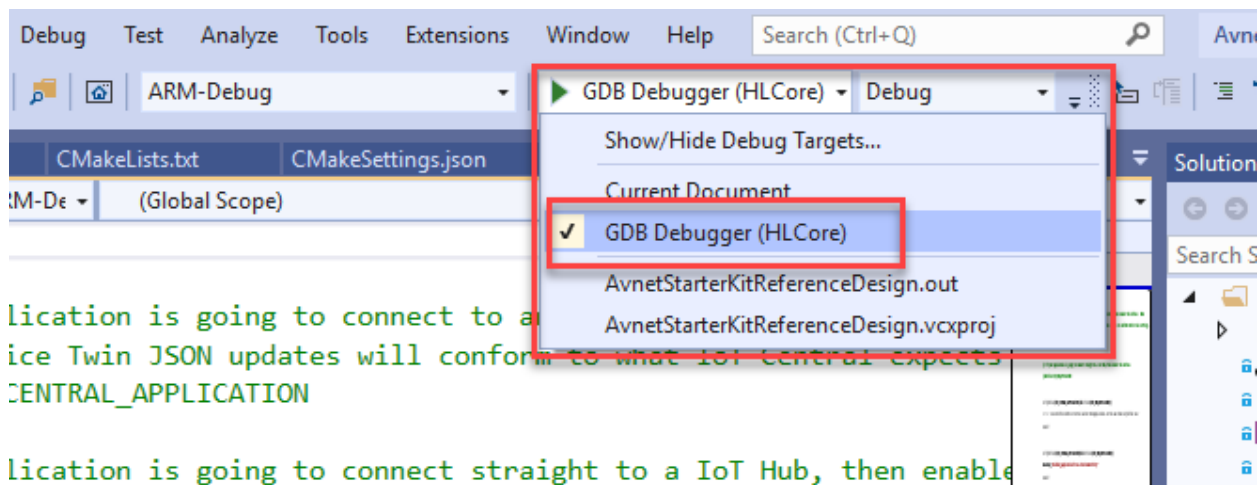
- Open the `build_options.h` file and confirm that both the `#defines` for `IOT_CENTRAL_APPLICATION` and `IOT_HUB_APPLICATION` are both commented out. Look around lines 5-8 for the `#define` pre-processor directives. This will build the no cloud connectivity configuration.

## Build and run the project

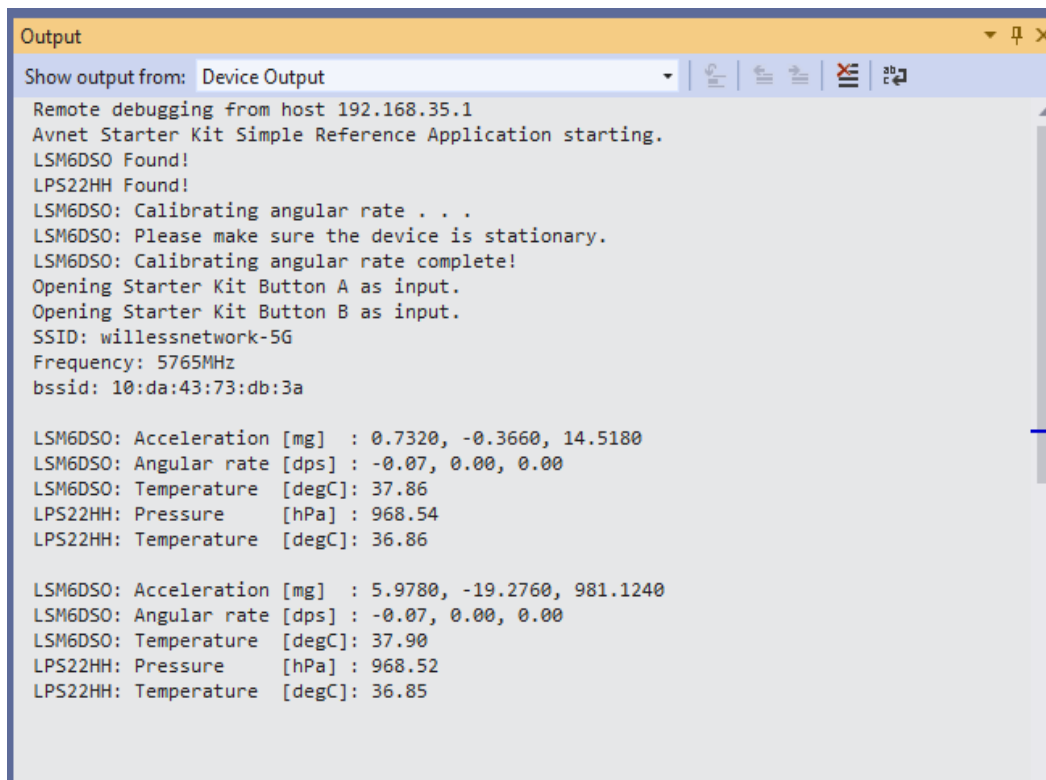


```
2
3 // If your application is going to connect to an IoT Central Application, then enable t
4 // enabled Device Twin JSON updates will conform to what IoT Central expects to confirm
5 // #define IOT_CENTRAL_APPLICATION
6
7 // If your application is going to connect straight to a IoT Hub, then enable this defi
8 // #define IOT_HUB_APPLICATION
9
10 #if (defined(IOT_CENTRAL_APPLICATION) && defined(IOT_HUB_APPLICATION))
11     #error "Can not define both IoT Central and IoT Hub Applications at the same time only
12     #endif
13
14 #if (!defined(IOT_CENTRAL_APPLICATION) && !defined(IOT_HUB_APPLICATION))
15     #warning "Building application for no cloud connectivity"
16     #endif
17
```

Now we can run the application. Confirm that your Avnet Starter Kit is connected to your PC select “GDB Debugger (HL Core)”, then click on the “GDB Debugger (HL Core)” button at the top of the VS application. If prompted to save your files click “OK.”







```
Output
Show output from: Device Output
Remote debugging from host 192.168.35.1
Avnet Starter Kit Simple Reference Application starting.
LSM6DSO Found!
LPS22HH Found!
LSM6DSO: Calibrating angular rate . . .
LSM6DSO: Please make sure the device is stationary.
LSM6DSO: Calibrating angular rate complete!
Opening Starter Kit Button A as input.
Opening Starter Kit Button B as input.
SSID: willessnetwork-5G
Frequency: 5765MHz
bssid: 10:da:43:73:db:3a

LSM6DSO: Acceleration [mg] : 0.7320, -0.3660, 14.5180
LSM6DSO: Angular rate [dps] : -0.07, 0.00, 0.00
LSM6DSO: Temperature [degC]: 37.86
LPS22HH: Pressure [hPa] : 968.54
LPS22HH: Temperature [degC]: 36.86

LSM6DSO: Acceleration [mg] : 5.9780, -19.2760, 981.1240
LSM6DSO: Angular rate [dps] : -0.07, 0.00, 0.00
LSM6DSO: Temperature [degC]: 37.90
LPS22HH: Pressure [hPa] : 968.52
LPS22HH: Temperature [degC]: 36.85
```

- VS will build, link, side load, and run your application on the Sphere Starter Kit, all with just one click!
- You should see your Output window open and debug will start to scroll in the window.
- Pick-up your Starter Kit up and slowly rotate in different planes to observe the Acceleration and Angular rates change as you move the device.
- If you press and release the A and B user buttons on the Starter Kit the application will detect the button events and output applicable debug messages.  
Note! Be careful not to press the RESET button! (ie. the button closest to the Sphere module)
- That's all there is to running an Azure Sphere application from Visual Studio.

## Developer Resources

In this section I'll introduce a few of the resources that are currently available to Azure Sphere developers.

### Microsoft Example Applications

One of the really great things about developing applications with Azure Sphere is that you have the Microsoft Azure Sphere engineering team behind you. This team develops example applications that can help you get a jump start on your Azure Sphere implementation. Each time this team releases a new Azure Sphere feature, you'll find an example application on GitHub that demonstrates how to use the feature. These examples focus on the featured functionality, so it's easy to see all the code necessary to implement each feature. This is a really great resource for Azure Sphere developers. The Microsoft Azure Sphere examples are available on GitHub [here](#).

## Hardware Abstraction Layer

Beginning with the 19.05 release, all the Microsoft GitHub example applications implement a hardware abstraction layer that allows the examples to work with any of the current Azure Sphere devices, modules, or development kits. Note that all the example applications default to use the Seeed Azure Sphere Development Kit, so you'll need to modify the project to target the Avnet Azure Sphere Starter Kit if you're using the Avnet Starter Kit. All the details to change the target hardware are documented by Microsoft and can be found [here](#). Lab 3 of this Technical Training Course contains more details on this feature.

Note that currently (12/24/19), none of the Ethernet examples on GitHub will work with the Avnet Azure Sphere Starter Kit. To use Ethernet on any Azure Sphere platform, you must side-load a board configuration image package that will enable the device to use the Ethernet interface. This procedure is documented [here](#). Currently, the only board configuration package released by Microsoft targets SPI interface isu0, and GPIO5 for the required interrupt signal. These interfaces are NOT available on the Starter Kit Click sockets.

## Avnet Example Applications

In addition to the Microsoft examples Avnet has been developing example projects and provides blogs to show you how to use them. Instead of listing the current blogs, I've created a blog that captures all the blogs! The "blog to rule all blogs" can be found [here](#).

This lab is also available in an online blog at:  
<http://avnet.me/mt3620-kit-OOB-ref-design-blog>

Instructor-led walkthrough of this lab is also available in **Lesson 3** of the **Azure Sphere Technical Training Course** on the Hackster.io website (but this not yet updated to a CMake project)  
<https://www.hackster.io/workshops/azure-sphere>

## 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
- Where to find additional Azure Sphere developer resources

## Revision History

Date	Version	Revision
25 Jun 2019	01	Preliminary release
05 Sep 2019	02	Added developer resources section
17 Dec 2019	03	Updated GitHub URL to point to new repo location
23 Dec 2019	04	Updates for CMAKE builds and 19.11
18 Jan 2020	05	Clean-up and readability edits