Hi, I'm Alan Hawse. Welcome back to Amazon FreeRTOS 101 with the Cypress 43907 and 54907 Wi-Fi development kits.

In this video, I'm going to talk about the application folder structure and how to debug AFR examples in WICED Studio.

Project files for WICED Studio are available in the Amazon FreeRTOS folder that you downloaded from Git. Here's the path:

amazon-freertos/projects/cypress/CYW943907AEVAL1F/wicedstudio/aws_demos/

These files are mapped to the root folder inside of the WICED Studio IDE project explorer window.

There are multiple demos that we include with the AWS code. By default, the MQTT_DEMO is enabled, but you can change the demo which ever one you like from a list.

To select a demo, first go to the project explorer and head to config_files/aws_demo_config.h.

Change the #define for CONFIG_MQTT_DEMO_ENABLED to any demo that you want to try. Only one demo can be enabled at a time, but no one will stop you if you want to combine two demos into one.

I have to say that I'm not, oh what you would say, always the best at following the rules... in fact the other day my wife was yelling at me...imaging that. Alan!!!! Why is it that you think the rules only apply to other people...

Anyway... do whatever you want to... I always do.

The file demos/include/iot_demo_runner.h maps the demo entry function to the corresponding function for the demo that you selected. In this case, the demo function is "RunMqttDemo". We will look at that function a bit later in this video.

The complete WICED Studio SDK is available under the amazon-freertos/vendors/cypress/WICED_SDK folder.

The SDK is linked into the project explorer under aws_demos/vendors/cypress/WICED_SDK.

Note that even if you use the latest WICED Studio IDE, Amazon FreeRTOS will still use the SDK from the amazon-freertos path so that everything works together properly.

Now let's look at the example code to see how it's structured. Application_code/main.c contains main() which is … get this.... the main entry point for the application. Main performs the hardware initialization that should be done prior to starting FreeROTS, it creates a logging task and then stars the FreeRTOS scheduler. After the scheduler is started, the application startup hook is invoked from the RTOS tick timer callback the first time it's called.

vApplicationDaemonTaskStartupHook() contains hardware initialization including initializing the Wi-Fi, generating a random seed, and doing the AWS certificate and private key provisioning, and then it starts the Demo task.

I said in the last video... and I'll say it again here... security matters... and we've made it so that you don't have any excuse not to have good security in your IoT devices.

Note that the pointer to the function that corresponds to the demo that you selected in the aws_demo_config.h file is passed as an argument to the task creation. In this case, it is mapped to RunMqttDemo. The same structure also passes in the network type which is Wi-Fi in this case, and it passes callbacks for connection and disconnection.

The runDemoTask first initializes and checks for the right interface. In our case that interface is Wi-Fi. More specifically, it's my kick butt 43907 Wi-Fi... look … it's easy to make Wi-Fi work on your desk... or in your lab... but to make Wi-Fi work well in the real world – that's a whole different story. The fastest way to get a boatload of bad product reviews on Amazon is by deploying flaky Wi-Fi... just don't do it. Friends don't let friends use flaky Wi-Fi!

Now, back to the code... next is an informational message tha's logged into the console.

When you ran the demo, you would have seen something like this…

```
[INFO ][DEMO][5742] Successfully initialized the demo. Network type for the demo: 1
```

Network type 1 here means Wi-Fi.

Once that's out of the way, control is passed to the appropriate demoFunction that was originally passed into the task as a pointer. Remember, in this case it is the function RunMqttDemo which is in iot_demo_mqtt.c.

Most of the action for this demo is accomplished in the RunMqttDemo.

After defining some variables, the MQTT library is initialized and an MQTT connection is established.

Then it subscribes to MQTT messages using the prefix of "iotdemo" followed by /topic/1, 2, 3, and 4.

Once that is completed, the project publishes all of the messages. A semaphore is used so that the firmware waits until all of the publishes have been completed before it continues on.

Once all of them are received it removes subscriptions and closes the connection, does a little bit of other cleanup, and then it exits.

Now that we have some idea of how a project is architected, let's dive deeply into the setup of debugging.

I'm going to use WICED Studio 6.2.1 but you can use WICED Studio 6.4 or later versions as well.

Start by choosing Run > Debug Configurations…

Under GDB Hardware Debugging click on "43xxx_Wi-Fi_Debug_Windows". Now Right-click and pick "Duplicate".

Change the name in the top of the window to "AFR Debug" or any other name you prefer.

In the "Main tab" click Browse under Project and select "aws_demos". Next, under C/C++ Application click "Search Project", select "last_built_elf" and click OK.

Before launching the debugger, we need to make one more change. Currently RTOS thread resolve during debug is not supported for AFR in WICED Studio. As a work-around, go to the WICED Studio installation folder to open the file shown here (43xxx_Wi-Fi\tools\OpenOCD\BCM4390x.cfg) in a file editor and go to the line that has:

```
$_TARGETNAME configure -rtos auto -rtos-wipe
```

Remove the "-rtos auto" from that line and save the file.

Now click Apply and then Debug. If you've connected the board and everything goes well, the debugger will start up and it will pause execution in the start_GCC.S file.

You can open iot_demo_mqtt.c and put a breakpoint at the line that says
"if(acknowledgementLength < 0)" inside the subscription callback function
by double clicking to the left of the line or by right-clicking and selecting Toggle Breakpoint.

This function is called by the MQTT library each time a message is received for a topic that you have subscribed to.

It should be noted that if you wait indefinitely on a breakpoint inside of a thread with a network connection in it, you have a much a larger chance of getting disconnected. It kind of turns out that timing in Wi-Fi networks actually matters and halting the processor for a long time probably isn't going to work well. So, stopping execution while inside of an active connection should be used sparingly.

Press the resume button. It will take several seconds to connect to your Wi-Fi access point and then to establish an MQTT connection with AWS. Once that is done, subscriptions are made and then finally you start to publish messages. Whenever a message from one of your subscriptions triggers the callback function, execution will stop at your breakpoint. Keep hitting resume and notice that it stops at the breakpoint each time a message is received from the broker.

That takes us to the end of this set of tutorial videos. I hope you enjoyed them and that you learned a lot about how to use Amazon FreeRTOS with the Cypress 43907 and 54907 to make your own awesome IoT connected devices.

We plan to make more videos to cover advanced topics and new Cypress devices as they come out, so check back again to see what's new.

You can post your comments and questions in our developer community or as always you are welcome to email me a personal comment or question to alan_hawse@cypress.com or tweet me @askioexpert with your comments, suggestions, criticisms and questions. Thank You!