Azure RTOS and Microchip ATSAME54-XPro Evaluation kit

By Sean D. Liming and John R. Malin
Annabooks – www.annabooks.com

May 2023

There are a number of Azure RTOS online guides to get started with different platforms. The ATSAME54-XPRO is one of the first platforms that demonstrated connecting to Azure IoT Central. If you follow the quick start online documents, you will be able to build the example application from the command line and get it to run. If you want to use the example application as a basis for a project, being able to debug by stepping through the code is going to be important. In this paper, we will walk through the example but set up the development environment to use Visual Studio Code.

Target Hardware: SAM E54 XPLAINED PRO EVALUATION KIT (ATSAME54-XPRO).

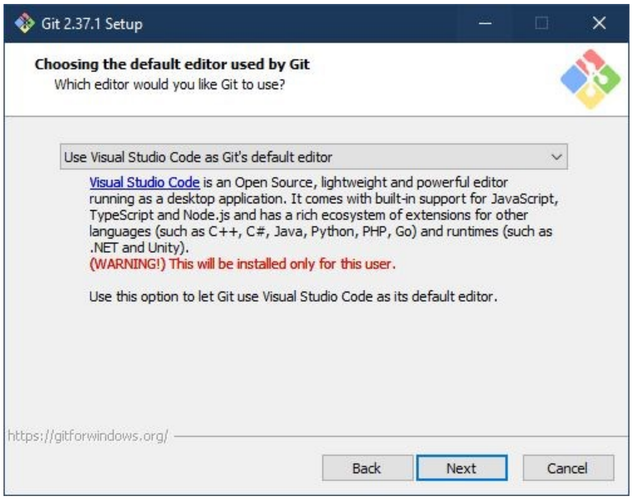
1 Tools Setup

For this setup we will need to download and install a few items.

1. Download and install Visual Studio Code: Visual Studio Code - Code Editing 1.69.2.
2. Once Visual Studio Code has been installed, install the following add-ons from the Visual Studio Code marketplace:

    - C/C++ - Visual Studio Marketplace
    - CMake Tools - Visual Studio Marketplace
    - CMake - Visual Studio Marketplace
    - Cortex-Debug - Visual Studio Marketplace
    - Embedded Tools - Visual Studio Marketplace

3. Install Git so we can download the Azure RTOS to get started building the files: Git - Downloads (git-scm.com).
    a. Accept the license, and click Next.
    b. Leave the install location as is, and click Next.
    c. Leave the Selected Components as they are, and click Next.
    d. Keep the State Menu Folder as is, and click Next.
    e. Set the default editor selection to be “Use Visual Studio Code as Git’s default editor”, and click Next.



Choosing the default editor used by Git
Which editor would you like Git to use?

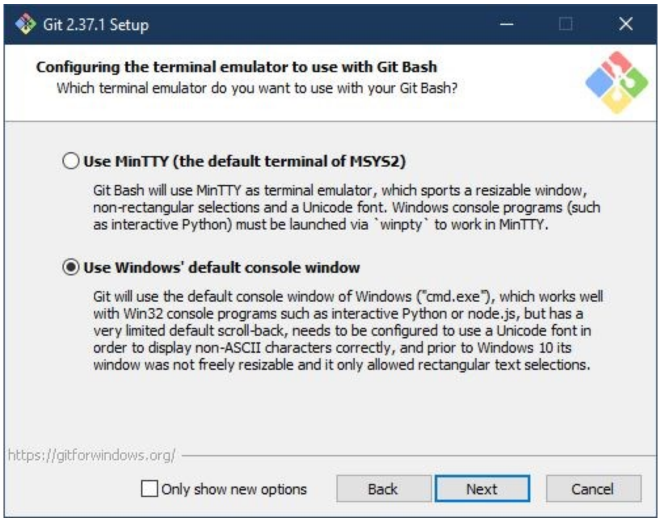
Use Visual Studio Code as Git's default editor

Visual Studio Code is an Open Source, lightweight and powerful editor running as a desktop application. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity).
(WARNING!) This will be installed only for this user.

Use this option to let Git use Visual Studio Code as its default editor.

https://gitforwindows.org/

f. Keep the default for initial branches, and click Next.
g. Keep the default PATH Environment, and click Next.
h. Keep the default OpenSSH selection, and click Next.
i. Select “Use Windows’ default console window”, and click Next.



Configuring the terminal emulator to use with Git Bash
Which terminal emulator do you want to use with your Git Bash?

○ **Use MinTTY (the default terminal of MSYS2)**

Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `winpty` to work in MinTTY.

● **Use Windows' default console window**

Git will use the default console window of Windows ("cmd.exe"), which works well with Win32 console programs such as interactive Python or node.js, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.

https://gitforwindows.org/

□ Only show new options

j. Keep the defaults for the next question, and click Next.
k. Keep the defaults for the next question, and click Next.
l. Keep the defaults for the next question, and click Next.
m. Keep the defaults for the extra options, and click Next.
n. Keep the defaults for the experimental options, and click Install.
o. Click Finish once the install completes.

4. Download and install a serial terminal program such as HyperTerminal, PuTTY, or Annabooks COM Terminal.
5. Reboot the computer.

2 Visual Studio Code Sample Application

This section covers the sample Azure RTOS getting-started sample but uses Visual Studio Code to implement the sample.

2.1 Download the Getting Started Files from GitHub

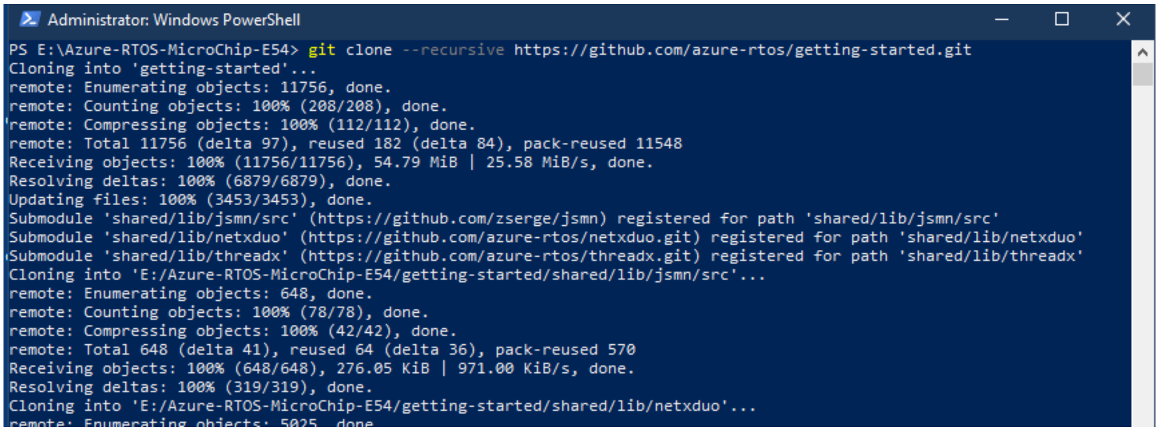
We need to get the getting-started repository that contains the Azure RTOS build example and the ports to the ATSAME54-XPRO and other development kits.

1. Create a directory called \Azure-RTOS-Microchip-E54
2. Open PowerShell.
3. Change the directory to the newly created folder:

   ```
   cd \Azure-RTOS-Microchip-E54
   ```

4. Run the following

   ```
   git clone --recursive https://github.com/azure-rtos/getting-started.git
   ```



```
PS E:\Azure-RTOS-MicroChip-E54> git clone --recursive https://github.com/azure-rtos/getting-started.git
Cloning into 'getting-started'...
remote: Enumerating objects: 11756, done.
remote: Counting objects: 100% (208/208), done.
remote: Compressing objects: 100% (112/112), done.
remote: Total 11756 (delta 97), reused 182 (delta 84), pack-reused 11548
Receiving objects: 100% (11756/11756), 54.79 MiB | 25.58 MiB/s, done.
Resolving deltas: 100% (6879/6879), done.
Updating files: 100% (3453/3453), done.
Submodule 'shared/lib/jsmn/src' (https://github.com/zserge/jsmn) registered for path 'shared/lib/jsmn/src'
Submodule 'shared/lib/netxduo' (https://github.com/azure-rtos/netxduo.git) registered for path 'shared/lib/netxduo'
Submodule 'shared/lib/threadx' (https://github.com/azure-rtos/threadx.git) registered for path 'shared/lib/threadx'
Cloning into 'E:/Azure-RTOS-MicroChip-E54/getting-started/shared/lib/jsmn/src'...
remote: Enumerating objects: 648, done.
remote: Counting objects: 100% (78/78), done.
remote: Compressing objects: 100% (42/42), done.
remote: Total 648 (delta 41), reused 64 (delta 36), pack-reused 570
Receiving objects: 100% (648/648), 276.05 KiB | 971.00 KiB/s, done.
Resolving deltas: 100% (319/319), done.
Cloning into 'E:/Azure-RTOS-MicroChip-E54/getting-started/shared/lib/netxduo'...
remote: Enumerating objects: 5025, done.
```

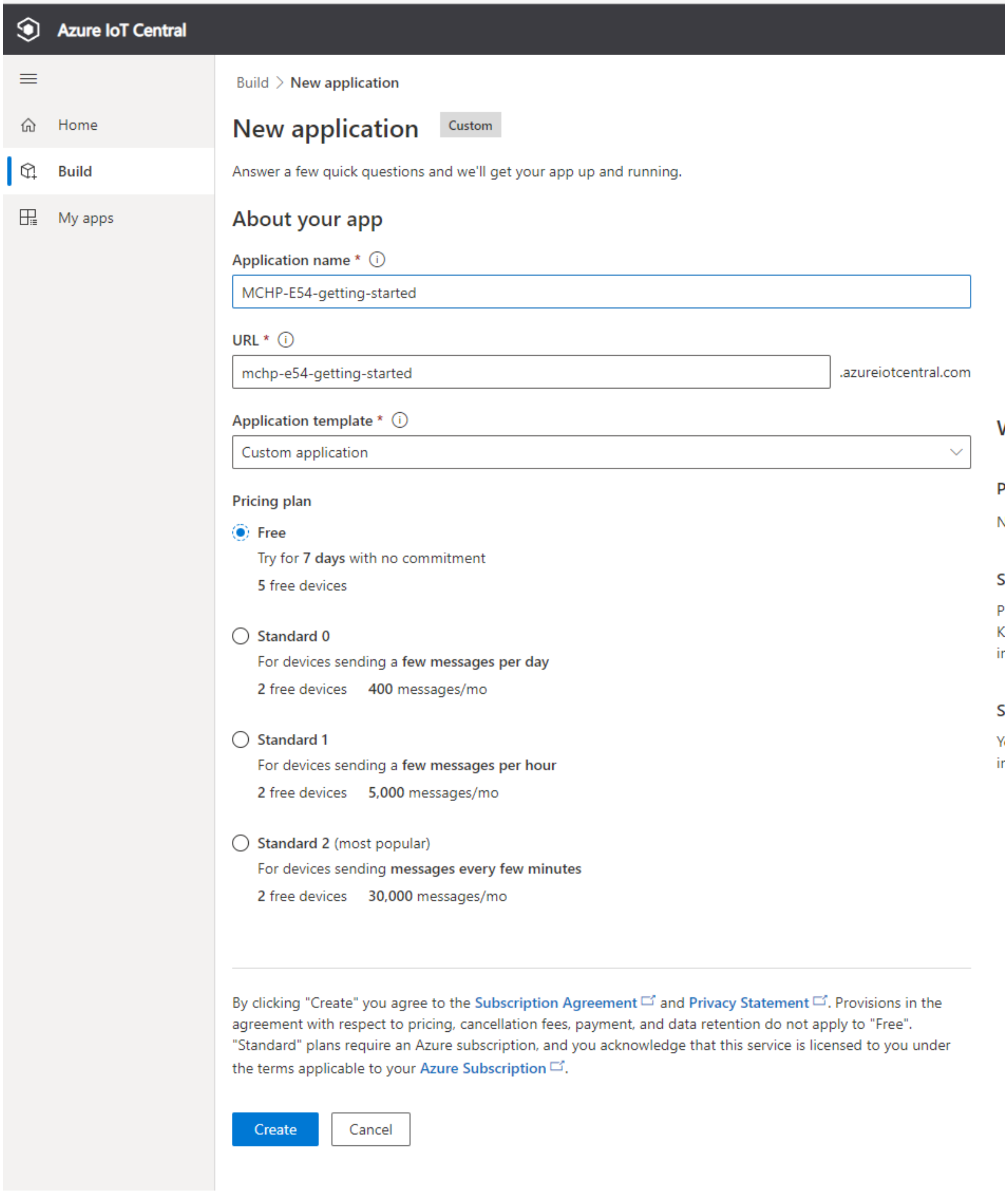
2.2 Create Azure IoT Central Application

Now, we need to set up the application on Azure IoT Central.

1. In a browser, open https://apps.azureiotcentral.com/home
2. Sign into the account or create an account.
3. Click on Build App.
4. In the Custom app tile, click Create app.

   Application Name: MCHP-E54-getting-started
   Pricing Plan: Free

5. Click Create.



Build > New application

New application   Custom

Answer a few quick questions and we'll get your app up and running.

About your app

Application name * : MCHP-E54-getting-started

URL * : mchp-e54-getting-started .azureiotcentral.com

Application template * : Custom application

Pricing plan

● Free
Try for **7 days** with no commitment
5 free devices

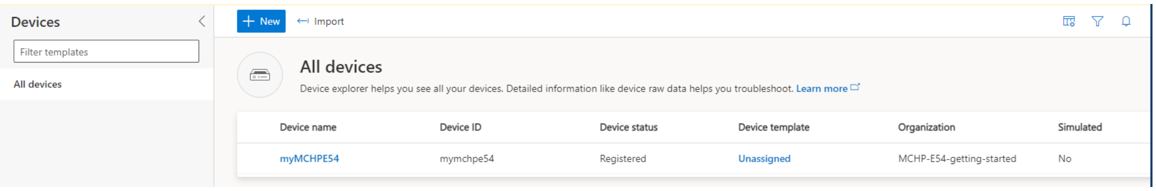
○ Standard 0
For devices sending a **few messages per day**
2 free devices   **400** messages/mo

○ Standard 1
For devices sending a **few messages per hour**
2 free devices   **5,000** messages/mo

○ Standard 2 (most popular)
For devices sending **messages every few minutes**
2 free devices   **30,000** messages/mo

By clicking "Create" you agree to the Subscription Agreement and Privacy Statement. Provisions in the agreement with respect to pricing, cancellation fees, payment, and data retention do not apply to "Free". "Standard" plans require an Azure subscription, and you acknowledge that this service is licensed to you under the terms applicable to your Azure Subscription.

6. Now, we need to add a device to the application, and click on the +New button that is above the All Devices section.
7. Enter the following:
   a. Device Name: myMCHPE54
   b. Device ID mymmymchpe54chpe54
8. Click Create.
9. The device will be created and listed under all devices

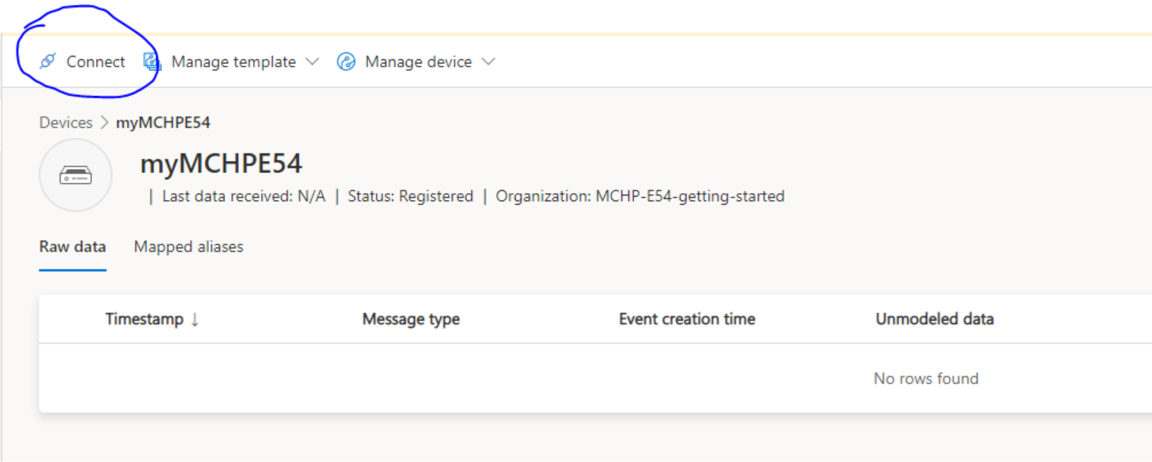


All devices

Device explorer helps you see all your devices. Detailed information like device raw data helps you troubleshoot. Learn more

Device name	Device ID	Device status	Device template	Organization	Simulated
myMCHPE54	mymchpe54	Registered	Unassigned	MCHP-E54-getting-started	No

10. Click on myMCHPE54. This will be the view of the data coming in.
11. Click on Connect at the top of the bar.



myMCHPE54

| Last data received: N/A | Status: Registered | Organization: MCHP-E54-getting-started

Timestamp ↓	Message type	Event creation time	Unmodeled data
			No rows found

12. A Device Connections group box appears. Copy the following information and paste it in a Notepad or Notepad++ temporary document. We will need this information in the next section.

    - ID scope
    - Device ID
    - Primary Key

13. Close the dialog when finished.

2.3 Building the Getting Started Sample App

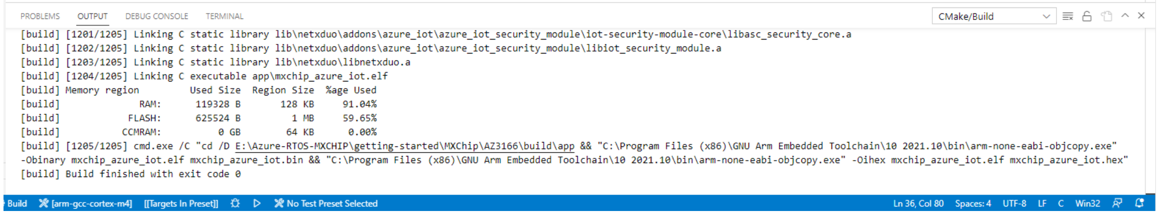
With the application created in Azure IoT Central and the device information collected to make the connection, we are ready to build the example.

1. Open File Explorer and change the directory to \Azure-RTOS-MicroChip-E54\getting-started\Microchip\ATSAME54-XPRO.
2. Double-click on ATSAME54-XPRO.code-workspace to open the workspace in Visual Studio Code.
3. You will be asked to trust the authors of the code, click Yes.
4. When asked for the toolchain at the top, accept arm-gcc-cortex-m4.
5. Under ATSAME54-XPRO\App, open Azure_config.h and fill in the information gathered from the Azure IoT Central application, as well as, your Wi-Fi connection settings:

Constant name	Value
IOT_DPS_ID_SCOPE	ID scope value
IOT_DPS_REGISTRATION_ID	Device ID value
IOT_DEVICE_SAS_KEY	Primary key value

6. Save the file.

7. At the bottom click on Build. It will take a few minutes, but the build should complete successfully.

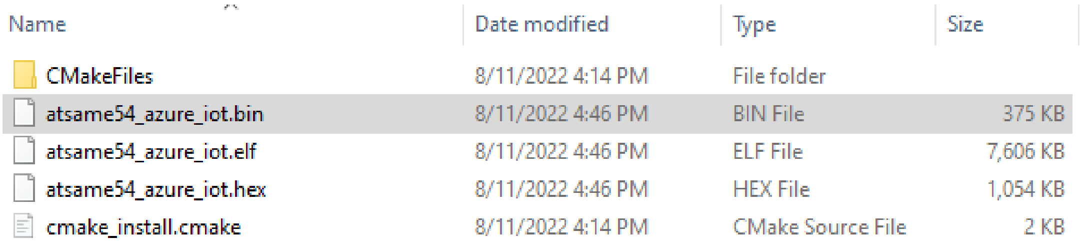


```
[build] [1201/1205] Linking C static library lib\netxduo\addons\azure_iot\azure_iot_security_module\iot-security-module-core\libasc_security_core.a
[build] [1202/1205] Linking C static library lib\netxduo\addons\azure_iot\azure_iot_security_module\libiot_security_module.a
[build] [1203/1205] Linking C static library lib\netxduo\libnetxduo.a
[build] [1204/1205] Linking C executable app\mxchip_azure_iot.elf
[build] Memory region         Used Size  Region Size  %age Used
[build]              RAM:      119328 B       128 KB     91.04%
[build]            FLASH:      625524 B         1 MB     59.65%
[build]           CCMRAM:           0 GB        64 KB      0.00%
[build] [1205/1205] cmd.exe /C "cd /D E:\Azure-RTOS-MXCHIP\getting-started\MXChip\AZ3166\build\app && "C:\Program Files (x86)\GNU Arm Embedded Toolchain\10 2021.10\bin\arm-none-eabi-objcopy.exe"
-Obinary mxchip_azure_iot.elf mxchip_azure_iot.bin && "C:\Program Files (x86)\GNU Arm Embedded Toolchain\10 2021.10\bin\arm-none-eabi-objcopy.exe" -Oihex mxchip_azure_iot.elf mxchip_azure_iot.hex"
[build] Build finished with exit code 0
```

2.4 *Program the ATSAME54-XPRO Board*

With the atsame54_azure_iot.bin built, programming the board is a simple copy and paste.

1. Make sure the board is connected to the development machine.
2. Open Microchip Command Prompt.
3. Change directory to \Azure-RTOS-MicroChip-E54\getting-started\Microchip\ATSAME54-XPRO\build\app.



Name	Date modified	Type	Size
CMakeFiles	8/11/2022 4:14 PM	File folder	
atsame54_azure_iot.bin	8/11/2022 4:46 PM	BIN File	375 KB
atsame54_azure_iot.elf	8/11/2022 4:46 PM	ELF File	7,606 KB
atsame54_azure_iot.hex	8/11/2022 4:46 PM	HEX File	1,054 KB
cmake_install.cmake	8/11/2022 4:14 PM	CMake Source File	2 KB

4. Run the following

atprogram --tool edbg --interface SWD --device ATSAME54P20A program --chiperase --file atsame54_azure_iot.bin --verify

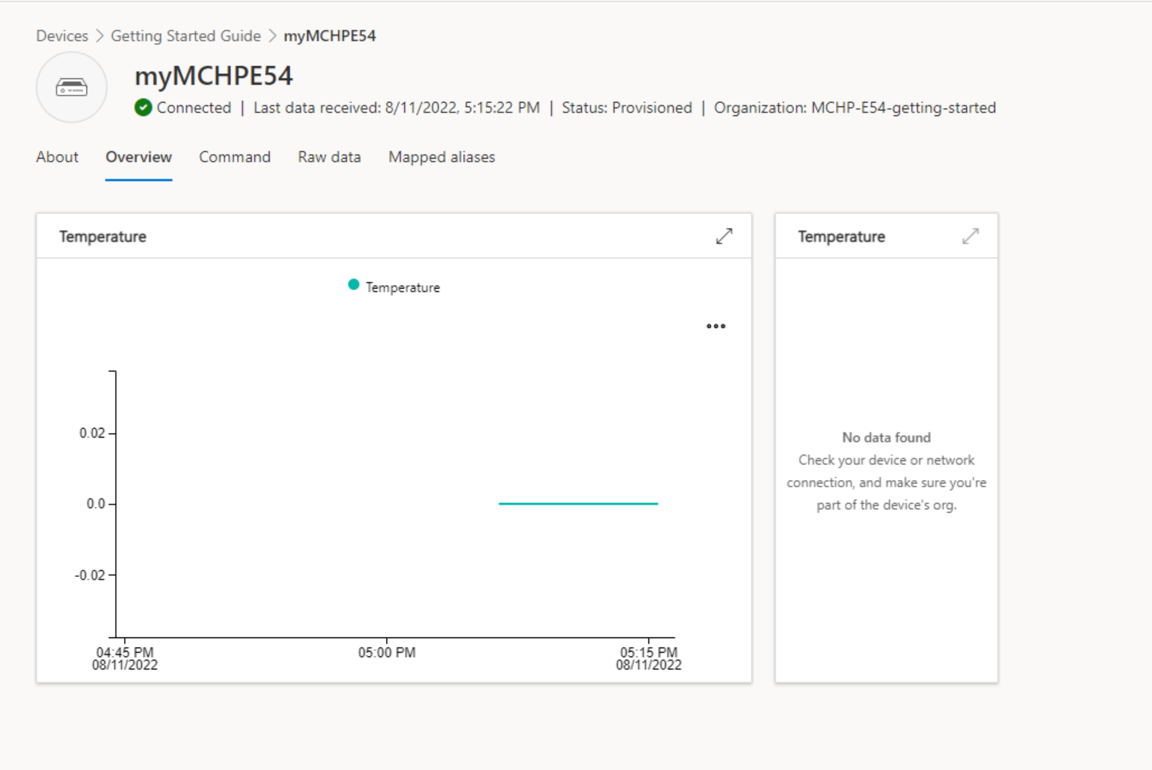
5. If asked to allow the application to go through the firewall, click Allow. Wait for the message that programming and verification have completed successfully before going to the next step.
6. Open a serial terminal program and connect to the COM port and set the baud rate to 115200 and enable Flow Control DTR/DSR (hardware).
7. Hit the reset button on the ATSAME54-XPRO.

If all goes well, you will see the terminal output with something similar to the following:

```
Starting Azure thread


Initializing DHCP
        MAC: FC:C2:3D:23:58:4B
        IP address: 192.168.1.239
        Mask: 255.255.255.0
        Gateway: 192.168.1.1
SUCCESS: DHCP initialized

Initializing DNS client
        DNS address: 192.168.1.1
SUCCESS: DNS client initialized
```

```
Initializing SNTP time sync
        SNTP server 0.pool.ntp.org
        SNTP time update: Aud 12, 2022 0:23:12.543 UTC
SUCCESS: SNTP initialized

Initializing Azure IoT DPS client
        DPS endpoint: global.azure-devices-provisioning.net
        DPS ID scope: 0ne00706F71
        Registration ID: mymchpe54
SUCCESS: Azure IoT DPS client initialized

Initializing Azure IoT Hub client
        Hub hostname: iotc-c131322c-97ad-4083-86e6-3a9776985e11.azure-devices.ne
t
        Device id: mymchpe54
        Model id: dtmi:azurertos:devkit:gsg;2
SUCCESS: Connected to IoT Hub

Receive properties: {"desired":{"$version":1},"reported":{"deviceInformation":{"
__t":"c","manufacturer":"Microchip","model":"ATSAME54-XPRO","swVersion":"1.0.0",
"osName":"Azure RTOS","processorArchitecture":"Arm Cortex M4","processorManufact
urer":"Microchip","totalStorage":1024,"totalMemory":256},"ledState":false,"telem
etryInterval":{"ac":200,"av":1,"value":10},"$version":27}}
Sending property: $iothub/twin/PATCH/properties/reported/?$rid=3{"deviceInformat
ion":{"__t":"c","manufacturer":"Microchip","model":"ATSAME54-XPRO","swVersion":"
1.0.0","osName":"Azure RTOS","processorArchitecture":"Arm Cortex M4","processorM
anufacturer":"Microchip","totalStorage":1024,"totalMemory":256}}
Sending property: $iothub/twin/PATCH/properties/reported/?$rid=5{"ledState":fals
e}
Sending property: $iothub/twin/PATCH/properties/reported/?$rid=7{"telemetryInter
val":{"ac":200,"av":1,"value":10}}

Starting Main loop
```

8. In Azure IoT Central, refresh the browser to see the myMCHPE54 device. Since there is no weather station hardware connected, there is no data.



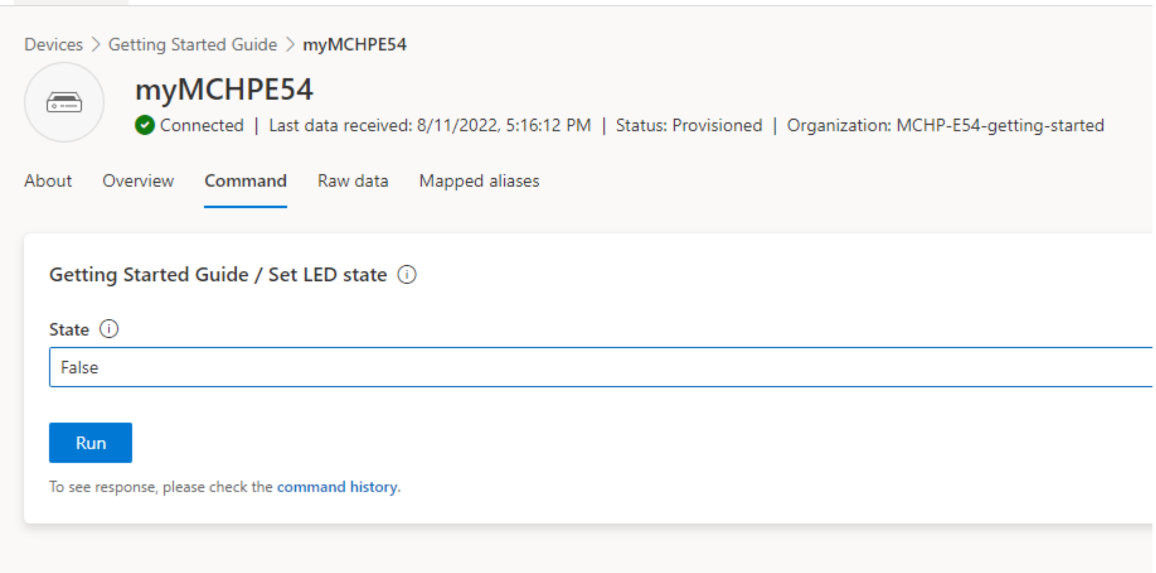
Devices > Getting Started Guide > myMCHPE54

myMCHPE54

Connected | Last data received: 8/11/2022, 5:15:22 PM | Status: Provisioned | Organization: MCHP-E54-getting-started

No data found
Check your device or network connection, and make sure you're part of the device's org.

9. Click on Command.
10. Set the State to False.
11. Click Run, and the LED on the board will turn off.
12. Set the State to True.
13. Click Run, and the LED on the board will turn on.



Devices > Getting Started Guide > myMCHPE54

myMCHPE54

Connected | Last data received: 8/11/2022, 5:16:12 PM | Status: Provisioned | Organization: MCHP-E54-getting-started

Getting Started Guide / Set LED state

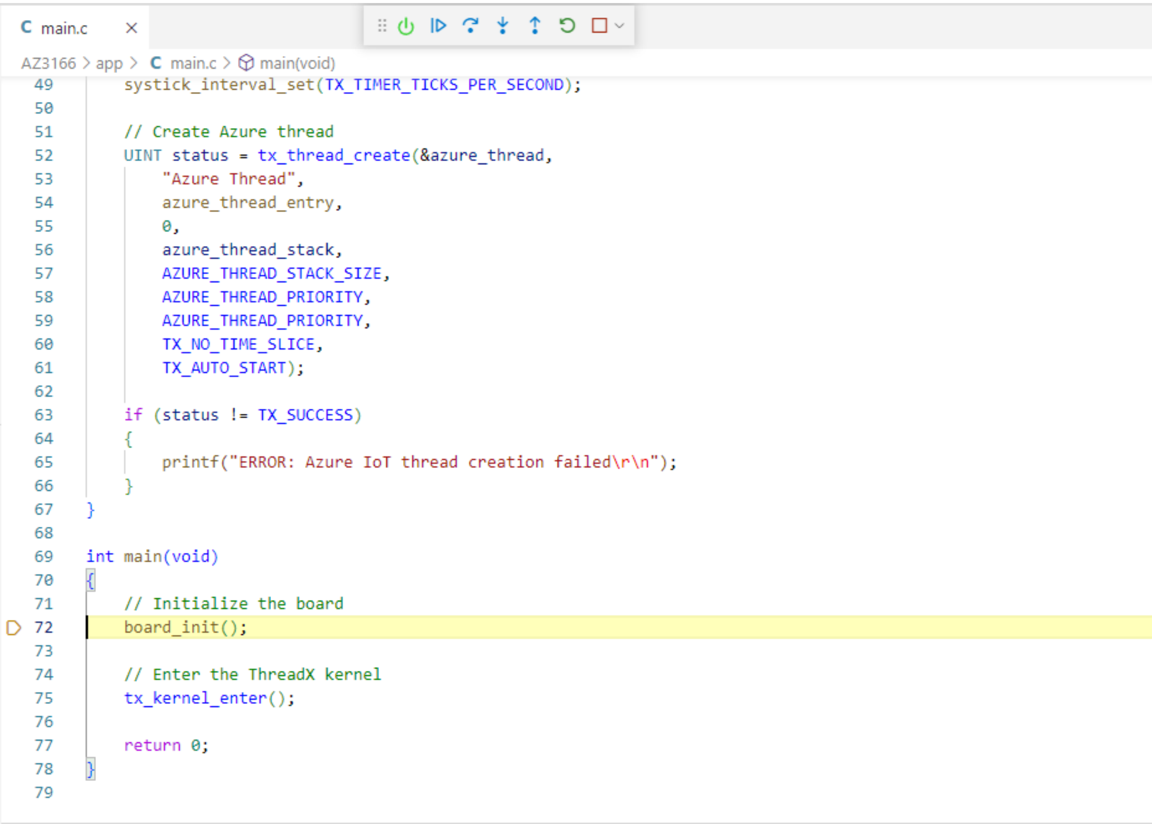
State: False

To see response, please check the command history.

2.5 *Debugging the Application*

Now, we will step through the code to see how it works.

1. In Visual Studio Code, hit F5.
2. The binary will be downloaded and a breakpoint will be hit within main.c.



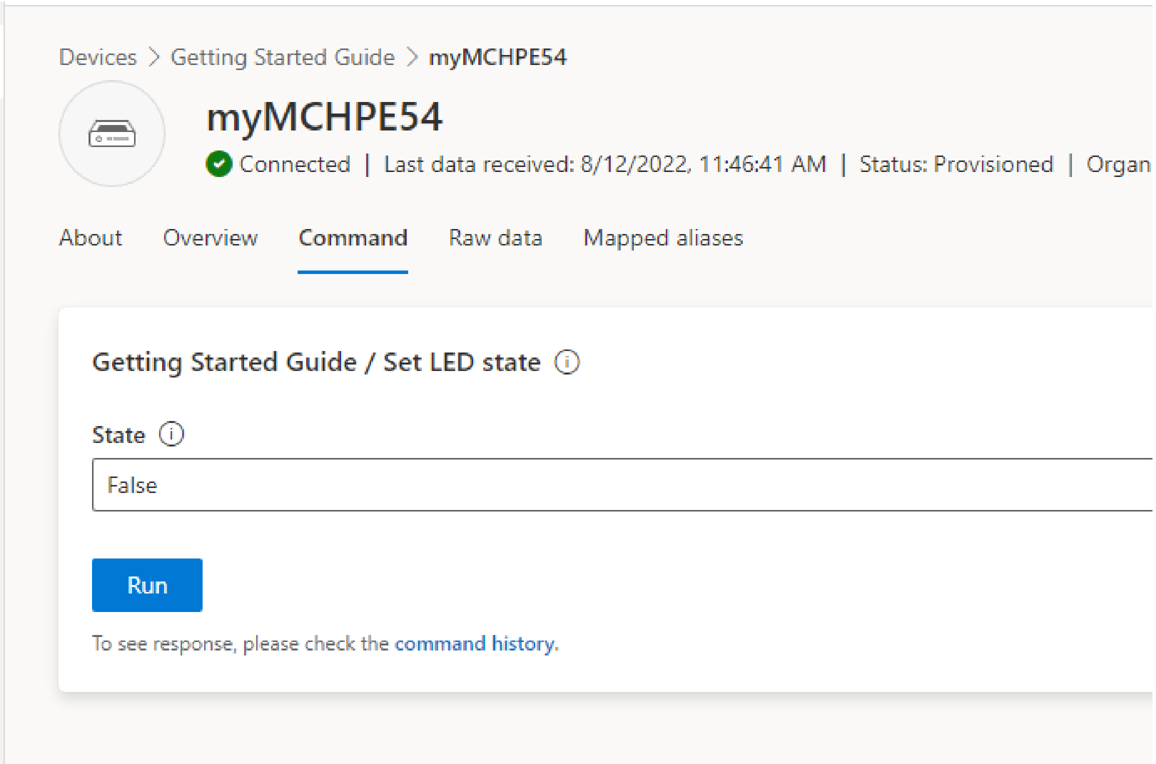
```
    systick_interval_set(TX_TIMER_TICKS_PER_SECOND);

    // Create Azure thread
    UINT status = tx_thread_create(&azure_thread,
        "Azure Thread",
        azure_thread_entry,
        0,
        azure_thread_stack,
        AZURE_THREAD_STACK_SIZE,
        AZURE_THREAD_PRIORITY,
        AZURE_THREAD_PRIORITY,
        TX_NO_TIME_SLICE,
        TX_AUTO_START);

    if (status != TX_SUCCESS)
    {
        printf("ERROR: Azure IoT thread creation failed\r\n");
    }
}

int main(void)
{
    // Initialize the board
    board_init();

    // Enter the ThreadX kernel
    tx_kernel_enter();

    return 0;
}
```

3. Click Step Over (F10) to move past the board initialization call.
4. Click Step Over (F10) and the application thread will kick off and run.
5. Stop the debugger (Shift+F5).

The files that comprise the core functionality of the application are:

- main.c – sets up and runs the thread.
- nx_client.c – creates the callback function to send telemetry and handle receive commands.
- Azure_iot_nx_client.c – this file has the main loop client_run(), which connects to Azure IoT Central and handles communications between the local application and the application on Azure IoT Central.

6. In main.c, set a breakpoint at line 38, which is the call to azure_iot_nx_client_entry.
7. Also, in nx_client.c, set another breakpoint at line 147, which is the call to turn the LED on or off.
8. Hit F5.
9. When the breakpoint hits in Main.c, hit F10 twice.
10. The debugger will break at line 34. Hit F11 to step into the to azure_iot_nx_client_entry call.
11. Hit F5 to allow the code to continue.
12. The debugger is now in the main loop in Azure_iot_nx_client.c. In Azure IoT Central, click on Command, set the LED State to True, and click Run.



Devices > Getting Started Guide > myMCHPE54

myMCHPE54

Connected | Last data received: 8/12/2022, 11:46:41 AM | Status: Provisioned | Organ

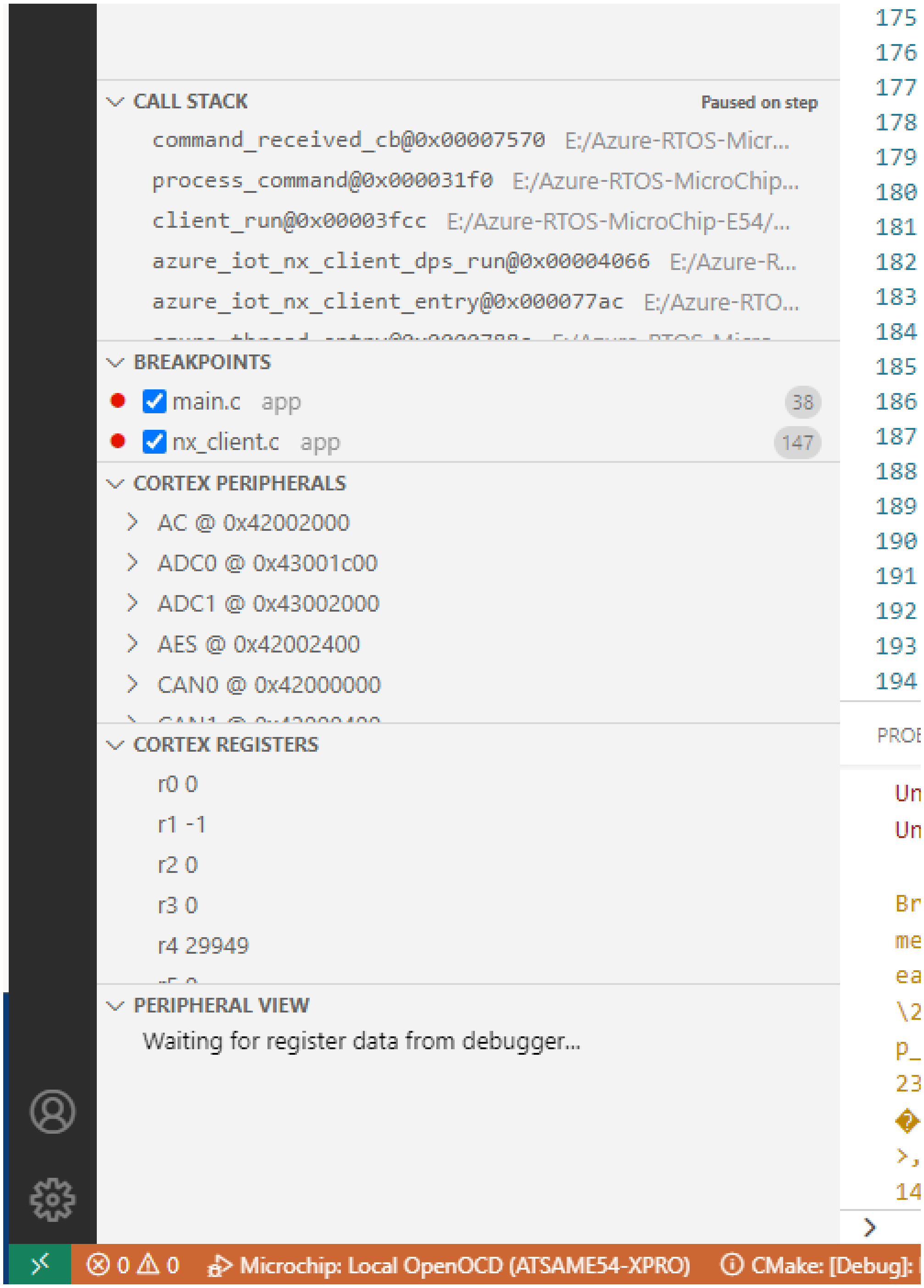
Getting Started Guide / Set LED state

State: False

To see response, please check the command history.

13. A breakpoint should be hit at line 147 in nx_client.c.
14. Hit F5 to continue debugging and the LED should turn on.
15. Change the state of the LED a few times and watch each time the breakpoint is hit.

If you have installed the embedded tools into Visual Studio Code, you will be able to see the Peripherals and Cortex Registers in the Debug section.



```
CALL STACK                                                    Paused on step
    command_received_cb@0x00007570  E:/Azure-RTOS-Micr...
    process_command@0x000031f0  E:/Azure-RTOS-MicroChip...
    client_run@0x00003fcc  E:/Azure-RTOS-MicroChip-E54/...
    azure_iot_nx_client_dps_run@0x00004066  E:/Azure-R...
    azure_iot_nx_client_entry@0x000077ac  E:/Azure-RTO...
BREAKPOINTS
 ● ☑ main.c  app                                                          38
 ● ☑ nx_client.c  app                                                     147
CORTEX PERIPHERALS
  > AC @ 0x42002000
  > ADC0 @ 0x43001c00
  > ADC1 @ 0x43002000
  > AES @ 0x42002400
  > CAN0 @ 0x42000000
CORTEX REGISTERS
    r0 0
    r1 -1
    r2 0
    r3 0
    r4 29949
PERIPHERAL VIEW
   Waiting for register data from debugger...

⊗ 0 ⚠ 0   Microchip: Local OpenOCD (ATSAME54-XPRO)   ⓘ CMake: [Debug]:
```

16. Hit Shift+F5 to stop debugging.

3 Conclusion

Sample projects are good starting points to get familiar with the software. The ability to step through the code and see the API calls during operation provides good insight when documentation is lacking. The paper here covered debugging with Visual Studio Code, but further development should be done using the Microchip MPLAB tools that provide a richer development experience and direct processor support.

References

More information on the Azure IoT SDKs can be found here.