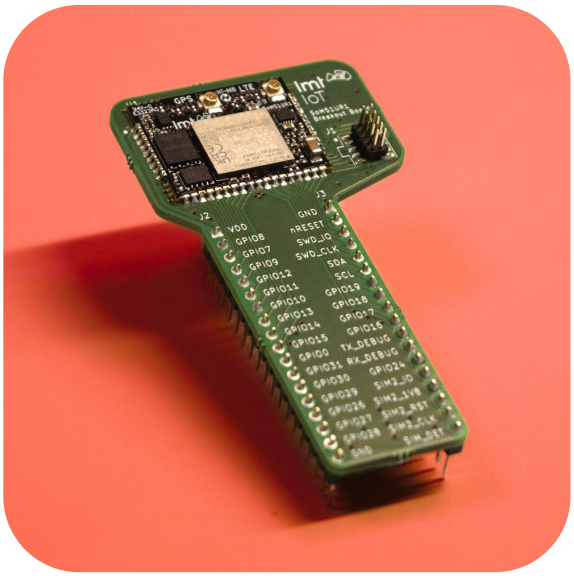
IoT Shortcut Evaluation Kit Tutorial



This guide walks you through the entire process of prototyping with the IoT Shortcut Evaluation Kit. You'll learn how to configure the development tools for creating custom firmware, assemble the hardware, and connect to our cloud solution for remote data uploads. To bring it all together, this tutorial uses a practical use case as a step-by-step example.

1 Foreword

This tutorial provides step-by-step instructions for creating an IoT device prototype using the Evaluation Kit hardware, developing its firmware, and uploading data to the LMT IoT Cloud via the cellular connection provided by the built-in global SIM card. Table 1 lists all items included with the IoT Shortcut Evaluation Kit. When your kit ships, LMT staff will send you related information, including this tutorial, and will register your user account(s) on the LMT IoT Cloud.

Packaged item	Official product name
Evaluation board	LMT IoT Shortcut hardware (nRF9151-based) soldered on a breakout board
Breadboard	Standard 400 point solderless breadboard
Programming cable	10-pin 2x5 Socket-Socket 1.27mm SWD cable
Antenna	Molex 824–2170 MHz cellular flexible antenna with U.FL connector
Identification label	—
Programming board (optional)	Nordic Semiconductor nRF9151 Development Kit
Power profiler (optional)	Nordic Semiconductor Power Profiler Kit II

Table 1: Contents of the IoT Shortcut Evaluation Kit packaging.

Some sections of this tutorial cover setting up the Software Development Kit (SDK) from Nordic Semiconductor, as the LMT IoT Shortcut is based on their nRF9151 System-in-Package (SiP). Feel free to skip these parts if you already have Nordic development tools installed or prefer a different environment than the one suggested here (Microsoft Visual Studio Code).

Similarly, other parts of the tutorial describe using a Nordic Semiconductor programming board to flash firmware onto the LMT evaluation board and using their power profiler tool for powering the board. If you already use your own firmware programming tools and power supply, you may skip these steps. However, these instructions are included for completeness and to assist those new to hardware development in ensuring all prerequisites are met.

Section 5 demonstrates building a hardware prototype with custom firmware, utilizing several I/O devices with the IoT Shortcut as the main processing and connectivity unit. Please note that the additional example hardware (sensors, LEDs, etc.) mentioned in this section is not included in the IoT Shortcut Evaluation Kit package (as listed in Table 1). This example serves to illustrate the I/O capabilities of the IoT Shortcut hardware and the benefits of the complete IoT Shortcut solution.

2 Setting Up Firmware Development Environment

2.1 VS Code and nRF Connect SDK

Since the IoT Shortcut hardware is based on the Nordic Semiconductor nRF9151 System-in-Package (SiP), developing firmware requires Nordic Semiconductor's software tools. Furthermore, the LMT IoT Shortcut SDK is built upon the Nordic SDK. Therefore, to create firmware for the LMT IoT Shortcut, you must install the nRF Connect SDK.

The recommended method for setting up the necessary nRF Connect SDK components is to install Microsoft Visual Studio Code along with the nRF Connect Extension Pack. **Follow "Installing nRF Connect SDK and VS Code" exercise of the Nordic Developer Academy to install the required tools.**

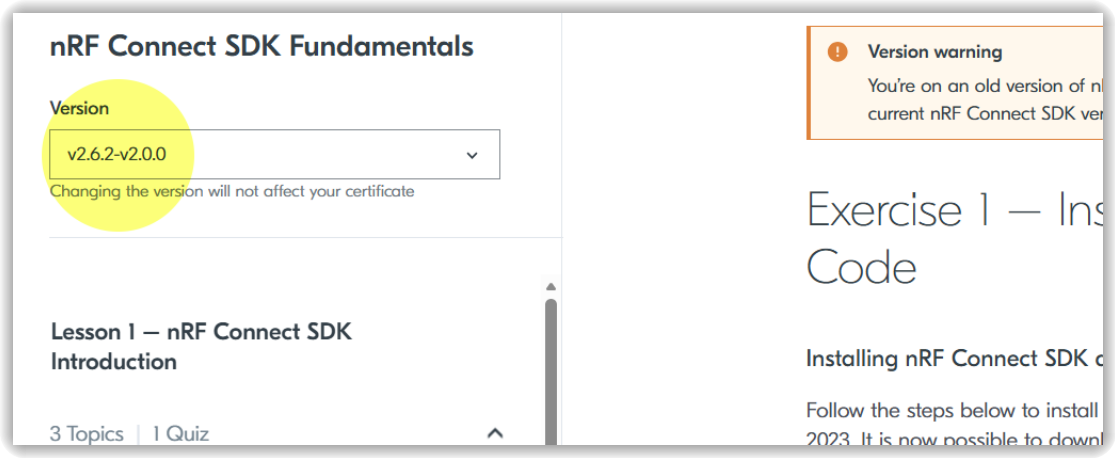


Figure 1: Selecting the correct nRF Connect SDK version for use with the LMT IoT Shortcut SDK.

Crucially, ensure that you **install SDK version 2.6.0**, as the LMT IoT Shortcut SDK is specifically based on this release. You will need to make this version selection several times during the setup process, starting with the dropdown menu in the Nordic Developer Academy (see Fig. 1).

2.2 IoT Shortcut SDK

Once the nRF Connect SDK prerequisites are installed, you can proceed to set up the LMT IoT Shortcut SDK. This SDK contains specialized software libraries, documentation, and firmware samples designed to simplify LMT IoT Cloud integration, and accelerate your overall firmware development process.

While the SDK can be obtained by cloning or downloading the release from its git repository, a much simpler method for Microsoft Visual Studio Code users is to install the dedicated IoT Shortcut SDK extension. To do this, open the VS Code Extensions view by clicking the Activity Bar icon or pressing `Ctrl+Shift+X`. Search for "LMT" as shown in Fig. 2, and install the "IoT Shortcut SDK" extension, which allows you to easily manage your SDK versions directly within the IDE.

To configure a specific SDK version, open the Command Palette by pressing `Ctrl+Shift+P` (Windows/Linux) or `Cmd+Shift+P` (macOS). Type and select "IoT Shortcut SDK: Install SDK" (Fig. 3), then choose your desired version from the dropdown list, with the latest version highly recommended. Finally, select a target installation directory. We strongly suggest using the base path of your system drive (e.g., `C:`) to prevent potential file path length errors. The extension will generate a new folder named `iot-shortcut-sdk-<version>` in this location, housing all the necessary libraries, documentation, and code samples for your LMT IoT Shortcut hardware development.

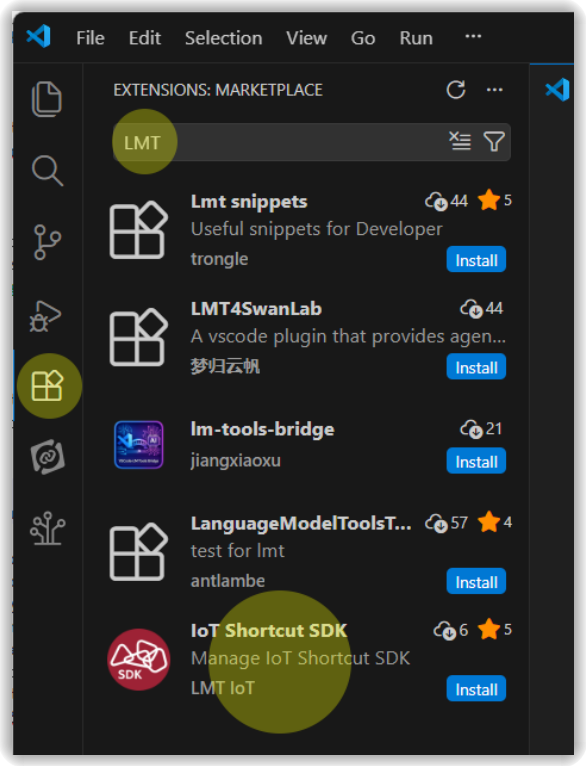


Figure 2: Installing VS Code extension.

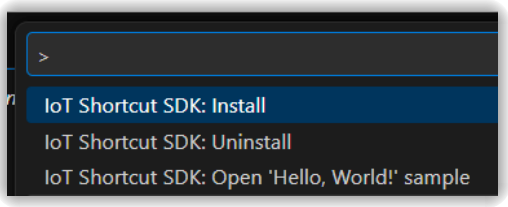


Figure 3: Options provided by the VS Code extension.

3 Accessing LMT IoT Cloud

3.1 Purpose

The LMT IoT Cloud is an essential part of the IoT Shortcut solution. The hardware SDK is optimized for easy integration with this platform, allowing you to upload data from the IoT Shortcut hardware with minimal firmware development. Once uploaded, this data can be used to build end-user web or mobile applications using the provided API and webhook integrations. The LMT IoT Cloud web interface serves as the central management hub. Here, you can set up, activate, deactivate, and assign devices to specific users. Within the platform, user and device management is organized into **tenants**, which are administrative divisions used to separate groups of **users** and **devices**.

3.2 First Time Login

During the onboarding process for the IoT Shortcut Evaluation Kit, LMT staff will create a tenant for your institution and one or more user accounts within this tenant for your evaluation. Registered users will then receive an email notification that an account has been created. Navigate to portal.lmt-iot.com and log in using your email address and the temporary password provided in the email. You will be prompted to change this temporary password during your first login.

3.3 Activating Evaluation Board Hardware

LMT staff will have already assigned the Evaluation Kit hardware to your user account. Navigate to *Device Management* → *Devices* to view your assigned devices; the Evaluation Kit will be included in this list. Locate the device entry for your Evaluation Kit. As shown in Fig. 4, open its options menu and select *Set up device*. If you have multiple devices or need to confirm you have the correct hardware, compare the serial number shown in the LMT IoT Cloud with the one printed on the identification label included in the IoT Shortcut Evaluation Kit packaging.

You will now see the device setup window (Fig. 5). Here, you must provide a name for your device and, crucially, define the structure of the data your Evaluation Kit will upload. The data tracks configured here must precisely match the data structure and order defined in your firmware. The names, units, and multipliers entered will determine how data is processed, stored, and displayed (e.g., in graphs and exports) within the LMT IoT Cloud. Unicode characters (like ° or ^{2}) are acceptable for unit notation. You must also specify a multiplier for each track. This multiplier is applied to the incoming integer data before it's stored. For example, a multiplier of 1 leaves the data unchanged. A multiplier of 1000 would convert incoming data in kilometers to meters, while 0.001 could convert millimeters to meters.

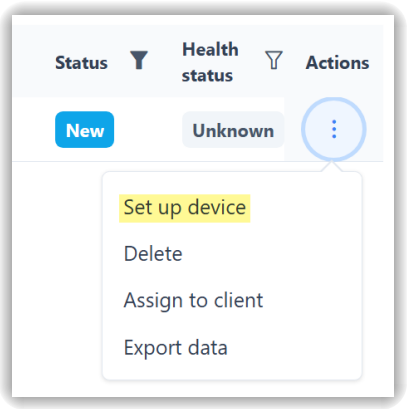


Figure 4: Device setup menu access.

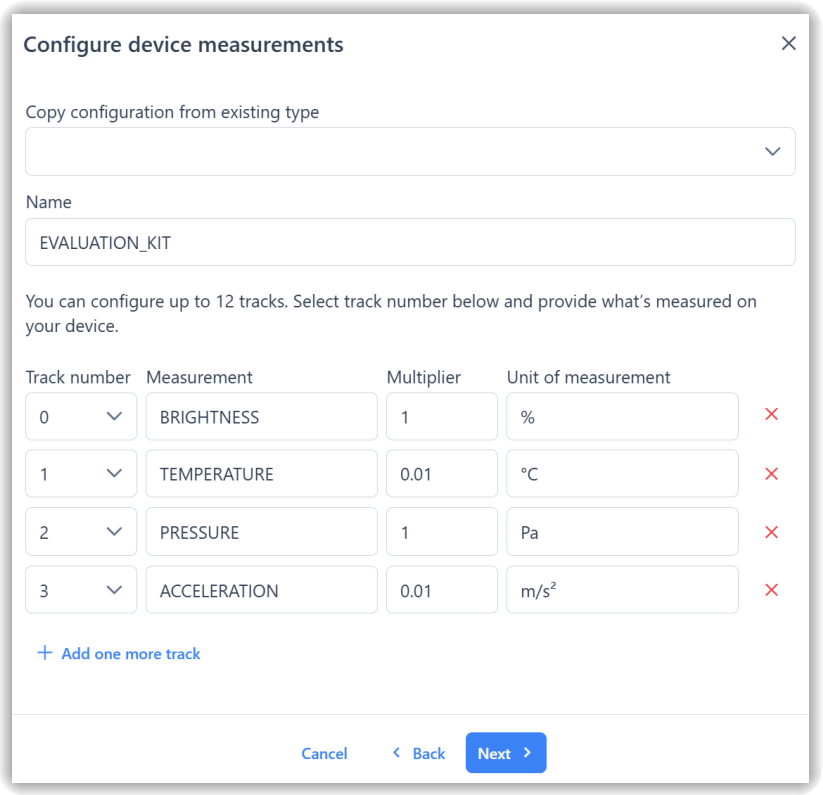


Figure 5: Data upload configuration.

For this tutorial's example (more detail in Section 5), we'll configure tracks for temperature, pressure, acceleration, and the potentiometer position (LED brightness). Since the example firmware will upload data only as integers, we'll upload temperature in centi-degrees Celsius and acceleration in cm/s^2 to preserve data resolution. To store these values in standard units (°C and m/s^2) on the LMT IoT Cloud, we will set the multiplier to 0.01 for the temperature track as well as for the acceleration track.

Pay close attention to the order and number of tracks: (1) the array indices in your firmware code must correspond exactly to the track order configured here; (2) mismatched array lengths between the firmware and this configuration will cause data uploads to fail; (3) incorrect track order will result in mislabeled data within the LMT IoT Cloud; (4) incorrect multiplier will leave misleading magnitude of sensor readings.

After confirming these settings, the device status will change from New to Active. This action automatically triggers the LMT IoT Cloud to use an external API to activate the global SIM card included in your IoT Shortcut Evaluation Kit. This activation is required; without it, the hardware will not be able to register on the mobile network after powering up.

4 Setting up Evaluation Board Hardware

4.1 Nordic Semiconductor Programming Board

Software Setup: If you are new to developing with Nordic Semiconductor devices, LMT provides a Nordic programming board for flashing firmware onto the LMT IoT Shortcut hardware. Begin by installing *nRF Connect for Desktop*. Once installed, open *nRF Connect for Desktop* and add the *Board Configurator* app from the app list.

Hardware Connection & Configuration: Connect the Nordic programming board to your computer using a USB-C cable and slide its power switch to *ON*. Launch the *Board Configurator* app. In the top-left menu, click *Select Device* and choose nRF9151 DK. You might want to enable the *Auto Reconnect* option if this is your primary Nordic board. Next, verify that the setting "VDD (nPM VOUT1)..." is set to 3.3 V (3300 mV). This voltage is required for the LMT IoT Shortcut hardware and ensures correct logic levels for communication. If you adjust this setting, click *Write config*. After the configuration is written, you can close the *Board Configurator* app.

Disconnecting Onboard SiP: Finally, locate the jumper header near the programming board's USB-C connector. Remove the jumper that connects the pins labeled `VDD_nRF` and `VDD_5V` (see Fig. 6). Removing this jumper disconnects power from the DK's onboard SiP, which prevents conflicts when programming the external LMT IoT Shortcut hardware.

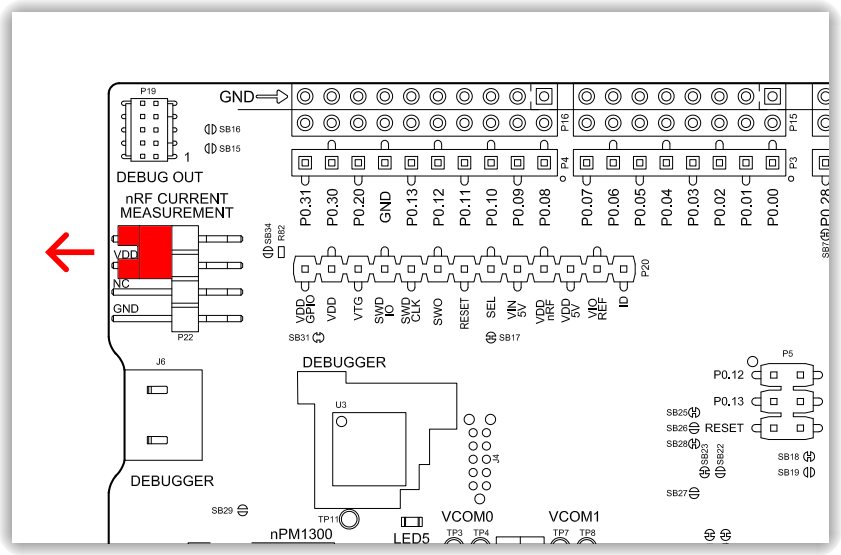


Figure 6: Jumper required be removed from the programming board marked in red.

Board Ready: The Nordic programming board is now ready for use with LMT IoT Shortcut. Proceed to Section 4.3 for instructions on connecting and programming the LMT IoT Shortcut hardware.

4.2 Nordic Semiconductor Power Profiler

Software Setup: Similar to setting up the programming board (Section 4.1), begin by installing *nRF Connect for Desktop* if you haven't already. Once installed, open it and add the *Power Profiler* app.

Hardware Connection & Selection: Connect the power profiler hardware to your computer using a USB cable. Crucially, ensure you plug the cable into the profiler hardware port labeled `USB DATA/POWER`, as this connection is required for configuration. Launch the *Power Profiler* app within *nRF Connect for Desktop*. In the top-left menu, click *Select Device* and choose PPK2.

App Functionality: The *Power Profiler* app allows you to enable/disable the power output, configure the profiler hardware to act either as a voltage source (using the `GND` and `VOUT` pins) or as an ammeter (using the `VIN` and `VOUT` pins), and log power consumption data at various sampling rates for externally connected devices.

Configuration for IoT Shortcut: For powering the IoT Shortcut hardware (as discussed further in Section 4.3), you will use the power profiler in *Source meter* mode. Set the output voltage to 3600 mV (3.6 V) and use the `GND` and `VOUT` output pins on the profiler hardware.

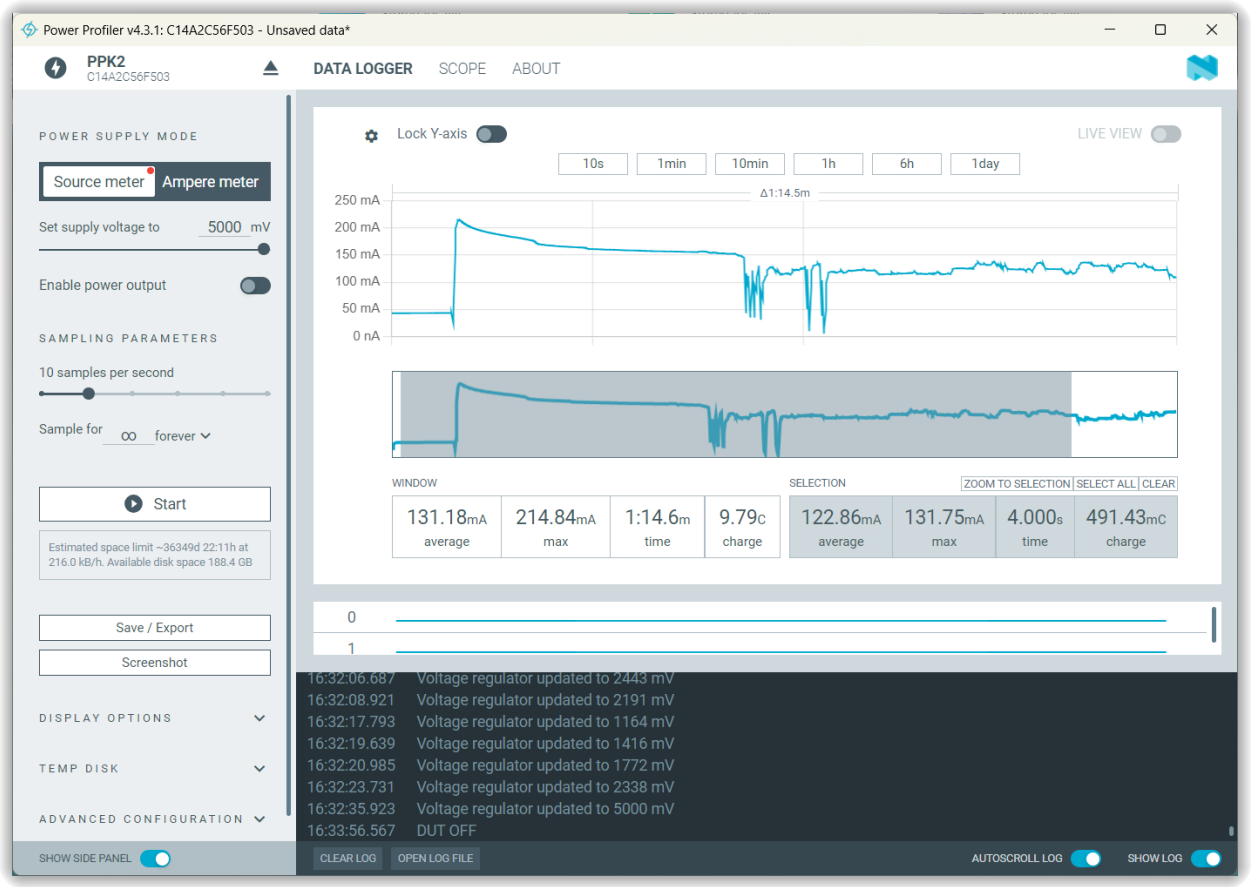


Figure 7: Example use of the *Power Profiler* app. An LED is deliberately supplied with excessive voltage, resulting in fluctuating current draw, indicating abnormal operation.

Recommendation for New Users: Before proceeding, if you are new to the power profiler hardware, we strongly recommend experimenting with it first. Try powering a simple circuit, like an LED on a breadboard, to familiarize yourself with the app's operation and capabilities before connecting the LMT IoT Shortcut hardware (as shown in Fig 8).

4.3 LMT IoT Shortcut Evaluation Board

This section guides you through connecting the IoT Shortcut Evaluation Kit hardware components and programming the LMT IoT Shortcut evaluation board (Fig. 8) with a sample firmware. This initial firmware will verify your hardware and software setup by establishing a connection to the LMT IoT Cloud using the provided credentials, but it will not yet upload any sensor data.

Connecting Power & Antenna: Before proceeding, ensure the power profiler voltage output is disabled and the programming board's power switch is set to OFF. This precaution helps prevent accidental short circuits or connection errors. Then place the LMT IoT Shortcut evaluation board onto a breadboard. Connect the provided cellular antenna to the

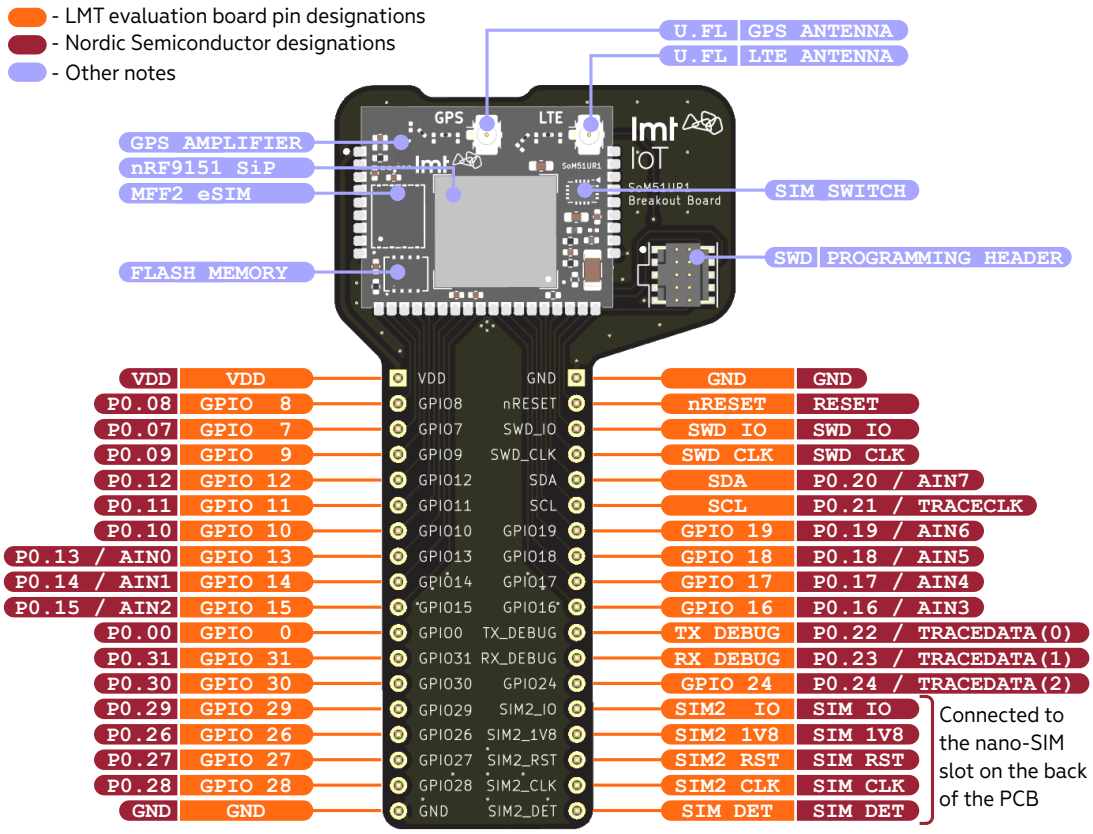


Figure 8: Pinout diagram of LMT evaluation board with reference to pin designations in nRF9151 documentation.

`LTE ANTENNA` port on the board. Then, connect the power profiler output leads: connect the profiler's `VOUT` pin to the evaluation board's `VDD` pin, and the profiler's `GND` pin to the evaluation board's `GND` pin.

Connecting the Programming Board: Next, connect the programming cable between the Nordic programming board and the LMT IoT Shortcut evaluation board. Plug one end into the `DEBUG OUT` socket on the programming board and the other end into the `PROGRAMMING HEADER` on the LMT board. Pay close attention to the orientation markings next to the header on the LMT board: the silkscreen line indicates the correct alignment for the notch on the cable plug, and the marked dot shows which side the colored strand (usually red) of the cable should face.

Powering Up: With all connections made, turn the programming board's power switch to ON and enable the 3600 mV output on the Power Profiler app. The LMT IoT Shortcut evaluation board should power up. You should observe a current consumption of approximately 4-6 mA on the Power Profiler app, indicating the circuitry is likely functioning correctly without obvious electrical faults.

Opening the Sample Project: In VS Code, open the `<sdk-path>/samples/hello_c` folder as your workspace. This firmware serves as a basic "Hello, World!" example designed solely for connectivity verification. It establishes a connection to the LMT IoT Cloud and then waits indefinitely, without uploading any data. Flashing this firmware first is a useful step to confirm that the SDK tools are correctly installed, the device credentials are valid, and the hardware is functioning properly.

Building the Firmware: Now, compile the firmware. In VS Code, open the *nRF Connect* extension view. Under the *APPLICATIONS* section, click *Add build configuration*. After a brief loading period, the build configuration menu will open. Under the *Board* setting, select the *Custom board* radio button and then choose `lmt_som_nrf9151_ns` from the dropdown list. The "ns" suffix indicates this target builds for the non-secure application domain, enabling secure boot and

firmware execution within the TrustZone non-secure region. Scroll to the bottom of the menu and click the *Generate and Build* button. This action will start the compilation of the firmware image and typically takes several minutes.

Flashing Firmware: After a successful build, the *Actions* section will appear in the *nRF Connect* extension view. Under this section, click the *Flash* action. This will program the compiled firmware image onto the LMT IoT Shortcut evaluation hardware.

Viewing Debug Output: Once flashing is complete, find the *Connected Devices* section at the bottom of the *nRF Connect* view. Locate your board (it should appear automatically) and click the plug icon next to *RTT* (see Fig. 9). A dropdown menu will appear at the top of the VS Code window; select *Search for the RTT control block automatically* (see Fig. 10). This action opens the RTT (Real-Time Transfer) console, where you can view debug output messages printed by the firmware running on the LMT IoT Shortcut hardware.

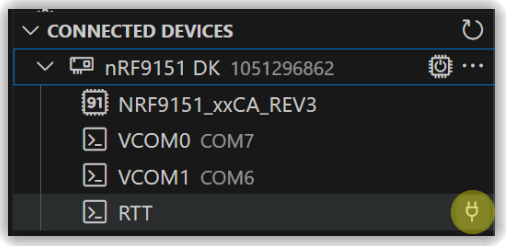


Figure 9: Connecting the RTT console

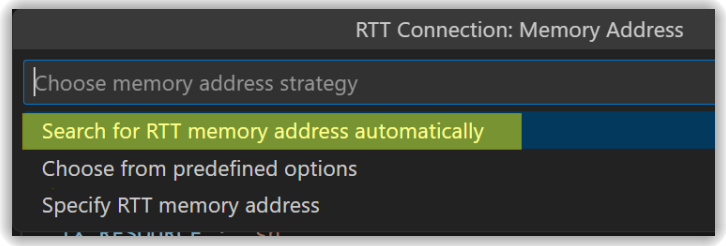


Figure 10: Searching automatically for RTT memory address.

Interpreting Debug Output: When observing the RTT console output after flashing (Fig. 11), you might initially see some errors. These typically indicate that valid time and date information has not yet been received from the cellular network. The device will then search for available cell towers and connection bands, **which may take a few minutes during the first connection**. Successful network registration is indicated by messages showing signal quality and the assigned IP address in the RTT logs. Following registration, the device will attempt to connect to the LMT IoT Cloud. In the `hello_c` example (which doesn't upload sensor data; the connection is rudimental), a successful connection attempt is confirmed by a message `Message ID <...> sent`. Subsequent network registrations are usually much faster, as the SIM card stores information about the last successful connection. You can optimize the firmware to lock onto specific cellular bands, which can accelerate the initial connection process by eliminating the need for the modem to scan all available bands.

5 Creating Firmware for Sensor Measurement and Cloud Upload

5.1 Example Description

The previous `hello_c` example confirmed that your full development environment (Nordic SDK and LMT IoT Shortcut SDK) is correctly set up, the hardware is functional, cellular connectivity works, and the LMT IoT Cloud is accessible. Now, you are encouraged to use the IoT Shortcut Evaluation Kit to develop your own IoT device prototype. To help you get started and illustrate a typical firmware structure, this section details a practical use case: the `ek_sensor_example` project, located in the `lmtSDK/samples` folder.

This example utilizes an Adafruit BMP390 (temperature and pressure) sensor and an LIS3DH (accelerometer) sensor, connected to the evaluation board via the I2C digital bus. This demonstrates how to integrate digital sensors. Additionally, to showcase analog I/O capabilities, we include a potentiometer (analog input) controlling the brightness of an LED via Pulse-Width Modulation (PWM, analog output). Fig. 12 shows the assembled prototype on a breadboard.

```
[00:00:00.257,263] <inf> FS_LOGGER: Partition 0 size: 4194304 bytes
[00:00:01.128,479] <inf> FS_LOGGER: /lfs is mounted
[00:03:00.258,697] <wrn> lte_lc: Connection evaluation failed with reason: 1
[00:03:00.258,728] <err> APP_LOG: Failed to get network connection quality, error code: 1
[00:03:00.258,728] <wrn> date_time: Valid time not currently available
[00:03:00.259,094] <wrn> date_time: Valid time not currently available
...
[00:03:32.073,577] <inf> APP_LOG: LTE cell changed: Cell ID: 2491420, Tracking area: 40171
[00:03:32.137,542] <inf> APP_LOG: RRC mode: Connected
[00:03:35.448,852] <inf> APP_LOG: Network registration status: Connected - roaming
[00:03:35.450,805] <inf> APP_LOG: PSM parameter update: TAU: 86400 s, Active time: 6 s
[00:03:35.607,299] <inf> APP_LOG: IPv4 Address found 18.194.48.99
[00:03:35.986,114] <inf> APP_LOG: Socket connected
[00:03:36.163,879] <inf> APP_LOG: RRC mode: Idle
[00:03:38.487,701] <inf> APP_LOG: Signal quality: SNR: 35, RSRP: 59, RSRQ: 24
...
[00:03:40.724,609] <inf> APP_LOG: Message ID 53089 sent
```

Figure 11: Output of the RTT console for the first minutes of using IoT Shortcut hardware with `hello_c` example.

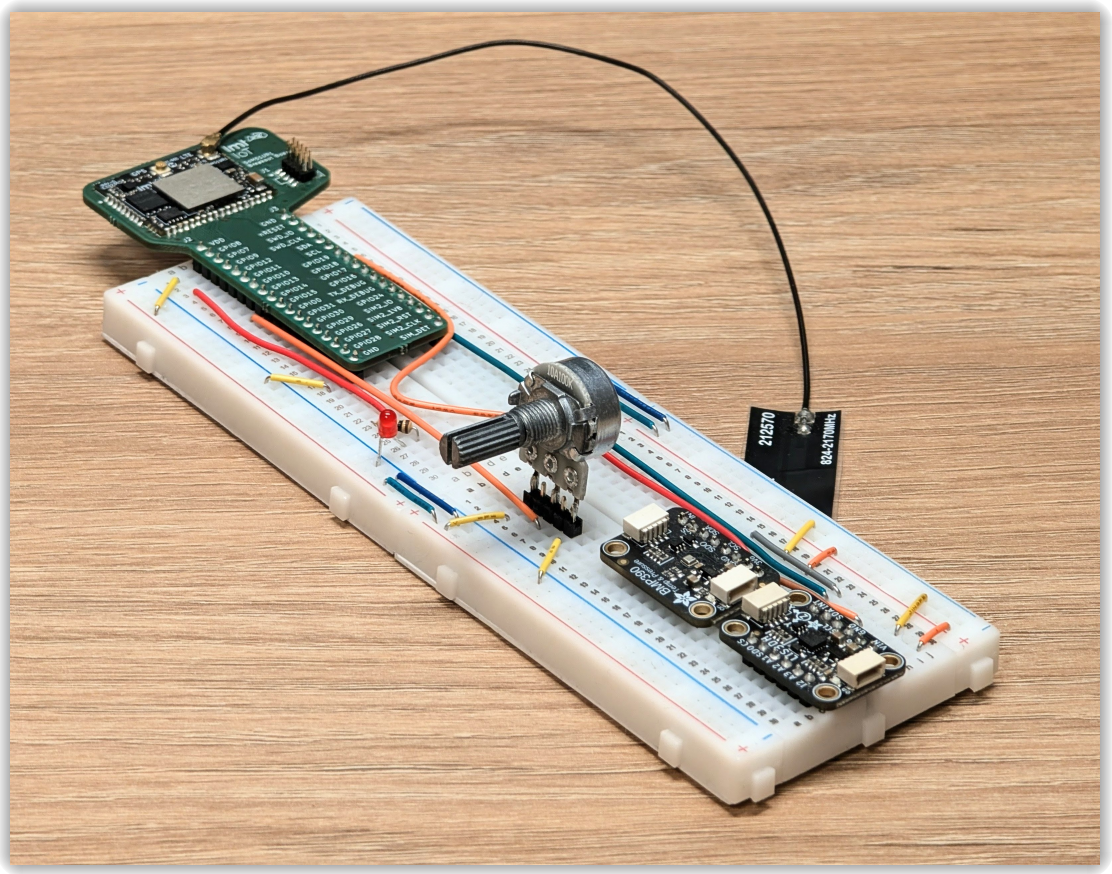


Figure 12: Photograph of the breadboard implementation of the circuitry shown in Fig. 13.

The goal is to upload sensor data to the LMT IoT Cloud **every five minutes**. The potentiometer's position will directly control the LED's brightness (PWM output). This brightness level (representing the potentiometer position) will be uploaded every five minutes along with the temperature and pressure readings from the BMP390 sensor, which are typical periodic measurements.

Reading accelerometer data only momentarily every five minutes is often not useful. Instead, we will configure the LIS3DH accelerometer to run continuously. The firmware will track the maximum total acceleration experienced during each five-minute interval and upload only this maximum value. Since the LIS3DH measures acceleration along three axes (a_x, a_y, a_z) separately, we need to calculate the magnitude of the acceleration vector ($|a|$) using the following formula before determining the maximum. This calculated maximum magnitude $|a|$ will be uploaded every five minutes.

$$|a| = \sqrt{a_x^2 + a_y^2 + a_z^2} \tag{1}$$

5.2 Connecting Peripherals and Preparing `.overlay` file

First, assemble the peripherals on the breadboard as shown in Fig. 12, following the schematic diagram in Fig. 13.

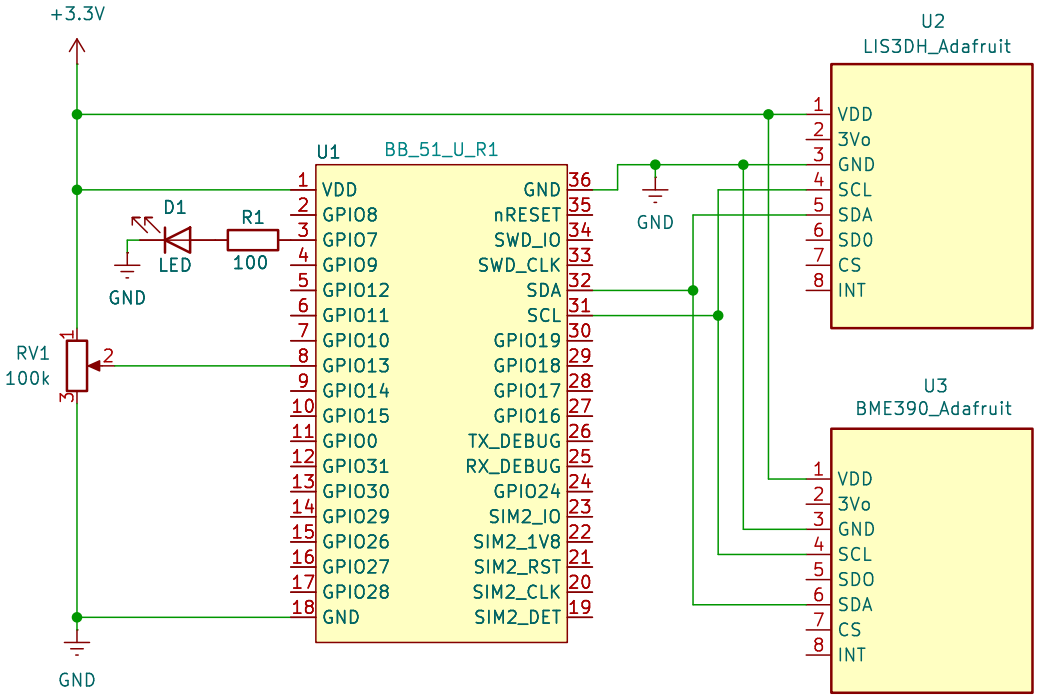


Figure 13: Schematic diagram of digital and analog input/output devices connected to the LMT evaluation board used as an example in this tutorial.

The first step in firmware configuration is enabling and defining the necessary hardware pins using a Zephyr overlay file. For this project, the relevant file is `ek_sensor_example/lmt_som_nrf9151_ns.overlay`. This file assigns functions to specific GPIO pins. For instance, the potentiometer is connected to `GPIO 13`, which corresponds to the `AIN 0` (Analog Input 0) function (refer to Fig. 8). The overlay file snippet in Fig. 14 shows how the ADC channel 0 is enabled and configured for this pin. Similar entries are required for the I2C pins (SDA, SCL), the PWM output pin for the LED, and any other peripherals used.

```
&adc {
    #address-cells = <1>;
    #size-cells = <0>;
    status = "okay";
    channel@0 {
        reg = <0>;
        zephyr,gain = "ADC_GAIN_1_6";
        zephyr,reference = "ADC_REF_INTERNAL";
        zephyr,acquisition-time = <ADC_ACQ_TIME_DEFAULT>;
        zephyr,input-positive = <NRF_SAADC_AIN0>; /* P0.13 */
        zephyr,resolution = <12>; //0-4095
    };
};
```

Figure 14: Partial contents of the `ek_sensor_example/lmt_som_nrf9151_ns.overlay` file, enabling `AIN0` (`P0.13`) for the potentiometer.

5.3 Main Operational Loop

The main operational logic resides in the `src/main.c` file. As shown in Fig. 15, the `main()` function begins by setting constants, such as the data upload period (5 minutes in this case). It then calls initialization functions for the LMT SDK (`lmtInit()`) and the specific application hardware/tasks (`appInit()`). Finally, it enters an infinite loop (`while(1)`) that periodically reads sensor data (`readAndRecordData()`) based on the `DATA_READ_PERIOD` constant, sleeping between reads to conserve power.

```
int main(void)
{
    ...
    // Set uplink period in minutes(how often data is sent to server)
    setUplinkTimeout(5);
    // Initialize the LMT SDK (sets up core system functions)
    lmtInit();
    // Initialize the user application (sensors, threads, etc.)
    appInit();

    // Main loop: read and record data every DATA_READ_PERIOD seconds
    while(1)
    {
        // Wait for the next data read period (adjusted for time spent processing)
        k_sleep(K_SECONDS(DATA_READ_PERIOD - (end_tick - start_tick) / 1000));

        start_tick = k_uptime_get();
        readAndRecordData();
        end_tick = k_uptime_get();
    }

    return 0;
}
```

Figure 15: The main loop in `src/main.c` controls periodic data reading.

5.4 Initializations and Parallel Threads

Examining the `appInit()` function within `src/main.c` (Fig. 16) reveals how background operations are managed using Zephyr threads. Separate threads are created for handling the accelerometer readings (`lis3dhTask`) and the potentiometer/PWM control (`adcPwmTask`). This allows these tasks to run concurrently with the main data reading and upload loop.

```
void appInit(void)
{
    ...

    // Start the LIS3DH accelerometer task in its own thread
    k_thread_create(&lis3dh_thread_data, lis3dh_stack_area,
    K_THREAD_STACK_SIZEOF(lis3dh_stack_area), lis3dhTask, NULL, NULL, NULL,
    PRIORITY_LIS3DH, 0, K_NO_WAIT);

    // Start the ADC/PWM task in its own thread
    k_thread_create(&adc_pwm_thread_data, adc_pwm_stack_area,
    K_THREAD_STACK_SIZEOF(adc_pwm_stack_area), adcPwmTask, NULL, NULL, NULL,
    PRIORITY_ADC_PWM, 0, K_NO_WAIT);

    ...
}
```

Figure 16: The `appInit` function creates separate threads for continuous accelerometer monitoring and ADC/PWM control.

Delving into the `src/lis3dh.c` file, the `lis3dhTask` function (Fig. 17) implements the continuous monitoring logic discussed earlier. Inside its `while(1)` loop, it repeatedly reads the raw x, y, z acceleration values, calculates the magnitude (vector modulus, Eq. 1), and updates the `max_accel` variable if the current reading is higher than the previously recorded maximum. The loop includes a short sleep (`k_sleep(K_MSEC(1))`) to control the sampling rate. The `max_accel` variable is then read by the main loop during the periodic `readAndRecordData()` call.

5.5 Uploading Data

Within the `readAndRecordData()` function in `main.c` (Fig. 18), sensor values are assigned to specific indices within the global `Data` array. Crucially, these array indices must exactly match the track numbers configured previously in the LMT IoT Cloud (shown in Fig. 5). For example, `Data[TEMPERATURE_INDEX]` (where `TEMPERATURE_INDEX` is defined as 1) holds the temperature value, corresponding to Track 1 in the cloud setup.

Note that the CoAP protocol stack currently used by the LMT SDK primarily supports integer data uploads. To preserve precision for floating-point sensor readings (like temperature or acceleration), the firmware multiplies the values before storing them as integers in the `Data` array (e.g., temperature is stored in centi-degrees Celsius, acceleration in cm/s^2). The corresponding scaling factor (multiplier) must then be configured for each track in the LMT IoT Cloud (as shown in Fig. 5) so that the LMT IoT Cloud can correctly interpret and display the data in the desired final units. Ensure the units defined in the firmware comments align with the final units configured in the LMT IoT Cloud after applying the multiplier.

5.6 Firmware in Action

After flashing the `ek_sensor_example` firmware onto the IoT Shortcut hardware (using the same method described in Section 4.3), observe the RTT console output. You will see periodic sensor readings logged, followed by messages indicating successful data uploads to the LMT IoT Cloud, as shown in the example log snippet in Fig. 19.

Log in to the LMT IoT Cloud to view your uploaded data. The platform automatically applies the multipliers you configured during device setup, ensuring the stored data is correctly scaled and ready for visualization. Fig. 20 shows an example screenshot of the acceleration data plotted over time within the LMT IoT Cloud interface. Now that data is successfully reaching the cloud, you can integrate these measurements into your final application — be it a mobile app, web application, or another system — using the provided REST API. **The complete API documentation is available here.**

```c
void lis3dhTask(void *arg1, void *arg2, void *arg3)
{

    ...

    while(1)
    {
        readLis3dh(); // Read new data into accel array
        // Convert sensor values to floating-point (in m/s^2)
        x = accel[0].val1 + accel[0].val2 / 1000000.0f;
        y = accel[1].val1 + accel[1].val2 / 1000000.0f;
        z = accel[2].val1 + accel[2].val2 / 1000000.0f;
        // Calculate the magnitude of the acceleration vector
        modulus = sqrtf(x * x + y * y + z * z);

        // Update the maximum if this reading is higher
        if(modulus > max_accel)
        {
            max_accel = modulus;
        }

        // Reset values for next iteration
        modulus = 0;
        x       = 0;
        y       = 0;
        z       = 0;
        // Sleep for 1 ms (1 kHz data rate)
        k_sleep(K_MSEC(1));
    }
}
```

Figure 17: The accelerometer thread continuously reads data, calculates magnitude, and stores the maximum value.

```c
// Indices for storing sensor data in the Data array
#define BRIGHTNESS_INDEX   0 // Potentiometer (knob) position
#define TEMPERATURE_INDEX  1 // Temperature from BMP390
#define PRESSURE_INDEX     2 // Pressure from BMP390
#define ACCELARATION_INDEX 3 // Maximum acceleration from LIS3DH

...

void readAndRecordData(void)
{
    ...
    Data[BRIGHTNESS_INDEX]  = brightness;  // Potentiometer position (0-100%)
    Data[TEMPERATURE_INDEX] = temperature; // Temperature (°C x 100)
    Data[PRESSURE_INDEX]    = pressure;    // Pressure (Pa)
    Data[ACCELARATION_INDEX] = (int32_t)(accelaration_max * 100); // Acc. (m/s^2 x 100)
    ...
}
```

Figure 18: Assigning scaled sensor data to the `Data` array. Indices must match the LMT IoT Cloud device setup.

```
...
[00:00:03.386,444] <inf> APP_LOG: LTE cell changed: Cell ID: 2491420, Tracking area: 40171
[00:00:03.449,859] <inf> APP_LOG: RRC mode: Connected
[00:00:07.561,187] <inf> APP_LOG: Network registration status: Connected - roaming
[00:00:07.563,171] <inf> APP_LOG: PSM parameter update: TAU: 86400 s, Active time: 6 s
[00:00:07.713,562] <inf> APP_LOG: IPv4 Address found 18.194.48.99
[00:00:08.177,947] <inf> APP_LOG: Socket connected
[00:00:08.178,466] <inf> APP_LOG: All system parts initialized successfully
...
[00:05:00.270,751] <inf> APP_LOG: Pot: 15%, Temp: 21.97 C, Pressure: 100698 Pa, Accel max:
    16.19 m/s^2
[00:05:11.498,596] <inf> APP_LOG: Signal quality: SNR: 32, RSRP: 58, RSRQ: 16
..
[00:05:11.597,137] <inf> APP_LOG: Going to send message ID 20043, size 63
[00:05:12.126,220] <inf> APP_LOG: RRC mode: Connected
[00:05:13.060,882] <inf> APP_LOG: No payload in response
[00:05:13.061,004] <inf> APP_LOG: Received response with matching ID: 20043
[00:05:13.061,096] <inf> APP_LOG: Message ID 20043 sent
```

Figure 19: Output of the RTT console for the first minutes of using IoT Shortcut hardware with sensor I/O example.

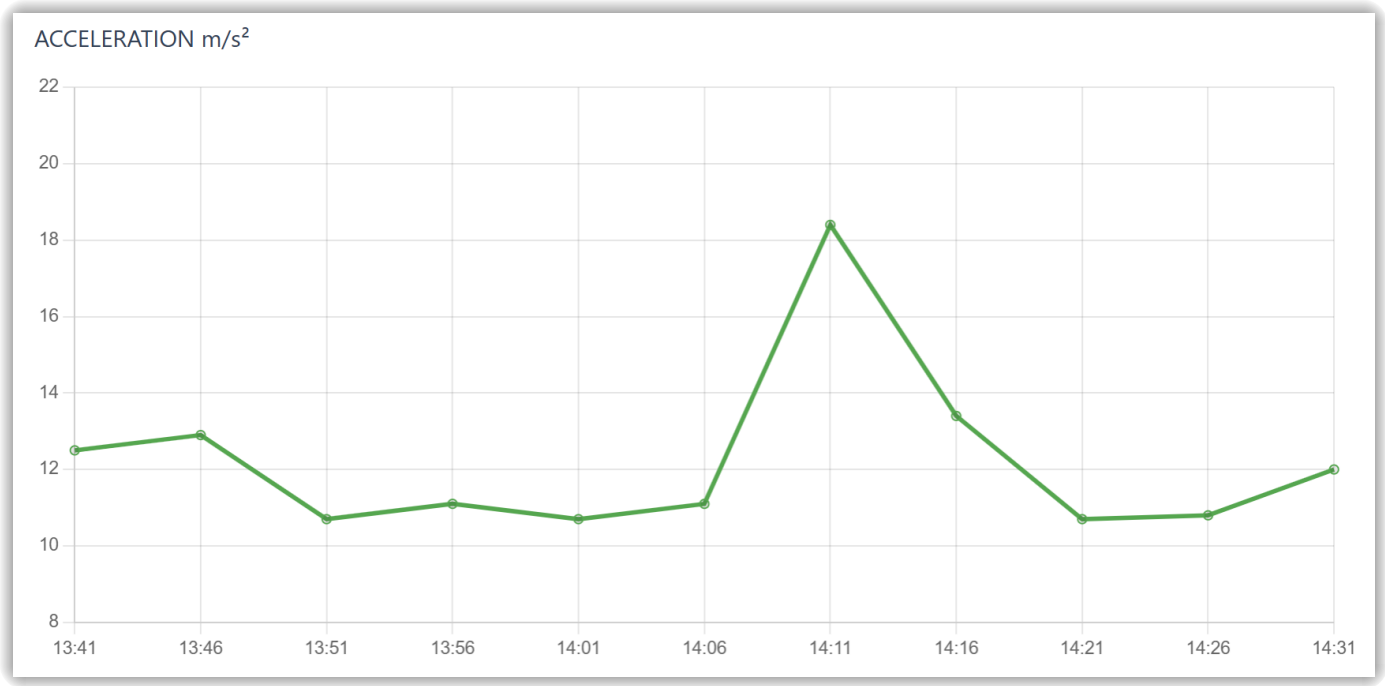


Figure 20: Screenshot showing data uploaded from the IoT Shortcut Evaluation Kit and displayed in the LMT IoT Cloud.

6 Use of nano-SIM card slot

IoT Shortcut Evaluation Kit already includes an MFF2 SIM on the PCB for global connectivity. However, if for any reason there is a need to use a different connectivity provider, the IoT Shortcut Evaluation Kit breakout board contains a nano-SIM slot. Fig. 22 shows how to insert this SIM card. Ensure that the SIM card's service plan supports LTE-M connectivity.

To switch to the nano-SIM slot instead of the on-board MFF2 SIM, the firmware edit shown in Fig. 21 is necessary. By default, the provided code examples use the MFF2 SIM populated on the PCB. However, if using the nano-SIM slot is desired, the `setActiveSim(ACTIVATE_ESIM)` function must be called before invoking `lmtInit()`, i.e. addition of code line 4 as seen in Fig. 21. In this context, the abbreviation `ESIM` stands for *external* SIM.

```
1   int main(void)
2   {
3       // Set SIM switch to use the
    nano-SIM slot
4       setActiveSim(ACTIVATE_ESIM);
5
6       lmtInit();
7       usrInit();
8
9       ...
10  }
11
```

Figure 21: Code line required to use the nano-SIM slot.

Figure 22: Inserting nano-SIM in the provided SIM slot.