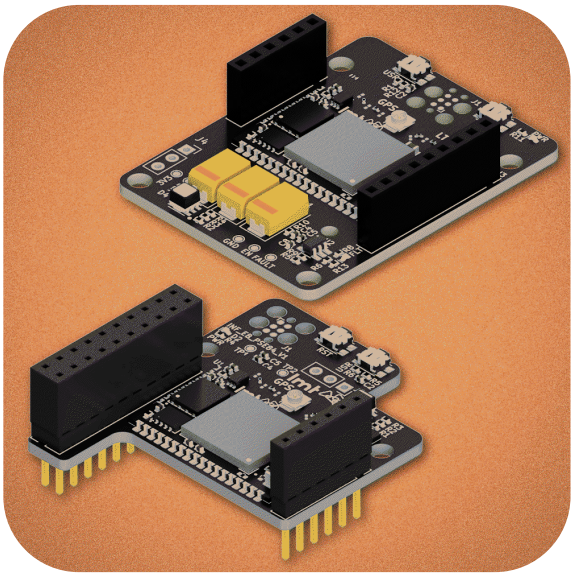
LMT Cellular LPWAN Extension Boards for Infineon PSOC AI Kits



Designed to mount directly onto the Infineon Edge AI Kits, the LMT extension boards provide seamless cellular connectivity for Edge AI applications. Powered by the LMT IoT Shortcut, each board includes a global SIM, native cloud integration, and works with a dedicated SDK featuring "serial-to-cloud gateway" functionality. This complete ecosystem allows developers to effortlessly transmit AI-processed data from the field to the cloud with minimal firmware development.

1 Ordering Information

LMT product number	Product name / Compatibility
`EBI151UFI`	LMT Cellular LPWAN Extension Board for Infineon PSOC Edge E84 AI Kit
`EBI251UFI`	LMT Cellular LPWAN Extension Board for Infineon PSOC 6 AI Kit

- Both product variations include a Molex 824–2170 MHz flexible cellular antenna with a U.FL connector. This antenna must be connected to the port labeled `LTE` before powering on the LMT board.
- The product packaging additionally includes unsoldered pin headers. These headers must be soldered onto the Infineon PSOC AI Kit to ensure proper mechanical mounting of the LMT extension board. See Section 6 for details.

2 Technical Specifications

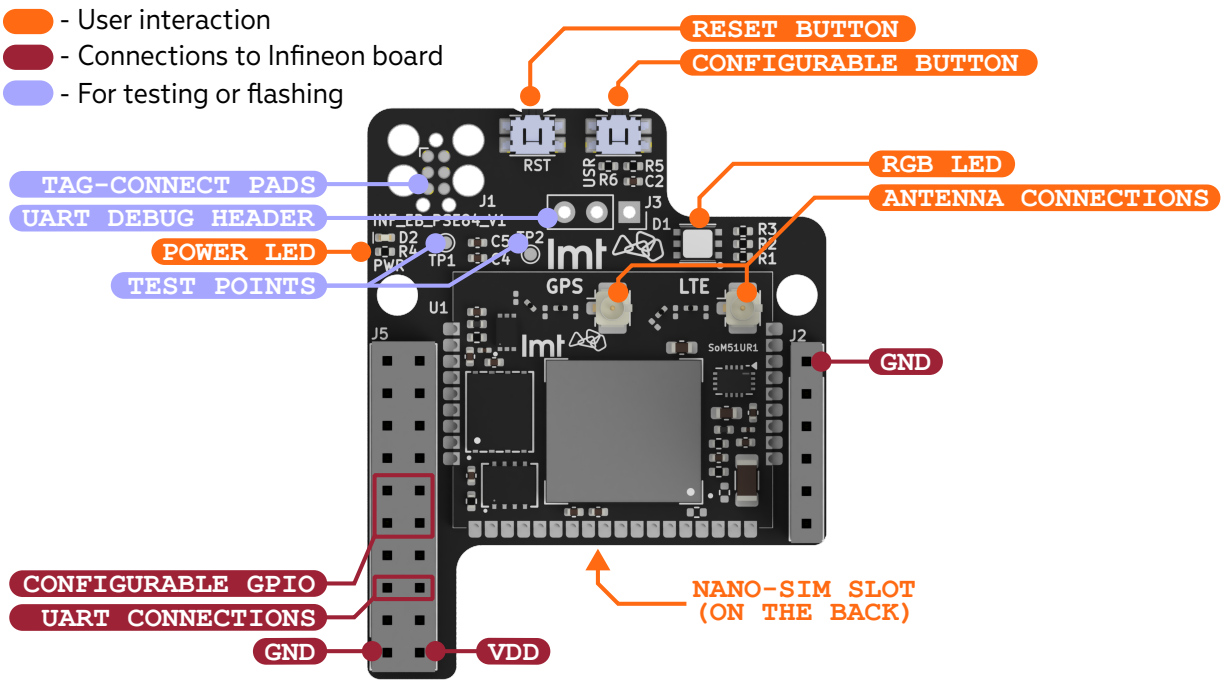
Main modem/MCU	Nordic Semiconductor nRF9151
Connectivity	LTE-M (Cat-M1) or NB-IoT (Cat-NB1/NB2)
SIM card configuration	Global MFF2 SIM included, nano-SIM slot provided
Deep sleep current consumption	<8 µA at 3.6 V supply
Single upload energy consumption	90 mJ or 25 µWh (100 B payload, excellent signal)

- These extension boards come pre-flashed with firmware and are pre-provisioned to the user's LMT IoT Cloud account. However, the user must implement the supported "serial-to-cloud gateway" commands within the host Infineon PSOC firmware. For integration details, refer to the LMT IoT SDK documentation and Section 6 of this document.
- The LMT extension boards support firmware-over-the-air (FOTA) updates. Please note that this functionality currently applies only to the LMT module and cannot be used to update the firmware of the host Infineon PSOC AI kit.

- The modules include an integrated global SIM card, providing LTE-M and NB-IoT connectivity in most countries worldwide. For specific coverage details, please contact the LMT IoT team.
- Users may opt to use a custom connectivity provider via the provided nano-SIM slot. Refer to the LMT IoT SDK documentation for instructions on how to toggle the active SIM.

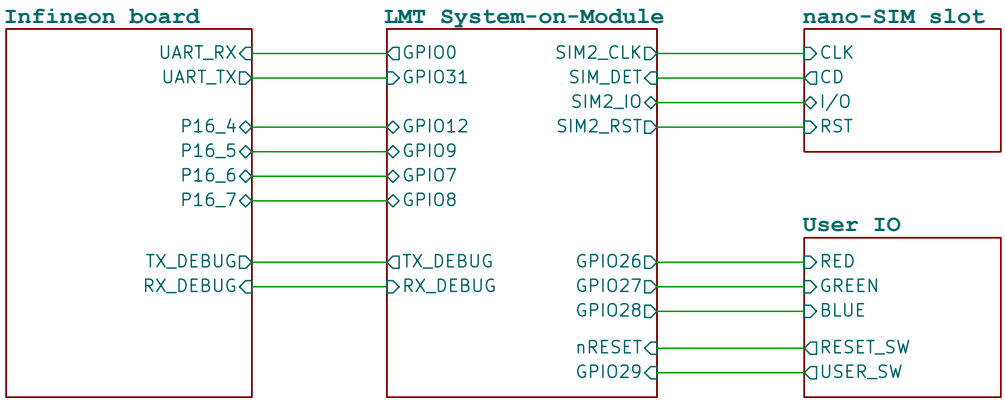
3 Extension Board for Edge E84 AI Kit (EBI151UFI)

3.1 Overview

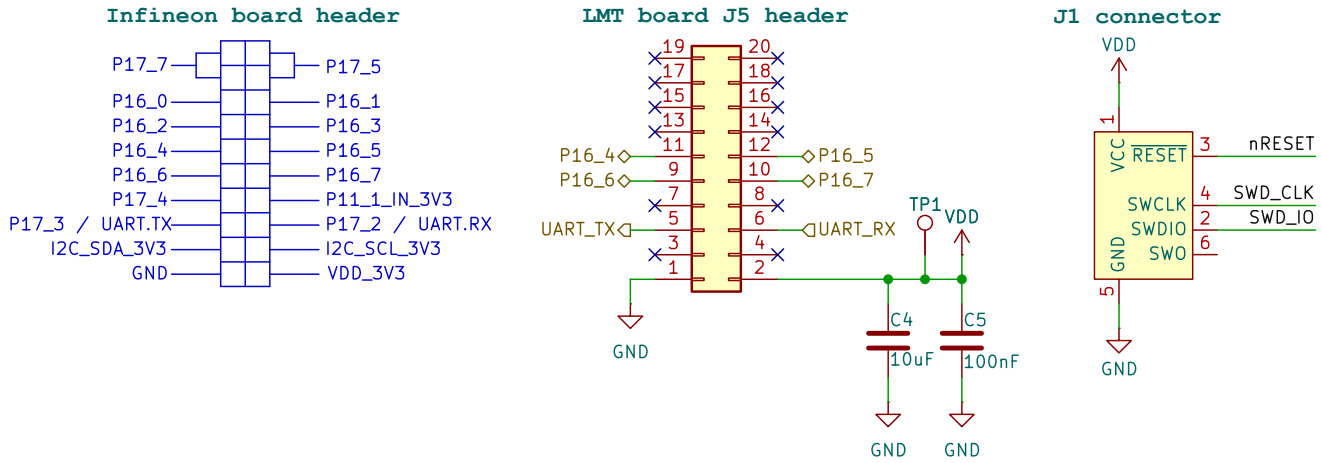


3.2 Design Notes

- The PCB is specifically shaped to prevent radio interference with the Wi-Fi antenna and radar sensor on the host Infineon board. Ensure this clearance area remains free of other metallic components.
- The board pinout is detailed below. For a comprehensive description of the LMT SoM pin designations, refer to the full datasheet. All unmarked pins on the headers serve strictly as pass-through connections and are electrically isolated from the LMT board.

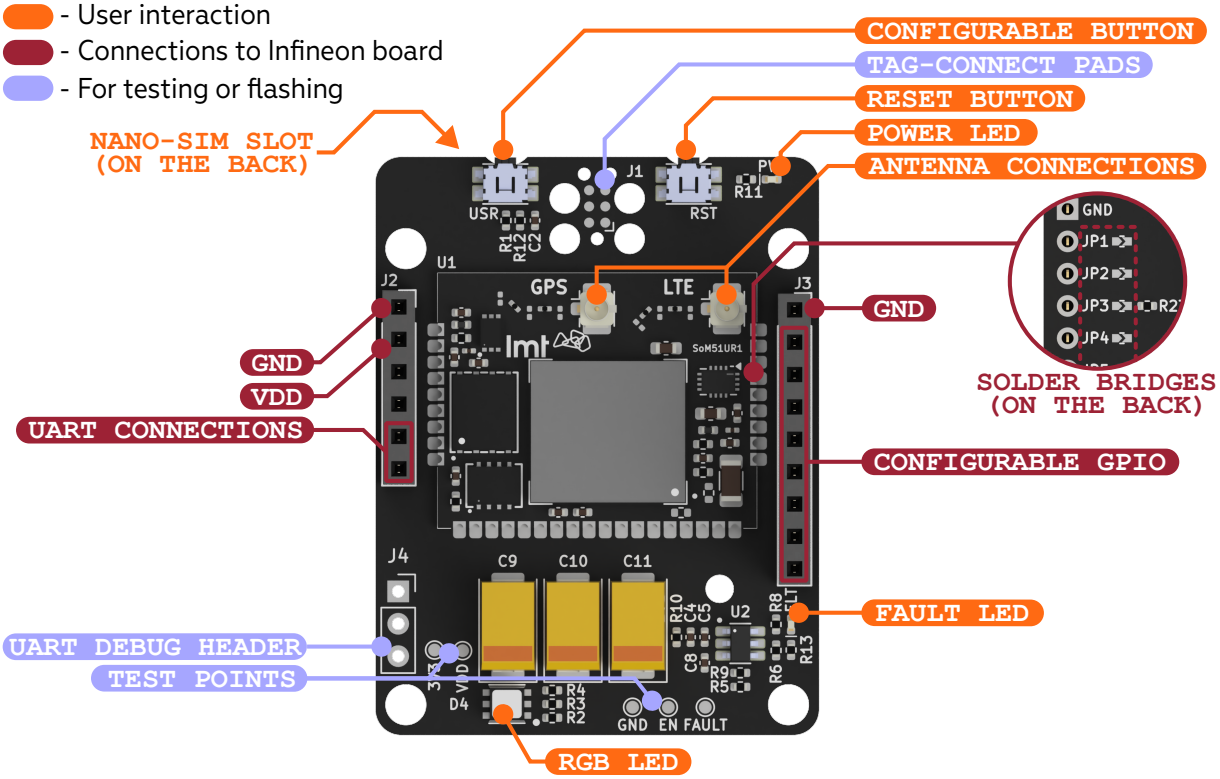


- Dedicated test points are routed to VDD and GND, allowing for convenient voltage level measurements without the need to probe the pin headers directly. Additionally, a power indicator LED is connected across these lines to visually confirm the active power state.
- Tag-Connect pads are routed to the nRESET, SWDIO, SWDCLK, VDD, and GND pins of the LMT module to facilitate custom firmware flashing and debugging. For programming interfaces, we recommend cables such as the TC2030-CTX or the space-saving TC2030-CTX-NL.



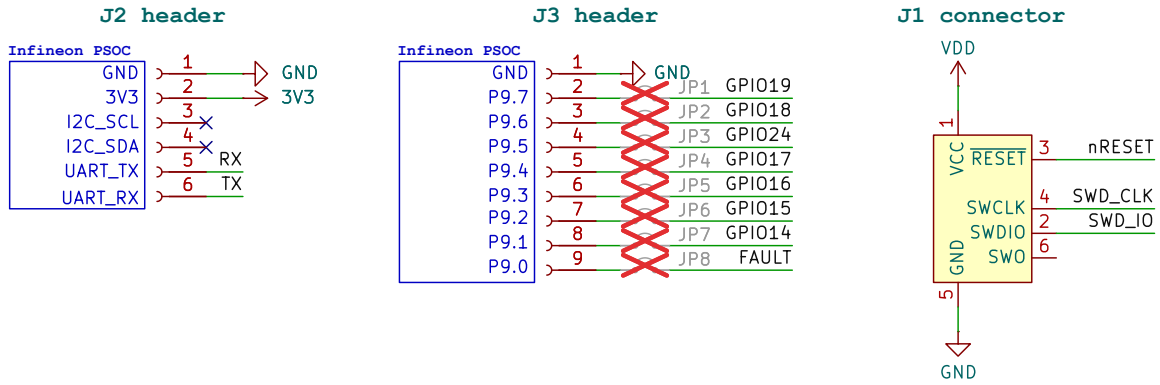
4 Extension Board for PSOC 6 AI Kit (EBI251UFI)

4.1 Overview



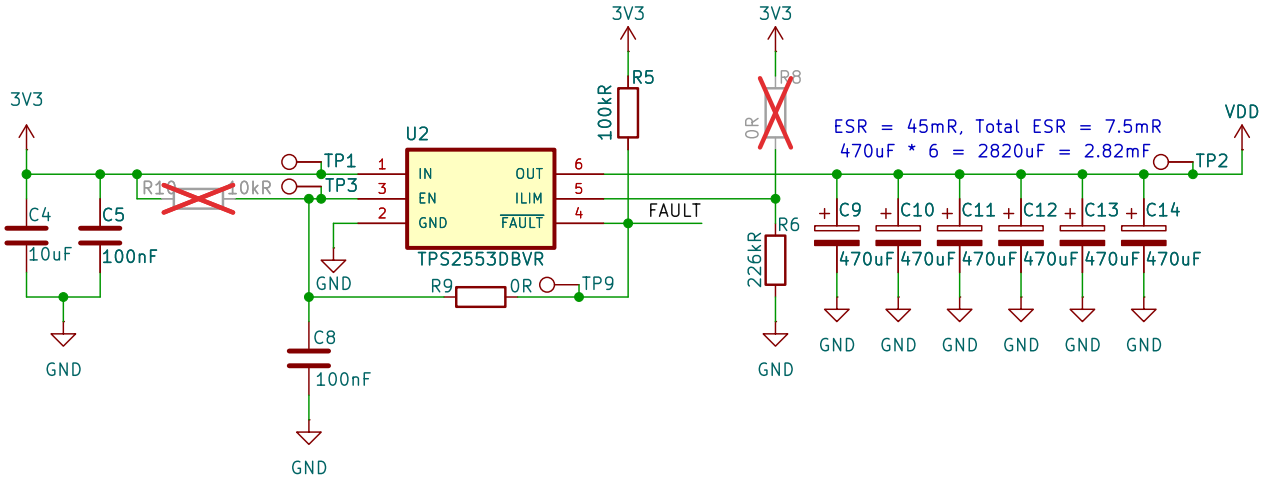
4.2 Design Notes

- The board pinout is detailed below. For a comprehensive description of the LMT SoM pin designations, refer to the full datasheet. All unmarked pins on the headers serve strictly as pass-through connections and are electrically isolated from the LMT board. However, bridging the designated solder jumpers (JP1–JP8) will connect some of the pins to specific I/O on both the Infineon and LMT modules for custom applications.



- Tag-Connect pads are routed to the nRESET, SWDIO, SWDCLK, VDD, and GND pins of the LMT module to facilitate custom firmware flashing and debugging. For programming interfaces, we recommend cables such as the TC2030-CTX or the space-saving TC2030-CTX-NL.

- Because the host Infineon board can supply a maximum of 120 mA to external devices, the LMT board's power circuitry incorporates a Texas Instruments TPS2553 power-distribution switch to cap the continuous current draw at this level. This protection prevents overloading the Infineon board's power supply. Such an overload could otherwise occur during extended cellular transmissions (e.g., sending large data payloads), which keeps the modem in active transmit mode long enough to deplete the onboard buffering capacitors designed to absorb transient current spikes.

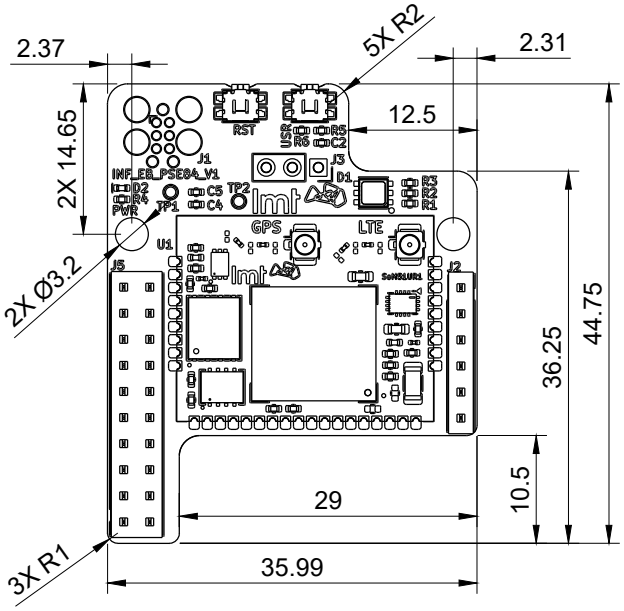


- The power limiter also provides short-circuit protection. In the event of a short circuit, the voltage supply to the module is immediately disconnected. When this occurs, the fault LED (FLT) will illuminate, and the corresponding test point (FAULT) will be asserted. After a brief delay, the circuit will automatically re-trigger the enable (EN) pin of the power switch to attempt restoring power. To make this fault status readable by the host Infineon board, bridge the JP8 solder jumper, which routes the FAULT signal to the Infineon board's P9.0 pin. For power supply debugging, two key test points are provided: the 3V3 test point (indicating the raw supply voltage from the Infineon board) and the VDD test point (confirming the active voltage output from the power limiter to the LMT SoM).

5 Mechanical Dimensions

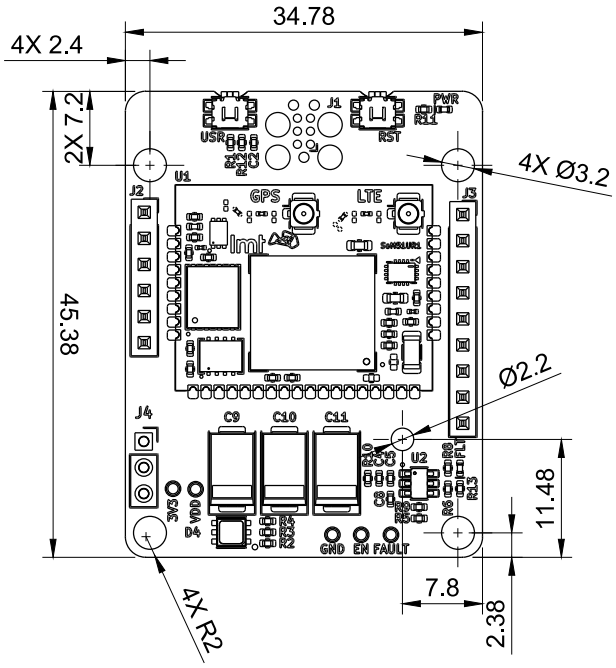
5.1 Board for Edge E84 AI Kit (EBI151UFI)

(Download .step file)



5.2 Board for PSOC 6 AI Kit (EBI251UFI)

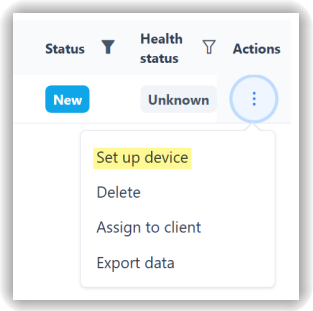
(Download .step file)

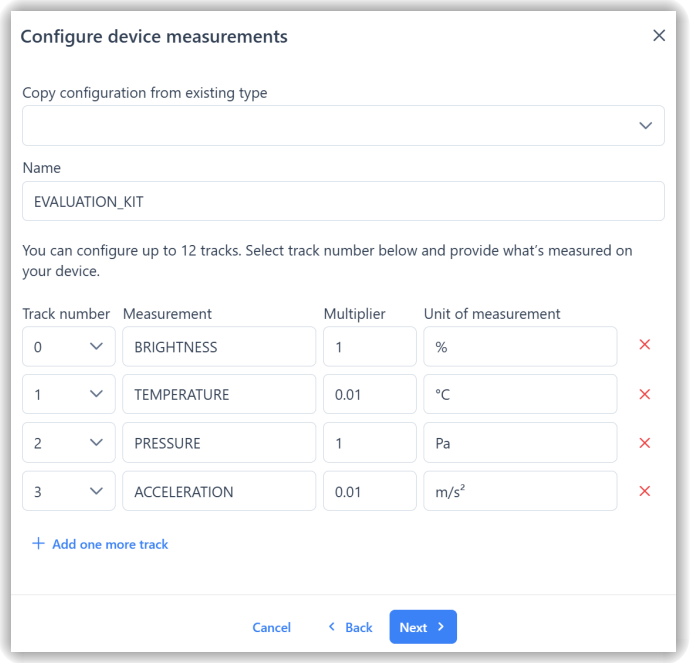


6 Usage and Integration Guidelines

6.1 Cloud and Firmware Configuration

1. **Verify Cloud Registration:** Verify the LMT board's presence on the LMT IoT Cloud. LMT personnel will have already pre-provisioned the shipped device to your user account.
2. **Configure Data Payload:** Using the dedicated configuration wizard on the cloud platform, define the structure of the data you intend to transmit as seen below (i.e., configure the CoAP protocol tracks for incoming messages).

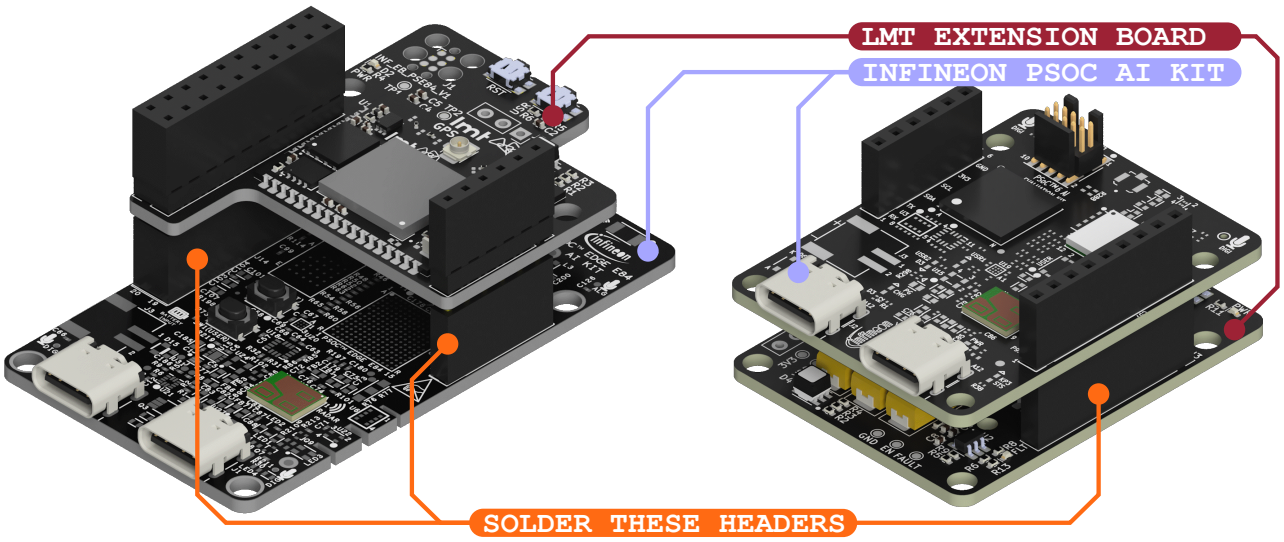




3. **Develop Host Firmware:** Develop the host Infineon firmware. Follow the LMT IoT SDK documentation guidelines to implement the "serial-to-cloud gateway" functionality, ensuring the transmitted data packet format matches the configuration defined in the cloud.

6.2 Hardware Mounting and Initialization

1. **Prepare the Extension Board:** Connect the provided U.FL cellular antenna to the LMT board. If additional GPIO routing is required for custom applications (applicable to the `EBI251UFI`), bridge the necessary solder pads before mounting. Note that the pre-flashed firmware supports fundamental data transfer commands as detailed in the LMT IoT SDK documentation and any GPIO pins connected using the solder bridges require custom firmware on the LMT board.
2. **Prepare the Infineon PSOC AI Kit:** Solder the included female pin headers onto the host Infineon PSOC AI Kit, as illustrated in the figure below. This ensures proper mechanical support for both sides of the LMT board and prevents physical strain on the primary header connections.
3. **Mount the Extension Board:** Flash your compiled application firmware onto the host Infineon board. Carefully mount the LMT extension board onto the Infineon host kit, as illustrated in the figure below. Apply power to the system and verify successful data uploads via the LMT IoT Cloud dashboard.



4. **If troubleshooting is necessary:** Utilize the UART debug header to inspect serial communication between the host Infineon board and the LMT module.
5. **If modifications to the LMT module's onboard firmware are required:** (e.g., to define custom user button actions or specific RGB LED behaviors), utilize the Tag-Connect pads for reprogramming.

7 AT Command Set

To communicate with the LMT Cellular LPWAN Extension Board, the host Infineon board UART should be configured to use a **115200 bps** baud rate in **8N1** mode. All commands must be prefixed with `AT+` and terminated with carriage return and newline characters (`<CR><NL>`).

7.1 Error Handling

There are two categories of errors encountered during incoming command processing: parsing errors and execution errors.

7.1.1 Parsing Errors

The incoming command stream is evaluated against the specific syntax described below. In the case of an inconsistency, the stream is rejected. Malformed commands are generally dropped (e.g., `AT+COLUMN=1,2,3,4,5,6,7,8,9,A<CR><NL>` will be ignored). If parameters match the syntax but fall out of acceptable ranges, the modem will report an error. Note: If an input stream becomes malformed, valid sub-strings might still execute (e.g., `AT+COLUMN AT+UPLOAD=1<CR><NL>` will execute as `AT+UPLOAD=1<CR><NL>`).

7.1.2 Execution Errors

During the execution of a properly formed command, the modem reports errors using two mechanisms:

- **Text Reporting (`<ERROR>`):** The modem outputs a message starting with `ERROR:` followed by descriptive text and an error code. This mechanism is used for all internally encountered errors (e.g., no connection, low battery), even if not directly triggered by the specific command.
- **Error Code Reporting:** For direct execution failures, the command output will return a negative integer representing the specific error code.

7.2 Data Storage

7.2.1 `COLUMN`

Syntax: `AT+COLUMN=i1,i2,i3,i4,i5,i6,i7,i8,i9,i10<CR><NL>`

Records a given measurement set into the internal data structure for later upload. The internal structure can hold up to 50 records if the data fits in the CoAP packet. When the data payload exceeds a single CoAP message, it is enqueued on the message stack. If the modem is initialized and connected, the data packet is uploaded immediately.

Parameters: `i1` … `i10` (32-bit integers). Do not skip commas for empty values (e.g., `COLUMN=1,0,-3,,,,,,`). Empty values default to 0.

Return: A positive number representing the remaining column space in the current CoAP packet. Returns `<ERROR>` on failure.

7.2.2 `ENCODEMESSAGE`

Syntax: `AT+ENCODEMESSAGE<CR><NL>`

Encodes the uplink message using the nanopb library. It does not enqueue the message or clear the measurements array. Use `GETENCODEDMSGLEN` to retrieve the encoded CoAP payload size.

Return: `OK` on success; negative error code on failure.

7.2.3 GETENCODEDMSGLEN

Syntax: AT+GETENCODEDMSGLEN<CR><NL>

Return: A positive number for encoded message size in bytes, or OK for zero. Negative error code on failure.

7.2.4 SETPERIOD

Syntax: AT+SETPERIOD=n<CR><NL>

Sets the measurement periodicity. Add the period first, then measurements. Duplicate periodicity entries are ignored.

Parameters: n (Periodicity in seconds).

Return: OK on success, or <ERROR>.

7.2.5 GETPERIOD

Syntax: AT+GETPERIOD<CR><NL>

Returns the last defined measurement periodicity.

Return: Positive number (seconds), or 0 (Default) if not set. Returns <ERROR> on failure.

7.2.6 GETRECORDSCOUNT

Syntax: AT+GETRECORDSCOUNT<CR><NL>

Return: Positive number indicating actual record count, or 0 (Default) if empty. Returns <ERROR> on failure.

7.3 Data Transmission

7.3.1 GETMAILERWAITMODE

Syntax: AT+GETMAILERWAITMODE<CR><NL>

Returns the current mailer wait mode. In mode 1 (default), the user must send the UPLOAD command to the LMT System-on-Module to initiate a data upload. In mode 0, the user registers data for upload using the COLUMN command, and the LMT System-on-Module automatically uploads the data to the server at regular intervals. If the connected network does not support the Power Saving Mode (PSM) feature, the time between uploads is determined by the GETNOPSMUPLINKTIMEOUT value (in hours). If the network supports PSM, the interval is determined by the GETUPLINKTIMEOUT value (in minutes).

Return: 0 (Uploads periodically) or 1 (Host is responsible for initializing upload via UPLOAD command).

7.3.2 SETMAILERWAITMODE

Syntax: AT+SETMAILERWAITMODE=b<CR><NL>

Parameters: 0 (Extension board uploads periodically) or 1 (Host is responsible for initializing upload via UPLOAD command).

Return: OK on success or <ERROR> on failure.

7.3.3 UPLOAD

Syntax: AT+UPLOAD=b<CR><NL>

Triggers a data upload. Maintains connection and uploads accumulated data, or queues it if the network is unavailable. Upload is not performed if there is no data.

Parameters: `0` (exclude radio signal strength from upload data; `AT+UPLOAD<CR><NL>` is equivalent to `AT+UPLOAD=0<CR><NL>`) or `1` (include radio signal strength).

Return: `OK` (Note: Returns `OK` even for failed uploads since this merely initiates the process).

7.4 Device Information & Status

7.4.1 GETCOAPDEVICENAME

Syntax: `AT+GETCOAPDEVICENAME<CR><NL>`

Return: The configured CoAP device name string, or `<ERROR>`.

7.4.2 GETDEVICESN

Syntax: `AT+GETDEVICESN<CR><NL>`

Return: The device's serial number string, or `<ERROR>`.

7.4.3 GETLASTBUILTSYSTEMMODE

Syntax: `AT+GETLASTBUILTSYSTEMMODE<CR><NL>`

Returns the last built system mode set string used for LMT System-on-Module modem initialization.

Return: `AT%XSYSTEMMODE=b1,b2,b3,u` (Default: `AT%XSYSTEMMODE 1,1,1,0`)

- `b1`: LTE-M mode flag
- `b2`: NB-IoT mode flag
- `b3`: GNSS flag
- `u`: LTE-M/NB-IoT preference

7.4.4 GETNETWORKQUALITY

Syntax: `AT+GETNETWORKQUALITY<CR><NL>`

Return: Three integer values representing `RSRP`, `RSRQ`, `SNR`, or `<ERROR>`. (Translate values according to official nRF Connect SDK documentation).

7.4.5 ISMODEMINITIALIZED

Syntax: `AT+ISMODEMINITIALIZED<CR><NL>`

Return: `1` (Default) if initialized, `0` otherwise.

7.4.6 ISSOCKETCONNECTED

Syntax: `AT+ISSOCKETCONNECTED<CR><NL>`

Return: `1` (Default) if connected to the IoT network, `0` otherwise.

7.5 IO Control

7.5.1 DISABLESERIALLOG / ENABLESERIALLOG

Syntax: `AT+DISABLESERIALLOG<CR><NL>` / `AT+ENABLESERIALLOG<CR><NL>`

Suspends or resumes the UART device using power management to disable/enable serial output logging. Useful for reducing power consumption.

Return: `OK` on success; negative error code on failure. Returns `-EALREADY` if already suspended/active (not treated as an error).

7.5.2 `PINxx`

Syntax: `AT+PINxx<CR><NL>`

Configures physical pin `P0.xx` as an input (pull-up resistor is automatically activated), reads it, and returns the raw input level. To identify the chip pin numbers, please refer to the LMT System-on-Module datasheet and pinouts shown in Sections 3–4.

Parameters: `xx` (Physical chip pin number).

Return: `0` or `1` (Actual measured pin level).

7.5.3 `POUTxx=y`

Syntax: `AT+POUTxx=y<CR><NL>`

Configures pin `P0.xx` as an output, sets the output level, and returns the physical level. To identify the chip pin numbers, please refer to the LMT System-on-Module datasheet and pinouts shown in Sections 3–4.

Parameters: `xx` (Pin number), `y` (Output level `0` or `1`).

Return: `0` or `1` (Actual measured pin level).

7.6 Network and Radio

7.6.1 `SETACTIVESIM`

Syntax: `AT+SETACTIVESIM=b<CR><NL>`

Sets the current SIM source. Please turn off the modem before changing the SIM.

Parameters: `0` (on-board MFF2 SIM active; default) or `1` (nano-SIM slot active).

Return: `OK` (success) or `<ERROR>`.

7.6.2 `GETACTIVESIM`

Syntax: `AT+GETACTIVESIM<CR><NL>`

Return: `0` (On-board MFF2 SIM active) or `1` (nano-SIM slot active) or `<ERROR>`.

7.6.3 `SETCOMMSMODE`

Syntax: `AT+SETCOMMSMODE=b<CR><NL>`

Manages the simplified LMT System-on-Module modem system mode when only one communication mode is allowed at a time.

Parameters: `0` (LTE-M; default) or `1` (NB-IoT).

Return: `OK` (success) or `<ERROR>`.

7.6.4 GETCOMMSMODE

Syntax: AT+GETCOMMSMODE<CR><NL>.

Return: 0 (LTE-M) or 1 (NB-IoT).

7.6.5 GETSYSTEMMODE

Syntax: AT+GETSYSTEMMODE<CR><NL>

Returns the currently configured (but potentially unapplied) system mode variable. This variable is a bitmask consisting of the LTE-M mode flag (bit 0), the NB-IoT flag (bit 1), the GNSS flag (bit 2), and the LTE-M/NB-IoT preference (bits 3 to 5).

For example, a returned value of 19 means that the command AT%XSYSTEMMODE=1,1,0,2 will be sent to the LMT System-on-Module modem during its next restart. To apply these pending changes, use the MODEMSHUTDOWN, SETPERIOD (if needed), COLUMN, and UPLOAD commands.

Return: Integer representing the bitmask (Default: 7).

Note: Granular flags for system mode (GET/SET) are also available for GNSSFLAG, LTEMFLAG, NBIOTFLAG, NTNFLAG, and LTEPREFERENCE. They all accept/return 0 or 1 (except Preference which takes SDK-specific integer values) and return OK upon setting.

7.6.6 SETSYSTEMMODE

Syntax: AT+SETSYSTEMMODE=u<CR><NL>

Parameters: System mode value built using field values according to the nRF Connect SDK.

Return: OK on success or negative error code on failure.

7.6.7 MODEMSHUTDOWN

Syntax: AT+MODEMSHUTDOWN<CR><NL>

Closes the socket and powers off the modem.

Return: OK on success or negative error code on failure.

7.6.8 GETPSMRAITIME / GETPSMTAUTIME

Syntax: AT+GETPSMRAITIME<CR><NL> / AT+GETPSMTAUTIME<CR><NL>

Return: Release Assistance Indication (RAI) or Tracking Area Update (TAU) times as 8-character binary strings.

7.6.9 SETPSMRAITIME / SETPSMTAUTIME

Syntax: AT+SETPSMRAITIME=b1...b8<CR><NL> / AT+SETPSMTAUTIME=b1...b8<CR><NL>

Sets the Release Assistance Indication (RAI) or Tracking Area Update (TAU) times as 8-character binary strings. Settings are applied during modem reconnection.

Parameters: 8-character binary string representing the RAI / TAU time according to the nRF Connect SDK; default for RAI "00000011"; default for TAU "00111000".

Return: OK on success or negative error code on failure.

7.7 Timing and Retry Management

All commands in this section follow a standard `GET<PARAM>` and `SET<PARAM>=n` structure. Setting parameters returns `OK` or a negative error code. Getting parameters returns a positive integer, `OK` (for 0), or a negative error code.

<PARAM>	Default	Range	Unit	Description
FILEULRETRIES	5	1–10	—	Number of retries for file uploads
LTECONNECTIONTIMEOUT	30	1–1800	s	Network connection timeout
MAXRESENDATTEMPTS	5	1–10	—	Max resend attempts before socket closes
MAXRESENDTIMEOUT	4	1–24	h	Max timeout for resending packets
NOPSMUPLINKTIMEOUT	4	1–24	h	Uplink timeout when PSM is unavailable
RESENDPACKETINITIALTIMEOUT	1	1–60	min	Initial timeout for resending packets
RESPONSEWAITTIMEOUT	6	—	s	CoAP response wait time (GET only; linked to RAI)
UPLINKTIMEOUT	30	5–1440	min	Device uplink timeout

8 Firmware Implementation Example

To demonstrate the integration of the LMT Cellular LPWAN Extension Board, we will adapt the standard Infineon `Hello_World` sample project for the `CY8CKIT-062S2-AI` host kit. This augmented firmware acts as a simulated sensor node that generates virtual sensor data, processes it locally, and periodically uploads summarized statistics to the cloud via the LMT extension board. You can download the source code **here**.

8.1 UART Communication Setup

The host Infineon board communicates with the LMT Gateway using its UART1 interface. Specifically, the Infineon pins `P10_0` and `P10_1` are configured as RX and TX, routing directly to the Gateway's corresponding TX and RX connections. To facilitate communication, basic blocking transmission functions are defined, allowing the host to easily send C-strings as AT commands to the LMT module.

```
uint32_t uart1_puts( char * str )
{
    uint8_t ch = '\0';
    char * ptr = str;
    while ((ch=*ptr++))
    {
        // uart1_putc handles blocking wait until FIFO is cleared
        uart1_putc( ch );
    }
    return --ptr-str;
}
```

Figure 1: Basic UART string transmission function used to send AT commands to the LMT Gateway.

8.2 Gateway Initialization

During the startup sequence, the system waits for an uptime of 15 seconds before attempting to configure the LMT module. This delay ensures the gateway is fully booted and ready to receive commands. Once ready, the host configures the gateway using three specific AT commands. The uplink timeout is set to 5 minutes, and the mailer wait mode is set to 0, which

instructs the gateway to automatically upload queued data at the specified periodic interval. Finally, an `AT+UPLOAD=1` command is sent to force the module to register and apply these new mailer settings.

```
bool lmt_iot_gw_setup(void)
{
    // Delay 15 s. Depends on the LTE signal quality.
    if ( uptime < 15 )
    {
        return false;
    }
    uart1_puts("AT+SETUPLINKTIMEOUT=5\r\n");
    uart1_puts("AT+SETMAILERWAITMODE=0\r\n");
    uart1_puts("AT+UPLOAD=1\r\n"); // UPLOAD to apply settings
    return true;
}
```

Figure 2: Gateway initialization sequence applying periodic upload settings.

8.3 Main Processing Loop and GPIO Control

The core logic of the application relies on a hardware timer configured to generate an interrupt every second (1 Hz). During each one-second tick, the system reads a virtual sensor value and tracks the minimum, maximum, and accumulated totals for later averaging. In addition to processing data, the firmware demonstrates direct control over the LMT Gateway's physical pins. During every timer tick, the host sends an `AT+POUT26` command to toggle the state of the Gateway's `P0.26` GPIO pin.

```
void lmt_iot_gw_gpio26_toggle(void)
{
    static bool state = false;
    state = !state;
    if (state)
    {
        uart1_puts("AT+POUT26=1\r\n");
    } else {
        uart1_puts("AT+POUT26=0\r\n");
    }
}
```

Figure 3: Toggling an external GPIO pin on the LMT board via AT commands.

8.4 Data Aggregation and Upload

To optimize data usage and power consumption, the firmware does not upload every single 1 Hz reading. Instead, it aggregates the data over a window of 100 samples. Once 100 samples are collected, the host calculates the average value and passes the minimum, average, and maximum values to the `lmt_gw_data_record` function.

This function uses `snprintf` to format the three data points into the `AT+COLUMN` command syntax. These values are stored in the gateway's internal buffer, and because the mailer wait mode was previously set to 0, the LMT module will automatically transmit this data bundle upon reaching its 5-minute timeout. If no data happens to be recorded in the buffer during a 5-minute interval, the gateway gracefully handles this by sending an empty packet to the cloud, serving as a system health heartbeat.

```c
void lmt_gw_data_record( int v1, int v2, int v3 )
{
    static char record[48]; // Buffer for 3 integer values

    int would_written = snprintf(record, sizeof(record),
    "AT+COLUMN=%d,%d,%d,,,,,,,\r\n", v1, v2, v3);

    if ( sizeof(record) <= would_written )
    {
        printf("Error: increase buffer size!");
    } else {
        uart1_puts(record);
    }
}
```

Figure 4: Formatting and queuing aggregated data into the gateway's upload buffer.

9 Integration with Edge AI Motion Classification

To further illustrate the capabilities of the LMT Cellular LPWAN Extension Board, we examine a more advanced use case integrating the board with an Edge AI application. Based on the Infineon `DEEPCRAFT_Deploy_Model_Motion` sample project, this firmware transforms the host kit into a sophisticated motion recognition node. The system continuously samples data from an onboard BMI270 IMU sensor via an I2C interface, processes the raw data using an embedded machine learning model, and classifies the motion into predefined labels in real time. You can download the source code **here**.

9.1 Visual Status Indication via Gateway GPIO

As the host Infineon board continuously runs its inference loop, it utilizes the LMT IoT Gateway's GPIO pins to provide simple visual status indications. Rather than relying on a separate timer interrupt, the main loop uses bitwise operations on a continuously incrementing `time_now` variable to periodically trigger state changes. The firmware toggles gateway pins `P0.28`, `P0.27`, and `P0.26` by sending `AT+POUT` commands over UART. This mechanism drives indicator LEDs to confirm normal operation and active serial communication between the Edge AI host and the LMT module, even without an attached serial terminal (Fig. 5).

9.2 Event-Driven Reporting and Rate Limiting

Unlike simpler periodic monitoring, this AI-driven application utilizes an event-driven reporting model. When the ML model successfully classifies a motion and detects a valid label, the system passes the identified label to the `report_event` function (Fig. 6). To prevent excessive cellular transmissions and conserve energy during rapid or repeated motions, this function implements a rate-limiting mechanism ensuring events are not reported more frequently than a defined `MIN_UL_PERIOD`.

If a new classification occurs too soon after a previous upload, the system buffers the event into an `unreported_event_number` variable to ensure that state changes are not lost. Once the timeout period expires, the function retrieves the event, formats the `AT+COLUMN` command, and verifies that the UART transmission FIFO buffer is empty. Finally, because this architecture relies on the host for network pacing, the transmission is immediately executed by issuing an `AT+UPLOAD` command directly to the LMT module.

```
if ( time_now & 128 ) {
    if ( 0 == rgb[0] ) {
        uart1_puts("AT+POUT28=1\r\n");
        rgb[0] = 1;
    }
} else {
    if ( 1 == rgb[0] ) {
        uart1_puts("AT+POUT28=0\r\n");
        rgb[0] = 0;
    }
}
if ( time_now & 256 ) {
    if ( 0 == rgb[1] ) {
        uart1_puts("AT+POUT27=1\r\n");
        rgb[1] = 1;
    }
} else {
    if ( 1 == rgb[1] ) {
        uart1_puts("AT+POUT27=0\r\n");
        rgb[1] = 0;
    }
}
```

Figure 5: Excerpt from the main loop showcasing the use of bitwise logic to sequence through the gateway's GPIO pins.

```c
static void report_event(int16_t event_number)
{
    static unsigned int last_event_UL_time = 0;
    static int16_t reported_event_number   = 0;
    static int16_t unreported_event_number = 0;

    unsigned int time_now = 0;
    time_now = xTaskGetTickCount() * portTICK_PERIOD_MS;

    static char command[40] = { 0 };

    printf("report_event: %d\r\n", event_number );
    if ( ( last_event_UL_time + MIN_UL_PERIOD ) > time_now ) // too soon
    {
        if ( event_number )
        {
            unreported_event_number = reported_event_number == event_number ?
            0 : event_number;
        }
    }
    else
    {
        event_number = ( ( 0 == event_number ) && unreported_event_number ) ?
        unreported_event_number :
        event_number;
        if ( event_number )
        {
            last_event_UL_time = time_now;
            reported_event_number = event_number;
            unreported_event_number = 0;
            snprintf( command, sizeof(command), "AT+COLUMN=%d,,,,,,,,,\r\n", event_number );
            uart1_puts( command );

            uint32_t uart1_status = Cy_SCB_UART_GetTxFifoStatus(SCB1);
            while ( !(CY_SCB_UART_TX_EMPTY&uart1_status) ) {
                printf("Waiting on previous command... ");
                Cy_SysLib_Delay(1);
            }
            uart1_puts("AT+UPLOAD\r\n");
        }
        else
        {
            printf("%40s\r\n", "");
            printf("%20s\r\n", "");
        }
    }
}
```

Figure 6: The reporting logic implemented in the example, featuring event buffering and rate limiting prior to transmitting data to the LMT IoT Cloud.