ONRAMP WIRELESS

TOTAL REACH

rACM Developer Guide

On-Ramp Wireless Incorporated
10920 Via Frontera, Suite 200
San Diego, CA 92127
U.S.A.

rACM Developer Guide

010-0105-00 Rev. A

November 1, 2013

Contents

Figures

Tables

Revision History

Revision	Release Date	Change Description
A	November 1, 2013	Initial release based on the racm_1.3.0 firmware.

PART 1: rACM Quick Start

Purpose

Part 1 of this guide describes the necessary steps to build, download, and test the reference Application Communication Module (rACM) software executing on the On-Ramp Wireless' battery-powered reference host application. It is a reference for external partners in the development of a sensor application on the reference host platform using On-Ramp Total Reach wireless technology.

By observing the rACM application as it joins the On-Ramp Total Reach Network and generates user-payload for various input stimuli, the end-user can better understand the battery-powered software architecture and implementation considerations required to integrate and develop third-party sensor applications.

NOTE: This document is derived from the first generation rACM platform (0.x.x firmware executing on a wAMI hardware platform – On-Ramp Wireless document 010-0042-00 Rev. B). For external customers who are familiar with the older platform, the following bullet items highlight the major differences presented in this guide:

- Updated Hardware Platform (see section 2)
- Key-Value pair reorganization (see section 6)
- Application Interface Configuration Framework (see section 7.11)
- Application Factory Data Framework (see section 7.12)

Scope

Part 1of this guide is intended to assist in a "quick start" and focuses on the build tools and scripts necessary to:

- Download and execute the rACM software application
- Observe the network join process
- Demonstrate the Over-the-Air (OTA) capabilities of the application.

References

Reference Platform Schematic Prints (510-0038-03 Schematic Prints, Sep 2013)

1 Build Environment

1.1 Supported Tool Chains

The rACM software supports the following build environments:

- GNU Compiler Collection (GCC) tools on Linux platform using Makefile. The cross compiler used on rACM were extracted from Sourcery Codebench source (version 4.6.1) by Mentor Graphics The tool chain can be found at their website under the name Codebench lite:
  http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/

  An account will be required. Look for ARM processors and use the EABDI Release link:
  http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/arm-eabi

  The rACM build process requires the release entitled Sourcery CodeBench Lite 2011.09-69. The Makefile looks for the GCC source in the *opt/CodeSourcery/arm-2011.09* path.
- IAR WARM IDE (version 6.30) running natively on a Windows platform.
- IAR ARM Compiler tools using Makefile a Windows platform using Cygwin.

The choice of which of these three build environments is left up to the user. However, for integrators who do not have previous experience with GCC and Linux-based debug capabilities (e.g., OpenOCD), we recommend that the IAR Workbench IDE be used. The rACM project file for the IAR EWARM contains all of the necessary configuration settings to quickly build, download and debug the rACM application via the Segger J-LINK JTAG Emulator.

Regardless of the actual choice of tool chain, the build environment also requires the use of Python 2.6 to successfully build the rACM project. In addition, the scripts used to send and monitor message exchanges over the rACM Host Interface UART are also Python-based. The following Python sources must be installed on the build platform (or a virtual machine must be used per section 1.2).

- matplotlib-0.99.1.win32-py2.6.exe
- numpy-1.3.0-win32-superpack-python2.6.exe
- pyserial-2.5.win32.exe
- python-2.6.3.msi
- pywin32-214.win32-py2.6.exe
- wxPython2.8-win32-unicode-2.8.10.1-py26.exe

NOTES:

1. On-Ramp project engineers can supply a zipped package including each of the aforementioned python sources upon request.
2. The installers listed above are for 32-bit Windows OS only. If you are developing on a Linux-based platform or 64-bit Windows OS, you must obtain the appropriate installers available on the internet.

1.2 Virtual Machine Based Tools

On-Ramp Wireless project engineers can supply a virtual machine (VM) containing the current rACM release and tools. With this approach, there is no need to search for all the right Python packages since they are pre-installed. The VM is platform-independent but we have the most experience using it on 64-bit Windows 7, with VMWare Player 5.0.1. We recommend configuring a VMWare player to allocate 2 GB of memory and 20 GB of hard disk space to the VM. The VM can take control of the host PC's serial port to connect to the rACM. On-Ramp Wireless requires the use of a particular USB-to-serial cable (FTDI TTL-232R-3V3) which is available through Digi-Key) to avoid problems experienced with other serial cables.

The rACM release on the VM can be found at:
`/opt/racm_1.3.0/`

The tools are found in:
`/opt/racm_1.3.0/python_tools`

The usage of tools such as host_app_ctrl.py can be found elsewhere in this document.

1.3 Supported Processor Targets

The rACM supports the following Kinetis Processor targets:

- K20 80-pin LQFP 72MHz package (K20D7)
- K20 80-pin LQFP 100Mhz package (K20DZ10)

The important differentiation (other than the maximum processor speed) that is not obvious is the Silicon die revision (1.x vs. 2.0). The K20DZ10 is an older die version that uses the 1.x die mask and the K20D7 is instantiated using the newer 2.0 die mask. Due to register changes affecting the PLL clock initialization, code compiled for the wrong target silicon die will NOT execute properly. As a failsafe, if the boot sequence determines that it is executing on an incompatible target processor (as identified by the Kinetis Silicon ID registers), it will disable watchdog, turn on the status LED, and remain in a software trap until the correct firmware can be downloaded via JTAG.

NOTE: It is recommended that if the end-user ports the reference application software on to a target MCU other than the two listed packages (e.g., a K10 variant), Freescale and On-Ramp Wireless application engineering should be consulted for a compatible target build. On a related note, any Kinetis processor target under consideration to run the rACM firmware MUST support the FlexNVM option in the second bank of flash.

1.4 Project Directory Structure

After the rACM source files have been extracted onto a local machine, the directory structure is organized as follows:

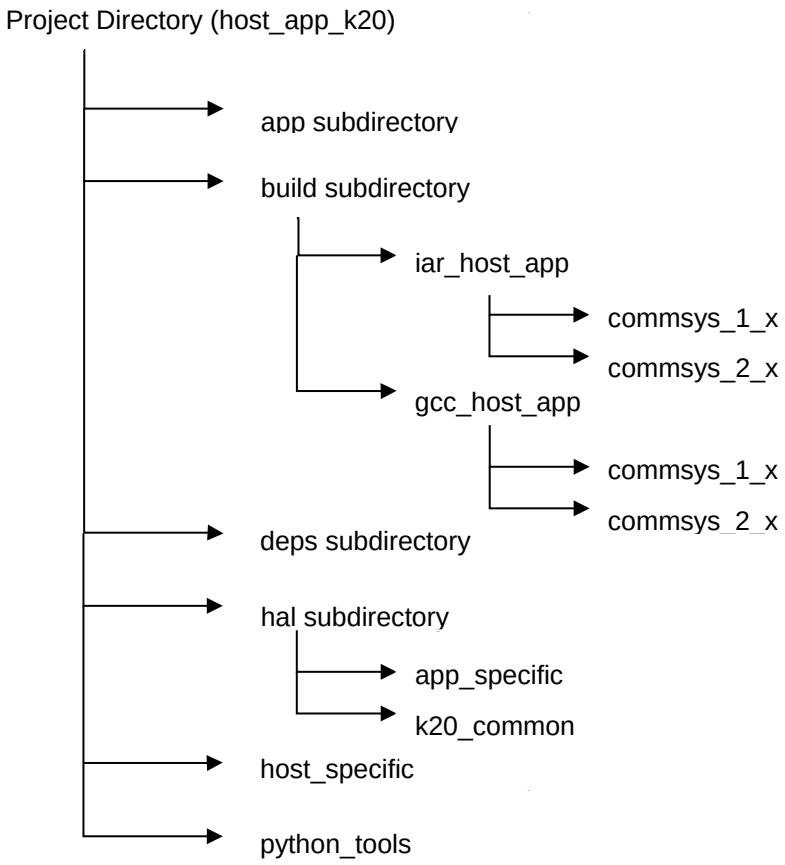


Figure 1. Reference Host Software Directory Structure

The subdirectories are summarized as follows:

- **app** – The set of application source code for the target build which implements the state handlers, message parsing/handling, and the necessary logging that supports the application type (e.g., water meter, pressure sensor, remote monitoring, etc.). These files do not interface directly with the hardware interfaces and, as such, are designed to be portable across different hardware platforms that are designed to perform the same functionality (e.g., a water meter application that can run on either a K10-based platform or a K20-based platform).
- **build** – This contains the project and/or makefiles used to compile and link the desired application and platform hardware abstraction layer. As noted, the reference host software currently supports both the IAR and GCC toolchains. The *gcc_host_app* and *iar_host_app* subdirectories within build also contain the firmware binaries, list files, and dictionary

output from the two different build tools for the different On-Ramp Wireless Communication Systems (see the next section).

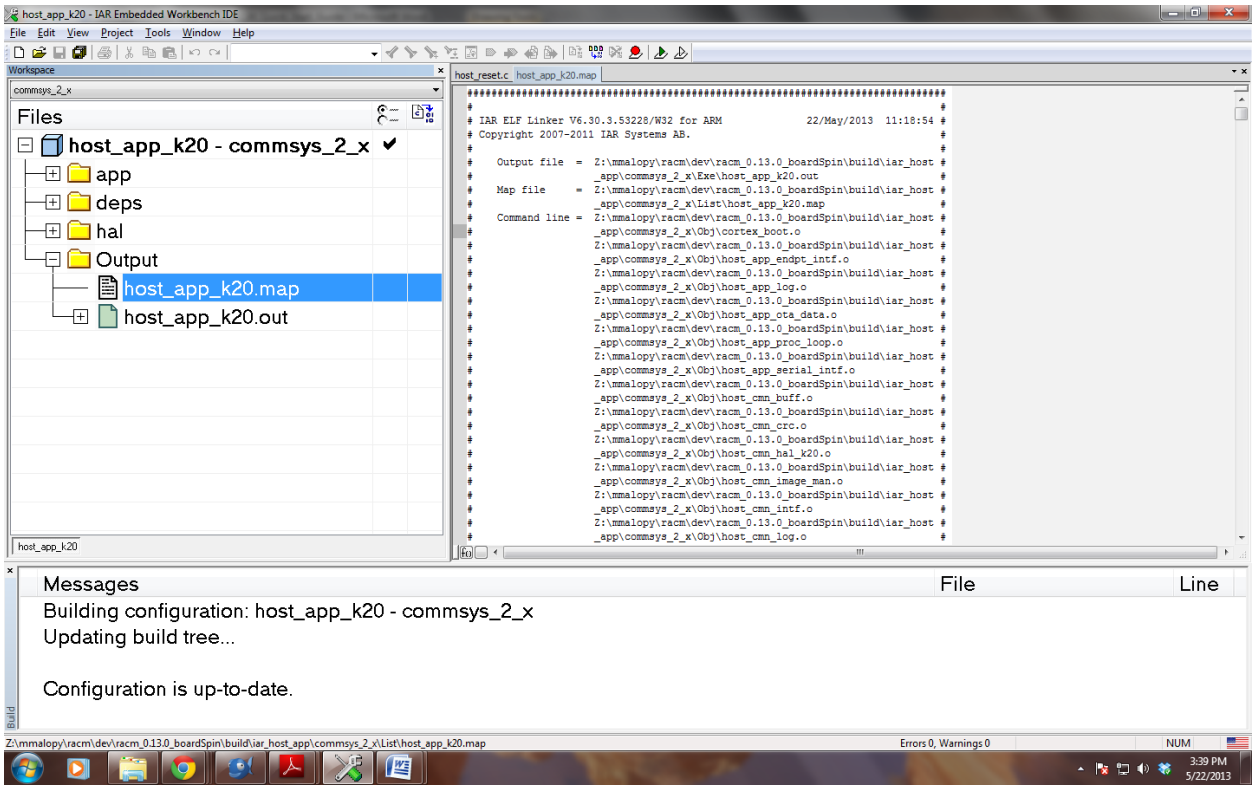
- **deps** – The external dependencies directory contains the shared/common source code used by On-Ramp Total Reach hardware platforms. For the reference host, this includes all of the Host-to-Node DSPI management routines as well as common message definitions. These are typically the software elements that make up the HOST_CMN library (see reference [1])
- **hal –** The Hardware Abstraction Layer (HAL) support functions/drivers that serve as the interface layer between the upper-level application functions and the actual hardware interfaces. These interface files are further divided into two subdirectories – *app_specific* contains interface drivers that are customized and/or unique to the rACM application. *k20_common* contains interface drivers that are common for any application executing on the Kinetis microcontroller unit (MCU) (e.g., flash drivers, board support functionality, etc.).
- **host_specific –** The host specific folder contains message descriptors, both for the C-source and Python scripts, which are specific to the rACM configuration and control commands sent over the Host Interface UART (e.g., Device Status Request, Log Retrieval command, etc.).
- **python_tools** – This subdirectory contains all of the Python scripts necessary to control and configure both the Node and host application through the Host UART. Of particular interest for the Node is the *ulp_node_monitor.py* script which contains all of the necessary GUI tools required to calibrate, configure, and enable the node to successfully track and join the network. For the rACM application, the *host_app_ctrl.py* script is used for configuration and control of the host. In addition, *host_app_logger.py* can also be used to monitor real-time UART logging as it relates to application events. Further details on the rACM control scripts are located in section 5.1).

1.5 Building the Application

Although the rACM release already contains the pre-built binaries for both the IAR and GCC tool chains, the application can still be (re)built using the following methods:

1.5.1 IAR Workbench IDE Project Manager Build

When using the IAR IDE, the *build* subdirectory contains the rACM project file *host_app_k20.eww*. After the project is loaded, select the drop-down configuration option for the appropriate target processor – the rACM platform assembled by On-Ramp Wireless is currently populated with the K20D7 MCU and defaults to the commsys2.x. After that, the 'Project->Build All' tab command performs all of the necessary steps to build the project output (i.e., target binaries, dictionary files, map files). All of the necessary build options and library configurations have previously been set-up and defined for the rACM project.



Details on the IAR Workbench IDE are found in the supporting IDE documentation, help menu, and online from the IAR website at http://www.iar.com/en/Products/IAR-Embedded-Workbench/ARM/.

1.5.2 Makefile Build

The rACM application can also be built from the command line.

- Linux platform – Using the GCC tool chain
- Windows platform – Using Cygwin with the IAR tool chain

Regardless of the platform, the Makefile syntax is the same:

- **make clean** – Clears/deletes the build output folder to force a fresh build.
- **make host_app** – Compiles/links the rACM application

The Makefile definition for each target (i.e., *gnu_host_app.mk or iar_host_app.mk*) contains all of the necessary rules for building the rACM project output.

1.6 Project Output

Regardless of the platform and/or build method, the output of the build process is identical and produces the following output:

- **Dict** – Contains the dictionary output (i.e*., host_app.logDict*) used by the Host application's logging methods. It is essentially a library of ASCII strings that are extracted from the source code during the build and associated with the Host Log Indicators that are broadcasted through the Host Interface UART. This method of logging allows the Host Application to log any user-defined string data and also keep the memory footprint caused by strings to be minimized.
- **Exe** – Contains the binary image (*host_app_k20.bin*) and debug image produced by the build process (*host_app_k20.out*). NOTE: The binary image is what is used when performing software upgrades through the Host Serial Interface using the **HOST_SW_UPGRADE** command (see section 5.1).
- **List** – Contains the pre-processor output of each source file as well as the map file as generated by the build process. The pre-processor output is required to support the generation of the dictionary output.
- **Obj** – The source objects, as generated, during the build process.

It is important to note that the tool chain determines which build subdirectory the project output is piped to:

- *gcc_host_app* – Project output for GCC-based tools on a Linux platform
- *iar_host_app* – Project output for IAR-based tools on a Windows platform (either Cygwin makefile or IAR IDE).

It is further directed into a subfolder for the currently configured On-Ramp Wireless Communication System (i.e., commsys_1_x or commsys_2_x subdirectories).

2 Reference Platform Hardware

The following illustration provides a high level depiction of the rACM hardware platform illustrating the two CPUs (i.e., application host and On-Ramp Wireless Node modem) and external interfaces that are under direct control of the Application Host.

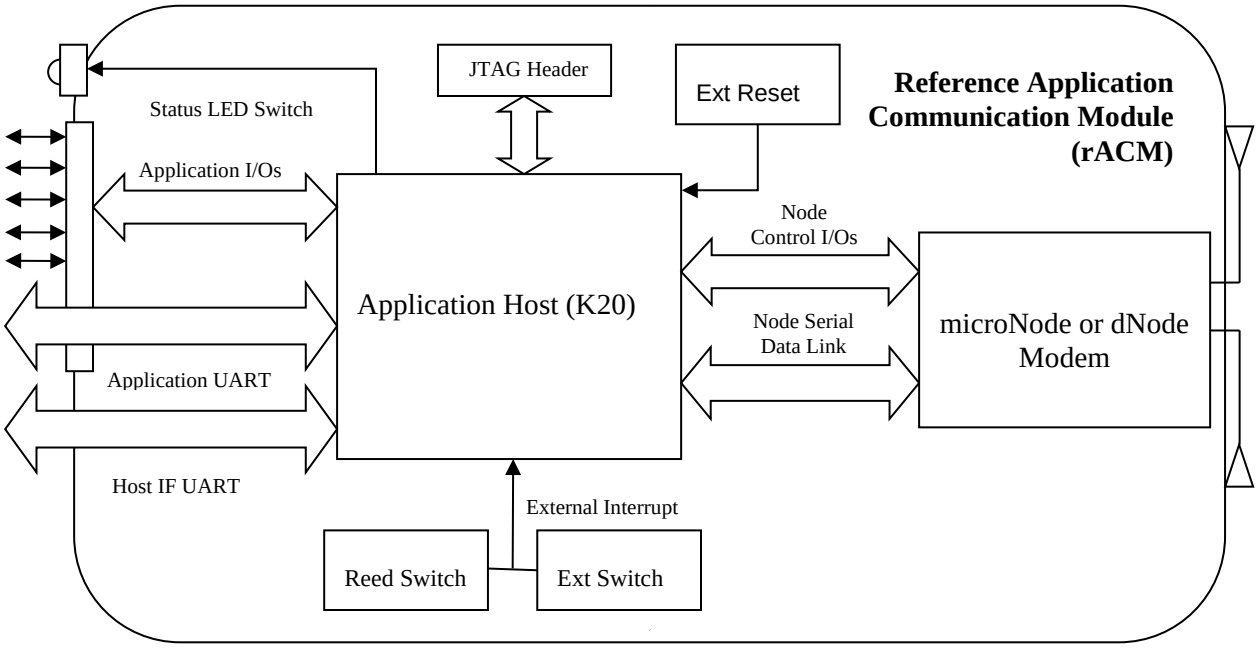


Figure 2. Reference Platform Block Diagram

Referring to the rACM hardware schematic (Appendix E), the application interfaces shown in the figure above fall into the following categories. All part reference designators below are from the schematic in Appendix E.

- **Application Host CPU** (U201) – The heart of the reference Application Communication Module (rACM) platform. The rACM firmware executes on this CPU (more details on this to follow).
- **ORW Node Modem** (X300 or J701/J703) - Page 3 of the schematics (Appendix E) entails the microNode or dNode component. Both Node devices have been verified to work with the rACM, but only one at a time. Nodes cannot share their SPI busses (more details on this to follow).
- **Status LED Switch** (D203) – A GPIO-controlled Green/Yellow LED which the application host uses to display status patterns, alarm states, and/or user-feedback. Its exact use is dependent on the application type.
- **Application I/O Header** (J207) – A bank of external application interface pins (in addition to ground and source voltage). All I/Os are ESD protected with the given protection diodes, as

well as the fact the Kinetis MCU can tolerate 5V directly as per specification. The application interface pins are defined as follows:

- APP_INTF1 (Kinetis port PTA19)
- APP_INTF2 (Kinetis port PTC3)
- APP_INTF3 (Kinetis port PTC5)
- APP_INTF4 (Kinetis port PTC6)
- APP_INTF5 (Kinetis port PTC0)
- ANA_IN0 (Kinetis pin15 – i.e., dedicated ADC input)
- A_UART_RX (Kinetis port PTD6 – see below)
- A_UART_TX (Kinetis port PTD7 – see below)

It should be noted that the ports on the Kinetis MCU are multiplexed to support a number of possible functions (even at run-time) and can also be configured to read/report the interface data at defined intervals. Depending on the application type, the listed pins can be configured for a combination of the following:

1. General Purpose Input/Output (GPIO)
2. Kinetis Flex Timer Module (FTM) output (e.g., single-ended or differential clock output, stepped patterns, Pulse-Width Modulation (PWM) signal generation, etc.).
3. Kinetis Analog-to-Digital Converter (ADC) input (single-ended or differential) – used for voltage or current sampling
4. Kinetis Comparator circuit (CMP) input
5. Kinetis Low-Power Timer (LPT) input – used for pulse counting or clock generation while the application is in deep sleep

By default, the rACM application currently configures and manages APP_INTF1 (PTA19) as a Low-Power Timer (LPT) input for use in monitoring and reporting pulse counts as user data (see section 4.2). All other APP_INTF pins are configured as unmonitored digital inputs.

Further details regarding the configuration of the Application Interface pins and a matrix of the functional capabilities are presented in section 7.11.

- **Application UART** (Kinetis ports PTD6 and PTD7) **–** A dedicated 3-wire serial interface for UART communications to an external device through the Application Header (e.g., GPS module, ModBus sensor interface, etc.). This interface is currently used in the rACM build to demonstrate serial loopack and/or OTA passthrough modes (see section 7.10).
- **Host Interface UART** (J206)**–** A dedicated 3-wire serial interface for UART communications to a Host PC for purposes of factory calibration, field provisioning and/or field debugging.
- **Reed Switch** (SW200 – magnetic switch on bottom front-edge of the reference board) or **Interrupt Switch** (SW201 – push button switch near application header) **–** A dedicated port (configured as a GPIO input) used to monitor an external interrupt event. Primarily intended for applications they may end up in storage before being deployed in the field where a manual toggle could be used to “activate” the device and initiate the network join process. The Reed Switch was deemed relevant such that if the K20 system was embedded (or potted) along with its battery, the Reed Switch may be necessary to awaken the K20 from a deep sleep and prompt an action.

- **Reset Switch** (SW202) – A push button switch near the application header used to reset the application host. This is a hard reset to the Kinetis CPU and will be handled by the rACM firmware as a Power-On-Reset with regards to the initialization of all firmware resources.
- **JTAG Header** (J205) – 10-pin IC Debug port used for on-target debugging or for flashing development builds (more on this interface to follow).
- **Voltage Supply Header** (J204) – Voltage input provided by an external AC-to-DC power adaptor. Input Range is 1.8V to 5.0V – with 3.3V being nominal.
- **VBatt Supply Header** (J201) 3.3 Volt battery input used to power the reference platform. The application host is specified to operate down in the 3.3V to 1.8 voltage range before resetting due to low-voltage.

2.1 Host CPU

The Freescale Kinetis MCU K20 variant was selected for a number of reasons for the basis of rACM:

- As a Cortex-M4, is very versatile as a true embedded device, allowing substantial growth in Firmware complexity.
- Supports many top name Operating Systems.
- Can be used for Ultra-Low power consumption.
- Wide range of input voltages and clock ranges.
- High level of security by locking FLASH.
- High integration with a wide range of peripherals.
- Good support from a wide range of tool vendors, including GCC-based compilers.
- FlexRAM, a valuable NVM storage for data in ultra-deep sleep modes and for loss of power.
- In general, each K20 pin can be multiplexed up to 8 different ways: ADCs, UARTS, I2C, GPIO, etc.

The 80-pin package of the K20 comes in a wide range of options, ranging from 64 kByte to 1MByte Flash storage. Additionally, the K20 comes in 50, 72, and 120 MHz options. The K20 can be selected for cost and performance. The existing 72 MHz, 256 kB is considered “middle of the road” and is a compromise of price and feature set. All K20-80 pin devices are pin-pin compatible.

The K20 has its own internal Voltage reference as well as U202 (3.0V). The internal K20 reference is less accurate, thus U202 was implemented. If the K20 internal reference is used, the U202 must be removed. U202 was selected to provide a Vref with 1uA of current, perfect for battery life. A Vref of 3.0V was chosen for a wide range of Vinput of the K20 ADCs. If the K20 will be run from Lithium batteries (J200 direct), it is possible a lower Vref voltage need to be used to ensure U202 stays in regulation. U202 nominally needs Vref+0.15V in order to maintain regulation. Thus a 1.8Vref would allow the general K20 VDD to go down to 1.95V which is optimal for most battery applications.

U201F provides a 32 kHz crystal input to the K20. This is a more stable clock input to the K20 than its internal RC clock oscillator. This 32 kHz is likely desired for a stable baud clock, if a UART interface is required. The K20 can execute off its own internal RC oscillator, yet it is very inaccurate, especially over Temperature.

2.2 AC Power

The Reference hardware platform nominally operates with a 3.3V DC source supplied externally through the AC Voltage or Vbatt Supply Header (J204 or J201). This can come from several different voltage sources delivered to the rACM platform.

- 19Ah lithium D cell (Tadiran TLP93311AL).
- 5V/3.3V AC to DC Adaptors (or Wall Warts)
- 2 Alkaline cells (3.0V, or lower).

Regardless of the actual source, the on-board TPS63000 buck/boost voltage regulator (U200) allows the rACM to be powered from batteries or standard wall warts, between 1.8 and 5.5VDC. The U200 converts the energy to a defined 3.3V for the CPU. This regulator type has the advantnge of very low quiescent current in Power Save mode (pin 7 Low) or excellent regulation with PWM mode (pin 7 High) at the expense of high quiescent current. This type of quiescent feature could be useful if the K20 needs to “clean” VDD (pin 7 High) such as Analog modes or a noisy VDD is tolerable (pin 7 Low) in K20 digital-only modes. The boost mode of U200 is useful if running from 2 Alkaline cells.

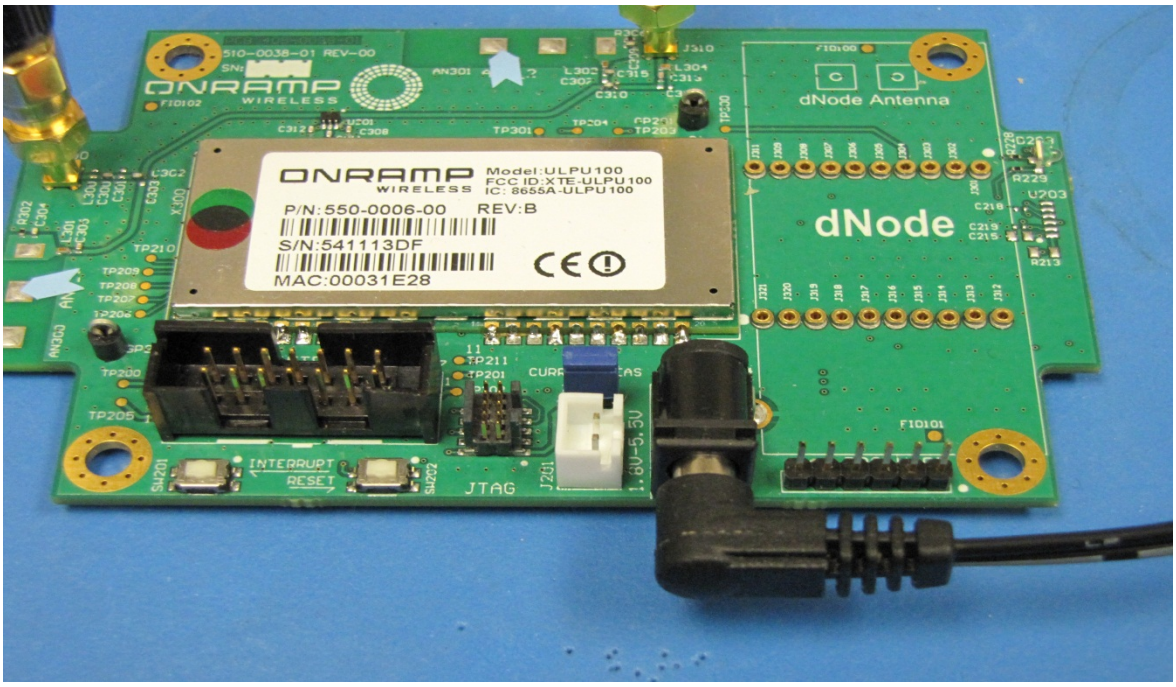
The headers J201 and J204 allow different input modes such as Wall warts (J204) or J201 (battery devices).

J200 is a jumper header with two main purposes:

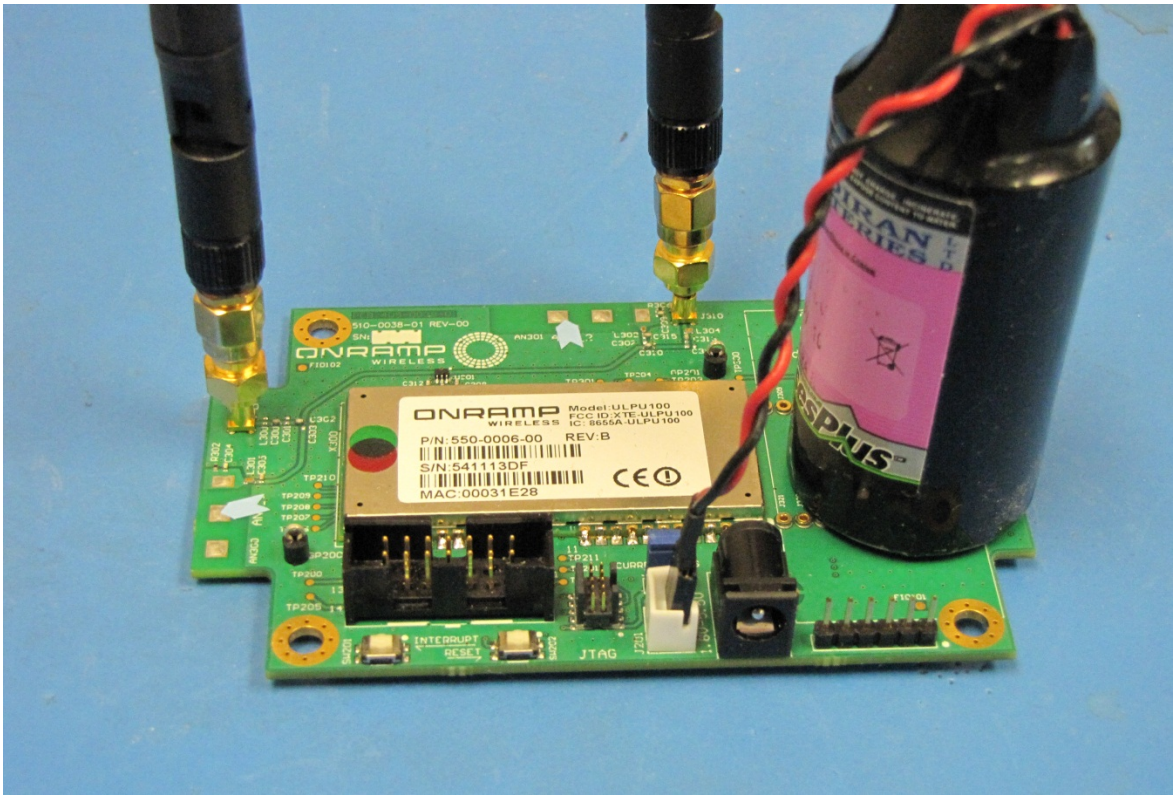
- Allow easy tap for current consumption profiling. This is accomplished by inserting an Ammeter between pins 1-2 of J200. When doing precision accumulated current to a device such as the rACM, it is crucial to consider the Series Resistance of the Ammeter. This seems obvious, but it is crucial to understand the huge dynamic range of the rACM K20/Node subsystem: 20uA to 300+mA. The internal resistance of the Ammeter plays a substantial roll in Vdrop of the voltage to the Device Under Test (DUT) which significantly affects current measurements.
- The second purpose of J200 is to bypass the U200 buck/boost regulator. In a true battery-only environment, the quiescent current of U200 is too significant for a meaningful (1+yr) battery life. Instead, the user can inject a nominal Lithium Thionyl Chloride voltage (i.e., 3.6VDC) directly into pins 2-3 of J200 to power up the K20 and microNode circuitry. This mode gives ultimate battery savings. It is necessary for the user to understand the variances within battery technologies and choose wisely. For instance, some Lithium battery types do not like large current surges.

Transistors Q200/Q201 are designed to provide a pulsed Vbatt to pin 2 of J207. It was found that some water meters require this as part of their activation sequence and this design was carried forward as a possible solution to that requirement.

The 5V AC to DC adaptor connection (J204) is shown in the following photo.



The following photo shows the Tadiran lithium D cell as the voltage source through the J201 header:



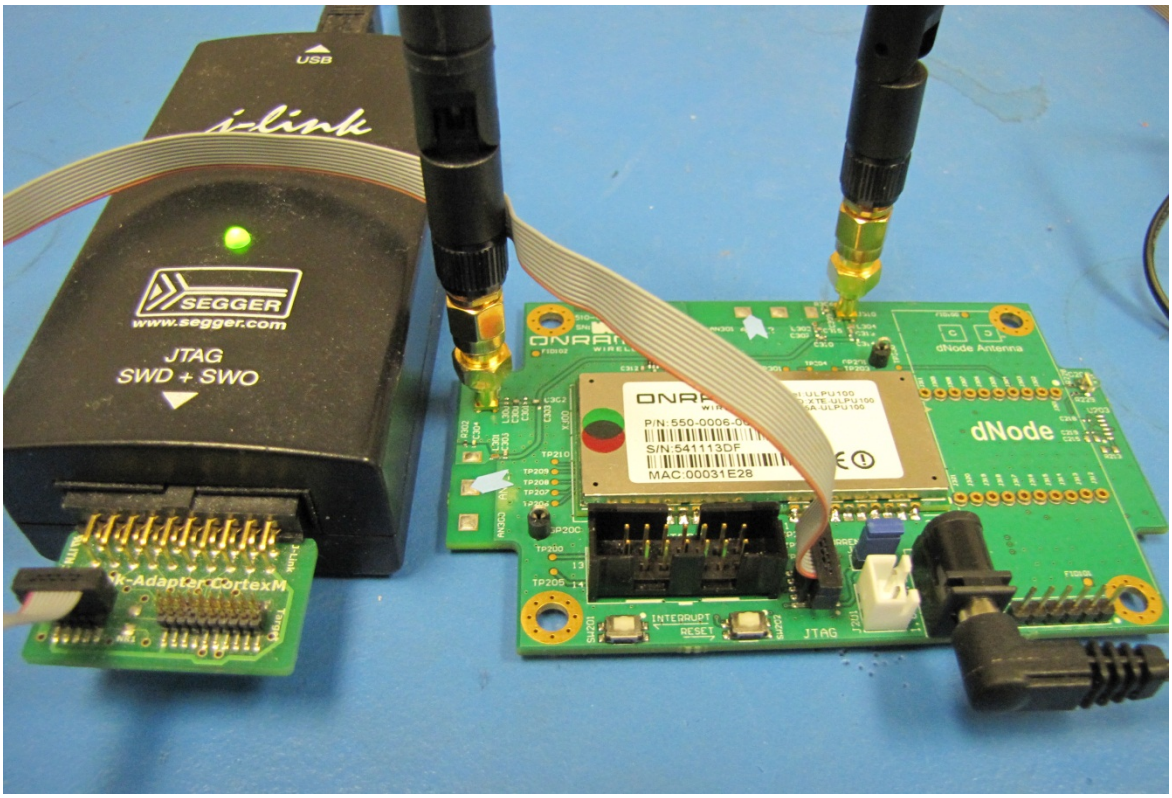
When power is applied to the reference platform (assuming that it has been previously flashed with the rACM firmware), the status LED (D203) flashes the initial Power-On Sequence (two half-second on-pulses every five seconds. See section 4.1).

2.3 JTAG Interface

For on-target debugging and/or developer and initial Flash programming of the application MCU on the Reference platform, an external emulator, capable of controlling the Cortex-M4 processor, can be used by connecting though the on-board JTAG header (J205). On-Ramp Wireless has used many JTAG Debuggers and has found the Segger series of devices to work well with our system. The Segger allows many SW tools to work seamlessly with the rACM, both purchased and Open Source.

If the IAR EWARM IDE is being used with the rACM, it is recommended that the Segger J-Link Emulator be selected for this purpose – the rACM IAR Project is preconfigured to communicate with the rACM platform through the J-Link adaptor. Further details on the Segger J-Link can be found at http://www.segger.com/jlink.html.

The following figure shows the J-Link emulator cabled to the rACM reference platform using the 10-pin cable connector that ships with the Segger unit.

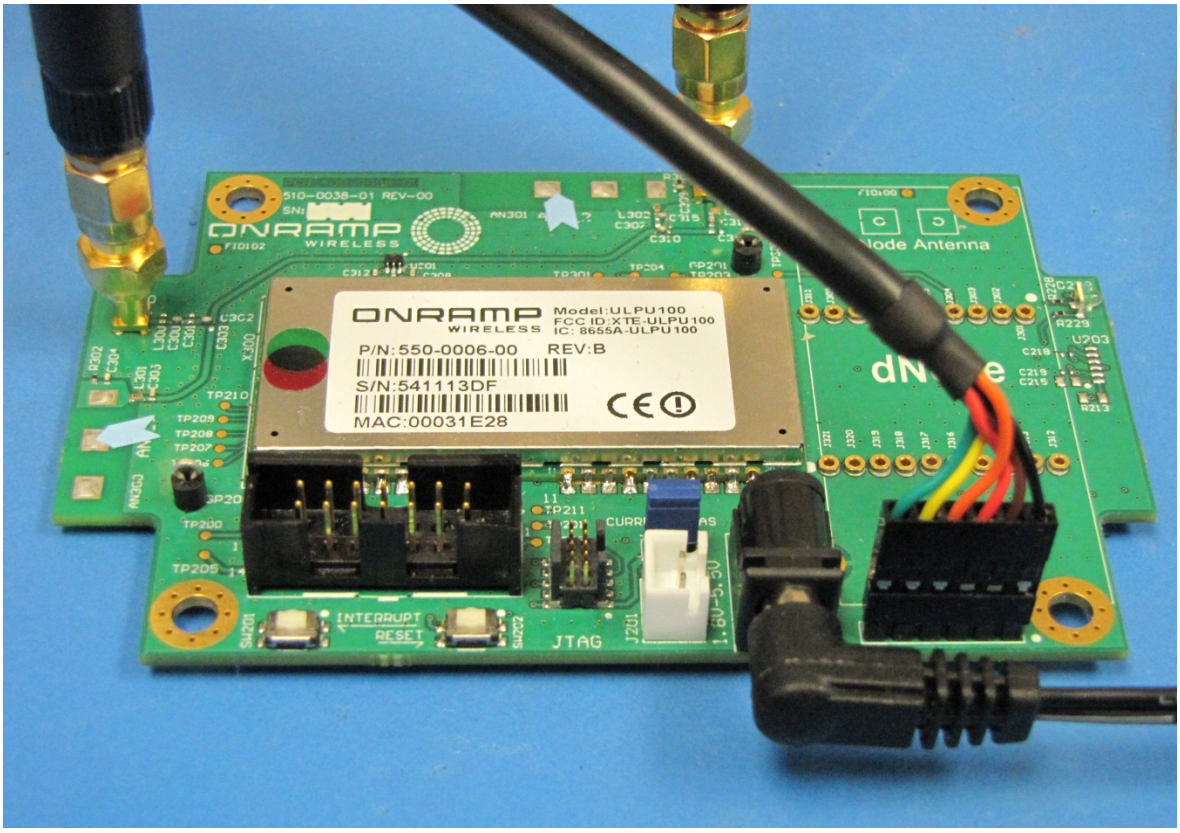


The JTAG header on the reference platform (J205) is keyed such that the JTAG interface pins on the connector tab are properly orientated.

2.4 Host UART Interface

All serial communications with the application host executing on the reference platform are done using an RS-232 serial interface. However, due to the reference platform executing at the 3.3V power domain, a USB to TTL convertor is required to safely convert the PC port to the levels that are safe for the application host. By default, the Host serial interface routed through the J206 header is configured to use Kinetis ports PTB10/PTB11 for Rx/Tx UART data.

The following figure shows the USB to TTL Serial cable connected to the reference platform through a 6-way 0.1” pitch single inline connector:



2.5 Antenna Connection

The rACM supports antenna diversity and includes an antenna port located on the back edge of the reference platform (i.e., opposite of the status LED and Reed Switch) and on the right “wing” of the reference platform (i.e., opposite of the AC Power input. Both antenna paths can be routed to use a standard micro-miniature coaxial RF connection (MMCX) on ports J300 and J310 or, alternatively, a ceramic chip antenna pad mounted on ports AN300 and AN301.

To switch between antenna ports, the user must re-solder capacitors:

- ANT1 - C301/C302
- ANT2 – C307/C310

Each capacitor must “point” to the specific desired antenna. The board is shipped with the MMCX connectors enabled. If the User desires the onboard antennas, some comments are:

- The antennas are generally tuned for “free air.” That is, if the rACM is placed in a housing next to ANY substance (i.e., polycarbonate enclosure), the optimal antenna “match” will shift. It is up to the User to determine how to match the antennas using the provided RLC pads and components to fit into the desired application and housing.
- The antennas have been placed at right angles, at approximately 2.5” apart (6.4 cm). This spacing is ¼ of the wavelength of 2.4 GHz, as well as the differing orientation helps with phase. If a User is to provide their own layout, these factors need to be considered and are required to optimize the true diversity of the Nodes.

It is the microNode that determines the antenna diversity switch times and optimization. The Node diversity can be enabled or disabled, as per the Node Provisioning process. This diversity configuration is stored in the Node’s FLASH as part of its user configuration. Diversity is always recommended for optimal reception. If Diversity is turned off, the Node defaults to ANT1.

The antenna platform components are used *only* when the microNode module is installed (see the next section). The dNode manages its own antenna ports.

2.6 Node Module Option

The rACM supports one of two On-Ramp Wireless Node module options:

1. **microNode Module:** This Node module leverages the rACM platform antenna path options/components . See the previous section.
2. **dNode Module:** This Node module is an FCC certified module with its own integrated antenna solution.

On-Ramp Wireless Application Engineering will recommend which module option is the best fit for a customized application using On-Ramp Wireless Total Reach technology. The reference platform can then be used for an initial prototype and development of an application using the appropriate Node module.

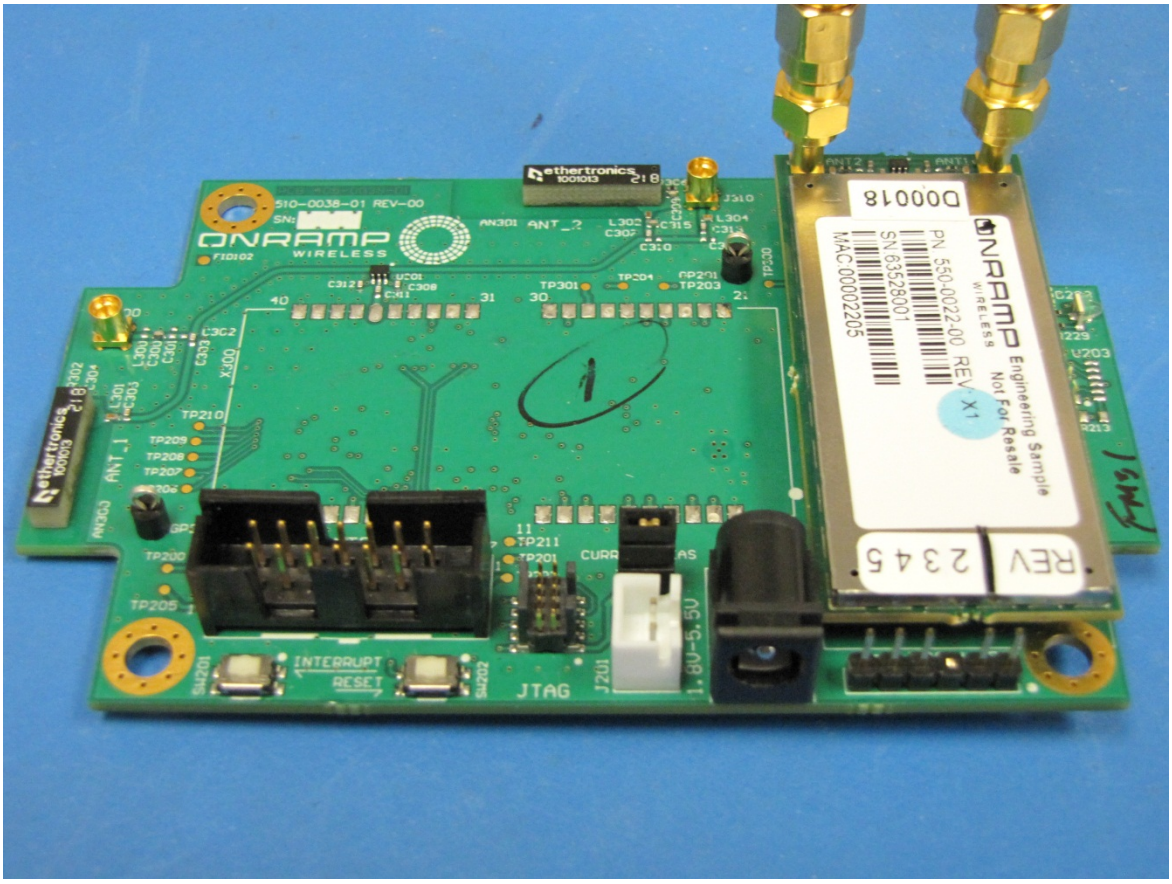
The microNode (X300) is laid out according to good design rules such as:

- Good thermal relief on rACM pads, as well as good ground
- Good bypassing with surges seen by a Node under a Wake/Sleep profile (C316/C317).
- Ability to turn off Node Power (pin X300-24) to allow for a Deep Sleep storage mode of a rACM-style module.

The microNode and dNodes are connected in a meaningful and standard way and it is recommended the user generally keep this interface. For instance, SRQ goes into a specific K20 port that allows the Node to awaken the K20 from its lowest of power modes. In a Deep Sleep mode, it is possible for:

- K20 to fall to a <5uA deep sleep mode
- Node to fall to its deepest sleep mode, of nominally <13uA.

The following photo shows the rACM platform populated with the dNode option. Previous photos show the microNode option.



As can be seen from the photo, the two coaxial RF antenna ports on the dNode are used instead of the on-board rACM antenna paths.

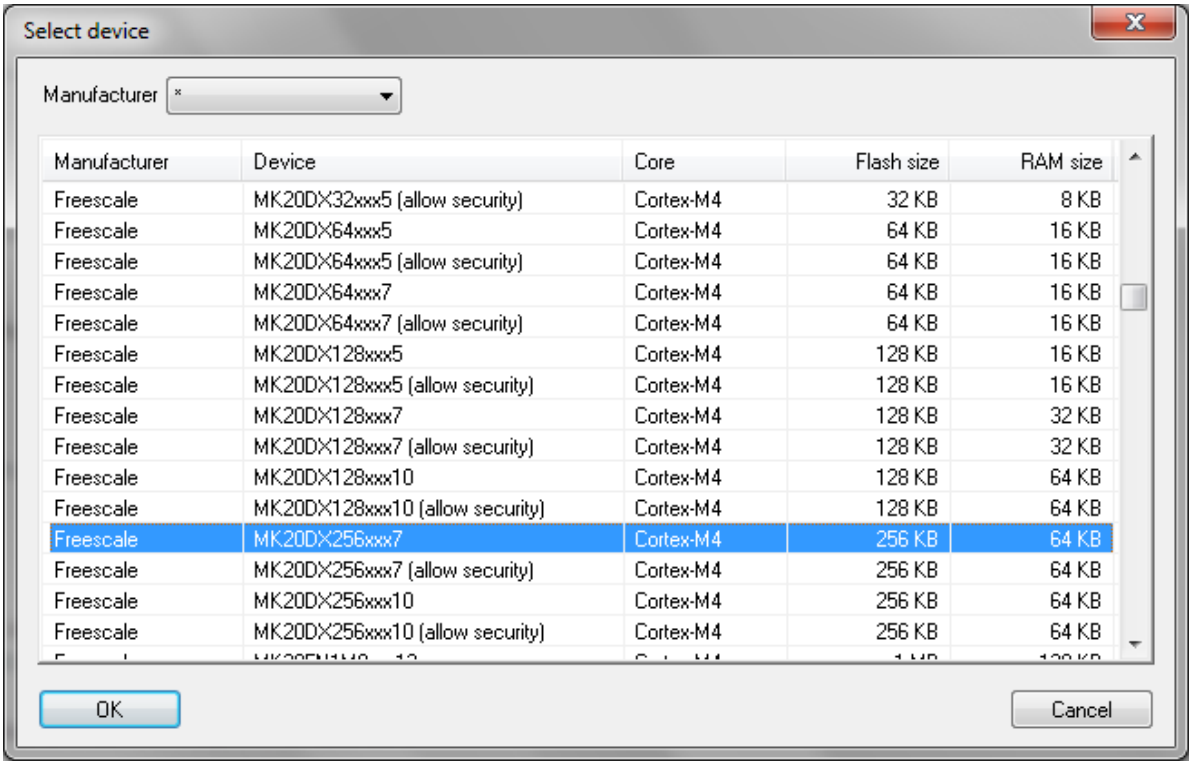
3 JTAG Code Download and Debug

During manufacturing test or application development, it is more efficient to download the application host software through the JTAG interface. The procedure for downloading and storing the rACM image to Flash varies based on the build method.

3.1 IAR EWARM Project

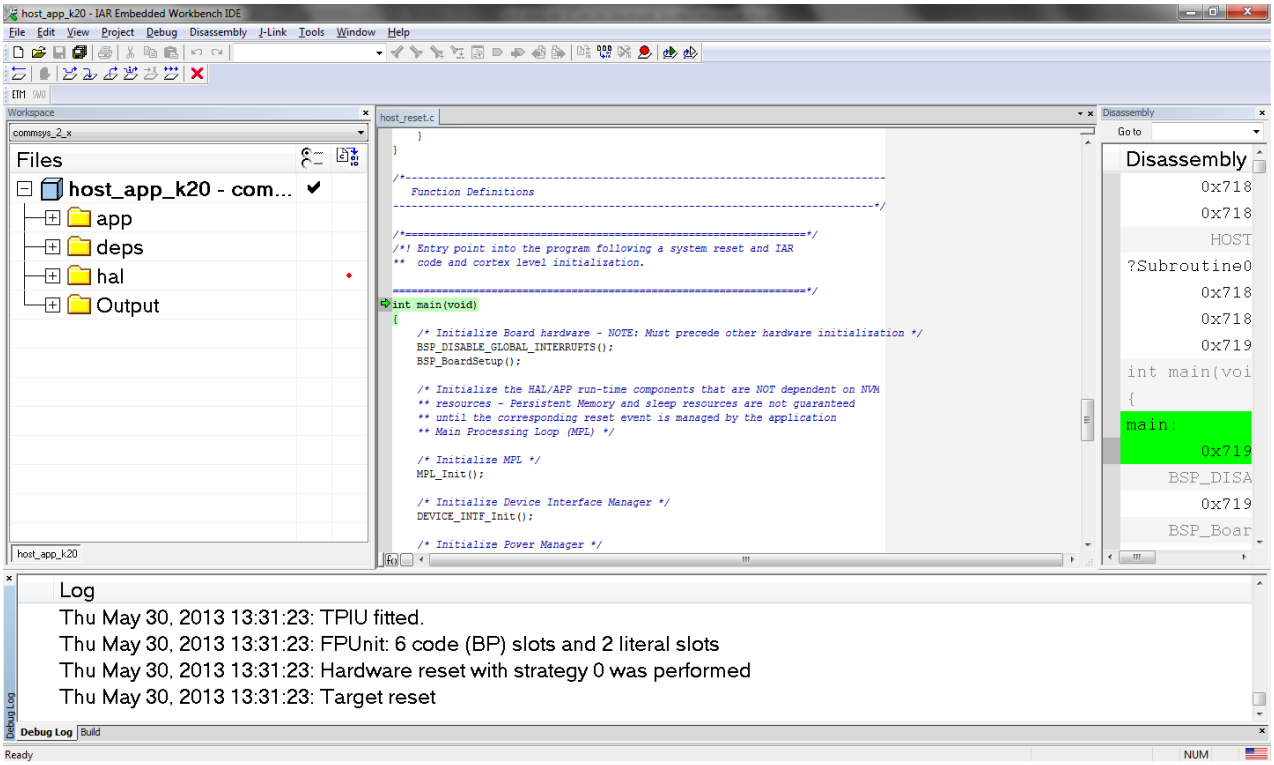
Perhaps the most straightforward method of downloading and debugging the rACM application is to use the rACM project which has been preconfigured with the necessary Flash loaders and J-Link interface settings necessary to perform this operation.

After downloading the project file (see section 1.5.1) and applying power to the reference platform, the 'Options->Download and Debug' command can be used to kick off the JTAG flashloader process (alternatively, the green-arrow command icon can be selected). When J-Link is invoked for the first time, it prompts the user to confirm the target processor as shown in the following figure.



Ensure that the MK20DX256xxx7 option is selected before proceeding with the download process. The rACM binary is less than 20K bytes. The actual download through the J-Link happens within a second at the configured clock rate of 48 MHz.

After the image is downloaded, the JTAG halts the processor at main() as the following figure shows:



NOTE: If the image downloaded through the JTAG is being done for development purposes (i.e., after code changes), we recommend that the application undergo a Power-On-Reset (POR) by removing the 3.3V source voltage following the successful upload. The ACM POR handler performs all of the necessary (re)initialization of the run-time variables. These may change as a result of software modifications. Otherwise, the rACM application host treats the JTAG reset as a recoverable scenario and attempts to resume steady-state operations. If the tracking variables have changed, this may lead to undefined behavior.

The IAR IDE contains all of the standard debug features (i.e., breakpoints, call stack trace, register display, memory display, etc.). Detailed steps for using the IAR Debugger is beyond the scope of this document. The EWARM Debugger manual is available within the IAR source directory or from the IAR website and contains detailed instructions and procedures for using the IAR IDE capabilities.

3.2 IAR Makefile Build

If the rACM application is built using the makefile process, the IAR Debugger can still be used to download and debug the .elf output file. The build directory contains a second project file called *host_app_k20_debug_only.eww*. Instead of source files, this project contains a single build file which can be downloaded in the same manner described for EWARM IDE project builds.

The only difference is that the user may be prompted to find the source file for debugging – depending on where the debugger is halted (i.e., *app*, *hal*, or *host_cmn* source directories).

4 rACM Operational Overview

The default rACM application build, when it first comes out of Power-On-Reset, operates under the following run-time parameters:

- rACM is configured for quick-start, without any user-prompt, the application powers on the Node and initiates the network join process.
- rACM is configured to enable and count pulses over the APP_INTF1 pin on the Application I/O header (J207) using the Low-Power Timer (LPT) Kinetis block.
- When rACM has joined the network (confirmed by monitoring the Status LED and/or rACM logger display), the application schedules a callback to read the LPT counter every hour on the hour. The value is time-stamped and saved into Non-Volatile Memory (NVM) for later OTA reporting at the Uplink Interval (UI).
- Each Uplink Interval (configured on a per-node bases), the rACM application reports to the backend all undelivered, time-stamped records in NVM in a single Tx Service Data Unit (SDU). If the UI is less than the read interval or if no read records are found in NVM, then the rACM reports the current read value at the time the UI is processed.
- When the rACM has joined the network, a swipe of the provisioning tool generates the asynchronous TEST_ALARM event. This, in turn, results in an asynchronous Tx SDU generated to the backend reporting the alarm state change.

  NOTE: All alarm state changes generated by the provisioning tool are subject to a 5-minute hysteresis period before an alarm state change can be generated again – this minimizes the amount of asynchronous SDU transfers generated by the application.
- The rACM application software is delivered to third-party integrators using the UNSECURED_POWERED application profile – i.e., Deep Sleep and UART Power Management are disabled by default on the rACM build. This is to allow unimpeded Host messaging capabilities that do not require the use of the interrupt switch or magnetic provisioning tool to wake the host application from deep sleep.

The following subsections discuss the reference platform interfaces that can be exercised during the course of steady-state operations on the rACM.

4.1 Status LED Patterns

The rACM uses the status LED to provide visual feedback during operational state changes and network join attempts. Distinct sequences are used to identify each state. The sequence consists of the 5-second pattern in the following table, repeated twice.

Table 1. Status LED Patterns

Name	Pattern
SHUTDOWN	1 x (½ sec ON + ½ sec OFF) + 4 sec OFF
PROVISION	2 x (½ sec ON + ½ sec OFF) + 3 sec OFF

Name	Pattern
JOIN	5 x (½ sec ON + ½ sec OFF)
NETWORK_DROP	1 x (2 ½ sec ON + 2 ½ sec OFF)

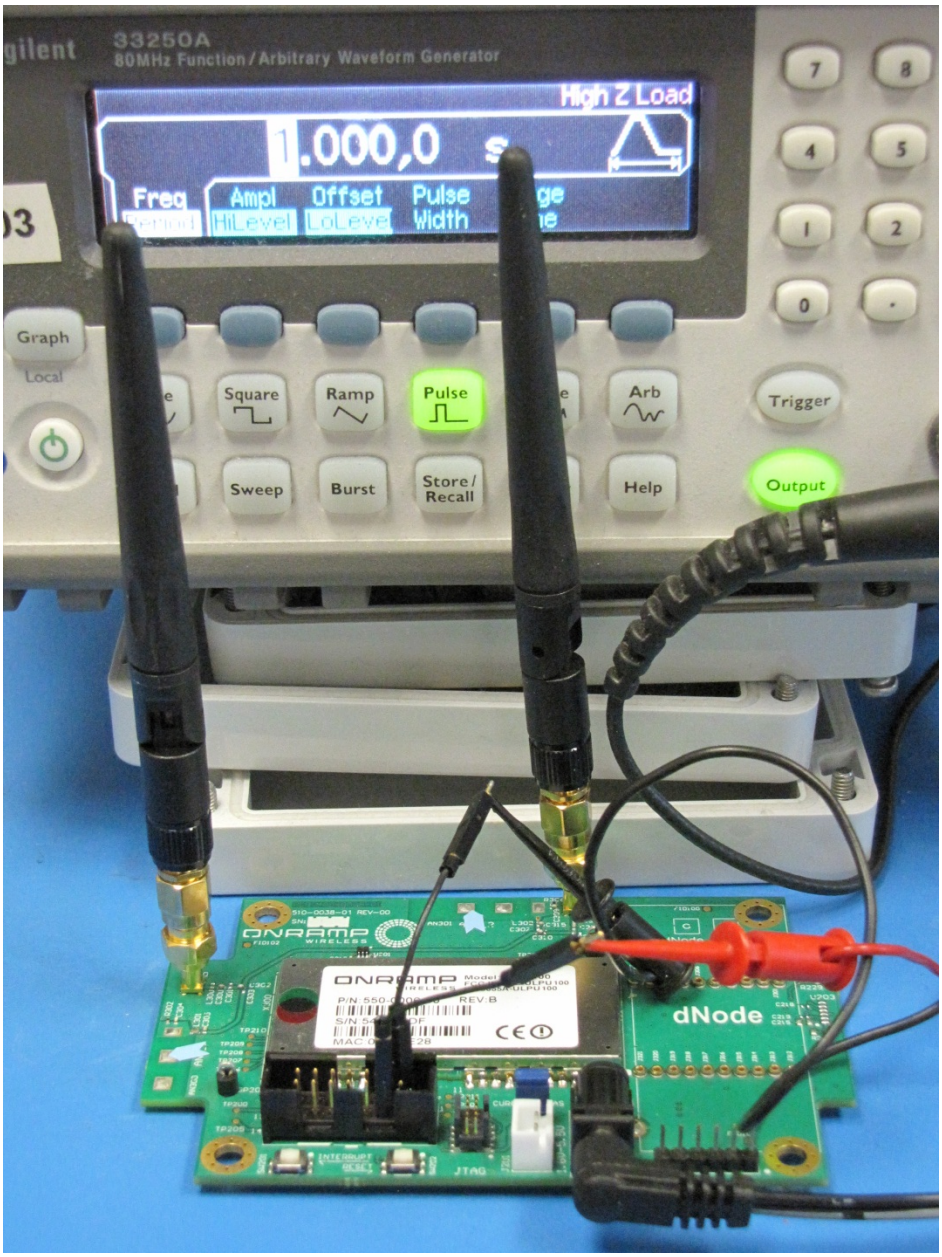
The rACM application autonomously generates the status LED pattern when it first powers on and when the device joins the network for the first time. The current device state is also reported in response to each swipe of the magnetic provisioning tool via the Reed Switch.

4.2 Pulse Counter Inputs

As mentioned in this section overview, the rACM's default sensor interface utilizes the low-powered pulse counter configured to monitor pin APP_INTF1 on the Application I/O header (J207). At each read interval, the application reads the LPT counter and stores the value into NVM. However, without any periodic pulses applied to this input, the value remains at zero which is not a very interesting demonstration of the rACM logging and reporting capabilities.

The recommended method to generate count data is to use a waveform generator and generate a periodic pulse with active high polarity and amplitude of 3.3V. The width of the pulse is typically expected to be around 100msec. The period of the pulse waveform is at the discretion of the end-user.

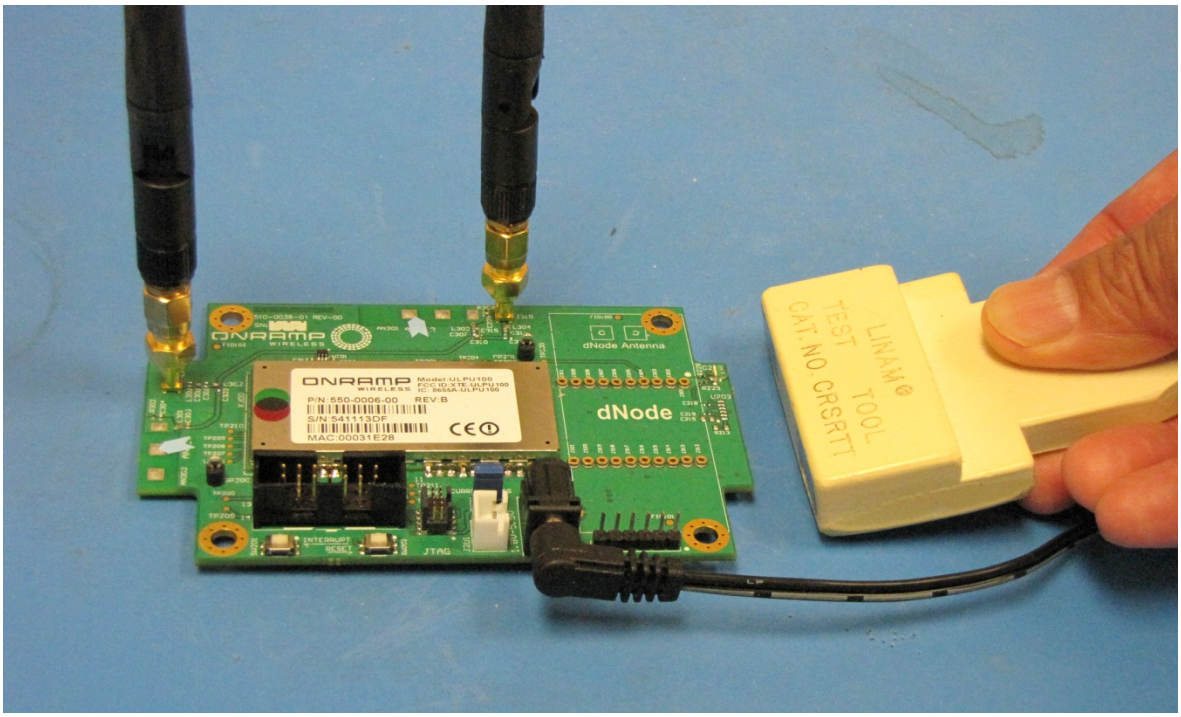
The following figure shows the output of the waveform generator (pulse_out and ground) connected to the reference platforms' Application I/O:



The current pulse count can be confirmed by using the GET_DEVICE_STATUS command via the Host Interface UART or by post-processing the Tx SDU payload data from the On-Ramp Total View application.

4.3 Alarm Inputs

As mentioned in the overview section of this chapter, asynchronous alarms can be demonstrated through the use of the magnetic provisioning tools with a single “swipe” across the Reed Switch circuit (SW200) mounted on the front edge of the reference platform (i.e. the edge of the platform where the status LED is mounted) as the following figure demonstrates.



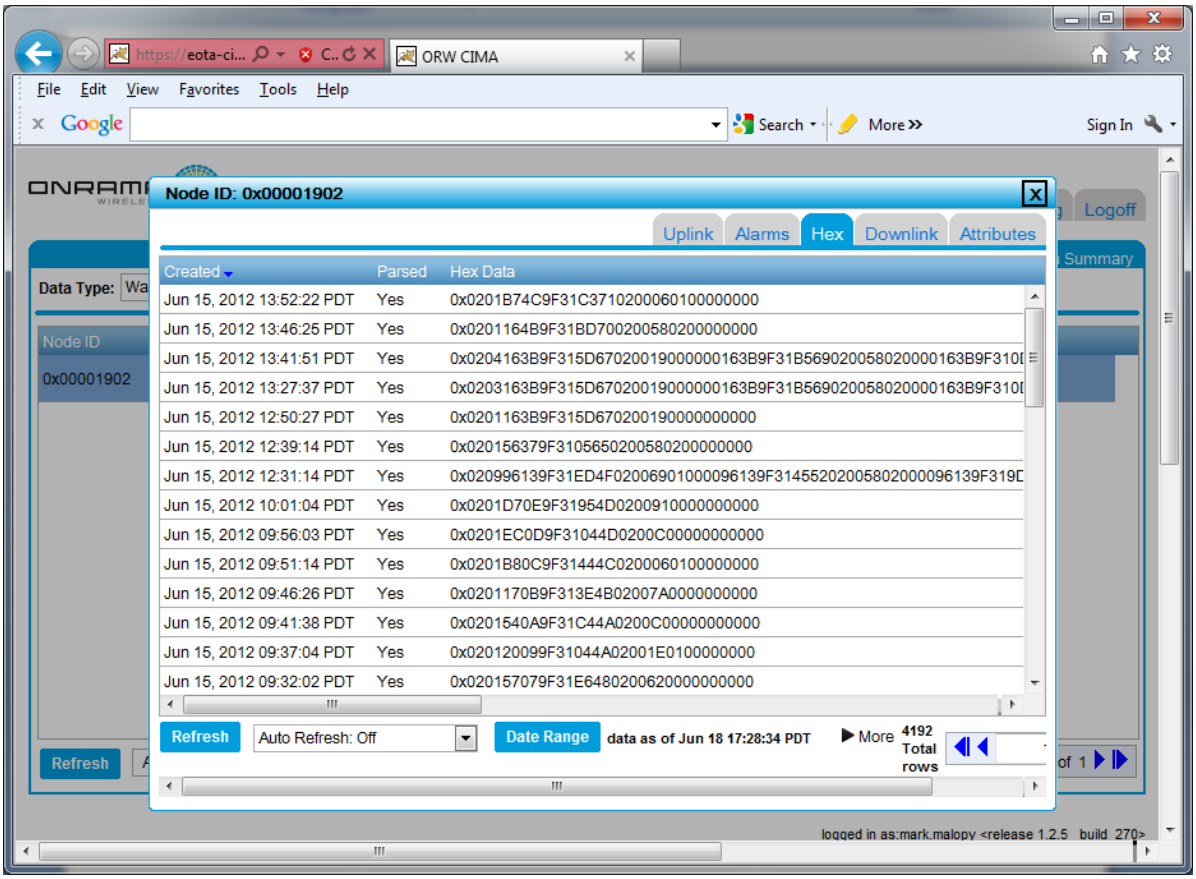
The Reed Switch swipe can be confirmed if the status LED responds with the current state pattern (see section 4.1). Alternatively, the external interrupt switch button (SW201) can be depressed to generate the same feature.

It is recommended that the rACM Logger application (see section 5.2) be monitored when this event occurs as the Reed Switch “swipe” triggers a state change for the TEST alarm type. If the Alarm Hysteresis is not in effect (default period of five minutes), then an asynchronous Tx SDU is generated with payload data that reports the alarm type and state along with a corresponding time stamp. The rACM application also manages conflicts if a Tx SDU is already in progress. It buffers the alarm data and waits for the pending network transaction to finish. By monitoring the rACM Logger, all of these types of application events and/or conflict management are displayed in real-time.

4.4 On-Ramp Total View Payload Verification

All Tx SDUs generated by the rACM application eventually get routed through the backend to the On-Ramp Total View application. It is assumed that, as part of the network configuration process, the Node on the reference platform has been added as a device by the On-Ramp Wireless Element Management System (EMS). Details about the On-Ramp Total Reach backend components are considered beyond the scope of this document.

By clicking on the Node ID associated with the rACM device, all Tx SDU payload data can be verified by monitoring the ‘Hex’ tab. Each Tx SDU received by the On-Ramp Total View application contains the time-stamped hexadecimal payload data as the following figure illustrates:



Part II of this document contains all of the message formats necessary to parse the raw hex data (see section 9.2).

5 rACM Host Tools

The *python_tools* directory all of the necessary Python scripts for interfacing with the rACM application through the Host Interface UART. All control scripts over the Host Interface use the following properties:

- Default baud rate of the Host UART is 115200bps. The `-b` option is used to override/set the baud rate to match any changes on the application host.
- Default Com port is COM0 (on a Windows platform) or /dev/ttyS0 (on a Linux platform). The `–d` option is used to override/set the actual port used to communicate with the application host.
- The `–h` or `–help` option displays the usage for the current control script.

5.1 rACM Control Script

The *host_app_ctrl.py* script is the primary utility script for configuring and status reporting of the rACM application. Here is a summary of the support command options.

Table 2. Support Command Options

Command	Description
HOST_VERSION	Reports the major, minor, and point release values of the rACM application currently executing on the reference platform.
ENTER_PASSCODE	If the UART security features are enabled (see chapter 6), this is the command is used to unlock the Host UART interface. This command (with a valid passcode) must precede any other rACM commands. In addition, a configurable security lockout feature can be entered into if the passcode is entered incorrectly in a set number of passcode attempts (see reference [1] for more details on the Host Interface security feature).
CONFIG_NODE_ATE	If the device is in the shutdown state, this command can be used to turn on the Node for purposes of network configuration and/or network debugging. If the rACM application is already in an operational state, this command is ignored.
GET_DEVICE_STATUS	The command can be used to report the status of the rACM application, its configuration parameters, and any current sensor read data (i.e., the absolute count value).

Command	Description
SET_APPLICATION_PROFILE	This command is used to configure the basic application behavior (i.e., profile) and is currently defined by the following: ■ FACTORY_MODE – Unsecured and powered device with application place in passthrough mode. ■ UNSECURED_POWERED – Unsecured and powered application. ■ UNSECURED_SLEEPY – Unsecured and deep sleep capable application. ■ SECURED_POWERED – Secured and powered application. ■ SECURED_SLEEPY – Secured and deep sleep capable application. Further details on the application profile can be found in section 7.9 in part II of this document. This setting is persistent across system resets and/or power cycles.
SET_ READ_INTERVAL	This command is used to configure the Read Interval rate and time synchronization offset. The rACM Software HLD contains the table of all supported read intervals. It should be noted that the time synchronization feature uses GMT as opposed to local timing. This setting is persistent across system resets and/or power cycles. The Read Interval can be configured for the following read blocks. ■ READ_BLOCK1 ■ READ_BLOCK2 Read blocks can be assigned to any number of application interfaces (see the SET_APP_INTF_READ_MAPPING command),
SET_POLL_INTERVAL	This command is used to configure the Poll Interval rate and time synchronized offset. Unlike the read interval, sensor data at the poll interval rate are not reported to the backend (i.e., they are for alarm threshold detection purposes only). As with the read interval, the time synchronization feature uses GMT as opposed to local timing. This setting is persistent across system resets and/or power cycles. The Poll Interval can be configured for the following poll blocks: ■ POLL_BLOCK1 ■ POLL_BLOCK2 ■ POLL_BLOCK3 Poll blocks can be assigned to any number of application interfaces (see the SET_APP_INTF_POLL_MAPPING command).
SET_ALARM_HYSTERESIS	This command is used to set the delay, in seconds, between generating alarm state changes (the default is 600 seconds – i.e., 5 minutes) that result in the generation of an asynchronous Tx SDU request. This feature helps minimize the amount of transmission spamming that may result from a alarm hardware circuit with transient “glitches”. This setting is persistent across system resets and/or power cycles.
CONFIG_LVD_ALARM	This command is used to configure Low-Voltage Detect (LVD) threshold alarm settings. This setting is persistent across system resets and/or power cycles.

Command	Description
SET_ NETWORK_DISCOVERY	This command is used to enable or disable the unsolicited generation of the Configuration Read Response message (opcode 0x4 – see reference [1]) when the device joins the network for the first time following a Power-On Reset (POR). This is a form of a “birthday boot” mechanism in which the application host’s configuration parameters can be used by the backend for display and/or device capability features. This setting is persistent across system resets and/or power cycles.
SET_UART_TIMEOUT	This command is used to set the inactivity timeout value, in seconds, at which point to power down the Host Interface UART for power savings purposes. This setting is persistent across system resets and/or power cycles.
SET_UART_PASSCODE	This command is used to set the UART Hexadecimal passcode value for Host Interface security purposes. This setting is persistent across system resets and/or power cycles.
SET_UART_PASSCODE_ALARM_THRESH	This command sets the alarm threshold for the Host UART pass-through security lockdown feature for a configurable number of invalid passcode attempts. It should be noted that the only way to unlock the security lockdown is from the backend via a Key-Value Pair. This setting is persistent across system resets and/or power cycles.
RESET_DEVICE	This command is used to clear and reset the following device parameters: ■ All device state parameters revert to the Shutdown state (i.e., device state, OTA state, host state). ■ Internal NVM log buffer is cleared/reset ■ All registered timer callbacks are flushed/cleared. ■ Field Configuration parameters/key-value pairs revert to software defaults.
HOST_UART_OVERRIDE	This command is used to enable/disable Host UART Power Control. This setting is persistent across system resets and/or power cycles.
SET_PROVISION_QUICK_START	This command is used to enable/disable the provisioned quick start state. When set, it essentially powers on the Node and initiates the network join process following a POR event. If cleared, the device remains in a shutdown state until a “swipe” of the magnetic provisioning tool occurs (i.e., during field deployment). This setting is persistent across system resets and/or power cycles.
SET_RHT_SUPPORT	This command is used to enable/disable the Reliable Host Transfer (RHT) protocol on all Host Interface serial packets (i.e., a retry mechanism to compensate for bit errors on the Host Interface serial link). If enabled, all command line scripts must include the *–wrapper=rht_v2* option for host messaging to properly function.

Command	Description
SET_APP_SERIAL_CTRL	This command is used for run-time configuration of the example APP UART Serial Parsing utility. It can be set for the following options: ■ LOCAL_ECHO – Echos any received ASCII bytes received over the Application UART). ■ OTA_PASSTHROUGH – Downlink and uplink OTA network transfer of ASCII character strings. Uplink strings are transmitted best-effort, one string at a time (i.e., no buffering of uplink data). ■ OTA_BUFFERED – Similar to OTA_PASSTHROUGH with the added buffering of all uplink strings (default circular buffer size of 4k).
SET_POR_SERIAL_CTRL	This command is used for Power-On-Reset (POR) initialization of the APP UART Serial Parsing utility. It can be disabled by setting it to NONE or through one of the following options: ■ LOCAL_ECHO – Echos any received ASCII bytes received over the Application UART). ■ OTA_PASSTHROUGH – Downlink and uplink OTA network transfer of ASCII character strings. Uplink strings are transmitted best-effort, one string at a time (i.e., no buffering of uplink data). ■ OTA_BUFFERED – Similar to OTA_PASSTHROUGH with the added buffering of all uplink strings (default circular buffer size of 4k).
SET_EXT_SWITCH_FUNCTION	This command is used to configure additional functionality with the external (reed) switch. The current options are: ■ PROVISION_ONLY – Only used for device provisioning/deployment. ■ ALARM_TOGGLE – In addition to provisioning, can be used to generate the ALARM0 state change. ■ OTA_CONFIG_GEN – In addition to provisioning, can be used to generate an OTA Config message This setting is persistent across system resets and/or power cycles.
SET_APP_INTF_MODE	This command is used to configure an application interface pin on header J207 for one of the following modes: ■ DIG_INPUT – GPIO Digital Input ■ DIG_OUTPUT – GPIO Digital Output ■ FUNC_MODE0 – Alternative Kinetis Function 0 ■ FUNC_MODE1 – Alternative Kinetis Function 1 Further details on the application interface modes, as well as their defaults, can be found in section 7.11 in part II of this document. This setting is persistent across system resets and/or power cycles.
SET_APP_INTF_OTA_DATA_ REPORT	This command is used to configure OTA reporting capability to an individual application interface – this includes the application header pins (APP_INTF1 – APP_INTF5), the dedicated analog interface (ANA_IN0), as well as virtual interfaces. Further details on application interface OTA reporting can be found in section 7.11 in part II of this document. This setting is persistent across system resets and/or power cycles.

Command	Description
SET_APP_INTF_READ_MAPPING	This command is used to assign an individual application interface with one of the configured read blocks (configured via the SET_READ_INTERVAL command). This allows periodic read updates on a given interface for purposes of OTA data reporting and/or alarm monitoring. This setting is persistent across system resets and/or power cycles.
SET_APP_INTF_POLL_MAPPING	This command is used to assign an individual application interface with one of the configured poll blocks (configured via the SET_POLL_INTERVAL command). This allows periodic poll updates on a given interface for purposes of alarm monitoring. This setting is persistent across system resets and/or power cycles.
SET_APP_INTF_DIG_IN_CFG	This command is used to configure the following GPIO Digital Input configuration capabilities for the application interface pins on header J207: ■ Interface internal resistor pull capability ■ Digital Input alarm type (i.e., none, active low/high) ■ GPIO Interrupt capability. Further details on the GPIO digital input capabilities can be found in section 7.11 in part II of this document. This setting is persistent across system resets and/or power cycles.
SET_APP_INTF_DIG_OUT_CFG	This command is used to configure the following GPIO Digital Output configuration capabilities for the application interface pins on header J207: ■ Open Drain capability ■ Drive Strength capability ■ Static Output Level. Further details on the GPIO digital output capabilities can be found in section 7.11 in part II of this document. This setting is persistent across system resets and/or power cycles.
SET_ADC_SAMPLE_CALC_PARAMS	This command is used to configure ADC sample calculation parameters for a selected ADC interface block which includes: ■ Sample Period window, in msec. ■ Number of subsamples per sample period window. Further details on the ADC sample sequence can be found in section 8.6.6 in part II of this document. This setting is persistent across system resets and/or power cycles.
SET_ADC_ALARM_TYPE	This command is used to configure the ADC alarm type (see section 8.6.6). This setting is persistent across system resets and/or power cycles.
SET_ADC_ALARM_UPPER_THRESH	This command is used to configure the ADC alarm upper threshold Hi/Lo water marks used to determine alarm state changes (see section 8.6.6). This setting is persistent across system resets and/or power cycles.
SET_ADC_ALARM_LOWER_THRESH	This command is used to configure the ADC alarm lower threshold Hi/Lo water marks used to determine alarm state changes (see section 8.6.6). This setting is persistent across system resets and/or power cycles.
HOST_SW_UPGRADE	This command is used to do a firmware upgrade, using the rACM binary image, through the Host UART Interface.
GET_FACTORY_CONFIG	This command is used to display the factory configuration data saved in rACM Non-Volatile Memory (NVM) at the PC Host.

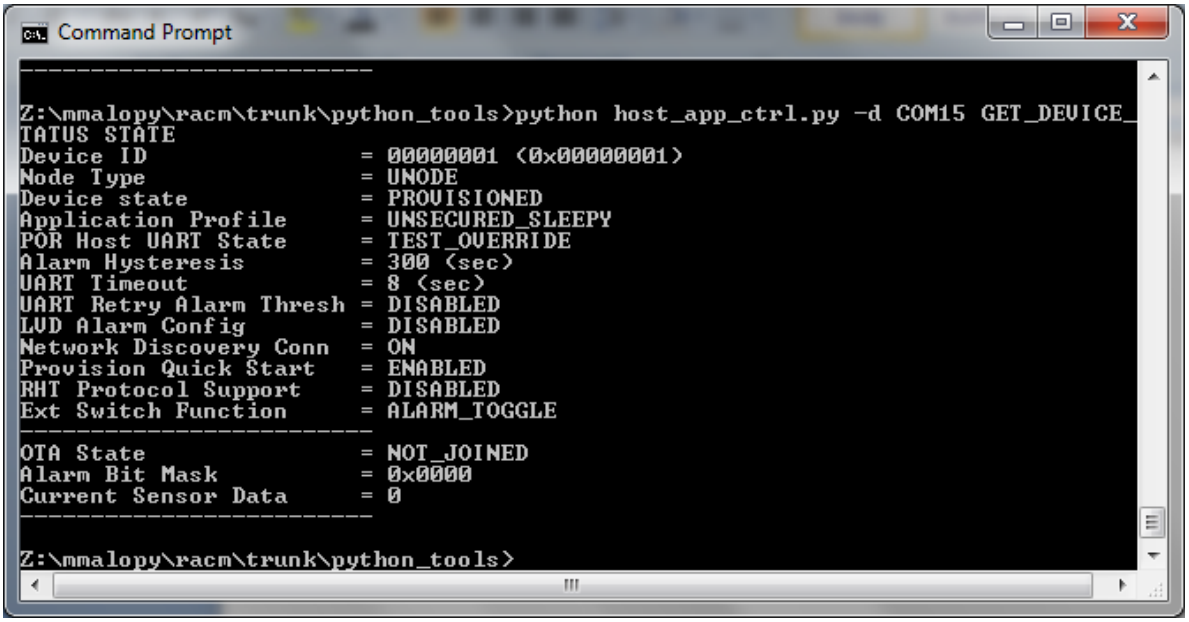
Command	Description
SET_FACTORY_CONFIG	This command is used to program the applicable section of rACM Non-Volatile Memory (NVM) with factory configuration data from a configuration .txt file on the PC host.
DUMP_READ_RECORDS	This command is used to dump all undelivered read records from rACM Non-volatile Memory (NVM) to the PC host.
DUMP_ALARM_RECORDS	This command is used to dump all undelivered alarm records from rACM NVM to the PC host.
DUMP_EXCEPTIONS	This command is used to dump all exception event records from rACM NVM to the PC host (e.g., Watchdog Timeout exceptions, Hard Faults, Logging failures, etc.).
DUMP_NODE_EVENTS	This command is used to dump all Node state changes from rACM NVM to the PC host.
CLEAR_EXCEPTIONS	This command is used to clear all exception event records from rACM NVM.
CLEAR_NODE_EVENTS	This command is used to clear all Node state changes from rACM NVM.
TEST_OTA_ALARM	This test command can be used to generate an asynchronous alarm type and state to the backend. However, Tx SDU generation is subject to the rACM state event handlers (i.e., it may be queued until a current Tx SDU request has completed). Unlike the application alarm events, alarms generated by this command are NOT subject to alarm hysteresis.
TEST_ADC_INTF_READ	This test command is used to initiate a read on one of the following ADC blocks for purposes of testing/validating an analog interface: ■ ADC1 – The analog input on ANA_IN0 ■ ADC2 – The analog input on APP_INTF5 (if configured in FUNC_MODE0) ■ VREAD1 – Voltage at the output of the rACM power regulator. ■ VREAD2 – Voltage at the input of the rACM power regulator. The user may specify the number of A/D samples iterations for the ADC interface or use ‘0’ to use configuration defaults.
TEST_DIGITAL_INTF_READ	This test command is used to initiate a read on all of the application interfaces on J207 configured as DIG_INPUT or DIG_OUTPUT.
TEST_DIGITAL_RELAY_TOGGLE	This test command is used to toggle the output level of a selected interface on J207 configured as a DIG_OUTPUT for a specified amount of time (255msec max).
SET_KEY_PAIR_VALUE	This command is used to change a single key-value pair (see chapter 6).This setting is persistent across system resets and/or power cycles.

5.1.1 Get Device Status Example

The GET_DEVICE_STATUS command is a useful for reporting the current snapshot of the rACM state variables and configuration settings. The following arguments are used to filter the configuration parameters:

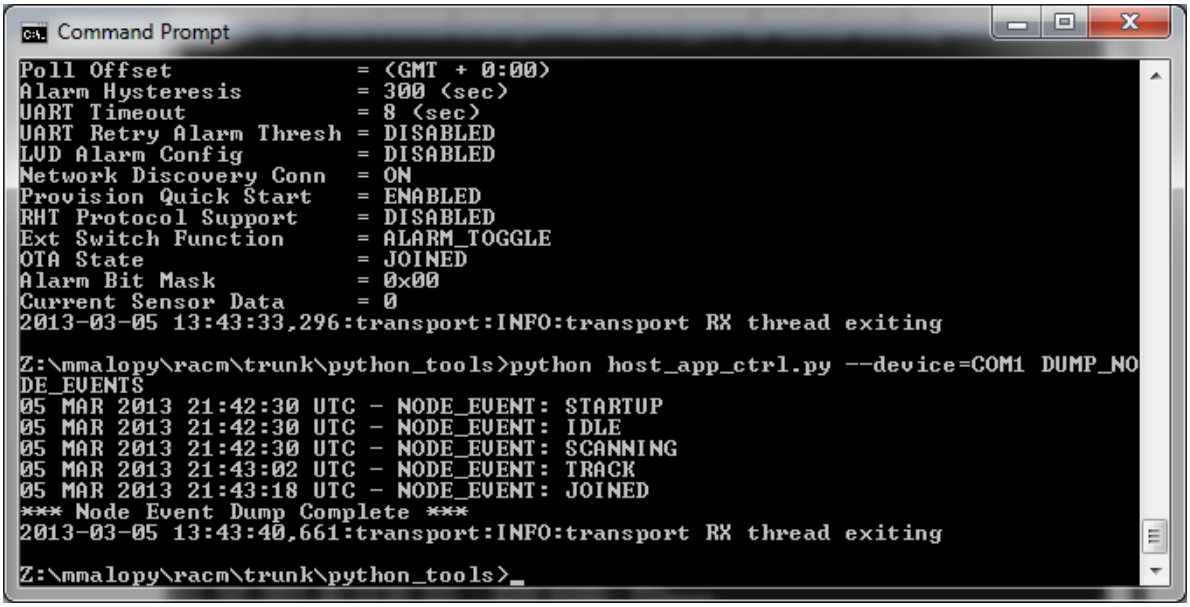
- **ALL** (default if no command line argument specified) – All rACM state and configuration information.
- **STATE** – The run-time state information of the rACM (e.g., OTA State, Host Interface State, etc.).
- **CORE** – The core feature set of rACM host processor configuration parameters (e.g., Application Profile, Low-Voltage Detect, etc.). These are features that are common across different application types that do not involve external interfaces.
- **INTERVAL_CFG** – The configuration parameters associated with the Read and Poll Intervals.
- **APP_INTF** – The configuration parameters associated with the application interfaces.
- **DIG_IN** – The configuration parameters associated with an application interfaces' GPIO Digital Input capabilities (e.g., internal impendence, alarm/interrupt capability, etc.).
- **DIG_OUT** – The configuration parameters associated with an application interfaces' GPIO Output capabilities (e.g., Output Level, Drive Strength, etc.).
- **ANALOG** – The configuration parameters associated with an application interfaces' ADC sampling capabilities (e.g., number of averaged samples, sample period, etc.).

The following screenshot shows the typical response to this command using the STATE option:



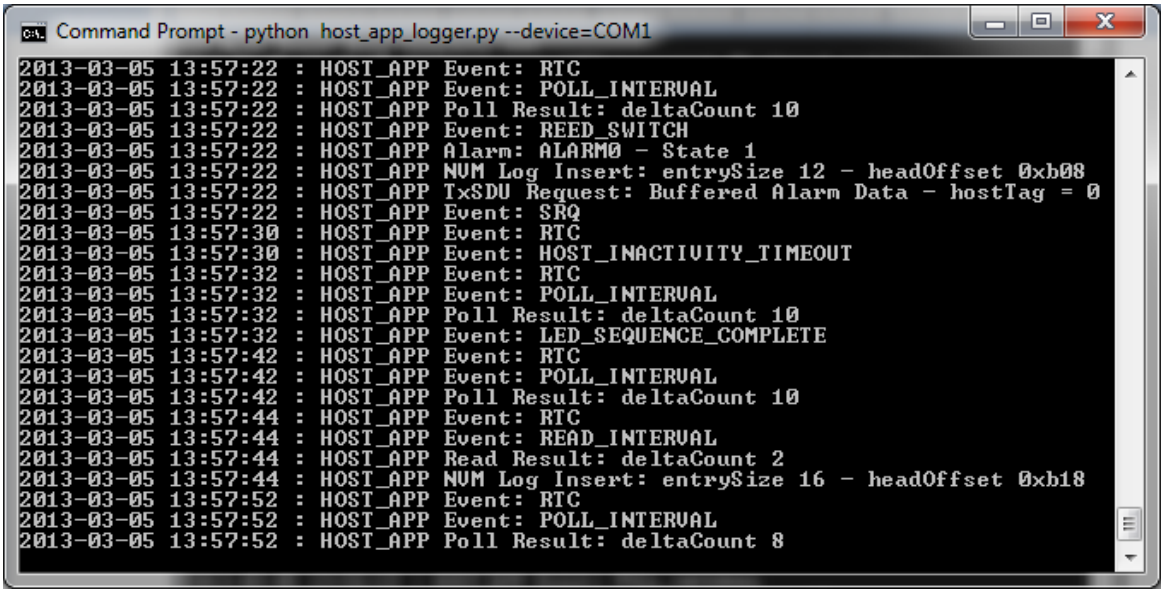
5.1.2 NVM Dump Example

The DUMP commands for read records, alarms, exceptions and node events are also very useful for debugging events for devices that are field deployed. Each record contains the GMT timestamp for when it was logged. The following figure shows an example of the Node events for a rACM device coming out of Power-On Reset:



5.2 rACM Logger Script

The *host_app_logger.py* script is allows for on-target monitoring of all of the rACM application events that occur in real-time through the Host UART. This is an excellent resource for understanding the flow of the rACM event handlers in response to various input stimuli. Each entry is time-stamped as they are received by the PC host to help provide an idea of the timing between the various event processing. The following screen capture shows the log output during an Alarm state change triggered through the Reed Switch.



All of the displayed strings by the logger utility are taken directly from the rACM source code via the Log Dictionary file generated as part of the build process (see section 1.6). By referencing the rACM logger event strings with the source code, one can quickly deduce and reconstruct the flow of the Main Processing Loop implementation.

NOTES:

1. To successfully process the rACM logger string output, the dictionary file in the build path MUST match the build that is flashed on target. A mismatch in the dictionary hash stamp that is reported over the Debug Log Indicators results in a parsing exception.
2. The rACM Logger application defaults to search for the rACM dictionary file in the project output generated by the iar_build (i.e*., build/iar_host_app/Dict*). The end-user can choose to override the default Dictionary path/filename by using the `-l` or `-logdict` command line option. For example:

   `--locdict=../build/gcc_host_app/commsys_2_x/Dict)`

6 rACM Configuration Options

Once the basic rACM implementation is understood, the end-user can choose to reconfigure the device configuration parameters. Using the same test methods and logging capabilities, the rACM application can be observed under the following conditions:

- rACM event flow with Deep Sleep enabled
- Different read intervals (e.g., 15 minute reads vs. 1 hour).
- Different time synchronization settings (e.g., 1 hour reads synchronized to at the 30 minute offset, 1 daily read to occur at 0:00 GMT, etc.).
- The UART Host Interface security feature can be enabled/tested to prevent unauthorized access.
- Quick start can be disabled to demonstrate the field provisioning process.

All of these configuration options can be read or modified using a designated Key-Value Pair for the rACM application. Each Key-Value pair can also be modified over the Host UART Interface or from the backend via the Write Configuration Command (opcode 0x3). All Key-Value pairs are stored in FlexNVM and protected with a 16-bit CRC.

NOTE: A DEVICE_RESET command causes all Key-Value pairs to revert to their software default values in NVM.

The following table summarizes the rACM Key-Value Pairs currently implemented – NOTE: The core key-value pairs (i.e., host configuration options that are common across different application types) are shaded – followed by application-specific key-value pairs (i.e., host configuration parameters that may change based on application implementation).

Table 3. rACM Key-Value Pairs

ORW ID	Configuration	Description
1	UART Timeout	Timeout in seconds Default = 10 seconds
2	UART Password0	Password[15:0]
3	UART Password1	Password[31:16]
4	UART Password2	Password[47:32]
5	UART Password3	Password[63:48]
6	UART Password Disable Threshold	Number of failures 0 = never disable
7	Low Battery Alarm Threshold	Threshold defined by LVD logic 0 = disabled 1 = 3.0V 2 = 2.9V 3 = 2.8V 4 = 2.7V 5 = 2.56V

ORW ID	Configuration	Description
8	Alarm Hysteresis	Number of seconds to disable alarm status reporting following an alarm state change. 0 – disables feature Default = 300 seconds
9	POR Default Host Interface State	Power-on-Reset Default state for Host UART. 0 = Inactive 2 = Active 3 = App Override
10	Network Discovery Connect	Feature to send all Configuration Key-Value pairs when the network is discovered for the first time: 0 = OFF 1 = ON
11	Provision Quick Start	Feature to skip provision step at Power-on-Reset and immediately initiate the network join process. 0 = OFF 1 = ON
12	RHT Support	Feature to enable the Reliable Host Transfer (RHT) protocol on all Host Interface serialized data. 0 = OFF 1 = ON
13	External Switch Functionality	Additional functions taken when the external reed switch is triggered: 0 = PROVISION_ONLY – External reed switch is only used for provision state control 1 = ALARM0_TOGGLE – Once the device is joined, the external reed switch will also generate an ALARM0 state change. 2 = OTA_CONFIG_RSP – Once the device is joined, the external reed switch will also generate the uplink Config Response message (see section 4.2.4).
14	Application Profile	Sets the general operational application parameters (see section 7.9): 0 = FACTORY_MODE 1 = UNSECURED_POWERED 2 = UNSECURED_SLEEPY 3 = SECURED_POWERED 4 = SECURED_SLEEPY
15	POR Serial Function	Sets the default POR configuration of the Application Serial Interface: 0 = NONE 1 = LOCAL_ECHO 3 = OTA_PASSTHROUGH 4 = OTA_BUFFERED

ORW ID	Configuration	Description
16	Application Interface Mode Bit 1:0 – APP_INTF1 Config Bit 3:2 – APP_INTF2 Config Bit 5:4 – APP_INTF3 Config Bit 7:6 – APP_INTF4 Config Bit 9:8 – APP_INTF5 Config Bit 15:10 - reserved	Sets the functional mode for the 5 application interface configurable inputs on the application header (J207). The possible settings for each interface pin are as follows (see section 7.11): 0 = DIG_INPUT (GPIO digital input) 1 = DIG_OUTPUT (GPIO digital output) 2 = FUNC_MODE0 (functional mode varies based on port capabilities) 3 = FUNC_MODE1 (functional mode varies based on port capabilities)
17	Over-The-Air (OTA) Report Config Bit 0 – APP_INTF1 Bit 1 – APP_INTF2 Bit 2 – APP_INTF3 Bit 3 – APP_INTF4 Bit 4 – APP_INTF5 Bit 5 – APP_INTF6 Bit 6 – APP_INTF7 Bit 7 – APP_INTF8 Bit 15:9 – reserved	Boolean configuration value that sets OTA reporting capability for each of the rACM application interfaces (i.e., read data reported at each UI interval data – see section 7.11): 0 – OTA reporting disabled for selected interface 1 – OTA reporting enabled for selected interface
18	Application Interface Read Mapping Bit 1:0 – APP_INTF1 Bit 3:2 – APP_INTF2 Bi t5:4 – APP_INTF3 Bit 7:6 – APP_INTF4 Bit 9:8 – APP_INTF5 Bit 11:10 – APP_INTF6 Bit 13:12 – APP_INTF7 Bit 15:14 – APP_INTF8	Sets the read block mapping/capability for each of the rACM application interfaces (i.e., which rate/time to initiate a read sequence- see section 7.11): 0 – NONE (no reads for selected interface) 1 – READ_BLOCK1 (Read interval rate as defined by key-value ID 22) 2 – READ_BLOCK2 (Read interval rate as defined by key-value ID 23) 3 – reserved (undefined)
19	Application Interface Poll Mapping Bit 1:0 – APP_INTF1 Bit 3:2 – APP_INTF2 Bit 5:4 – APP_INTF3 Bit 7:6 – APP_INTF4 Bit 9:8 – APP_INTF5 Bit 11:10 – APP_INTF6 Bit 13:12 – APP_INTF7 Bit 15:14 – APP_INTF8	Sets the poll block mapping/capability for each of the rACM application interfaces (i.e., which rate/time to initiate a poll sequence- see section 7.11): 0 – NONE (no polls for selected interface) 1 – POLL_BLOCK1 (Poll interval rate as defined by key-value ID 24) 2 – POLL_BLOCK2 (Poll interval rate as defined by key-value ID 25) 3 – POLL_BLOCK3 (Poll interval rate as defined by key-value ID 26)

ORW ID	Configuration	Description
20	Read Block 1 Interval Bit 3:0 – Interval Duration Bit 15:4 – Interval Minute Offset (from 0:00 GMT)	Duration Default = 2 (1 hour) 0 = 15 min 8 = 12 hour 1 = 30 min 9 = 24 hour 2 = 1 hour 10 = 30 sec 3 = 2 hour 11= 1 min 4 = 3 hour 12 = 2 min 5 = 4 hour 13 = 4 min 6 = 6 hour 14 = 8 min 7 = 8 hour 15 = none Minute Offset Default = 0x000 (0:00 GMT) **NOTE:** If Minute Offset = 0x3FF, read intervals are asynchronously scheduled for when the rACM first joins the network.
21	Read Block 2 Interval Bit 3:0 – Interval Duration Bit 15:4 – Interval Minute Offset (from 0:00 GMT)	Same definitions as Read Block 1 Interval setting (ORW ID 22)
22	Poll Block 1 Interval Bit 3:0 – Interval Duration Bit 15:4 – Interval Minute Offset (from 0:00 GMT)	Duration Default = 15 (none) 0 = 1 sec 8 = 10 min 1 = 2 sec 9 = 15 min 2 = 5 sec 10 = 30 min 3 = 10 sec 11 = 1 hour 4 = 30 sec 12 = 2 hour 5 = 1 min 13 = 3 hour 6 = 2 min 14 = 4 hour 7 = 5 min 15 = none Minute Offset Default = 0x000 (0:00 GMT) **NOTE:** If Minute Offset = 0x3FF, poll intervals are asynchronously scheduled for when the rACM first joins the network.
23	Poll Block 2 Interval Bit 3:0 – Interval Duration Bit 15:4 – Interval Minute Offset (from 0:00 GMT)	Same definitions as Poll Block 1 Interval setting (ORW ID 24)
24	Poll Block 3 Interval Bit 3:0 – Interval Duration Bit 15:4 – Interval Minute Offset (from 0:00 GMT)	Same definitions as Poll Block 1 Interval setting (ORW ID 24)
25	APP_INTF1 Digital Configuration Bit 7:0 – Kinetis PORT_PCR setting Bit 8 – Digital Output level setting Bit 10:9 – Digital Input alarm type Bit 11 – Digital Input interrupt flag Bit 15:12 – reserved	Sets both the Digital Input and Digital output capabilities for the APP_INTF1 interface pin (see section 7.11).

ORW ID	Configuration	Description
26	APP_INTF2 Digital Configuration Bit 7:0 – Kinetis PORT_PCR setting Bit 8 – Digital Output level setting Bit 10:9 – Digital Input alarm type Bit 11 – Digital Input interrupt flag Bit 15:12 – reserved	Sets both the Digital Input and Digital output capabilities for the APP_INTF2 interface pin (see section 7.11).
27	APP_INTF3 Digital Configuration Bit 7:0 – Kinetis PORT_PCR setting Bit 8 – Digital Output level setting Bit 10:9 – Digital Input alarm type Bit 11 – Digital Input interrupt flag Bit 15:12 – reserved	Sets both the Digital Input and Digital output capabilities for the APP_INTF3 interface pin (see section 7.11).
28	APP_INTF4 Digital Configuration Bit 7:0 – Kinetis PORT_PCR setting Bit 8 – Digital Output level setting Bit 10:9 – Digital Input alarm type Bit 11 – Digital Input interrupt flag Bit 15:12 – reserved	Sets both the Digital Input and Digital output capabilities for the APP_INTF4 interface pin (see section 7.11).
29	APP_INTF5 Digital Configuration Bit 7:0 – Kinetis PORT_PCR setting Bit 8 – Digital Output level setting Bit 10:9 – Digital Input alarm type Bit 11 – Digital Input interrupt flag Bit 15:12 – reserved	Sets both the Digital Input and Digital output capabilities for the APP_INTF5 interface pin (see section 7.11).
30	ADC1 Alarm Type	Analog alarm configuration for the ADC1 block (APP_INTF6): 0 – Disabled 1 – High Threshold boundary exceeded alarm condition 2 – Low Threshold boundary exceeded alarm condition 3 – Out-of-Bounds alarm condition 4 – In-Bounds alarm condition.
31	ADC1 Upper Alarm Threshold Hi	The Upper Threshold High hysteresis level for all alarm condition boundary checks for any poll/read on the ADC1 block (APP_INTF6).
32	ADC1 Upper Alarm Threshold Lo	The Upper Threshold Low hysteresis level for all alarm condition boundary checks for any poll/read on the ADC1 block (APP_INTF6).
33	ADC1 Lower Alarm Threshold Hi	The Lower Threshold High hysteresis level for all alarm condition boundary checks for any poll/read on the ADC1 block (APP_INTF6).
34	ADC1 Lower Alarm Threshold Lo	The Lower Threshold Low hysteresis level for all alarm condition boundary checks for any poll/read on the ADC1 block (APP_INTF6).
35	ADC1 Sample Calculation Params Bit 7:0 – Sample Period (in Hz) Bit 11:8 - # of Subsamples Bit 15:12 – Sample iterations.	Bit-packed configuration field that configures the sampling period on the ADC1 block. The total number of samples is used in determining the artihmetic mean value of for a poll/read sequence.

ORW ID	Configuration	Description
36	ADC2 Alarm Type	Analog alarm configuration for the ADC2 block (APP_INTF5 in FUNC_MODE0): 0 – Disabled 1 – High Threshold boundary exceeded alarm condition 2 – Low Threshold boundary exceeded alarm condition 3 – Out-of-Bounds alarm condition 4 – In-Bounds alarm condition.
37	ADC2 Upper Alarm Threshold Hi	The Upper Threshold High hysteresis level for all alarm condition boundary checks for any poll/read on the ADC1 block (APP_INTF5 in FUNC_MODE0).
38	ADC2 Upper Alarm Threshold Lo	The Upper Threshold Low hysteresis level for all alarm condition boundary checks for any poll/read on the ADC1 block (APP_INTF5 in FUNC_MODE0).
39	ADC2 Lower Alarm Threshold Hi	The Lower Threshold High hysteresis level for all alarm condition boundary checks for any poll/read on the ADC1 block (APP_INTF5 in FUNC_MODE0).
40	ADC2 Lower Alarm Threshold Lo	The Lower Threshold Low hysteresis level for all alarm condition boundary checks for any poll/read on the ADC1 block (APP_INTF5 in FUNC_MODE0).
41	ADC2 Sample Calculation Params Bit 7:0 – Sample Period (in Hz) Bit 11:8 - # of Subsamples Bit 15:12 – Sample iterations.	Bit-packed configuration field that configures the sampling period on the ADC2 block. The total number of samples is used in determining the artihmetic mean value of for a poll/read sequence.
0x81	OTA Alarm Clear Test	Simulates/generates an OTA Alarm Clear record event for the following: 0 = APP_INTF1 1 = APP_INTF2 2 = APP_INTF3 3 = APP_INTF4 4 = APP_INTF5 5 = APP_INTF6 6 = APP_INTF7 7 = APP_INTF8 8 – LOW_BATTERY 9 - SECURITY 10 – LOG_NVM 11 – SLEEP_NVM 12 - RUNTIME 13 - undefined 14 - undefined 15 - undefined
0x82	Security Alarm Clear	Clear the Security Alarm state on the device (regardless of data field).
0x83	Exception Record Clear	Clear stored Exception records on the device.
0x84	Node Indicator Record Clear	Clear stored Node Indicator records on the device.
0x85	OTA Exception Record Request	Send all stored Exceptions records on the device to the backend. When the OTA exception dump is successfully acknowledged, the records will be cleared.
0x86	Relay Toggle Request Bit 7:0 – Application Interface ID Bit 15:8 – Toggle duration (in msec)	Toggle the digital output on a specified Application Interface (1 through 5 are only valid interface IDs) for a specified delay period (max of 255msec). NOTE: Will have no effect on interfaces that are not configured as DIG_OUTPUT.

PART 2: rACM Software Components

Purpose

Part 2 of this guide describes the software components that make up the On-Ramp Wireless (ORW) reference Application Communication Module (rACM). This software serves as a design template for use by third-party integrators and application developers to aid in the design and development of different sensor applications using the On-Ramp Total Reach proprietary wireless technology. The software design template is optimized for very low power usage applications and has been adapted for battery-powered systems.

Scope

Part 2 of this guide identifies each of the functional software blocks used on the reference application host processor that are responsible for managing and monitoring external sensor interfaces as well as the Node module. This document also identifies the interfaces and system resources (e.g., memory, drivers, timer blocks, etc.) required for each of the aforementioned software blocks. The default Over-the-Air (OTA) message payloads sent by the reference ACM to backend components of the On-Ramp Total Reach Network (i.e., AP, Gateway, EMS, and On-Ramp Total View) are also described.

References

1. The reference ACM (rACM) is targeted to run on the Freescale K20 microcontroller unit (MCU) built around an ARM Cortex-M4 core processor. As such, further details on this processor and the various system modules it supports can be found in the reference manual for this family of processors at: http://cache.freescale.com/files/32bit/doc/ref_manual/K20P81M100SF2RM.pdf
2. Further information on the Cortex M-4 processor can be found at: http://www.arm.com/products/processors/cortex-m/cortex-m4-processor.php
3. *Node Host Message Specification (014-0020-00)* – Provides details relating to Node Host messages.
4. *Node Application Programming Guide (010-0073-00)* **–** Provides details regarding the Node Host network interactions.

7 rACM Architecture Overview

7.1 Software Design Requirements

The reference Application Communication Module (rACM) supports external third-party integrators. The following software design requirements are considered:

1. The application processor utilized by the Reference ACM uses the Freescale Kinetis MCU—specifically, the 81-pin LQFP **MK20DX256VLK7**. rACM power management implementation features are designed around the Kinetis low-power modes.
2. The rACM application is primarily developed as a battery-powered device that is required to be operating in the lowest power-mode state as possible (this applies to the Node as well). Tentative requirements suggest that low-powered sensor applications based on the rACM design may be required to operate up to 20 years without a battery change.
3. The rACM application is required to demonstrate the ability to poll/read a simple sensor interface on the Reference Board's application I/O header. Specifically this consists of a pulse counter using the Kinetis Low-Power Timer (LPT) module. Pulse counts are reported at the uplink interval (UI). The end-user can configure the application to read/report other sensor data types (e.g., analog 0-3V samples, digital states, comparator levels, etc.).
4. The rACM pulse count data is kept as unit-less "counts." This allows for re-use for different types of sensors and/or monitoring applications (e.g., gas meters).
5. The rACM sensor reads can be scheduled for a specified interval period and specified interval offset that is synchronized with network timing.
6. The rACM stores all sensor reads in NVM before being reported OTA. The read data can also be retrieved and reported through the rACM Host UART interface.
7. The rACM is required to demonstrate asynchronous alarm generation (e.g., tamper alarm, burst detection, etc.) and report the alarm state to the backend. The magnetic Reed Switch can be configured for setting/clearing this alarm capability.
8. The rACM supports both the Linux-based GCC and windows-based IAR build tool environments.
9. The rACM application is optimized for code size in support of cost reduction efforts (i.e., smaller physical memory sizes equals lower cost).
10. The rACM application interfaces with and supports all of the On-Ramp Wireless Node management functionality provided by the HOST_CMN interfaces (e.g., Node Endpoint messaging, logging, host firmware upgrades, OTA firmware upgrades, etc.).
11. The rACM application supports a set of supporting Host UART scripts to configure the application and to provide UART logging support.
12. The rACM deliverables include a comprehensive developer's guide and application notes that includes detailed descriptions of the software interfaces and drivers.

7.2 Third Party Software

The application host on the ACM platform is implemented on the Freescale K20 MCU and makes use of the following resources provided by the vendor:

- **BSP** – The Freescale Board Support Package (BSP) consists of a hardware abstraction layer for the Kinetis hardware drivers. Further details on these code/interfaces can also be found in the MQX RTOS User's Guide. Although the MQX RTOS is not utilized on the rACM application, much of the low-level hardware initialization routines (e.g., cortex register settings, bare-metal drivers, etc.) are reused whenever possible.
- **IAR Link Libraries** – The rACM application, as with all On-Ramp Wireless application software, supports the build environment provided by the IAR Embedded Workbench IDE. As such, the application makes use of low-level library functions and link definitions required to correctly initialize program data executing out of SRAM.
- **Mentor Graphics Sourcery Tools** – The rACM application also supports the GNU tool chain on a Linux-based environment. As such, the application makes use of low-level library functions from the Sourcery Tools targeted for the Kinetis platform to support the initial "bring-up" of the application.
- **Python Programming Language** – The rACM application, as with all On-Ramp Wireless applications, requires the use of python scripts for a number of applications:
  - Pre/post build actions on the rACM application. All of the supported tool chains require Python to be installed to properly build the source code.
  - UART command and control message support for rACM application
  - UART configuration and provisioning support for the On-Ramp Wireless Node
  - UART Radio-Link diagnostic reports
  - UART real-time logging capabilities
  - UART log retrieval capabilities

For the most part, implementation details on third party software are beyond the scope of this HLD. For further details about the libraries and/or software APIs for these components, refer to the vendor's website or source materials.

7.3 Application Overview

For a high level depiction of the rACM architecture, refer to Figure 2. Reference Platform Block Diagram. This figure illustrates the two CPUs (i.e., application host and On-Ramp Wireless Node modem) and external interfaces that are under direct control by the Application Host.

7.4 Run-Time Environment

To help meet the memory and power savings requirements (some would claim constraints), the rACM application will NOT use an embedded real-time operating system (e.g., MQX, eCos, etc.). Instead, the rACM application is implemented as a simple run-time executive consisting of a

looping thread with a non-preemptive run-to-completion task model. This is identified as the Process Thread. Interrupt handling is done in the order they are processed by the cortex vector table – termed the Interrupt Thread. Nested interrupts through the Cortex Nested Vector Interrupt Controller (NVIC) are NOT supported in the rACM application in order to keep Stack memory usage under tight control. Each interrupt must also run-to-completion before another can be serviced.

The majority of the rACM application is run from the context of the Process Thread and is implemented with the following cortex processor run-time considerations:

- All process code is compiled and executed in Thumb mode in order to help reduce program size (i.e., 16-bit instructions).
- Process code runs in cortex Thread Mode.
- Process code uses the Main Stack Pointer (MSP).
- All MCU power savings states can only be configured by the process thread (i.e., Cortex light sleep and MCU deep sleep commands).
- Process code is configured to use privileged instruction access (simplifies any cortex processor interactions by software).
- Process code can be preempted by cortex NVIC interrupts.

All MCU and/or core interrupts take advantage of the Cortex Interrupt hardware vector controller, such that each interrupt results in a stack push and the program counter is loaded with an address from a vector which is used to branch to a set of ISR handlers specific to the rACM application. All ISR handlers are limited to setting/sending an event variable to the Main Processing Loop for subsequent handling in the Process Thread (implementation details during initial integration may require special handling where identified). The following run-time considerations apply to code executing in the Interrupt Thread:

- Interrupt code runs in cortex Handler Mode with privileged instruction access.
- Interrupt code uses the Main Stack Pointer (MSP).
- All Interrupts are configured for the same priority in the NVIC.
- Nested interrupts are NOT enabled or supported by the rACM application.

The above run-time settings result in efficient switching between the application background processing and the interrupt service routines. Also, and more importantly, this scheme minimizes the amount of code overhead to support this type of context switching. The limited amount of software processing required by the vast majority of sensor types targeted by the rACM application helps to support this type of process model.

7.5 Application Operational Overview

The On-Ramp Wireless Application Hosts' primary responsibilities can be broken down into the following management components:

- Determining when to power on the Node to initiate the network join process. In general, this is done when the rACM application is installed in the field.

- Support the configuration and Over-the-Air (OTA) reporting of sensor data of up to 8 separate application interfaces – 5 of the interfaces are fully configurable input/output (i/o), a dedicated interface for monitoring a 0-3Volt analog input, a dedicated interface for serial data exchange, and an "virtual" register interface for use in Modbus-type applications.
- Determining at which rate to poll and store response data from externally interfaced sensor(s), i.e., the read interval. Two separate and configurable read interval blocks are available for use on the rACM and can be assigned/mapped to any of the application interfaces.
- Determining at which rate poll data from externally interfaced sensor(s) for alarm threshold detection, i.e., the poll interval. Unlike the read interval, a polled measurement does not get stored into non-volatile memory. Three separate and configurable poll interval blocks are available for use on the rACM and can be assigned/mapped to any of the application interfaces.
- At the Node uplink interval, form and send an OTA payload packet consisting of the sensor read data. In general, uplink interval is greater than the read interval (e.g., a read interval of every 15 minutes and an uplink interval of once per day). The OTA data payload packet normally consists of chained records of read data.
- Asynchronous application events are also monitored by the application. In general, these are time-critical events that necessitate an immediate OTA payload packet sent to the On-Ramp Wireless Network in the form of an alarm record (e.g., Low Voltage Alarms, Tamper Alarms, etc.).
- When idle, the application places the host processor into the applicable low-power mode for its current operational state (i.e., light sleep vs. deep sleep).

The On-Ramp Wireless Application Host also provides the following supplementary functionality:

- NVM Factory configuration options
- NVM Application configuration options
- Real-time logging through the Host UART for on-target debug
- NVM logging of all sensor reads to support field debugging and/or backlog retrieval following an On-Ramp Wireless Network outage
- NVM logging of time-stamped ORW Node Event indicators to support field debugging
- NVM logging of application host exception event to support field debugging
- UART pass-through operations to support configuration and control of the Node
- UART and OTA application firmware upgrade capabilities

7.6 Software Functional Blocks

The following figure provides a high-level depiction of the software blocks required to manage the rACM application.

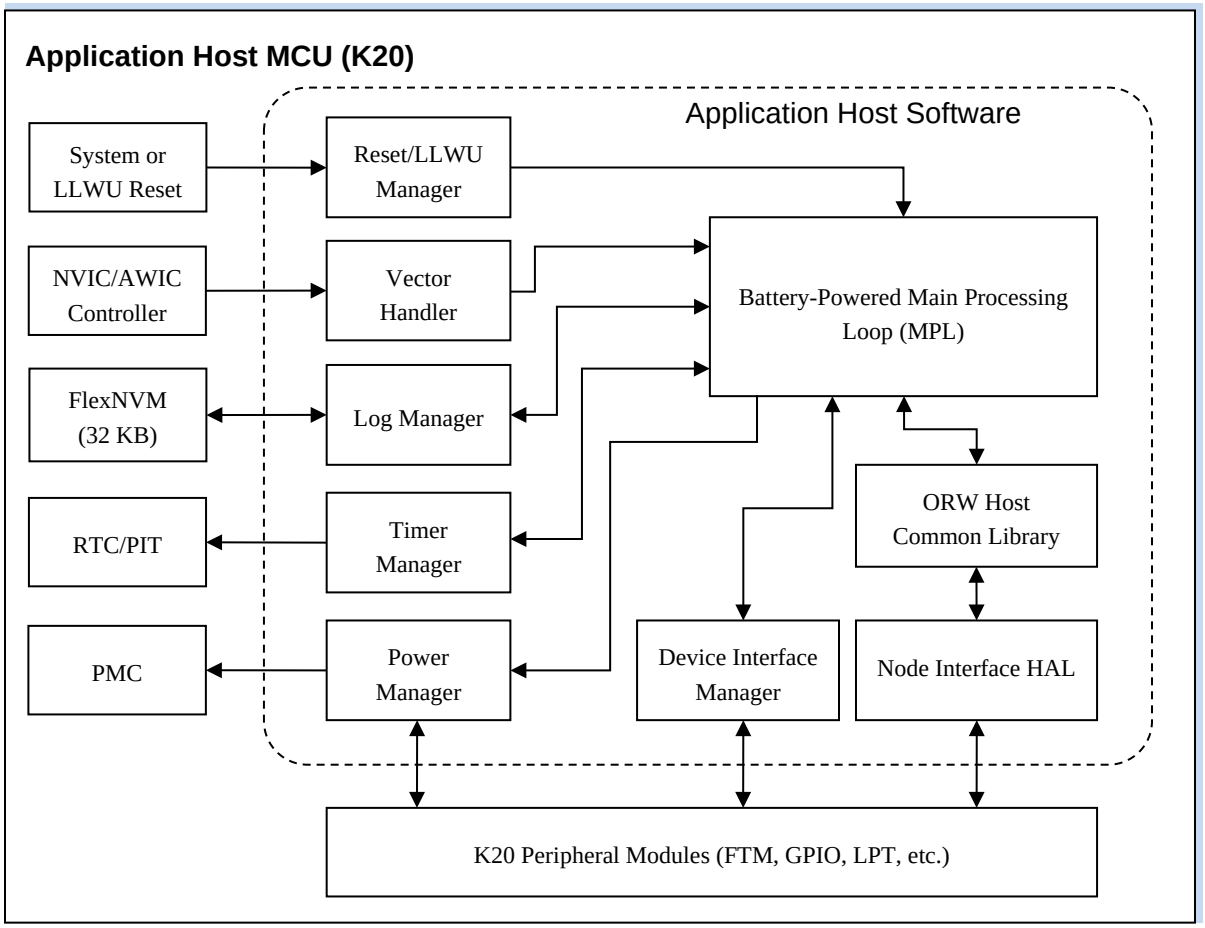


Figure 3. rACM Software Functional Blocks

The software functional blocks can be summarized as follows:

- **Reset Manager (RM)** – This is the software management routines that handle the initial entry point of the firmware when it first comes out of system reset. It is responsible for all MCU initialization as well as the determination of the type of reset (e.g., initial battery power, deep sleep interval, alarm, or unplanned assert condition). After performing reset management, it then kicks off the main processing loop.
- **Board Support Package (BSP)** – This is the set of drivers invoked by the Reset Manager following a system reset that performs all of the necessary MCU initialization required for software execution and control of the application resources. The list of driver resources managed by the BSP include the following:
  1. MCU watchdog management
  2. External 32K TCXO Initialization

3. MCU clock tree initialization
4. MCU interrupt management
5. MCU memory protection management

- **Vector Handler (VH)** – The rACM must handle all interrupt processing generated by the Cortex. This includes the installation of all associated vectors as well as any IRQ processing. Application Cortex interrupts are quickly cleared and event flags sent down to the main processing loops for subsequent handling. Cortex exception interrupts are logged to NVM and later reported as an exception event following a software reset.
- **Main Processing Loop (MPL)** – This is main processing loop for battery-powered sensors and is responsible for the following functions:

  1. Device State Management
  2. Sensor Event Handling (e.g., scheduled reads, alarms, etc.)
  3. Sleep Request Processing

  The HLD goes into much greater detail on the processing loop.

  NOTE: The processing loop is agnostic to the sensor type and is designed to be reused/adapted by other On-Ramp Wireless battery powered devices. As such, the MPL functional block *does not* interact directly with the Kinetis hardware but, instead, invokes other software functional blocks implemented as hardware abstraction layers for use by the application.

- **Over-The-Air Handlers (OTA)** – This is the source code that is responsible for parsing both the downlink and uplink message payloads and is invoked by MPL state-event handlers.
- **Factory Calibration Handlers (CAL_DATA)** – This is the source code that is used to provide a simple calibration framework for validating, reading, and writing Factory Calibration data blocks to persistent memory. Application hooks and data structures allow for multiple versions of calibration data. Application-specific firmware, derived from the rACM framework, can reuse these methods or replace them with their own proprietary methods of managing factory calibration data.
- **Host Common Interface Library (HOST_CMN)** – The Host common library is a set of source files used by the application host to manage the Node. This includes Digital SPI protocol handlers, Host Message endpoint command handlers, firmware upgrade management, etc. Developers familiar with UNIL will recognize the HOST_CMN component as a stripped-down set of core interface handlers (i.e., UNIL-lite). Other than the external API interfaces, the implementation details of the HOST_CMN library are considered beyond the scope of this document.
- **Node Interface HAL (host_cmn_hal_k20.c) –** The Hardware Abstraction Layer (HAL) used by HOST_CMN to interface with the Node. Essentially, a set of drivers to manage the serial interface with the microNode (e.g., GPIOs, SPI, etc.).
- **Host Interface (HOST_INTF)** – The application wrapper used by HOST_CMN to relay Node-to-Host events (e.g., Tx Service Data Unit [SDU] indicators, Rx SDU payloads, Node status indicators, etc.) In addition, The Host Interface module also supports all Host Specific endpoint command handlers.

- **Device Interface Manager (DEVICE_INTF) –** This is essentially a hardware abstraction layer for the external hardware interfaces on the ACM platform. The rACM application supports the following interfaces:

  1. *Low Power Pulse Counter* – Used as to demonstrate sensor reads. A waveform generator can be easily hooked to the application I/O header to drive the pulse counter without any other hardware.
  2. *Reed Switch* – Used for detecting external swipes of a magnetic provisioning tool. The Reed Switch is also used to demonstrate the asynchronous alarm generation capabilities of the rACM application (when the node has joined the On-Ramp Wireless Network).

  Additional interfaces, based on application requirements, need to be added to the Device Interface Manager (e.g., digital and/or analog inputs/outputs, serial interfaces).

- **Power Manager (PM)** – A hardware abstraction layer for managing the cortex resources for the lowest current draw possible required by the device. There are essentially three device power mode states for the application to manage:

  1. *Running state*
  2. *Light Sleep* – The core is in standby/sleep but can wake quickly as a result of core processor interrupts (e.g., ORW Node SPI transactions, etc.).
  3. *Deep Sleep* – The core/peripherals are in the lowest sleep state and can only be brought out of reset by MCU-specific low-leakage interrupts (e.g., Real Time Clock, sensor alarms, etc.).

- **Timer Manager (TM)** – The hardware abstraction layer responsible for managing callbacks for use by the application.
- **Log Manager (LM)** – The hardware abstraction layer responsible for logging and retrieving device events to NVM (e.g., sensor updates, device alarm states, etc.).

Further implementation details and requirements on each of the software functional blocks are found in chapter 1 of this document.

7.7 rACM Device States

The application host software maintains a number of state machines in its main processing loop for management of the device and associated peripherals. These are summarized as follows:

- **Device State** – The high level device state of the application (e.g., Shutdown, Deployed, OTA active, etc.). State transitions are triggered by an external user by using a magnetic provisioning tool over the device's magnetic Reed switch (or through Host interface commands received over the Host UART).
- **OTA State** – The state of the On-Ramp Wireless modem, only active after the device has been deployed. State transitions are triggered by events generated by the NODE_INTF component based on On-Ramp Wireless Network activity.
- **Host IF State** – The Host UART can be secured, powered, or placed in override mode based on the needs of the application. Both Host and On-Ramp Wireless Node configuration and debug commands exchanged using the Host UART Interface.

The following sections provide a summary of the aforementioned state machines as they exist in the reference host software. They may be expanded, modified and/or re-implemented based on the needs of an application.

7.7.1 Top Level Device State

The following figure shows the top level device state machine utilized by the reference host software. The main processing loop executing on the application host MCU initializes the various Kinetis peripherals and interface pins based on the Device State in which it is currently operating. The Device State is stored in persistent memory, is retained across all deep sleep operations, and is only reset upon battery removal (i.e., the Power-on Reset (POR) system reset).

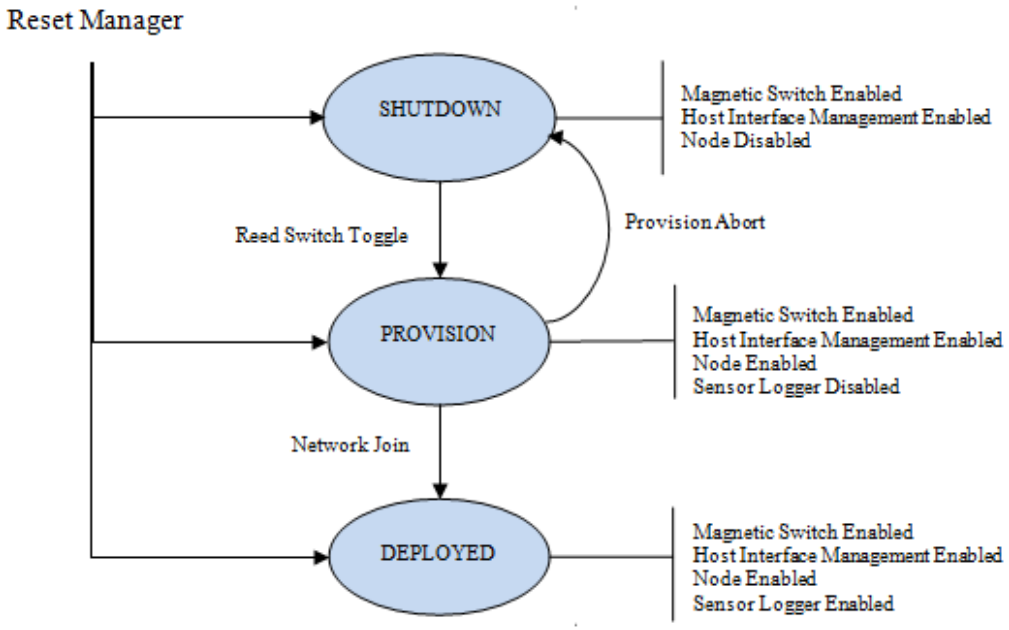


Figure 4. Top Level Device State Machine

The top level device states and operations are briefly summarized as follows:

- **SHUTDOWN** – The initial state of the reference host software whenever a battery is installed for the first time. Since the time between factory provisioning and actual on-site installation cannot be determined, the device is essentially running in its lowest power state at all times and only responds to a “swipe” of a magnetic provisioning tool (aka wand). Only the Host Interface management loop is active at this time in the main processing loop.
- **PROVISIONED** – This state is transitioned into during on-site installation of the reference host software (or the application built upon it) following a “swipe” with the magnetic provisioning tool. The field installer has a brief window of opportunity (10 seconds in the default build, i.e., the duration of the provisioning LED pattern) to abort the provisioning step by another pass of the provisioning tool. If not aborted, the microNode is released from reset and initialized, through HOST_CMN, to start the On-Ramp Wireless Total Reach system join process. **NOTE:** The Host application “PROVISIONED” State should not be confused

with node provisioning which is an entirely separate process. The “PROVISIONED” host state is simply the transition between node power-up and network join.

- **DEPLOYED** – When the node has reported that it has joined the On-Ramp Wireless Network, the reference host software gets a network time reference and all steady-state application processing can begin. Sensor logging is enabled, application interfaces are monitored, and the user data can be formatted for OTA communications with the backend during the network uplink interval.

It should be noted that the top level device state is stored in persistent memory so that the Reset Manager can quickly transition to the last known state following a system reset (either as a result of a deep sleep interrupt or unplanned exception). Should the battery be removed/replaced at any time, the device state reverts back to the Shutdown state and requires re-provisioning.

7.7.2 Host Interface State

As a battery operated (and potentially secured) device, the reference host software actively manages the external Host Interface (typically implemented using a DB9 serial connector as the following state diagram shows:

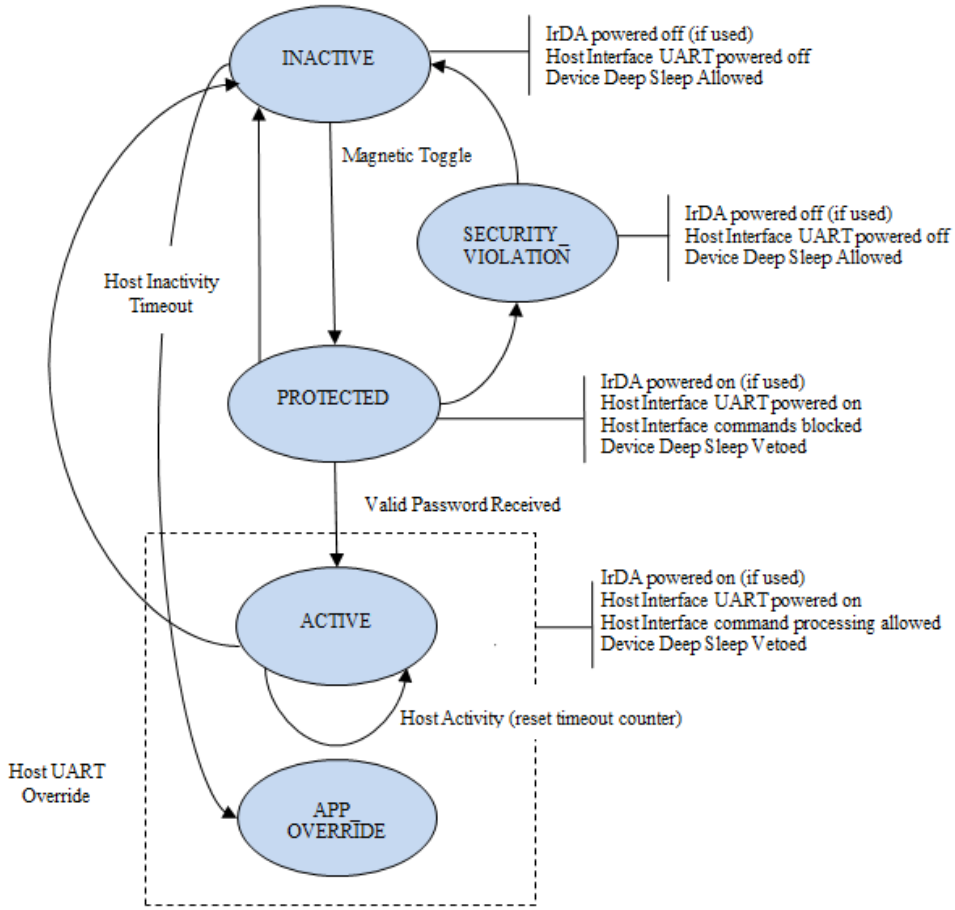


Figure 5. Host Interface State Machine

The host interface states are fairly obvious:

- **INACTIVE** – The default state of the Host Interface is to keep the host interface UART block disabled to minimize current draw. Only when the Reed Switch detects a "swipe" of the magnetic provisioning tool will it result in a transition to the Protected state.
- **SECURITY_VIOLATION** – This is essentially the security "lock-down" state of the Host UART which keeps the Host UART resources powered off despite any external attempt to activate it (i.e., the Reed Switch). It is entered into, if the configured number of invalid password attempts has been exceeded while in the Protected state. It can only be cleared through the backend by sending a configuration key-pair value to clear the security alarm. At this point, the Host UART returns to the Inactive state.

  NOTE: This Host Interface state is only used when using one of the SECURED application profiles (see section 7.9)
- **PROTECTED** – This state is entered into whenever there is a detection of a magnetic provisioning tool "swipe". The external IrDA device and Host UART are powered on and the main processing loop polls for any incoming message bytes in the UART buffer. An Inactivity timer determines when to revert back to the inactive state (currently defaulted to 10 seconds). However, the Host Interface message processing is blocked until receipt of a Host Password message (with a 64-bit password field). When a valid password has been received, the Host Interface transitions to the Active state.

  NOTES:

  1. The device is prevented from going to the lowest power savings state supported by the Kinetis at this time (VLLS) to allow buffered data/responses to be retained in SRAM during inactive periods on the UART.
  2. This Host Interface state is only used when using one of the SECURED application profiles (see section 7.9)
- **ACTIVE** – This state is entered into whenever a valid password command has been sent through the Host Interface. The external IrDA device and Host UART remain powered on and the main processing loop polls for any incoming message bytes in the UART buffer. An Inactivity timer determines when to revert back to the inactive state (currently defaulted to 10 seconds). Any host message received through the IrDA host interface results in the inactivity timer getting reset.

  NOTE: The device is prevented from going to the lowest power savings state supported by the Kinetis at this time (VLLS) to allow buffered data/responses to be retained in SRAM during inactive periods on the UART.
- **APP_OVERRIDE** – Essentially the same state as Active with regards to its operation. However, it is essentially paired with the operational state of the application (i.e., anytime the application processor is awake), and so is the Host UART. This setting is entered into based on the Host UART Override configuration option.

It should be noted that the Host Interface can be active in any of the top level device states (see previous sections).

7.7.3 Over-The-Air (OTA) State

After the reference host software has been provisioned, the application host can now bring the microNode out of reset and invoke HOST_CMN to initiate the network join process. For the most part, the OTA State largely mimics the Node State (with the exception of when user data is queued for transmission) as the following state diagram illustrates:

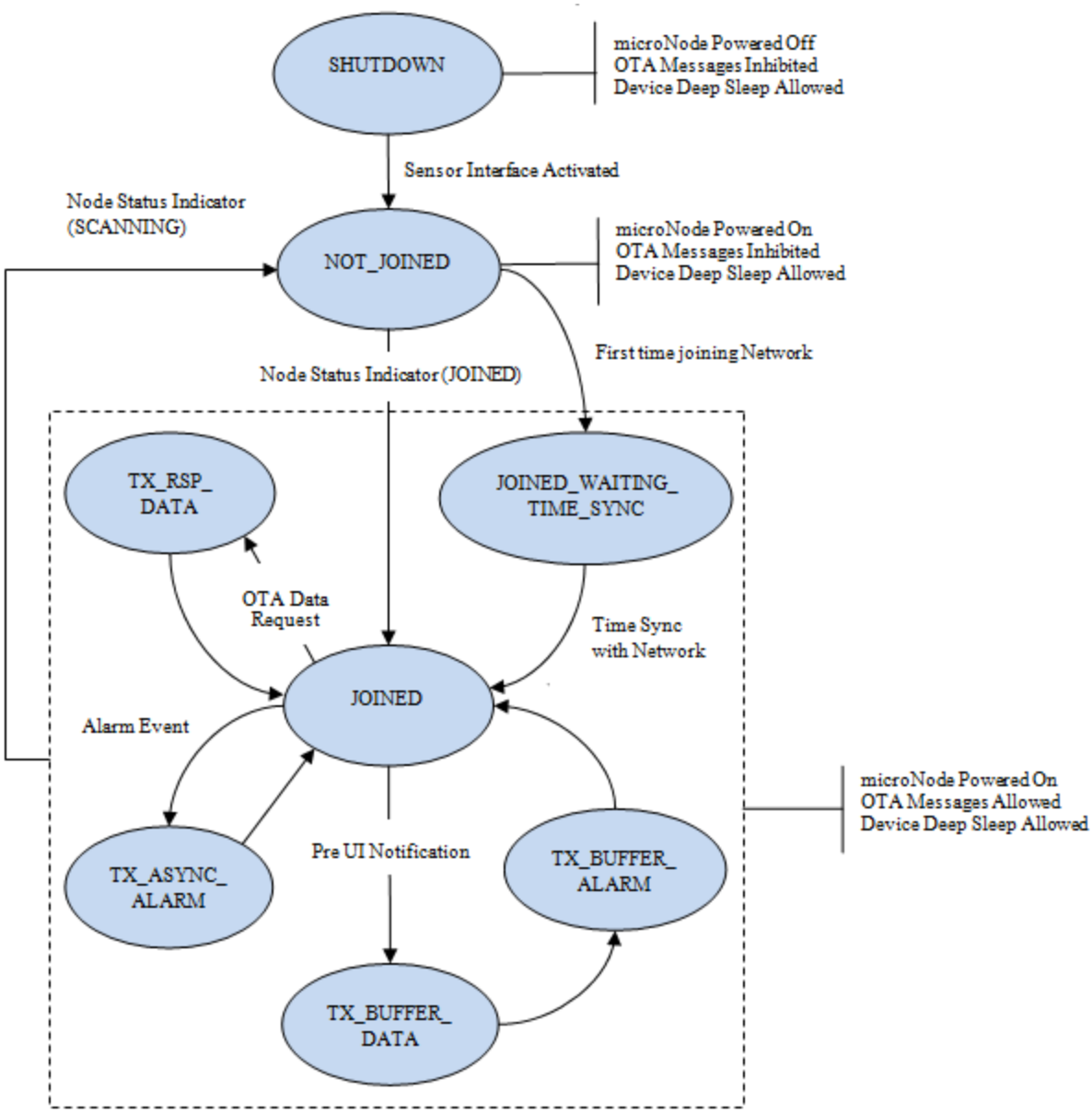


Figure 6. OTA State Machine

The Node OTA states can be summarized as follows:

- **SHUTDOWN** – The Node is kept powered off during the shutdown state until the application goes through the on-site installation procedure (i.e., the field provisioning process). NODE_INTF and all OTA message generation are inhibited at this time.

- **NOT_JOINED** – Transitioned into at any point following field provisioning but the Node is not joined with the On-Ramp Wireless Network (e.g., scanning following node initialization or as a result of frame tracking timeouts). The Node may be active at this time sweeping through its system determination algorithm. However, the Node has no link with the backend. As a result, all OTA message generation (priority and best effort) are inhibited at this time.
- **JOINED_WAITING_TIME_SYNC** – The Node has acquired and registered with the On-Ramp Wireless Network for the first time coming out of Power-On Reset (POR). Steady state operations on the application cannot begin until the first Time Sync Indication has been received from the Node (which in turn acquires a network time reference as a result of its registration with the On-Ramp Wireless Network).
- **JOINED** – The Node has acquired and registered with the On-Ramp Wireless Network and is most likely awaiting the next UI/LI interval as specified by the On-Ramp Wireless Gateway. The Main Processing Loop is allowed to generate asynchronous messages to the network (e.g., alarms) or prepare best effort exchanges with the backend (e.g., application user data) as a prelude to the next scheduled uplink interval.
- **TX_ASYNC_ALARM** – The Main Processing Loop has detected an alarm condition and requested a transfer of alarm data up to the backend through the Node interface to report the alarm type and alarm state. When the node has completed the transfer of the alarm event (whether through a network acknowledgment or retry timeout), a callback event sent to the MPL causes the OTA state to revert to JOINED. In the event the OTA transfer is unsuccessful, the alarm event is logged to memory for subsequent retransmission at the next UI opportunity.
- **TX_BUFFER_DATA** – The Main Processing Loop has requested a transfer of sensor read data up to the backend through the Node interface as a result of a Pre-UI Notification event received from the Node **OR** as a result of an uplink-bound serial data string if OTA_PASSTHROUGH is enabled (see section 7.10). When the node has completed the transfer of the user data (whether through a network acknowledgment or retry timeout), a callback event sent to the MPL causes the OTA state to transition to TX_BUFFER_ALARM (if alarm events are present in NVM) or revert to JOINED.
- **TX_BUFFER_ALARM** – The Main Processing Loop has requested a transfer of buffered alarm event data up to the backend through the Node interface as a result of a Pre-UI Notification event received from the Node. When the node has completed the transfer of the alarm data (whether through a network acknowledgment or retry timeout), a callback event sent to the MPL causes the OTA state to revert to JOINED.
- **TX_RSP_DATA** – The Main Processing Loop has processed an event which requires the generation of OTA data to the backend. This occurs either as a result of network discovery (i.e., when the device is joining the On-Ramp Wireless Network for the first time) or as a result of a solicited request from the backend (i.e., the Device Config Request OTA message). When the node has completed the transfer of the OTA response data, (whether through a network acknowledgment or retry timeout), a callback event sent to MPL causes the OTA state to revert to JOINED.

Note that while in the JOINED or active transmit states, any Node State indications that suggest a network drop (e.g., scanning) or Node assert (e.g., startup) causes the OTA state to transition

to NOT_JOINED. The system determination process in the Node restarts and alerts the host when the Node rejoins the On-Ramp Wireless Network. Pending transfers timeout and are re-sent at a later UI period.

7.8 rACM Memory Resources

The application host software has the use of following memory resources on the MK20DX256VLK7:

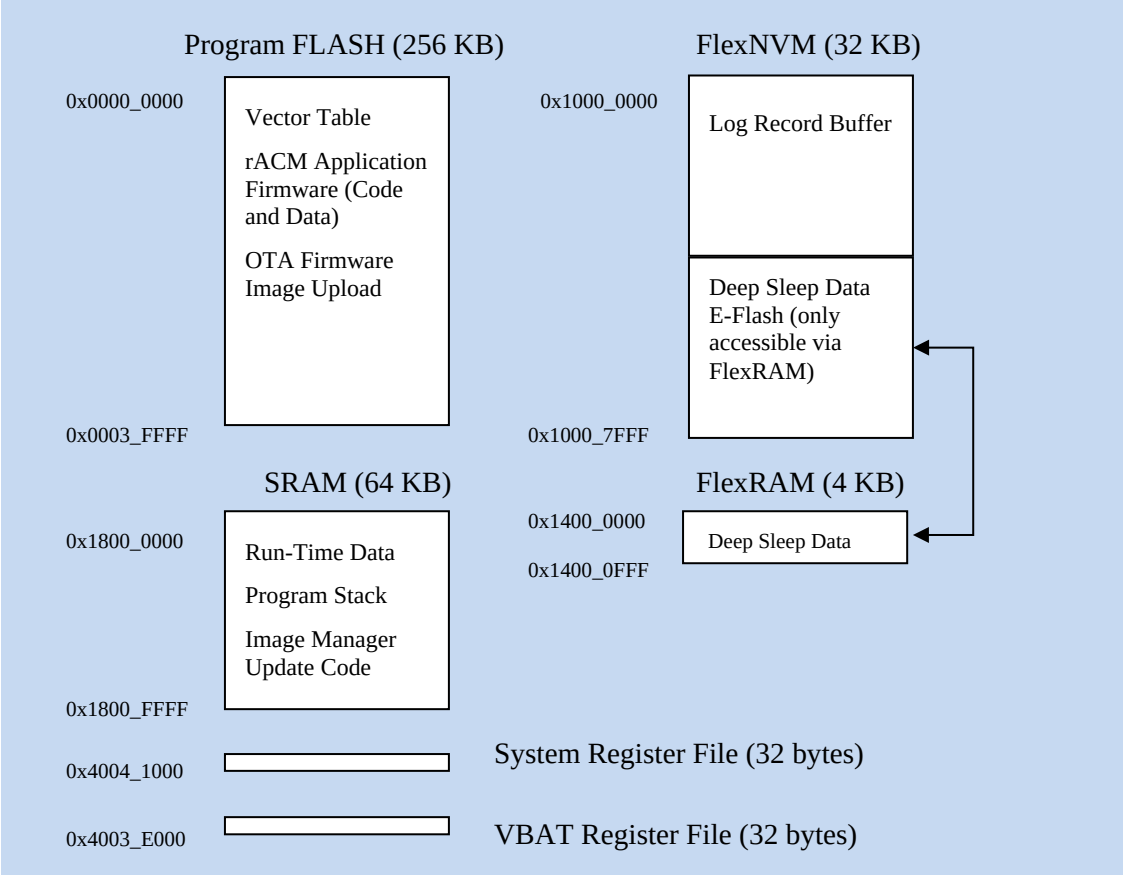


Figure 7. rACM Memory Resources

Listed to the right of each memory block is the system 32-bit address space used by the Kinetis MCU. **NOTE:** This is subject to change depending on the actual Kinetis package configuration.

7.8.1 Program FLASH

The MK20DX256VLK7 MCU package selected for the rACM platform has an internal block of 256KB program flash consisting of 2KB writable sectors.

Wait state configuration for Program Flash is done by the Reset Manager software block at startup by setting the appropriate control fields in the Kinetis Flash Memory Controller (FMC) block (see chapter 27 from reference [1]).

Program updates to flash, either from the Host Interface or Image Manager, is done with by setting the control registers in the Kinetis Flash Memory Module (FTFL) using special rACM HAL driver code running out of SRAM.

The rACM application currently links the following objects into the first half of Program Flash:

- **Vector Table** – All interrupts are vectored by the Cortex processor into the vector table linked into Program Flash starting at address 0x0000_0000. The first instruction executed by the processor following system reset is also vectored to address 0x000_0000. The first 1024 bytes of Program Flash is dedicated for the vector table which essentially consists of ARM branch instructions into program code space.
- **Program Code Space** – The ACM firmware is executed in place from Program Flash (with the exception of the Image Manager software routines which execute from SRAM during any updates to the Program Flash bank). The code start is offset into Program Flash by the size of the vector table (i.e., 1024-byte offset).
- **Program Constant Data** – Essentially treated as program code space by the IAR or GCC Linker. All constant data is accessed by the processor directly from Program Flash.
- **Program Initialized Data** – All initialized data is also stored in Program Flash and is copied by into SRAM by the Reset Manager during initialization.

All of the listed data objects are managed by the IAR Embedded Workbench as set in the linker configuration file used by the rACM Application IAR build.

The second half of Program Flash is reserved for use for any firmware uploaded from either the HOST_CMN endpoint handlers or from the HOST_CMN OTA Image Manager. The Kinetis does not permit simultaneous read and write operations from Program Flash. As a result, software accesses to the image portion of Program Flash (i.e., the Image Software NVM updates) must be executed in place from SRAM.

7.8.2 FlexNVM

The second bank of Flash in the Kinetis MCU package selected for the ACM platform consists of an internal block of 32KB Flex NVM also consisting of 2KB writable sectors.

As with Program FLASH, initial configuration is done by the Reset Manager using the Kinetis FMC module.

The rACM application uses this bank of Flash for the storage of the following data:

- **rACM Log Record Buffer** – 16KB of FlexNVM is reserved for use by the Log Manager software block for keeping track of events (e.g., reads, alarm conditions, etc.). Logged information is eventually sent to the backend during scheduled UI periods. The Log Record Buffer portion of FlexNVM is configured as traditional Data Flash.
- **Deep Sleep Persistent Data** – The second 16KB of FlexNVM is reserved for use by the Sleep Manager to save any data that needs to be persistent across periods of deep sleep (e.g., timer queue entries, HOST_CMN state data, etc.). This block of FlexNVM is configured using the Kinetis Enhanced EEPROM feature (i.e., E-Flash) and cannot be directly accessed by the core processor. All accesses to and from E-Flash are done through the 4KB FlexRAM block which performs automatic wear-leveling to improve the endurance cycle of FlexNVM. Deep Sleep Persistent Data can further be broken down into the following blocks:

- **Field Configuration Data** – All of the configuration key-value pairs used to update the rACM operational parameters (see section 6, Table 3). Field configuration data can be set both through the Host serial interface or from the a downlink message sent by the backend (i.e., the Config Write Request).

  NOTE: Field configuration data will revert to software defaults upon reception of the host serial RESET_DEVICE command.

- **Factory Configuration Data** – Configuration data that is set as part of the factory provisioning process (e.g., Device Identifier, Node Type, analog calibration parameters – see section 7.12 for the details on the default factory configuration data used on the rACM firmware).

  NOTE: Factory configuration data will be *preserved* upon reception of the host serial RESET_DEVICE command.

- **Run Time Data –** Tracking variables used by the rACM main processing loop and/or device interface manager that are critical for steady state operations (e.g., State variables, timer callbacks, log manager pointers). These variables allow the rACM to quickly recover from a system reset – either planned (e.g., deep sleep LLWU) or unplanned (e.g., watchdog reset).

Simultaneous read accesses to Program FLASH and read/writes to FlexNVM are permitted by the Kinetis making code execution out and NVM data management transparent to the rACM application.

7.8.3 SRAM

The K20 MCU package selected for the ACM device utilizes an internal 64KB block of SRAM. A portion of this SRAM block can also be clocked by the Kinetis during Deep Sleep keeping some of the program data persistent (at the cost of some current draw). SRAM is currently allocated for the following ACM application data:

- **Non-Persistent Data** – Essentially the firmware data objects (initialized, .bss, and non-initialized) are used by the application firmware. The exception is constant read-only data which is accessed directly from Program Flash.
- **Image Manager Code** – The HOST_CMN Image Manager, which performs write operations to Program Flash, executes from SRAM. This gets around the operation restriction on the Kinetis MCU prohibiting simultaneous read/write operations.
- **Stack Memory** – The ACM application uses a single block memory stack for use by the two cortex processor modes (main/ MSP and process/PSP). Because nested interrupts are NOT supported on the application, the stack is relatively small (initial estimate of 500 bytes).

  NOTE: Further details on the cortex stack registers and modes can be found in reference [2].

All of the listed data objects are managed by the IAR Embedded Workbench in the linker configuration file for the ACM Application firmware build. The Stack Memory pointers are initialized in the appropriate boot vector at startup.

It is important to note that data objects located in SRAM are not preserved across periods of Deep Sleep on the Kinetis MCU. The system vector handler, which occurs as a result of LLWU interrupt, causes the SRAM to be re-initialized during the ACM boot sequence.

7.8.4 FlexRAM

The K20 MCU package selected for the ACM device also includes an internal block of 4K FlexRAM memory which can be configured in one of two ways:

- Traditional RAM
- EEPROM emulation scheme (EEE)

The FlexRAM is configured for EEE mode and be used for the update and retrieval of any data objects that must be preserved during periods of Deep Sleep (e.g., timer callback queues, HOST_CMN states, etc.). To increase the number of endurance cycles of the 32KB E-Flash reserved for Deep Sleep persistent data, FlexRAM is configured to only allow 512B of EEE data. The Power Manager software block is responsible for the management of all persistent data stored into E-Flash (see section 8.7). Refer to the Kinetis reference manual for further details on Enhanced EEPROM functionality (see reference [1]).

7.8.5 MCU Register Files

The Kinetis processor family, including the K20, also includes two, 32-byte register files that are accessed by the internal peripheral bridge:

- **System Register File** – 32 bytes of persistent memory that is powered on in all MCU power modes and is only cleared as a result of a power-on reset (POR).
- **VBAT Register File** – 32 bytes of persistent memory that is powered on in all MCU power modes and is only cleared as result of battery removal.

The rACM application currently keeps track of the following Main Processer Loop variables in these two register files:

- **State Information** – The current state of each of the ACM device state machines (i.e., Top Level Device, Host Interface, and OTA states. See section 7.5).
- **Software Reset Exception Data** – Should the application software receive a cortex fault that prevents software program execution, the application vector handlers log the exception type into the system register files and recover the MCU by issuing a Software System Reset. The reset handlers can use this data for logging exception event into NVM for subsequent post-processing and debug efforts.
- **Latest Interface Read Data** – An NVM copy of the latest application interface data which can be recovered in the event of an exception reset of Log NVM update failure.
- **Timer IDs** – Various callbacks/timeouts used by the application are managed in the system file registers and are used in coordination with the Timer Manager (see section 8.8).

All Register File data elements are defined within the scope of the Device Manager. In addition, they are accessed using a set of macros. Unlike other Non-Volatile memory resources, these file registers can be accessed without restriction (i.e., similar to random access memory).

7.9 Application Profile

The rACM has a top-level configuration setting (see Table 16) that can be used to define the application capabilities. The application profile is used to gate certain transitions between the device state machines – for example, an unsecured application profile will prevent transitions to the PROTECTED host interface state.

The following application profiles are defined for the rACM device:

- **FACTORY_MODE** – Device is unsecured, continuously powered, and the Host Interface is turned on by default. In addition, the Host Communication Framework will be left in Passthrough mode such that all Node Event/Messages will bypass the application specific endpoint and get routed to the PC Host. As the name suggests, this application profile is intended to support ATE/DVT procedures. The rACM will default to this state following the very first Power-On-Reset (POR) following factory flash.
- **UNSECURED_POWERED** – The device is unsecured, continuously powered, and the Host Interface is turned on by default. Unlike factory mode, both the PC Host and Application Specific endpoints receive Node Event/Messages. The application will prevent the rACM from entering low power modes – however, light sleep is still allowed. Because the device is unsecured, the logger utility and/or node control scripts will function without password protection.
- **UNSECURED_SLEEPY** – The device is unsecured but, during application idle periods, will attempt to enter low power modes (aka deep sleep). Launching any PC Host test commands (e.g., the application logger, Node Monitor, etc.), will require a swipe of the magnetic provisioning tool to wake the application host from sleep (for as long as the Node Interface Inactivity Timer remains active). Because, the device is unsecured, the host interface will still function (when awake) without password protection.
- **SECURED_POWERED** – Although continuously powered, the Host Interface will only function once a valid passcode is entered via the Host Application Control scripts (see the rACM quick start guide for a description of the python-based control scripts).
- **SECURED_SLEEPY** – The device is both secured and operating, when applicable) in a low power mode of operation. It takes both a swipe of the magnetic provisioning tool to wake the application host from sleep and a valid passcode command before the application host control scripts will be able to function.

Any custom software derived from the rACM baseline software will need to take into consideration application requirements via ease of access when choosing an application profile.

7.10 Application Serial Interface Template

The rACM includes an application serial template interface module that is intended to aid third-party integrators to quickly bring up a UART serial interface for use in communicating with external sensors or co-processor. In addition, the rACM can be configured to use an asynchronous data model in which the network node serves as a “pipe” for low-rate serial data streams between the backend and the application UART interface.

The following components make up this template application:

- **Application Serial Interface APIs** – Implemented in app/host_app_serial_intf.c, this file includes all of the APIs necessary to initialize, enable, disable, and process Rx/Tx data bytes over the Application UART. Each API is heavily commented and includes user application notes to aid in tailoring the software for third-party custom interfaces.

  NOTE: The default rACM Application Serial Interface Template is implemented out of the box to interface with an external terminal utility (e.g., minicom, putty, etc.). This is primarily for demonstration purposes only – it is expected that the application serial interface will be modified/tailored by third party integrators for a real sensor device or co-processor.

- **OTA Serial Pass-through Message Format** – Serial data streams can be sent intact in both the downlink and uplink direction to the networked host application via an existing rACM message format (see section 9.2.8). This message format can be leveraged for similar Over-The-Air (OTA) serial transfer schemes.

- **MPL Event Processing for OTA Pass-through** – If configured, MPL has all of the event processing necessary to support the "piping" of serial data in both the uplink and downlink directions – this is done with the TX_SERIAL_PASSTHROUGH event handler for serial data transfers in the uplink direction as well as an Rx SDU handler for the OTA Serial Passthrough packet in the downlink direction.

- **Host Application Configuration Commands** – The Host Application control script (see section 5.1) contains two commands that manages the Application Serial Interface:

  - **SET_APP_SERIAL_CTRL –** Used for dynamic run-time configuration of the Application Serial Interface.

  - **SET_POR_SERIAL_CTRL –** Used for setting the persistent Power-on-reset (POR) configuration of the Application Serial Interface.

The Application Serial Interface Template supports three modes of operation:

1. **NONE –** The Application Serial Interface is unused and is powered off.
2. **LOCAL_ECHO** – This is a local loop-back mode where an incoming character stream from the Application UART is logged and then transmitted back out through the same UART. In the default rACM baseline, a character string is determined once a carriage return (i.e., ascii 0x0D) is detected. This is primarily intended for basic interface unit-testing/validation of the application serial interface (i.e., baud rate configuration, UART character framing, polarity, etc.).
3. **OTA_PASSTHROUGH** – Not only does this mode enable the Application Serial Interface, but it will also re-configure the rACMs data model from a synchronous behavior (i.e., Data payloads generated by the Pre-UI Notification) to an asynchronous behavior that is triggered by incoming data streams from either the uplink or downlink direction. In the default rACM baseline, an incoming character stream from the Application Serial Interface is determined once a carriage return (i.e., ascii 0x0D) is detected. The transmission of uplink-bound character string will be gated by the OTA State Machine (see Figure 6) with the following caveats:

   - If the OTA state is NOT_JOINED, the uplink bound character string is discarded/ignored.

   - If the OTA state is in one of the active transmit states (e.g., TX_ASYNC_ALARM, TX_RSP_DATA, etc.), then the data string request will be buffered and the string will be

transmitted at the once the OTA state transitions back to JOINED. Multiple string requests will be ignored until then.

NOTE: When the OTA_PASSTHROUGH state is enabled, the PRE_UI_INDICATOR event will be ignored by MPL.

4. **OTA_BUFFERED** – This application serial interface mode performs all of the same features and capabilities as the OTA_PASSHTHROUGH mode along with the following additional enhancements:

   - All serial stream packets received from the Application Serial Interface are stored in a circular buffer in internal memory. The default size for this circular buffer is 4Kbytes.
   - Serial stream packets stored the application circular buffer are delivered, in the order they are received, to the backend using OTA Passthrough uplink packets.
   - Unlike OTA_PASSTHROUGH mode, serial stream packets are only freed/released from the internal circular buffer when the OTA Passthrough uplink packets are acknowledged by the On-Ramp Wireless network Access Point (i.e., reliable delivery).
   - The rACM Main Processor Loop (MPL) will attempt to continuously transmit OTA Passthrough uplink packets as long as the application circular buffer has undelivered/unacknowledged stream packet data.

   The OTA_BUFFERED mode is intended for POWERED application types only as the circular buffer used for this implementation is located in internal SRAM (i.e., the buffered data is in non-persistent memory).

   NOTE: When the OTA_BUFFERED state is enabled, the PRE_UI_INDICATOR event will be ignored by MPL.

   An example sequence diagram of events for the OTA Passthrough processing on rACM can be found in the Appendix C in section C.8.

7.11 Application Interface Configuration Framework

One of requirements of the Reference Applications' firmware implementation is to provide a configuration framework for managing a to-be-determined number and type of application interfaces. As currently implemented, the unmodified rACM firmware supports the following interfaces:

- Five separate and fully configurable application header interfaces (APP_INTF1 – APP_INTF5) that takes advantage of the Kinetis port muxing capabilities to leverage different functional blocks (see Table 4).
- A dedicated interface on the application header used for measuring a 0-3Volt analog input.
- A dedicated application UART interface (see previous section) used for communicating with serial-based peer application coprocessors (e.g., GPS solution, DNP3 solution, etc.).
- A virtual interface consisting of a 32-bit application-specific data stored in the Kinetis File System Register (e.g., a modbus register value).

The following illustration shows a high level composite structure diagram of the configurable application interfaces supported on the rACM platform:

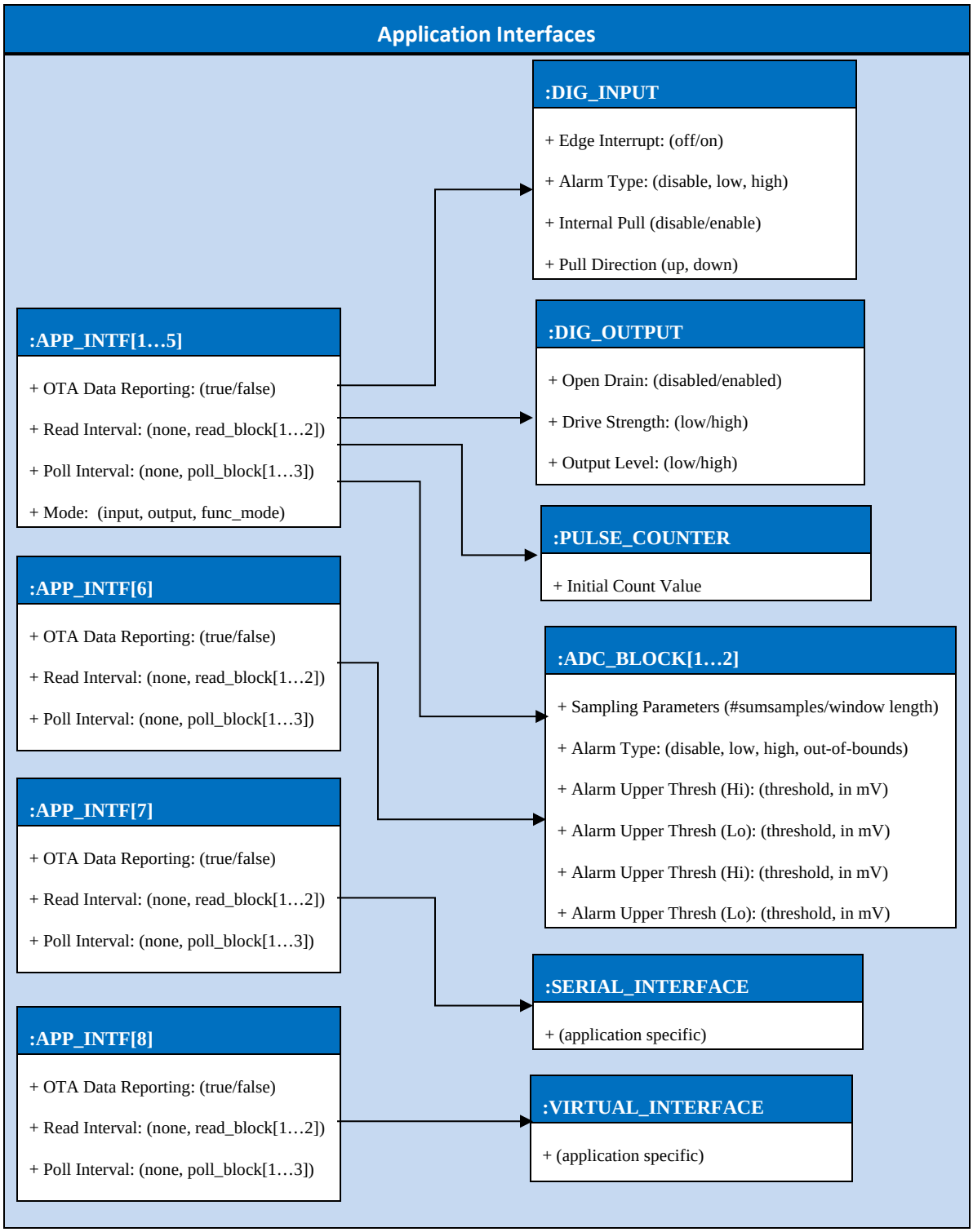


Figure 8. rACM Application Interface Structure

Referring to Figure 8, each application interface can be look on as a type of metaclass which all share the following methods.

- OTA Data Reporting at the Pre-UI Event – either instantaneous reads or buffered reads from memory.
- OTA Asynchronous Alarm Reporting – alarm definition/threshold checks defined based on interface type (e.g., high/low polarity for digital inputs, analog Hi/Lo thresholds for ADC inputs, etc.).
- Read Interval mapping which allows for interface measurements to be scheduled and stored to memory at a defined interval rate and/or time-of-day.
- Poll Interval mapping which allows for interface measurements to be scheduled at defined interval rate and/or time-of-day for alarm threshold checks.

Each application interface is then mapped to a stereotype class which further defines its run-time behavior. The Device Interface Manager is responsible for managing these interface classes through the use of initialization, read, and polling methods. The latest rACM firmware release supports the following interface types:

- **Digital Input Interface**: This interface type leverages the Kinetis Port/GPIO resources as described in section 8.6.4. This includes interrupt management and alarm handling for changes in the Boolean input state.
- **Digital Output Interface**: This interface type leverages the Kinetis Port/GPIO resources as described in section 8.6.5. Output levels are held to a static state, even while the application is transitioning in/out of deep sleep (for SLEEPY application profiles).
- **Pulse Counter Interface**: This interface type leverages the Kinetis Low-Power Timer block resource as described in section 8.6.1. The pulse counter is fully functional even or SLEEPY application profiles.
- **Analog-to-Digital (ADC) Input Interface:** This interface type leverages the Kinetis ADC resources as described in section 8.6.6. This interface also supports alarm threshold checks for each read/poll method.
- **Serial Interface:** This is an interface type reserved for application-specific devices that utilizes a UART-based serial interface (e.g., GPS co-processor, a sensor that leverages a DNP3-style serial protocol, etc.). It is the responsibility of the external integrator to define the serial protocol and OTA data format that is applicable to the remote sensor the rACM-based application is interfaced with. The On-Ramp Wireless application engineering team is available to provide technical advice and/or reference examples on how this type of interface can be implemented within the rACM framework.
- **Virtual Interface:** Similar to the serial interface, the virtual interface is also reserved for application-specific devices that could be used to map/update sensor data to a reserved memory location in persistent memory which could be accessed by the backend in a SCADA-type implementation. On the rACM baseline, any OTA reporting of the virtual interface would result in a data packet containing a reserved 32-bit location in the File System registers.

7.11.1 Multi-Purpose Application I/Os

As discussed in the section overview, the first five application interfaces supported by the rACM platform (APP_INTF1 – APP_INTF5) are multi-purpose I/Os and support an additional method/configuration option used to map the port to one of 4 possible interface modes:

- **DIG_INPUT** – Digital Input Interface. As mentioned in the section overview, deep sleep interrupt support for digital inputs are limited to selected interfaces (see Table 4).
- **DIG_OUTPUT** – Digital output interface.
- **FUNC_MODE0** – Kinetis port mapping to one of the internal Kinetis functional blocks (see Table 4).
- **FUNC_MODE1** – Kinetis port mapping to an alternate internal Kinetis functional blocks (see Table 4).

Functional mapping assignment is done through a dedicated configuration key-value pair (ORW ID#18 from Table 3).

The following table provides more detailed information on the interfaces and includes: Kinetis port/pin assignment, deep sleep interrupt capability, functional mode assignment, and the default configuration in software after initial factory flash:

Table 4. Fully Configurable Application I/O Settings

Interface	Kinetis Port(s)	Deep Sleep DIG_INPUT Int Capability?	FUNC_MODE0	FUNC_MODE1	Default Mode
APP_INTF1	PTA19	NO	LPT_BLOCK	*(reserved)*	FUNC_MODE0
APP_INTF2	PTC3	YES	*(reserved)*	*(reserved)*	DIG_INPUT
APP_INTF3	PTC5	YES	*(reserved)*	*(reserved)*	DIG_INPUT
APP_INTF4	PTC6	YES	*(reserved)*	*(reserved)*	DIG_INPUT
APP_INTF5	PTC0	NO	ADC_BLOCK2	*(reserved)*	DIG_INPUT

7.11.2 Fixed Application I/O Settings

For completeness, the following table provides information on the remaining application interfaces that are fixed to functional block/interface class (and cannot be re-assigned):

Table 5. Fixed Application I/O Settings

Interface	Kinetis Port(s)	Deep Sleep DIG_INPUT Int Capability?	FUNC_MODE0	FUNC_MODE1	Default Mode
APP_INTF6	ADC0_DP3	N/A	N/A	N/A	ADC_BLOCK1
APP_INTF7	PTB10/ PTB11	N/A	N/A	N/A	SERIAL_INTF
APP_INTF8	N/A*	N/A	N/A	N/A	VIRTUAL_INTF

7.11.3 Application Interface OTA Data Reporting

The Interface class that an application I/O is mapped to is responsible for determining the type of measurement unit and payload data size that is necessary to convey any read information to the backend via the Over-The-Air (OTA) Sensor Data History Message (see section 9.2.7):

- **Digital Input Interface**: OTA Data for an interface configured as a digital input will store and report a 1-byte Boolean state indicating the input level of the selected interface (sensor identifier of 0x090).
- **Digital Output Interface**: OTA Data for an interface configured as a digital output will store and report a 1-byte Boolean state indicating the output level of the selected interface (sensor type identifier of 0x090).
- **Pulse Counter Interface**: OTA Data for an interface configured to monitor a pulse counter input will store and report a 32-bit absolute count value (sensor type identifier of 0x012)
- **Analog-to-Digital (ADC) Input Interface:** OTA Data for an interface configured to monitor an analog-to-digital input will convert, store, and report a 16-bit signed value in DC milli-volts representing the 0-3V voltage input range (sensor type identifier of 0x035).
- **Serial Interface:** This interface is reserved for an application-specific implementation. OTA Data reporting for a serial interface is ignored in the rACM baseline.
- **Virtual Interface:** OTA Data for a virtual interface is configured to report an undefined, application-specific 32-bit value reserved in the Kinetis File System Register (sensor type identifier of 0xF03).

7.12 Application Factory Data Framework

The Reference Application also includes a software framework for managing a block of data in the Kinetis MCU for use as Factory Configuration/Calibration Data. The code is contained in a single file object, *host_app_cal_data.o*, with all public methods and data structures prefixed with '*CAL_DATA_*'. The methods provided by the rACM firmware manage a small block of memory (max size of 128bytes) physically located in FlexNVM (see section 7.8.2 for an overview of FlexNVM).

NOTE: Applications derived from the rACM firmware can choose to build upon the provided methods/scripts or recoded in its entirety. For complex applications, the 128-byte size limit in FlexNVM may be too prohibitive – in this case, sectors in internal Program or Data Flash can be borrowed for use in storing factory configuration/calibration data.

The Factory Data Framework includes support for the following software methods:

- **Data Read Pointer**: The Factory Data Framework provides a global accessor pointer to the Factory Data in NVM for use by the application in accessing the individual data elements.
- **Data Update/Write API**: A software method used for storing an updated copy of the factory data block to NVM.
- **Power-On Reset Verify API**: A software method used for verifying a factory data block in NVM following a Power-On Reset (POR) event. If a checksum integrity check on the data

block fails, the software will initialize NVM with a default copy. If the version element is stale/obsolete, a method for upgrading to a current version of the data block is provided.

- **Set Default API**: A software method used for updating NVM with software defaults.
- **Host PC Python Support**: The *host_app_ctrl.py* python script includes methods for reading updated factory NVM data (GET_FACTORY_DATA and SET_FACTORY_DATA).
- **Factory Data File Ingestor**: Similar to the Node firmware methods, the rACM provides a Host PC method that reads a human-readable text file for defining the factory data configuration values and then leverages the Host PC python scripts for programming the values on-target.

The following table(s) list the Factory Data configuration block elements, based on configuration version, currently supported by the rACM software:

Table 6. rACM Factory Configuration/Calibration Data

CAL_DATA_FlashConfig_v1_t	
version	Factory data block version. Must be set to '1' for this version of the flash configuration block.
nodeType	The node type currently populated on the rACM platform (as defined by orw_msg_nodeType_t). Valid values are: 1 – UNODE 2 – DNODE
deviceId	A 32-bit device identifier (separate from nodeId)
metaDataLength	The OTR commsys_2.x network supports up to 12 "Meta" data bytes that are sent in-band from the node as part of the network join process – it use is currently undefined/unsupported by the rACM firmware. This field (if used) is the number of "Meta" data bytes to transfer to the backend.
metaDataBytes[0…11]	This is the array of "Meta" data bytes (12 maximum).
adcScaleFactor[0…1]	Used by the two ADC inputs on the rACM, this is the fixed-point scale factor used for converting a 16-bit A/D count to an extrapolated measurement unit (see section 8.6.6 for more details on the ADC block implementation).
adcResultShift[0…1]	Used by the two ADC inputs on the rACM, this is the fixed-point result shift used for converting a 16-bit A/D count to an extrapolated measurement unit (see section 8.6.6 for more details on the ADC block implementation).
adcOffset[0…1]	Used by the two ADC inputs on the rACM, this is used as the offset adjustment for any extrapolated measurement (see section 8.6.6 for more details on the ADC block implementation).
adcUnitType[0…1]	This is the unit of measurement for any extrapolated measurement on the two ADC inputs on the rACM. The unit type field is defined by the same enum used in OTA Data Reports (i.e., *OTA_InterfaceTypeUnits_t* as defined in section 9.2.7, Table 23).

8 rACM Functional Blocks

8.1 Reset Manager (RM)

The Reset Manager is essentially the entry point of the rACM Application whenever a system reset is received by the Kinetis processor core. Currently, the RM software block identifies and processes the following types of system resets:

- **Power-On-Reset (POR)** – Expected when battery power is provided for the first time.
- **Low-Voltage Detect (LVD) Reset** – Expected reset when the battery input voltage dips below a configurable threshold.
- **Low-Leakage Wakeup (LLWU) Reset** – Expected reset when the MCU comes out of one of the deep sleep modes either due to a RTC counter match or external GPIO pin configured for LLWU detection. This is considered a normal operational feature of the ACM application.
- **Watchdog Reset** – Used to safeguard against the Main Processing Loop software block getting stuck in a prolonged or infinite loop.
- **Exception Reset** – Basically the remaining Kinetis system resets that are only generated as a result of a processing exception (e.g., software, hardware, loss-of-clock, etc.).

The System Reset is passed in as the first entry in the Main Processing Loop (MPL) event queue for subsequent processing by the ACM state machines (see section 8.3).

In addition to identify the system reset source, the Reset Manager is responsible for all of the initial configurations required for code execution – this includes the cortex core register settings, memory wait states, clock domains, as well as any run-time processor power configurations that do not persist across system resets. In addition to hardware, the RM is also responsible for setting up the run-time software environment (e.g., SRAM data initialization, Stack initialization for the Process and Exception threads, the setting of privileged instruction access, etc.).

After all Kinetis system resources have been initialized, the Reset Manager unconditionally branches to the Main Processing Loop (MPL). The MPL is then responsible for bringing up additional Kinetis resources and/or peripherals as required through the Device Interface Manager.

All public functions in the Reset Manager contain the *RM* prefix.

8.2 Vector Handler (VH)

The Vector Handler functional block is essentially all of the software that is executed in the context of the interrupt thread. This consists of the following:

- The Cortex Vector table, starting at address 0x0000_0000 in Program Flash containing the set of vector branch instructions
- All Interrupt handlers specific to the ACM application

- Default interrupt handlers for the remaining set of vectors that are not expected in the ACM application

rACM application interrupt handlers are limited to inserting an event entry to the MPL Event Queue (see the next section) and performing all of the necessary steps to clear the source of the MCU interrupt and quickly reverting context to the Process Thread.

All interrupt handlers in the Vector Handler contain the *VM* prefix.

8.3 Main Processing Loop (MPL)

This is the software functional block that is essentially the heart of the rACM Application and is responsible for managing all of the events and interfaces associated with sensor functionality. It is implemented as a hierarchical state machine engine implementation using an event queue as the input that drives the state machine engines. All events are processed in the order they are received/generated by the rACM. When all queued events have been processed to completion, the MPL can then invoke the Power Manager to place the Kinetis MCU into the appropriate sleep state (light or deep sleep).

This following pseudo code helps illustrate this process:

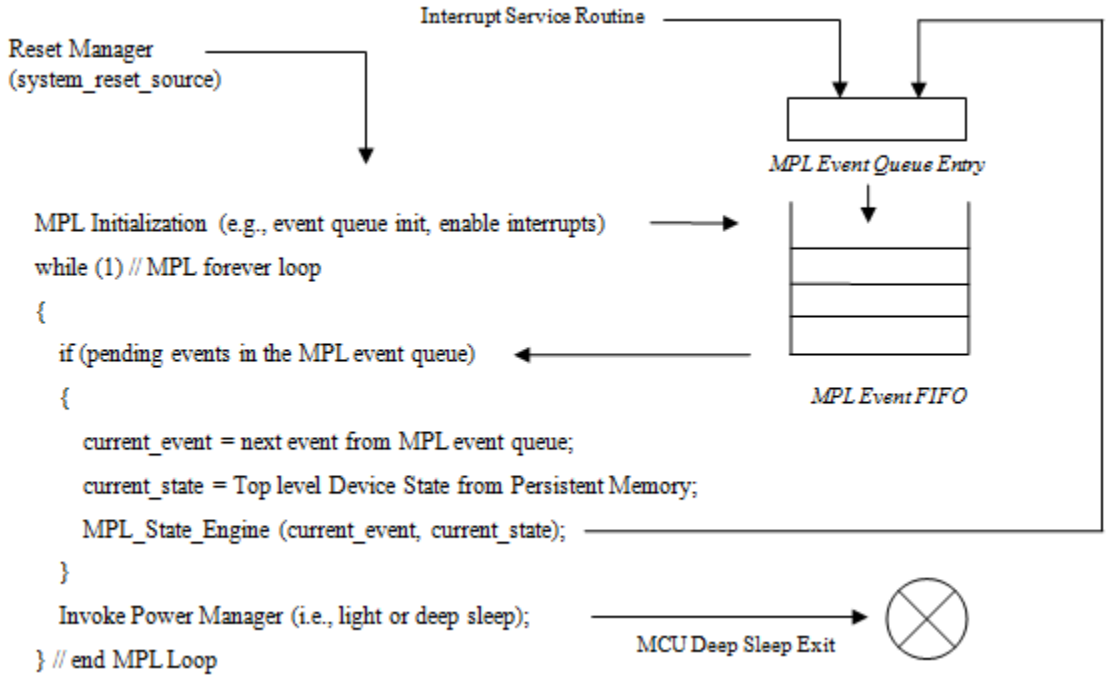


Figure 9. Main Processing Loop

The Reset Manager (see section 8.1) is the initial entry point of the rACM Application. After configuring the processor resources, it branches into MPL passing into it a parameter which identifies the system reset source. This is used as the first entry within the MPL Event Queue. Additional Events are generated and inserted in the queue either in response to an asynchronous rACM or as an action from the MPL State Engine processing.

After the Main Processing Loop is first branched into from the Reset Manager, the following resources need to be initialized:

- The MPL Event Queue (a software FIFO) is created and initialized with an entry for the RM System Reset Source. If necessary, events from persistent memory can also be restored and loaded into the MPL Event Queue. The current design does not anticipate this, even for software exceptions.
- The Kinetis Watchdog Timer is configured and enabled (or optionally disabled based on the configuration parameters).
- System Interrupts are enabled allowing the Interrupt Thread to run.

Following MPL Initialization, the MPL process loop is entered into and continuously performs the following steps:

- Kick the Hardware watchdog.
- Flush the current MPL entry from the MPL event queue.
- Run the MPL State Engine with the current event and current top level Device State (see section 7.7.1). The handlers within the MPL State Engine interface with the rACM peripherals as needed based on the current state and event. The top level state machine also kicks off the host interface and OTA state engines as required.
- When the MPL event queue is empty, the Power Manager is invoked to determine how to manage the rACM. In most cases, the Power Manager places the Kinetis into Deep Sleep with the next timer event programmed into the Kinetis RTC to wake the processor back up via an LLWU System Reset. If Deep Sleep has been vetoed by the MPL State Engine handlers (e.g., pending transactions within the HOST_CMN interface), then light sleep is enabled preserving the data in SRAM until the core receives an interrupt (at which point the interrupt thread inserts an event in the MPL event queue). Further details on the Power Manager can be found in section 8.7).

Detailed state machine transitions in the MPL State Engine are currently beyond the scope of the High Level Design and are subject to change as implementation details work themselves out. This type of detail is presented in an application developer guide; however, the following table lists the expected application events that are processed by MPL.

Table 7. rACM Application Events Processed by MPL

System Reset Events	
POR_Sys_Reset	Power-on Reset – expected during factory calibration and/or installation of battery for first time. MPL enables Reed Switch for LLWU and places the system into Deep Sleep.
LVD_Sys_Reset	Low-Voltage-Detect Reset – exception condition for failing power supply and/or battery. MPL logs the event and attempts to notify the backend.
LLWU_Sys_Reset	Low-Leakage Wakeup Event – wakeup from deep sleep. MPL determines the LLWU wakeup source (e.g., timer, alarm, Node SRQ, etc.) and generates the appropriate event to it (i.e., inserts it into the event queue).
Software_Sys_Reset	Software Initiated Reset – used to recover from cortex exceptions or during an image manager firmware switchover. Software resets initiated by cortex exception result in exception events being logged to NVM for subsequent post-processing.

Core_Sys_Reset	Core Initiated Reset – unrecoverable processor faults during run-time execution result in a core reset. MPL logs the event to NVM for subsequent post-processing.
JTAG_Sys_Reset	JTAG Initiated Reset – lab debug use only. MPL only takes the necessary steps to restore hardware port settings.
External_Pin_Sys_Reset	External Pin Reset – lab debug use only. MPL treats this event in the same manner as a POR reset with regards to the initialization of all software resources.
Watchdog_Sys_Reset	Watchdog exceptions are logged and all hardware port settings are restored. The sensor attempts to resume steady-state using persistent application parameters.
LOC_Sys_Reset	Core Loss-of-Clock (LOC) Initiated Reset – unrecoverable processor faults during run-time execution result in a core reset. MPL logs the event to NVM for subsequent post-processing.
rACM Asynchronous Events	
Reed_Switch_Int	The rACM has detected a magnetic “swipe”or – the host interface and provisioning sub-state are affected by this event depending on the Top Level Device state. In addition, the rACM application uses the Reed Switch event to demonstrate the asynchronous alarm capability of the rACM platform.
LPT_Overrun_Int	When configured to use the Pulse Interface, the Kinetis utilizes a low-power Timer (LPTMR), 16-bit counter to keep track of flow rate. A high-water mark threshold is used by MPL to manage any potential overrun conditions.
SRQ_Int	The HAL has detected the Node SRQ line being toggled indicating that the Node has an unsolicited Node Host Interface message for the Application Host to process. This invokes the HOST_CMN Library to initiate the serial arbitration sequence.
LVD_Warning	The HAL has detected that the external battery voltage is below a certain threshold indicating a failing battery. If the Top Level Device state is DEPLOYED, the event is logged and an alarm event is generated OTA via HOST_CMN.
Image_Switchover	The HOST_CMN Image Manager has sent an event to MPL indicating that it has received and validated an uploaded firmware image in Flash Memory. MPL prepares the application for a switch to the new image which is followed by a software initiated system reset.
Node_Intf_Failure	The HOST_CMN manager has sent an event to MPL indicating that the node is non-responsive (after several retries). MPL logs the exception to NVM and updates the sensor device state accordingly.
TOUT_Interrupt	The HAL has detected the TOUT interrupt condition on rACM. Used in conjunction with the Time Synchronization Indicator, this synchronizes sensor time-of-day information with On-Ramp Wireless Network timing.
Log_NVM_Failure	The Flash Manager has experienced a failure in updating physical Flash for one of the sensor events. MPL generates an alarm event to the On-Ramp Wireless Network and discontinues the logging of event data.
Tx_Serial_Passthrough	The Application Serial interface is requesting that a character string be transferred OTA in the uplink direction. MPL, depending on the OTA state, will form and send the OTA Serial Passthrough packet.
rACM Synchronous Events	
UART_Tx_Complete	Used to notify the MPL that the transfer of outgoing data over the Host Serial Interface is complete – used to allow the Host Interface state machine to clear deep sleep veto.

SPI_Txfr_Complete	Used to notify that HOST_CMN SPI transfer is complete. The HOST_CMN state machine generates the appropriate actions necessary based on the Node Host SPI arbitration sequence.
LED_Sequence_Complete	Used to notify MPL that the any on-going LED pattern sequence has finished to completion.
rACM Timer Event	
Read_Interval	A timer event indicating that the MPL must initiate a sensor read that is mapped to the Read Block that initiated the event.
Host_Inactivity_Timeout	A timer event indicating that the Host UART Interface has been inactive for the specified timeout period and needs to be shutdown.
Provisioning_Timeout	A timer event during the on-site deployment procedure to perform the sensor interface detection scheme initiated by MPL.
Node_Inactivity_Timeout	A timer event indicating that the Node Interface has been inactive for the specified timeout period. The MPL attempts to reset the node to re-establish communications.
Time_Sync_Timeout	A timer event indicating that the Time Synchronization sequence with the On-Ramp Wireless Network has not completed with a specified time. MPL either retries immediately or waits until the next UI.
Poll_interval	A timer event indicating that the MPL must initiate a sensor poll that is mapped to the Poll Block that initiated the event.
Node Interface Events	
UI_Notification	The HOST_CMN Event Manager generates this event in response to a scheduled UI on the On-Ramp Wireless Network. The MPL uses this event to prepare to send any OTA synchronous data to the backend (e.g., sensor read logs).
Tx_SDU_Complete	The HOST_CMN Event Manager notifies the MPL when the current best-effort SDU for the current buffered sensor read log data for the current UI is acknowledged as received by the backend.
Node_State_Change	The HOST_CMN Event Manager generates an event state to MPL to help in management of the OTA State Engine (e.g., scanning, join, etc.).
Time_Sync_Ind	On request, the Node responds with network timing information as maintained by the Node. The Log Manager uses this information for time stamping the appropriate log entries in NVM.
Rx_SDU_Ind	The HOST_CMN Event Manager notifies MPL when it has a downlink message for the application to post-process (e.g., sensor application capabilities, image packet, or image switchover indication).
Board Specific Events	
RTC_Int	The Kinetis Real-Time 32K Counter has generated an interrupt due to a counter match. The MPL invokes the Timer Manager to determine the queued timer event(s) and generates the appropriate event indicator into the MPL event queue.
PIT_Int	The Kinetis Periodic Interval Timer (PIT) has generated an interrupt due to its expiration. The MPL invokes the Timer Manager to determine the subsequent actions required depending on queued timer event(s).
Deep_Sleep_Abort	A race condition has occurred between a Kinetis hardware interrupt while the Power Manager was preparing the Kinetis resources for deep sleep. MPL needs to take the appropriate recovery steps to resume steady-state operations.
Invalid_Wakeup_Int	The sensor application has determined that the device has woken out of deep sleep using an unrecognized/unmanaged interrupt source (possibly due to misconfiguration). MPL logs an exception for this type of event.

Sys_Tick	The interrupt handlers for the Cortex System Tick will generate this event to the Main Processing Loop at a predetermined periodic interval rate. The default value for this period is 25.6sec which the rACM firmware uses to refresh the Kinetis hardware watchdog.
Application Specific Events	
ADC_Sample_Complete	An event generated by the device interface manager when a non-blocking ADC sample sequence has completed (NOTE: A read sequence might consist of several sample sequences).
ADC_Read_Complete	An event generated by the device interface manager when a non-blocking ADC read sequence has completed.
ADC_Read_Timeout	A failsafe event generated by an RTC callback initiated by the device interface manager in the event that a non-blocking ADC read sequence could not complete.
Digital_Input_Interrupt	Interrupt events generated by cortex processor for digital inputs configured to for an edge-level interrupt (deep sleep or out-of-sleep).
Alarm_State_Change	An event generated by the device interface manager when an alarm state change is detected as a result of a read/poll. This allows MPL to manage alarm conditions/hysteresis with regards to OTA asynchronous alarm generation.

As seen indicated from the above table, events processed by the Main Processing Loop (MPL) can be categorized as follows:

- **System Reset Events –** The source of the system reset as determined by the RM when the Kinetis comes out of reset. This is used as the first queued event in the MPL Event Queue FIFO.
- **rACM Asynchronous Events –** Asynchronous rACM events that can occur at any point when the rACM is in steady-state operation. All asynchronous rACM events are capable of waking the Kinetis from deep sleep (i.e., handled by the LLWU block).
- **rACM Synchronous Events –** Events that are used to signal when an rACM operation has completed. These are used for hardware functions (e.g., SPI transfers, Flex Timer transfers) that cannot operate when the Kinetis is in a low-leakage power mode (i.e., the processing is in light sleep in the MPL).
- **rACM Timer Events –** Callback handlers register with the Timer Manager when the 32K RTC block generates a counter match. Timer Events are capable of waking the Kinetis as an LLWU interrupt.
- **Node Interface Events –** Node-> Host message events generated by the Node Interface Library used by the rACM to interact with the On-Ramp Wireless Network.
- **Board Specific Events –** Events that are specific to the platform the sensor application software is executing on. On the On-Ramp Wireless rACM platform, these are events unique to the Freescale Kinetis MCU.
- **Application Specific Events –** Events that are specific to the application. On the On-Rampe Wireless rACM firmware, these are events that are necessary to manage the Analog-to-Digital Conversion (ADC) sensor inputs, digital input interrupts, and alarm state changes. Applications that build upon the rACM framework may add/remove their own application-specific events as needed.

As stated before, the application integration guide goes into much greater detail on the event driven state machine used in MPL including all of the sensor application management functionality implementation. All public functions in the Main Processing Loop contain the *MPL* prefix.

8.4 Host Common Library

The Node Interface Library is a set of APIs that manage all transactions to and from the On-Ramp Total Reach Network (e.g., node status, OTA messages, UI notification, etc.). As an external software library, the interfaces to the HOST_CMN software components are discussed in section 9.1.

Because there are separate design documents regarding this library, details regarding the Node Interface implementation are considered beyond the scope of this document.

8.5 Host Common Hardware Abstraction Layer (HAL)

This is the supporting set of hardware drivers that are used by HOST_CMN and are unique to the ACM application. In most instances, this consists of a set of driver codes that manages the various Kinetis peripherals required to drive external interfaces between the ACM application and the Node.

All public functions in the Kinetis HAL contain the *HOST_CMN* prefix.

8.5.1 HAL Node Control Signals

To support message exchanges from the ACM Application and the Host over the SPI controller, a set of control signals must be supported by the HOST_CMN HAL for the following pins:

- **MRQ** – The Master Request output signal used by the application host to wake the Node out of deep sleep as a prelude to kicking off an arbitration sequence over the host↔node DSPI controller.
- **SRDY –** The Slave Ready input signal used as a “handshake” from the Node to the host to indicate that it is powered on and prepared to process an arbitration sequence over the host↔node DSPI controller.
- **SRQ** – The Slave Request input signal used by Node to alert the application host that Rx Data is prepared and requires an arbitration sequence to be initiated over the host↔node DSPI controller.

All of the aforementioned HAL Node Control signals use pins configured by the Kinetis GPIO controller. However, only the SRQ GPIO is configured as an LLWU interrupt, i.e., the SRQ is capable of waking up the rACM from deep sleep.

8.5.2 HAL Node Power Switch

The microNode is held in reset by the rACM until the device can go through the on-site provisioning sequence used to determine the type of flow sensor interface. When the device is

successfully installed, the microNode is pulled out of reset and HOST_CMN kicks off the initialization process to allow the Node to scan and join the On-Ramp Wireless Network.

The Node Power Switch uses a pin driven by the Kinetis GPIO controller. As an output pin, there are no additional features required by the Node Power Switch other than it retains its output configuration while the Kinetis operates in the various deep sleep modes.

8.5.3 HAL SPI Driver

The host↔node driver is implemented using the single SPI controller on the Kinetis (see reference [1], chapter 49 for configuration and operational details). All data transfers with the host are full duplex and use an arbitration protocol that is beyond the scope of this).

The HAL SPI Driver must support the following functions:

- **SPI Hardware Initialization**

  NOTE: Functionality may need to be done by the Reset Manager and/or MPL OTA State Machine due to transitions in and out of deep sleep)

- **SPI Enable**
- **Exchange SPI Message** – The Kinetis polling loop used to initiate a full duplex SPI transfer based on Rx/Tx buffers configured by HOST_CMN. Since the HAL driver uses a polling loop to guarantee that the SPI transfer completes, light sleep is also configured in the Kinetis MCU during this time.
- **SPI Disable**

  NOTE: This step must precede HOST_CMN's deep sleep veto flag clearing.

The Kinetis SPI Controller used by the HAL cannot operate in the Low Leakage Power modes. As a result, deep sleep modes (i.e., LLS and VLLS) must be vetoed during while the circuit is active.

8.5.4 HAL Timer Resources

The Kinetis HAL uses APIs provided by the Timer Manager software for purposes of enabling, disabling, and callback processing of all timer based events. The Timer Manager consists of a set of queued timer events and interfaces with the Kinetis 32K RTC to generate a time count match IRQ for post-processing by MPL (see section 8.8 for further details on TM).

8.6 Device Interface Manager (DEVICE_INTF)

This is essentially a hardware abstraction layer for all external interfaces that are NOT associated with controlling the Node. All of the necessary Kinetis drivers are invoked by this functional block as well as the scheduling of any callback event necessary to drive the MPL State engine.

All public functions in the Device Interface Manager contain the *DEVICE_INTF* prefix and can be broken down into the following functional APIs:

- Power-On-Reset initialization of interface resources
- Reset Recovery initialization of interface resources

- Reconfiguration of interface resources
- Start a read sequence on an interface resource.
- Return the alarm state (if applicable) on an interface resource.

8.6.1 Pulse Counter Interface

The Pulse Counter interface is conveniently implemented on the Kinetis using the Low Power Timer (LPTMR) module configured to increment an internal 16-bit counter on rising/falling edge of an input pulse waveform.

The LPTMR block can operate while the Device is in Deep Sleep, even in the lowest power control modes (VLLS1 – see section 8.7). There is no need to manage this interface in any of the power mode transitions. In addition, the LPTMR retains its current count during the different system resets – the only exceptions being the POR or LVD System Reset conditions.

The MPL interacts with the Pulse Counter Interface in the following manner:

- *Read_Timer* event triggers MPL to take the current LPTMR count value and save the count value to the Sensor application log in NVM.
- LPTMR Counter roll-overs are also managed by MPL (Alternatively, the count could be reset for every *Read_Timer* event).

8.6.2 Reed Switch Interface

The Reed Switch Interface is used to detect an external magnetic provisioning device which helps the rACM during on-site deployment and to activate the Host UART serial interface. The Reed switch drives one of the Kinetis GPIO configured to detect and generate an interrupt of the appropriate edge trigger.

The GPIO used for the Reed Switch Interface is also configured as an LLWU interrupt, i.e., it is capable of waking up the device from deep sleep.

When the Reed Switch IRQ is detected, the corresponding ISR clears the source of the interrupt and notify MPL by posting the appropriate event to its event queue.

The Reset Manager (RM) powers on and configures the Reed Switch GPIO following a POR system event. Thereafter, it remains operational regardless of the Top Level Device State.

8.6.3 Status LED Interface

The Status LED interface is driver by single output capable of being driven by the FTM block. The LED is driven for the following:

- The ‘SHUTDOWN’ pattern sequence
- The ‘PROVISION’ pattern sequence.
- The ‘JOIN’ pattern sequence.

The driver for the Status LED interface configures the Flex Timer Module (FTM) to generate the desired output pattern sequence. After the sequence is finished, an IRQ is generated and an

event is posted to MPL indicating that the LED Status pattern has completed so that the FTM block can be powered back down to conserve battery current.

The FTM block used by the Status LED interface cannot operate in the Kinetis Low Leakage Power modes. As a result, deep sleep mode (i.e., LLS and VLLS) must be vetoed during while the LED sequence is active.

The MPL interacts with the Pulse Counter Interface in the following manner:

- *Reed_Switch_Int* – Verification pulse for the magnetic "swipe" is generated. The sensor application interface provisioning sequence is generated by the MPL Top Level Device State Engine based on magnetic "swipes" counts.
- *LED_Sequence_Complete* – Indicator that the current LED pattern sequence has finished and that the FTM block/channel can be powered down.

8.6.4 Digital Input Interface

For application interfaces on the J207 header (APP_INTF1 – APP_INTF5) configured to act as a digital input, the following hardware resources are managed by the rACM Device Interface Manager:

- Individual Port Management configurations affecting:
- Asynchronous Edge-triggered Interrupt capability
- Internal pull-control settings (pull-up, pull-down, none)
- Individual Alarm Management configurations:
- Disabled - no OTA Async Alarm generation.
- Active High - generate an OTA Async Alarm during a low->high edge transition.
- Active Low – generate an OTA Async Alarm during a high->low edge transition.

NOTE: On rACM-based applications using the Sleepy profile, not all digital application interfaces are capable of interrupt the host processor while in deep sleep – see section 7.11 for a list of the deep-sleep capable digital inputs.

8.6.5 Digital Output Interface

For application interfaces on the J207 header (APP_INTF1 – APP_INTF5) configured to act as a digital output the following hardware resources are managed by the rACM Device Interface Manager:

- Individual Port Management configurations affecting:
- Open drain output capability.
- Drive strength capability
- Output Level (i.e., static hold at a digital low or digital high)

8.6.6 ADC Interface

To support analog input measurements, the following Kinetis Analog-to-Digital Conversion (ADC) resources are managed by the rACM Device Interface Manager:

- ADC Single-Ended port management for the following application interfaces:
  - ADC_BLOCK1 – Configuration/channel settings dedicated to sampling an analog input on APP_INTF6. This physically maps to the ANA_IN0 pin on the J207 application header.
  - ADC_BLOCK2 – Configuration/channel settings dedicated to sampling an analog input on APP_INTF5 configured for FUNC_MODE0. This physically maps to the PTC3 kinetis port routed through the J207 application header.
- ADC Block Management for the following:
  - Power Savings – following a system reset, the ADC block is configured, but kept powered-off until an analog read is initiated by the Main Processing Loop.
  - ADC Initialization – following a system reset, the ADC block is initialized and configured for the following operational parameters:
    - 16-bit A/D Sampling across a 3.0V External Reference
    - Long Sample Sequence
    - Hardware averaging over 32-samples
  - ADC Auto-Calibration – as part of the ADC block initialization, the ADC auto-calibration sequence is initiated to determine an initial gain and offset data points before starting a conversion on an external input.
  - ADC Read Sequence – All of the necessary management functions to initiate a read sequence (in coordination with the application control I/Os) and read the results from hardware once the conversion has completed.
- Flex Timer Module Management is also included part of the rACM ADC Drivers to support sample captures across a defined sampling period. In this mode, the FTM is configured for use as periodic clock that generates a hardware trigger for the ADC conversion hardware (as opposed to being generated via software).

In addition to the hardware resources listed above, the analog interface drivers are also responsible for managing the following software resources:

- Arithmetic Averaging (used for mean calculations)
- Alarm Level Hysteresis Management
- ADC Interrupt Service Routine

The Device Interface Manager (DIM) ensures that all of these resources are properly initialized when coming out of reset (POR or deep sleep) and ready for use by the Main Processing Loop.

8.7 Power Manager (PM)

The Power Manager software is responsible for placing the rACM into the lowest power state as possible for the current run-time operation thereby minimizing current draw on the battery – a critical function of the battery powered device as minimizing current draw correlates directly with extended battery life.

The methods used by the Power Manager are tightly couple with the Kinetis Power Manager Controller hardware (see chapter 7 of reference [1]). When the event queue used by the Main Processing Loop (MPL) is empty, the PM is invoked to place the MCU and all associated peripherals into the optimal current-draw state based on any pending ACM activity (e.g., light sleep or deep sleep). This decision logic is aided in part with the use of sleep "veto" flags by the various software functional blocks to indicate the state of various peripherals or memory buffers.

In addition to managing the Kinetis Power Manager Controller, the PM software is also responsible for the storage of rACM persistent data into the Deep Sleep EEE memory within the Kinetis FlexNVM block (see section 7.7). EEE memory allows data that is critical to the operation of the rACM to be preserved across periods of Very Low Leakage modes of operation on the Kinetis MCU where SRAM is powered off. Special logic in the EEE FlexNVM block allows efficient use of Flash Data memory with MCY support that automatically manages wear leveling and endurance cycles required for the estimated 20-year life of the product.

All public functions, in the Power Manager software function block, contain the *PM* prefix.

8.7.1 Kinetis Power Modes

Only the following Kinetis power modes are considered for the ACM device:

- **Normal Run** – Default run-mode of the core processor and peripherals when coming out of reset and when code is being executing. On-chip voltage regulator is on and all peripherals are allowed to function.
- **VLPR (Very Low power Run)** – An optional run-mode of the core processor that could be utilized by the rACM after coming out of reset. The chip, core, bus and peripheral clocks are all operating at a reduced clock frequency. The Low Voltage Detect (LVD) circuit is also non-functional at this state. On-target profiling determines whether Normal Run or VLPR mode is more efficient for purposes of extending battery life (i.e., VLPR is reduced power consumption at the cost of longer code execution periods).
- **Normal Wait (or VLPW)** – aka light sleep. The core is placed in sleep, but registers, SRAM, and peripherals are all clocked and fully functional. An interrupt wakes the core from sleep. On the rACM, this state is most likely entered into by MPL when the only pending activity is peripheral data transfers (e.g., SPI data exchanges with the Node, sensor application reads on the 3-wire interface, etc.).
- **LLS (Low Leakage Stop)** – aka deep sleep with state retention. Most peripherals are in state retention mode (exceptions are RTC, LPTimer, LLWU) and are inactive. SRAM memory is clocked and persistent. Only the LLWU module recovers the system from deep sleep. The rACM uses this state while waiting for network acknowledgments on uplink data transfers

(e.g., UI data transfers). This allows the rACM to retain message buffers for use in updating the Log Manager based on the results of such data transfers (which may last for minutes).

- **VLLS1 (Very Low Leakage Stop1)** – aka deep sleep. Only deep-sleep capable peripherals are operating (i.e., RTC, LPTimer, LLWU). All other peripherals, voltage regulators, and SRAM memory are powered off and require a LLWU interrupt to recover the system. This is the desired low-current state of the rACM when not actively managing the flow sensor and/or Node modem.

The Kinetis reference manual (see reference [1]) contains additional details about these power savings states as well as estimated current draw and it should be obvious that VLLS1 draws the least amount of current (i.e., 2uA or less).

Whenever the MPL has emptied its event queue, the Power Manager always attempts to place the application host into the lowest power savings state. If the lowest power state has been "vetoed" by any of the ACM's software blocks (e.g., Device interface Manager during data transfers), then the Power Manager checks the status of the next level of sleep (VLLS1 → LLS → Wait).

8.7.2 Veto Flags

Power Mode transition logic in the PM is done exclusively through the use of a 32-bit data object made up of bit "veto" flags. Each bit flag is reserved/assigned for the various ACM functional operations (e.g., sensor read, Host SPI data exchange, HOST_CMN uplink data transfer, etc.).

There is a 32-bit veto flag associated with each of the deep sleep modes supported by the Power Manager. In the case of the Kinetis mode controller used on the application host, they are as follows:

- PM_VetoFlag_DeepSleep
- PM_VetoFlag_Stop

It is the responsibility of the rACM software handlers to understand the power mode requirements of any on-going operation and set the appropriate veto flags accordingly. For example, when the Device Interface Manager has initiated an LED pattern sequence by programming the PIT Timer, it requires that the peripheral be actively clocked. As a result it sets its corresponding bit in BOTH the VLLS1 and LLS veto flags. When the LED sequence has completed (i.e., the PIT interrupt has occurred and MPL has post-processed the LED sequence Complete event), the Device Interface Manager clears the veto flags.

Only when the associated 32-bit veto flag is cleared (i.e., has a zero value), will the corresponding sleep state be entered into. This makes the Power Manager transitional logic quite simple as the following flow diagram illustrates.

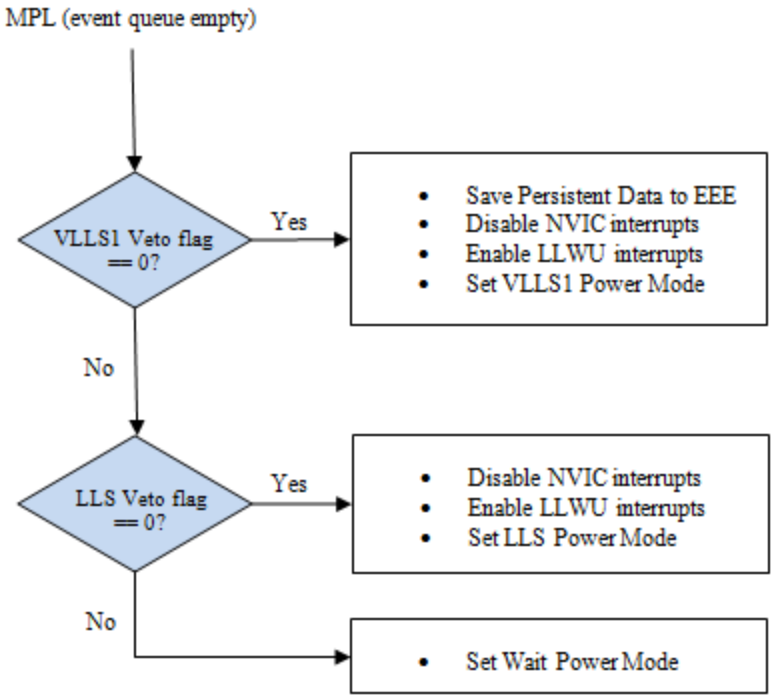


Figure 10. Power Manager Transition Flow

8.7.3 Light Sleep Transitions

Transitions into light sleep are relatively simple, the watchdog timer is temporarily disables and the wait mode is programmed into the Kinetis Mode Controller.

An interrupt causes the core to come back out of deep sleep; in most cases an event is inserted into the MPL event queue as a result of the interrupt service routine.

8.7.4 Deep Sleep Transitions

When the Power Manager determines that deep sleep is allowed (VLLS1 or LLS), the following steps must be taken to properly manage the Cortex processors and MCU resources:

- Cortex IRQ and FIQ are disabled to prevent lost interrupt events while the power manager preps the MCU for deep sleep. It is important to note that only LLWU interrupts should be generated by the system when deep sleep is permitted by the rACM software functional blocks.
- The Low Leakage Wakeup (LLWU) module is enabled. It is important to note that it is presumed that the MPL has properly set up LLWU interrupts prior to the system entering deep sleep (e.g., RTC timer and/or SRQ interrupt line). Failure to program an LLWU interrupt prior to a deep sleep request essentially keeps the rACM in a perpetual sleep state.
- The Kinetis Mode Controller is programmed for the appropriate power mode. The MCU can only be recovered from deep sleep through a LLWU system reset.

Transitions into the VLSS1 deep sleep power mode requires the additional step to save persistent memory into EEE Flash (see the next section).

8.7.5 Deep Sleep Persistent Memory

As described in the Architecture Overview (see section 7.7), there are two memory regions utilized by the rACM to store data that is persistent across long periods of deep sleep:

- 64 bytes System File Registers
- 32KB of Data Flash configured as FlexNVM (accessed via FlexRAM)

The System File Registers are used to save the following rACM data that are not dependent on On-Ramp Wireless Network configuration parameters:

- Absolute “count” value
- Device States

The System File Registers are accessed through the peripheral bus. They require no additional hardware management to ensure that system file data is written into hardware.

Additional persistent memory that is dependent on On-Ramp Wireless Network configuration parameters require more memory then the 64-byte System File Registers can provide. FlexNVM memory, accessed through the 2KB FlexRAM interface is reserved for the following:

- rACM Network Configuration parameters (protected via CRC)
- Time Manager callback queues (see section 8.8)
- Log Manager buffer pointers (see section 8.9)
- Image Manager firmware pointers (used by HOST_CMN)

E-Flash memory (when configured during the factory provisioning process) is loaded from FlexNVM into FlexRAM by the core following any system reset. The FlexRAM data can then read from or written in a traditional random access scheme. However, during a transition into deep sleep, the Power Manager ensures that the E-Flash FlexNVM has been properly synchronized with any updated data in FlexRAM by using the Flash Memory Controller (FMC).

The IAR link files are configured to link in all deep sleep persistent data, using the *PersistentData* prefix, into the FlexRAM address range.

It should be noted that FlexRAM data MUST be treated as uninitialized data by the linker to avoid FlexRam bus faults (and corrupted data) during the startup sequence.

8.8 Timer Manager (TM)

The Timer Manager is responsible for managing the timer resources on the Kinetis MCU for use by the rACM. The following Timer resources are supported:

- High resolution (i.e., microseconds) timers using the Kinetis Periodic Interrupt Timer (PIT) module.

- Low resolution (i.e., whole seconds) timers using the Kinetis Real Time Counter (RTC) module.

The Timer Manager provides a set of APIs for use by the rACM that is limited to the following services:

- Timing Information is maintained using the Universal Time Coordinated (UTC) format (i.e., time of day and calendar
- Allows the registration of a callback routine to occur at a specified delay (microseconds or seconds) that is relative to current time.
- All timer requests are one-shot ONLY. The Timer Manager does not support periodic callback generation.
- The Timer Manager queues a limited number of simultaneous requests internally, all multiple timer requests to run concurrently.
- The Timer Manager returns a timer block identifier value for use by calling the application for cancelling a callback request.
- The user application can cancel a registered timer using the timer block identifier.
- When a relative delay expires (due to a timer module count match), an interrupt is generated which, in turn, results in an inserted event entry into the MPL event queue for post-processing.
- The timer resources are available in ALL ACM device states.

Restricting features that are a normal to the software implementation of a typical Timer Manager are primarily intended to keep the rACM code size reduced.

The Timer Manager also supports the time synchronization feature with the On-Ramp Wireless Network through a set of utility functions invoked by the Node Time Synchronization and TOUT interrupt. In this method, the RTC counter is “latched” to a given ORW frame boundary.

All public functions in the Timer Manager contain the *TM* prefix.

8.8.1 High Resolution Timer Management

All timer requests that require microsecond resolution (up to a maximum value of 1 second) are implemented by the Timer Manager to use the Kinetis Periodic Interrupt Timer (PIT) module (see chapter 38 from reference [1]).

The following implementation restrictions are placed on the implementation and use of the High Resolution Timer resources:

- Up to three concurrent High Resolution Timers can run simultaneously. This is based on the PIT hardware implementation on the Kinetis MCU.
- Each PIT timer generates its own interrupt/event to the MPL. The registered callback is associated with each interrupt/event type as they are processed by the event queue (i.e., there is added latency in the actual execution of the callback depending on the backlog of queued/simultaneous events.

NOTE: Because of this restriction, if callback execution at a desired time is a critical requirement, then a delay loop in place of a Timer Manager callback request is the preferable solution. It is the responsibility of the application to take this into consideration.

- The PIT does not operate in the Kinetis Low Leakage power modes. As a result, deep sleep must be vetoed when high resolution callbacks are pending. The Timer Manager issues its own deep sleep veto flag to the Power Manager due to this restriction.
- When there are no registered High Resolution callback requests, the PIT module is powered off.

On the rACM, it is expected that HOST_CMN is the only resource used (of the High Resolution Timer resources) to aid in SPI arbitration during message exchanges with the On-Ramp Wireless Node.

8.8.2 Low Resolution Timer Requests

Most event callbacks on the rACM are typically on the order of minutes (if not hours). As such, a low resolution timer callback request makes use of the Kinetis Real Time Clock (RTC) module (see chapter 41 from reference [1]).

The RTC module is clocked by the external 32k crystal and is powered by VBAT. Its second counter continues to run even when the Kinetis MCU is operating in the Low Leakage power modes. As such, the RTC is responsible for most of the LLWU interrupts when the rACM is operating in the DEPLOYED state.

The following implementation considerations are placed on the implementation and use of the Low Resolution Timer resource:

- Up to 10 concurrent Low Resolution Timers can run simultaneously (e.g., flow sensor interval callbacks, UI Pre-Notification callbacks, IrDA timeout callbacks, etc.).
- The RTC only generates one interrupt. It is the responsibility of the Timer Manager to parse all registered timer events and program the RTC alarm match counter with the nearest timer expiration count.
- RTC interrupts are configured to also generate a LLWU interrupt source.
- When enabled, the RTC module is always powered on (both the external 32K oscillator and the timer module).
- The Timer Callback Management queue for Low Resolution callback requests MUST be located in deep sleep persistent memory (see section 8.7).

8.8.3 Time Synchronization

To support the synchronization of sensor reads with network Universal Time Coordinated (UTC), the sensor application needs to support the following interfaces with the Node:

- **Time Sync Indicator** – A solicited message received from the node containing UTC information that is applicable to the next On-Ramp Wireless Frame boundary (synchronized with the forthcoming TOUT interrupt).

- **TOUT hardware interrupt** – A dedicated hardware pin that is pulsed by the Node at an ORW frame boundary where the Time Sync Indicator information is applied to.

The Time Manager supports a number of utility functions that supports these Time Sync interfaces so that the Low Resolution Time (i.e., the RTC 1-second counter) is properly latched with UTC. These support functions perform the following:

- **Time Reference Set** – a utility function invoked by the Time Sync Indicator MPL event. Its main purpose is to buffer the UTC timing information for eventual "latching" on the TOUT interrupt.
- **RTC Latch** – A utility function invoked by the TOUT IRQ. Its only purpose is to reset the RTC 1-second counter so that its epoch it tightly synchronized with the TOUT interrupt. This is accomplished by resetting the Kinetis RTC 32K prescaler register (each 1-second "tick" is incremented when the 32K prescaler register rolls).
- **Time Reference Latch** – A utility function invoked by the TOUT MPL event. Its purpose is to take the buffered UTC data and apply the data as an updated Time Reference for the current RTC Count value. All timing-related functions now use the updated information for purposes of time-stamping and/or the scheduling of events.

A fourth utility function is also used by the Time Manager to be used by MPL to schedule sensor Reads at a specified interval offset (per the MPL configuration data). This is done by computing the delta between current time and next interval reading and referencing the current RTC count to determine the desired RTC count delta.

Time Synchronization with the On-Ramp Wireless Network is done under the following network events:

- The first On-Ramp Wireless Network joined indication (i.e., the JOINED_WAITING_FOR_TIME_SYNC OTA state).
- Immediately following OTA Data Payload transfers that occurs at each UI (i.e., when the node has its network time re-synchronized with the On-Ramp Wireless Network).

8.9 Log Manager (LM)

The Log Manager is responsible for storing the following rACM events into NVM as they are generated:

- Sensor count reads for each rACM read interval period
- rACM Alarm Events
- Node State Indicators
- rACM Exception Resets

The rACM applications also supports a large backlog of buffered event logs stored in NVM during periods of network outages. This buffered data can be recoverable for the later reconstruction and reporting of rACM events that occurred during these network outages.

In addition, the Log Manager is also required to manage pointers into the Log Record Buffer that identify the head and the tail of the Log record strings stored in NVM. This is coordinated with

the Main Processing Loop when preparing Uplink Data SDUs to the Node (via HOST_CMN) using the logged sensor data. It also aids in Flash block erasures as the Log Record Buffer eventually needs to roll over.

Log Data must also be able to be accessed by the Host UART interface to allow on-site debugging and/or analysis of rACM events.

All public functions in the Log Manager contain the *LM* prefix.

8.9.1 Log Record Buffer Implementation

The 16K Log Record Buffer in FLexNVM is configured as D-Flash using the traditional method of perform erase-write Flash data and implemented as a simple ring buffer. Each record is managed as a data stream of a known byte size with an identifier prefix. All records are also required to be byte aligned to avoid the use of read-modify-writes during log updates.

As with any circular buffer implementation, the Log Manager needs to keep a pointer to the head of the record data (to aid in the insertion of new record byte streams) and a pointer to the tail of the data (to aid in determination of overruns). An additional Log Manager pointer is also used to aid in SDU message generation when rACM is uploading sensor application activity records to the backhaul. It must be noted that the pointers for managing the Log Record Buffer MUST be in placed in persistent memory to ensure accurate record keeping.

Unlike traditional circular buffer implementation, the Log Manager must also be aware of the 2K sector boundaries as the buffer starts to roll-over (see section 8.9.4). Updates over old record data are not permitted in Flash memory. Instead, the flash sector where the data stream is overwriting must be erased in its entirety.

8.9.2 Log Record Format

The Log Record data stream is organized into the following record events categories:

- **Read Data** – Record includes an interface ID, time stamp, 32-bit primary read value0, and a 32-bit supplementary read value (if required). The default behavior of the rACM application is to store the absolute count value from the Low Power Timer (LPT) on APP_INTF1 into the value0 field.
- **Alarm Event Data** – Outstanding alarm events that are NOT acknowledged by the On-Ramp Wireless Network are logged in the record buffer for later upload to the network. Each Alarm event also includes a timestamp. (12-byte record).
- **ORW Node State Event Data** – Any state changes reported by HOST_CMN is logged as an event and reported to the backhaul. (8-byte record).
- **Application Exception Data** – Any unexpected system resets are logged as an exception and reported to the backhaul. (8-byte record).
- **Pending Alarm Event Data** – Identical to the Alarm Event record, these are used to manage asynchronous alarm conditions as they occur in real time but are in the process of getting transmitted to the network. The status of the pending alarm transfer via the asynchronous OTA alarm message determines whether or not the Pending Alarm record is used in

generating an undelivered Alarm Event record (which is used for attempted uploads to the network at a later time when channel conditions have improved).

With the exception of a log record "prefix" (i.e., a header field), the stored record data stream in the Log Record buffer closely matches the OTA messages generated by the rACM application (see section 9.2). Unlike OTA messages however, all log records are fixed sized (no variable length entries) and 32-bit word aligned. This is done allow to efficient SDU message generation when the Node requests payload data for a scheduled uplink interval.

As far as the amount of memory allowed for record backlogs, the following assumptions and record data streams are presumed:

- Read-format record at 15 minute intervals: 10 * (24 * 4 – 1) * 16bytes = 15360 bytes
- 1 Alarm record per day: 10 * 12 = 120 bytes
- 2 Node record per day: 10 * 2 * 8 = 160 bytes
- 1 Exception record per day: 10 * 8 = 80 bytes

Considering there should be NO alarms/exceptions on the rACM, this is a rather high estimate. In addition, the default read interval is at 1 hour. Regardless, the total estimate of 15720 bytes for this record log usage is just around the 16K of memory provided by the Log Manager's circular buffer implementation. To increase the backlog capability, the read format record can easily be modified down to 12 bytes by simply eliminating the supplementary data field.

The following subsections breaks down the format of each log record for reference.

8.9.2.1 Read Record

Table 8. Read Record

Record Byte Number	Description	Valid Values
0	Record Prefix	"R" ASCII Character
1	InterfaceID	The Application Interface ID associated with the read record.
2-5	Timestamp	Year/Month/Day/Seconds w- accuracy of +-10 seconds.
6 –9	Primary Read Value0	32-bits reserved for primary read data. Default format uses an absolute unsigned count value.
10 – 13	Supplementary Read Value1	32-bits reserved for supplemental read data. Default format is a signed delta count value from a previous read record entry.
14-15	Pad	Reserved – keeps record 32-bit aligned

8.9.2.2 Alarm Event Record

Table 9. Alarm Event Record

Message Byte Number	Description	Valid Values
0	Record Prefix	"A" ASCII Character
1	Bit 7:0 – Alarm Type 0 – App Interface1 1 – App Interface 2 2 – App Interface 3 3 – App Interface 4 4 – App Interface 5 5 – App Interface 6 6 – App Interface 7 7 – App Interface 8 8 – Alarm2 9 – Low Battery 10 – Security 11 – Log NVM 12 – Sleep NVM 13 – Exception* 14 - Test Bit 8 – Alarm State	Alarm Identifier *** NOTE:** When the exception status is set, an 8-bit enumerated status is appended to the alarm. In the case of chained messages the field is repeated in every alarm where the exception status remains set. Alarm State (0 – cleared, 1 – set)
2-5	Timestamp	Encoded Year/Month/Day/Seconds with accuracy of +/-10 seconds.
6-9	Alarm Data	16-bit battery level or 8-bit sensor register count
10-11	Pad	Reserved – keeps record 32-bit aligned

8.9.2.3 Node Event Record

Table 10. Node Event Record

Message Byte Number	Description	Valid Values
0	Record Prefix	"N" ASCII Character
1	Node Event	Node Event Enumerations: 0 – NIL 1 – Startup 2 – Idle 3 - Scanning 4 - Track 5 - Join
2-5	Timestamp	Encoded Year/Month/Day/Seconds with accuracy of +/-10 seconds.
6-7	Pad	Reserved – keeps record 32-bit aligned

8.9.2.4 Exception Event Record

Table 11. Exception Event Record

Message Byte Number	Description	Valid Values
0	Record Prefix	“E” ASCII Character
1	Exception Event	Exception Event enumerations are defined in Table 16 (see section 9.2.1 on alarm enums)
2-5	Timestamp	Encoded Year/Month/Day/Seconds with accuracy of +/-10 seconds.

8.9.2.5 Pending Alarm Event Record

Table 12. Pending Alarm Event Record

Message Byte Number	Description	Valid Values
0	Record Prefix	“P” ASCII Character
1	Bit 7:0 – Alarm Type 0 – App Interface1 1 – App Interface 2 2 – App Interface 3 3 – App Interface 4 4 – App Interface 5 5 – App Interface 6 6 – App Interface 7 7 – App Interface 8 8 – Alarm2 9 – Low Battery 10 – Security 11 – Log NVM 12 – Sleep NVM 13 – Exception* 14 - Test Bit 8 – Alarm State	Alarm Identifier *** NOTE 1:** When the exception status is set, an 8-bit enumerated status is appended to the alarm. In the case of chained messages the field is repeated in every alarm where the exception status remains set. Alarm State (0 – cleared, 1 – set)
2-5	Timestamp	Encoded Year/Month/Day/Seconds with accuracy of +/-10 seconds.
6-9	Alarm Data	16-bit battery level or 32-bit read value

8.9.3 Log Record Pointer Management

In order to manage the log record circular buffer, the following tracking variables are used by the Log Manager.

- **Log Base Pointer** – This hard-coded constant used in program memory is the start of the Log Record buffer. It is the absolute 32-bit system address on the Kinetis where the 16K offset into the FlexNVM Flash bank is located at (0x1000_0000).
- **Head Offset Pointer –** This is the 16-bit offset from the Log Base Pointer where log records are to be written to. Due to Log Record Buffer rollover (see the next section) the address pointed to is in an erased location of Flash memory (i.e., All 1's). The Head Offset Pointer is located in persistent memory. The Head Offset Pointer is updated following any log record insertion by the Log Manager.
- **Tail Offset Pointer –** This is the 16-bit offset from the Log Base Pointer where the tail of the log record circular buffer is located at. This is ALWAYS located at a 2K flash sector address boundary as memory from the circular buffer is freed up by the sector erase sequence. As with the head Offset point, this value also needs to be stored in persistent memory. The Tail Offset pointer is only updated follow sector erase operations as a result of the Log Manager's buffer rollover algorithm
- **Undelivered Data Offset Pointer –** This is the 16-bit offset from the Log Base Pointer is the start location of log records that are to form the basis of the OTA Read Data message that occurs at the next scheduled UI (providing the Node is still tracking the On-Ramp Wireless Network). As OTA SDUs are acknowledge by the On-Ramp Wireless Network, this offset pointer advances by the number of records sent. Under normal network operations, the upload offset point is also referencing a Sensor Read record entry. Should the Tail pointer ever exceed the Tail offset Pointer, a rACM exception event is generated (see the next section). Again, as with all offset pointers, this needs to be stored in persistent memory.
- **Undelivered Alarm Offset Pointer –** Similar to the data upload pointer, this is the start of the log records that are to form the basis of the OTA Alarm history message sent in a separate SDU at the next scheduled UI.
- **Pending Alarm Offset Pointer** – This 16-bit offset from the Log Base Pointer is used to aid in the management of Pending Alarm Records during race conditions in the OTA state where a previous Tx SDU may be ongoing while an alarm event has occurred.
- **Pending Alarm Sent Offset Pointer** – This 16-bit offset from the Log Base Pointer is used to aid in the management of Pending Alarm Records after their use in OTA Alarm Message payload generation. Should the OTA Alarm message fail to get transferred over the On-Ramp Wireless Network, then these Pending Alarm records can be quickly converted to undelivered Alarm records.
- **Pending Sensor Read Offset Pointer** – This 16-bit offset from the Log Base Pointer is used to aid in the management of Pending Read Records that occur while the Sensor is in the process of transmitting the OTA Read Interval Data.

There are other helper "pointers" to aid in the transfer of records from the Log Record buffer into OTA SDU messages but, because the system is prevented from entering deep sleep, these variables are not needed in persistent memory (and therefore are not discussed in any detail).

8.9.4 Log Record Buffer Overwrite Management

The Log Manager must ensure that all log records inserted into the Log Record ring buffer are inserted in the Flash memory that has been properly erased. The Long Manager ensures that as log record insertions cross Flash memory sector boundary locations, that the next sector is erased before proceeding with the sector update. Under normal circumstances (where the Node does not drop off the On-Ramp Wireless Network) the Upload Offset Pointer is located near or at the Head Offset Pointer (as data is uploaded to the backhaul at each UI). The following example helps describes the record buffer rollover management under normal circumstances.

1. The following offset pointers are set as follows:
   - HeadOffsetPtr = 0x7FC (i.e., 4 bytes before the start of the next 2k sector)
   - TailOffsetPtr = 0x800 (i.e., 2k sector boundary)
   - UndeliveredOffsetPtr = 0x400 (updated at last UI OTA SDU Acknowledgment)
2. The Main Processing Loop has determined that an asynchronous alarm needs to be logged to NVM (record size of 12 bytes) and invokes the Log Manager to insert the record.
3. The Log Manager processes the record insertion request. Because the inserted record results in the Tail Offset Pointer getting advanced, it performs the overwrite management sequence.
4. The Log Manager first checks to see if the sector erase sequence initiated the Tail sector boundary results in lost record data as indicated by the UploadOffsetPtr. In our example, it does not, and the sector erase can proceed without consequence.
5. The Log Manager invokes the sector erase procedure in the Kinetis FTM block for the Flash sector referenced by the Tail Offset Pointer. The Log Manager polls for the flash operation to complete (without going back into the MPL).
6. The Log Manager now invokes the Kinetis FTM block for writing the alarm record to Flash starting at the Head Offset Pointer. The Log Manager polls for the flash writes to complete for the entire record update.
7. The Log Manager updates the offset pointer following the record insertion to the following:
   - HeadOffsetPtr = 0x808 (following the insertion of the alarm record)
   - TailOffsetPtr = 0x1000 (advanced to the next sector boundary)
   - UndeliveredOffsetPtr = 0x400 (not affected by record insertion)

Should the node drop off the network for an extended period of time and the upload offset pointer may not have advanced, there remains the possibility that the application host will start to lose the backlog of rACM events. This cannot be prevented for outage periods exceeding 10 days. However, the Log Manager must log the more recent events and advance the upload pointer to ensure OTA message generation can resume as a result of lost records from a sector erase as the following example shows:

1. The following offset pointers are set as follows:
   - HeadOffsetPtr = 0x7FC (i.e., 4 bytes before the start of the next 2K sector)
   - TailOffsetPtr = 0x800 (i.e., 2K sector boundary)
   - UndeliveredOffsetPtr = 0x90C (updated at last UI OTA SDU Acknowledgement)

2. The Main Processing Loop has determined that an asynchronous alarm needs to be logged to NVM (record size of 12 bytes) and invokes the Log Manager to insert the record.
3. The Log Manager processes the record insertion request. Because the inserted record results in the Tail Offset Pointer getting advanced, it performs the overwrite management sequence.
4. Unlike the previous example, the Log Manager detects that the sector erase sequence results in overwriting log records that have not been sent OTA. The Log Manager parses the Log Record Buffer until it gets to the first valid record in the next sector boundary. The Upload Offset Pointer is advanced to this record location.
5. The Log Manager invokes the sector erase procedure in the Kinetis FTM block for the Flash sector referenced by the Tail Offset Pointer. The Log Manager polls for the flash operation to complete (without going back into the MPL).
6. The Log Manager now invokes the Kinetis FTM block for writing the alarm record to Flash starting at the Head Offset Pointer. The Log Manager polls for the flash writes to complete for the entire record update.
7. Because record data that has not been uploaded to the backhaul has been overwritten, the Log Manager also inserts an Exception Event record to Flash starting at the updated Head Offset Pointer. As before, the LM polls for this operation to complete.
8. The update Log Manager offset pointers are as follows:
    - HeadOffsetPtr = 0x810 (following the insertion of both the alarm and exception records)
    - TailOffsetPtr = 0x1000 (advanced to the next sector boundary)
    - UndeliveredOffsetPtr = 0x1008 (start of the next valid record following updated sector boundary)

It should be noted that the Log Manager also must check for and manage address ranges that exceed the physical 32-bit range of FlexNVM and wrap them around to the start of the Log Record Buffer.

8.9.5 Timestamp Management

For records that require a timestamp (most asynchronous events), the following resources on the rACM are used:

- The Time reference is synchronized with the On-Ramp Wireless Network at each UI opportunity using the HOST_CMN interface. By synchronizing the network time request during SDU payload transfer minimizes current consumption since the Node has already been powered on at this time. Any updated time reference is saved into persistent memory for use by the Log Manager.
- The low-power Kinetis Real-Time Clock (RTC) block contains a 32-bit seconds counter that can be used to determine an offset from the save reference offset for use in getting an updated time stamp for asynchronous events. The second counter is reset to zero following any network time reference from the On-Ramp Wireless Network.

9 rACM External Interfaces

9.1 Host Interface

The host↔node interface consists of a standard 4-wire SPI interface where the host is the SPI master, with the addition of two sleep signaling lines (MRQ/SRDY) and a line for the slave to request transfers (SRQ). A message protocol is implemented over this interface. For node↔host interface definition, refer to references [3] and [4] in chapter 1.

On the ACM application host, the HOST_CMN software library component is responsible for all message exchanges over the Host Interface. As long as the HOST_CMN software is active, the Main Processing Loop (see Figure 9) is responsible for performing an iterative pass through the HOST_CMN Manager. When the Host Interface has completed all pending message transactions, the run flag is cleared and the ACM application can enter into the appropriate sleep mode.

Other software interfaces with the HOST_CMN software library include the following:

- **Pre-UI Notification Event** – HOST_CMN notifies the application when it has received the Node->Host Pre-UI notification event. This indicator is handled by the HOST_INTF software subcomponent which, in turn, posts the corresponding event for subsequent processing by the MPL event handlers.
- **Node State Indicator Event** – HOST_CMN notifies the application when it has received a change in the Node's operational status (e.g., SCANNING, TRACK, or JOINED states). The HOST_INTF handler for this event posts this Node State change for subsequent processing by the MPL event handlers.
- **Node Reset Event** – Under exception conditions where the HOST_CMN cannot establish a message exchange with the Node (after several retry attempts), it resets the node as a last-ditch recovery attempt. HOST_INTF receives this indicator and, in turn, posts the corresponding event for subsequent processing of the MPL event handlers.
- **Host Common Persistent State Change** – The HOST_CMN software engine maintains a number of state and tracking variables that need to be persistent across the application low-power modes. However, as a set of external library functions, it does not have the necessary implementation details on how to interface with the persistent memory controller (in the case of the ACM host, this is through the FlexNVM interface). As a result, whenever any of these persistent variables are modified, an API handled by the HOST_INTF subcomponent is called which, in turn, updates the HOST_CMN set of persistent state data from SRAM into NVM.
- **Host Common Tx SDU Payload Queue Request –** Under normal use cases, it is expected that the ACM Application will only be required to initiate a Host->Node message exchange whenever an OTA message needs to originate from the end-device. The HOST_CMN Manager provides an API to service this type of request which is normally originated from the MPL state handlers (primarily based on the OTA state). Further details on the Tx SDU OTA message formats are found in section 9.2.

NOTE: The state machines in MPL are designed to manage only a single SDU payload at a time. This simplifies the design and minimizes the amount of memory that is required if multiple Tx SDU requests are forwarded to the On-Ramp Wireless Network.

- **Host Common Tx SDU Result Indicator** – When a Tx SDU request is forwarded to the Node by HOST_CMN, it is queued into the Node MAC's software state machine where the subsequent actions are taken to transmit the SDU to the backend. This could take several ORW frames, all depending on the RF link condition. The status of the Tx SDU is eventually reported by HOST_CMN to the HOST_INTF subcomponent which, in turn, posts the Tx SDU event and its result to the MPL event handlers.
- **Host Common Rx SDU Indicator** – Any downlink traffic that is originated from the backend to the ACM device is received at a designated LI interval (as configured in the network. On the ACM design, downlink traffic is primarily concerned with the reconfiguration of the sensor application's operational key-value pairs (see section 9.2). As with all indicators from HOST_CMN, the Rx SDU indicator is handled by a callback within HOST_INTF which is responsible for parsing the Rx SDU payload and taking any appropriate action based on the data therein.

9.2 Over-the-Air Interface

The OTA interface protocol is defined by the MAC specification. For more details, refer to reference [3] in chapter 1. The following messages are defined by the rACM application:

- Alarm Uplink Message (e.g., asynchronous Tamper Alarm)
- Data History Uplink Message (e.g., sensor data logs sent at the scheduled UI)
- Alarm History Uplink Message (e.g., alarm conditions that occurred when network is down)
- Device Configuration Downlink Message (e.g., sensor application interval, alarm thresholds, etc.)
- Firmware Version Request Downlink Message
- Firmware Version Response Uplink Message
- Node Reset Command Downlink Message

Every OTA message has a header that is used to identify what type of message it is and optionally how big its data payload is. The header is followed by the data payload. The header plus the payload must be a multiple of eight.

The generic format of the OTA message is illustrated below.

Table 13. OTA Message Format

Byte 0	Byte 1	Byte 2	Byte 3	…	Byte N+2
Header		Payload			
Op Code	Payload Data Byte 0	Payload Data Byte 1	Payload Data Byte 2	…	Payload Data Byte N

The Op Code field specifies what type of data is being communicated within this message. This field is unsigned and one byte, having a range of 0 – 255. Not all values within this range are defined. If a message with an undefined Op Code is received at either end, the message shall be ignored. The following table lists the defined Op Codes on the rACM application.

Table 14. rACM Message Op Codes

Message Type	Op Code	Direction	Delivery Reliability	Description
Alarm	0x1	Uplink	Reliable Asynchronous	Used to report device alarms.
Data	0x2	Uplink	Reliable Scheduled	Used to report sensor application data and device state.
Configuration Write	0x3	Downlink	Reliable Scheduled	Used to configure device.
Configuration Read Request	0x4	Downlink	Reliable Scheduled	Used to request a configuration value read.
Configuration Read Response	0x4	Uplink	Reliable Scheduled	Used to report configuration value.
Reset Command	0x5	Downlink	Reliable Scheduled	Used to request a host and/or node reset.
Sensor Data History	0x6	Uplink	Reliable Scheduled	Used to report all pending sensor data.
Serial Passthrough	0x7	Downlink and Uplink	Reliable Asynchronous	Used as a tunnel for low-rate serialized character strings

Uplink messages can be either Scheduled, which means messages are queued up and wait to be transmitted until the Update Interval occurs, or they can be Asynchronous, which means messages are sent as soon as possible regardless of the Update Interval.

Downlink messages are queued at the AP and are transmitted when the next uplink message from the device is successfully received at the AP. Under normal operating conditions the worst case delay occurs when the downlink message is queued immediately after the Update interval and remains in the queue until the next Update interval.

NOTE: Multi-field bytes (e.g., 16-bit or 32-bit values) are in little endian format (i.e., least significant byte to most significant byte ordered from left → right).

9.2.1 Alarm Message

As the ACM application detects an urgent alarm condition that must be reported to the backhaul, a single Alarm SDU (with the appropriate alarm status bit settings) is formed and sent to the Node for immediate transmission to the backhaul via HOST_CMN.

If the AP/Gateway acknowledges the receipt of the Alarm SDU, normal steady-state operations resume following a notification to the MPL by HOST_CMN.

If the Alarm Message is NACKed by the On-Ramp Wireless Network, the alarm event is then saved into the Log Record Buffer by the Log Manager for subsequent retransmission during the next UI period.

Alarm events that fail to get reported to the backend using the asynchronous alarm SDU are saved in the Log Record Buffer for retransmission attempts at the next scheduled UI. Multiple alarm events in the log record buffer can be "chained" together in a single SDU. Like the Data message, the following considerations are in affect during message construction:

- The Alarm History message must fit into a single SDU (i.e., max payload of 468 bytes).
- Alarm History records are ordered from oldest record in the log buffer to the most recent.
- Only the backlog of alarm history records that fit into the max SDU size are transmitted for the current UI. Further alarm history backlogs are required to be sent at the following UI.
- Only when the Alarm History message is acknowledged as received by the On-Ramp Wireless Network from HOST_CMN, will the alarm upload pointer advance though the Log Record buffer.

If there are NO alarm record events in the Log Record Buffer, the SDU is aborted and the alarm upload point advances to the head of the chain buffer (i.e., this uplink message does not necessarily need to be sent every UI).

The format of the rACM Alarm Message and the fields within are as follows:

Table 15. Alarm Message Format

Message Byte Number	Description	Valid Values
0	Header	1
1	Alarm Count	Number of chained alarm records.
2	Bit[6:0] – Alarm Type 0 – App Interface1 1 – App Interface 2 2 – App Interface 3 3 – App Interface 4 4 – App Interface 5 5 – App Interface 6 6 – App Interface 7 7 – App Interface 8 8 – Alarm2 9 – Low Battery 10 – Security 11 – Log NVM 12 – Sleep NVM 13 – Exception* 14 - Test Bit[7] – Alarm State	Alarm Identifier *** NOTE 1:** When the exception status is set, an 8-bit enumerated status is appended to the alarm. In the case of chained messages the field is repeated in every alarm where the exception status remains set. The exception enumerated values are defined in Table 16. Alarm State (0 – cleared, 1 – set)

Message Byte Number	Description	Valid Values
3-6	Timestamp Bit 5:0 – seconds Bit 11:6 – minutes Bit 16:12 – hour Bit 21:17 – day Bit 25:22 – month Bit 31:26 – year offset from 2000	Accuracy of >1 second.
7 and/or 7/8/11-n	Enumerated Status and/or Next chained alarm	8-bit enumerated status and/or Alarm status + timestamp +

The run-time exception events that the rACM application will report are defined in the following table:

Table 16. Exception Events

Exception Identifier (hex)	Description
MPL State-Event Exceptions (ID Range 0x00-0x0F)	
0x00	Invalid SRQ Event
0x01	Invalid Tx SDU Complete Event
MPL Event Queue Exceptions (ID Range 0x10-0x1F)	
0x010	Event Queue Overflow
Tx SDU Data Network Exceptions (0x20-0x2F)	
0x20	Data SDU Tx Failure
0x21	Buffered Alarm SDU Tx Failure
0x22	Async Alarm SDU Tx Failure
Tx SDU Data Request Exceptions (0x30-0x3F)	
0x30	Data SDU Pack Failure
0x31	SDU Payload Access Conflict
MPL Timer Exceptions (0x40-0x4F)	
0x40	Timer Queue Overflow
0x41	Stale Timer Event
0x42	Timer Invalid Clear
SPID Exceptions (0x50-0x5F)	
0x50	SPID Transfer Conflict
0x51	SPID Transfer Failure
Node Interface Exceptions (0x60-0x6F)	
0x60	Node Interface Inactivity Timeout
0x61	Node Interface SPI Message Timeout
0x62	Node Interface ACK Timeout
System Reset Exceptions (0x70-0x7F)	
0x70	Software Reset Unknown
0x71	Invalid ISR Vector

Exception Identifier (hex)	Description
0x72	Invalid NMI
0x73	Hard Fault
0x74	MPU Fault
0x75	Bus Fault
0x76	Usage Fault
0x77	LVD System Reset
0x78	Core Lockup Reset
0x79	Watchdog Reset
0x7A	LOC Reset
0x7B	Invalid Wakeup Interrupt
Rx SDU Exceptions (0x80-0x8F)	
0x80	Rx SDU Invalid Downlink Opcode
0x81	Rx SDU Invalid Message Class
Time Sync Exceptions (0x90-0x9F)	
0x90	Time Sync Timeout
0x91	Time Sync TOUT Latch Error
0x92	Time Sync Payload Access Conflict
0x93	PIT In Use Error
Application Specific (i.e., reserved) Data (0xA0 – 0xFF)	
0xA0-0xFF	*Defined by custom application*

9.2.2 Data Message (deprecated)

The Data message is used by the rACM to report the backlog of all sensor application reads to the backend for subsequent post-processing. This message is constructed and sent at the scheduled uplink interval of the Node. In coordination with the Log Manager, all events from the last acknowledged UI (as tracked using the LM Upload Offset pointer) to the head of the log record buffer are formatted into a single SDU to be transmitted by the Node. There are several considerations regarding the Data History message construction:

- The Data History message must fit into a single SDU (i.e., max payload of 468 bytes).
- Data History read records are ordered from oldest record in the log buffer to the most recent.
- If the backlog of count records exceeds the max payload restriction, then the message is reconstructed using the next interval count (e.g., 15 minutes to 30 minutes read intervals). Until all of the data can fit a single SDU, this process repeats itself.

  NOTE: As subsequent UIs are missed, the backlog grows larger (up to a maximum of 33 days). The resolution of count history available at CIMA for long network outages is reduced when the node reacquires the On-Ramp Wireless Network.
- Only when the Data History message is acknowledged from HOST_CMN, will the data upload pointer advance though the Log Record buffer.

This implementation scheme serves two purposes:

- There is no log record buffer or OTA message fragmentation to manage in software (i.e., keeps the code simple and streamlined)
- Backlogs can be recovered and uploaded to CIMA in the case of network outages (again, at the cost of count history resolution).

The format of the rACM Data Message and the fields within are as follows:

Table 17. Data Message Format

Message Byte Number	Description	Valid Values
0	Header	2
1	Data Count	Number of chained data records.
2-5	Timestamp Bit 5:0 – seconds Bit 11:6 – minutes Bit 16:12 – hour Bit 21:17 – day Bit 25:22 – month Bit 31:26 – year offset from 2000	Accuracy of >1 second.
6 –9	ReadData0	32-bits reserved for primary sensor read data. Default format uses an absolute unsigned count value.
10-13	ReadData1	32-bits reserved for supplemental sensor read data. Default format is a signed delta count value from a previous read record entry.
14 – n	Next chained data record	Timestamp + readData0 + readData1

9.2.3 Configuration Write Message

This is a downlink message used to communicate device configuration from the network. Each configuration field in the device is addressed by a unique On-Ramp Wireless Identification (ORW ID). A configuration message contains a sequence of key-value pairs where the key is the 8-bit ORW ID and the value is the 16-bit value to assign to the configuration mapped to the key. The number of key-value pairs in a single configuration message can vary from one key-value pair to all valid pairs.

The format of the rACM Configuration Write message and the fields within are as follows:

Table 18. Configuration Message Format

Message Byte Number	Description	Valid Values
0	Header	3
1	Config Count	Number of chained key-value pairs,
2	Key [0]	ORW ID 0-255

Message Byte Number	Description	Valid Values
3-4	Value [0]	
5	Key [1]	ORW ID 0-255
6-7	Value [1]	
8-n		

Valid configuration key-value pair mappings are shown in table Table 3 from Part I of this document.

NOTE: Key Value pairs starting at 0x80 are command-only messages and do not require its value to be set as a persistent configuration on the sensor application.

9.2.4 Configuration Request Message

This is a downlink message used to query the device configuration. This is sent as a Reliable Delivery message. When the device receives this message, it responds with a "Configuration Response" Message in the uplink.

Table 19. Configuration Request Message Format

Message Byte Number	Description	Valid Values
0	Header: Op Code	4
1-7	Reserved	Reserved for future use. Value = 0

9.2.5 Configuration Response Message

This is an uplink message used to respond to a "Configuration Request" Message. This is sent as a Reliable Delivery message. Every time the device receives the "Configuration Request" Message, it replies with this message. The message contents the entire device configuration starting with the first configuration value (0) and ending with the last configuration value (m). Each value is a 2-byte field.

NOTE: If the 'Network Discovery Connect' configuration option is enabled, the rACM autonomously generates this unsolicited response when it joins the network for the first time.

Table 20. Configuration Response Message Format

Message Byte Number	Description	Valid Values
0	Header: Op Code	4
1	Major Revision	Populated from revision value in version.h
2	Minor Revision	Populated from revision value in version.h
3	Point Revision	Populated from revision value in version.h
4	Config Count	Number of chained configuration values
5-6	Configuration Value[0]	

Message Byte Number	Description	Valid Values
6-7	Configuration Value[1]	
…		
n-n+1	Configuration Value[m]	
m = n+2	Sensor Count	Number of chained sensor values
m+1-m+2	Sensor X Header Bit 3:0 – Identification Bit 15:4 – Type*	Identifies which sensor and type the following immediate read data is from *Note: See table 17 for a list of supported sensor types.
m+3 – m+x	Sensor X Immediate ReadData	Variable Length Bytes (based on interface type – see table 17) for Sensor X immediate read.
m+(x+1)- m+(x+2)	Sensor Y Header Bit 3:0 – Identification Bit 15:4 – Type*	Identifies which sensor and type the following immediate read data is from *Note: See table 17 for a list of supported sensor types.
m+(x+3) – m+((x+3)+y)	Sensor Y Immediate ReadData	Variable Length Bytes (based on interface type – see table 17) for Sensor Y immediate read.
…	…	Repeated for the total number of sensors reported in the Sensor Count field.

9.2.6 Node Reset Command Message

This is a downlink message used to instruct the host and/or the node to reset. This is sent as a Reliable Delivery message. When the node receives this message, it resets the appropriate processors of the node as noted in the following table.

Table 21. Node Reset Command Message Format

Message Byte Number	Description	Valid Values
0	Header: Op Code	5
1	Static Key 1	0x5A
2	Static Key 2	0xE5
3	Reset Host	1 resets the host, 0 does not reset the host. Any other value invalidates this message.
4	Reset Node	1 resets the Node, 0 does not reset the Node. Any other value invalidates this message.
5-7	Reserved	Reserved for future use. Value = 0

9.2.7 Sensor Data History Message

The sensor data history message is used by the rACM to report the backlog of all read data to the backend for subsequent post-processing. This message is formatted at the scheduled uplink interval of the Node. In coordination with the Log Manager, all events from the last acknowledged UI (as tracked using the LM Upload Offset pointer) to the head of the log record

buffer are formatted into a single SDU to be transmitted by the Node. There are several considerations regarding the Sensor Data History message construction:

- The Sensor Data History message must fit into a single SDU (i.e., max payload of 468 bytes).
- Sensor Data for up to four separate interfaces are concatenated together – each with a separate chain buffer. However, the CPMON currently only supports a single analog sensor interface.
- Data History read records are ordered from oldest record in the log buffer to the most recent.
- If the backlog of count records exceeds the max payload restriction, then the message is reconstructed using the next interval count (e.g., 15 minutes to 30 minutes read intervals). Until all of the data can fit a single SDU, this process repeats itself across all of the concatenated sensor interfaces.

  NOTE: As subsequent UIs are missed, the backlog grows larger (up to a maximum of 33 days). The resolution of count history available at CIMA for long network outages is reduced when the node reacquires the On-Ramp Wireless Network.

- Only when the Data History message is acknowledged from HOST_CMN, will the data upload pointer advance though the Log Record buffer.

This implementation scheme serves two purposes:

- There is no log record buffer or OTA message fragmentation to manage in software (i.e., keeps the code simple and streamlined)
- Backlogs can be recovered and uploaded to CIMA in the case of network outages (again, at the cost of count history resolution).

The format of the CPMon Sensor Data Message and the fields within are as follows:

Table 22. Sensor Data History Message Format

Message Byte Number	Description	Valid Values
0	Header	6
1	Sensor Count	Number of chained sensors
2-5	Sensor X Header Bit 3:0 – Identification Bit 15:4 – Type* Bit 23:16 – Read Interval Bit 31:24- Read Count	Identifies which sensor the following read data is from, the type of sensor which provides the data format, the time between reads and the number of reads. *Note: See table 17 for a list of supported sensor types.
6-9	Sensor X Timestamp Bit 5:0 – seconds Bit 11:6 – minutes Bit 16:12 – hour Bit 21:17 – day Bit 25:22 – month Bit 31:26 – year offset from 2000	Accuracy of >1 second.

Message Byte Number	Description	Valid Values
10-13[1]	Sensor X ReadData0	Variable Length Bytes (based on interface type – see table 17) for Sensor X read data aligned to timestamp.
14-17	Sensor X ReadData1	Variable Length Bytes (based on interface type – see table 17) for Sensor X read data aligned to timestamp+readInterval.
M – M+3	Sensor X ReadDataN	Variable Length Bytes (based on interface type – see table 17) for Sensor X read data aligned to timestamp+(N-1)*readInterval.
M+4 – M+7	Sensor Y Header	(same format as Sensor X Header)
M+8 – M+11	Sensor Y Timestamp	(same format as Sensor X TimeStamp)
M+12 – M+15	Sensor Y ReadData0	(same format as Sensor X ReadData0
M+16 …	…	(Repeated for the total number of sensors reported in the Sensor Count field)

The Interface Type used in the Sensor Header field serves two purposes in that it identifies:

- The type of units the analog sensor interface is reported (e.g., DC Voltage using units of millivolts).
- The byte size of each Data field.

rACM-based applications currently support the following interface types:

Table 23. Sensor Interface Types

Sensor Interface Identifier (hex)	Description	Read Data Payload Length (bytes)
0x000	8-bit signed data, in engineering units or eu (i.e., undefined units of measurement).	1
0x001	16-bit signed data, in engineering units or eu (i.e., undefined units of measurement).	2
0x002	32-bit signed data, in engineering units or eu (i.e., undefined units of measurement).	4
0x003	64-bit signed data, in engineering units or eu (i.e., undefined units of measurement).	8
0x004	32-bit signed data followed by 4-bit signed delta values in eu (i.e., engineering units).	4 (first), 4-bit nibble for each record thereafter
0x005	32-bit signed data followed by 8-bit signed delta values in eu (i.e., engineering units).	4 (first), 1 byte for each record thereafter
0x006	32-bit signed data followed by 12-bit signed delta values in eu (i.e., engineering units).	4 (first), 3 nibbles for each record thereafter

[1] The bolded text in the table reflects values that may change in later versions of the software; i.e., they define byte offsets of variable length (e.g., x+3 or x+n).

Sensor Interface Identifier (hex)	Description	Read Data Payload Length (bytes)
0x007	32-bit signed data followed by16-bit signed delta values in eu (i.e., engineering units).	4 (first), 2 bytes for each record thereafter
Pulse Count Types (ID Range 0x010-0x01F)		
0x010	8-bit unsigned pulse counts.	1
0x011	16-bit unsigned pulse counts.	2
0x012	32-bit unsigned pulse counts	4
0x013	64-bit unsigned pulse counts	8
0x014	32-bit unsigned absolute followed by 4-bit unsigned delta pulse count values.	4 (first), 4-bit nibble for each record thereafter
0x015	32-bit unsigned absolute followed by 8-bit unsigned delta pulse count values.	4 (first), 1 byte for each record thereafter
0x016	32-bit unsigned absolute followed by 12-bit unsigned delta pulse count values.	4 (first), 3 nibbles for each record thereafter
0x017	32-bit unsigned absolute followed by 16-bit unsigned delta pulse count values.	4 (first), 2 bytes for each record thereafter
A/D Count Types (0x020-0x02F)		
0x020	8-bit unsigned A/D counts.	1
0x021	16-bit unsigned A/D counts.	2
0x022	16-bit unsigned absolute followed by 4-bit signed delta A/D count values.	2 (first), 4-bit nibble for each record thereafter
0x023	16-bit unsigned absolute followed by 8-bit signed delta A/D count values.	2 (first), 1 byte for each record thereafter
0x024	16-bit unsigned absolute followed by 12-bit signed delta A/D count values.	2 (first), 3 nibbles for each record thereafter
0x025	16-bit unsigned absolute followed by 16-bit signed delta A/D count values.	2 (first), 2 bytes for each record thereafter
Voltage Types (0x030-0x03F)		
0x030	8-bit signed DC Volts	1
0x031	16-bit signed DC Volts	2
0x032	8-bit unsigned AC Volts	1
0x033	16-bit unsigned AC Volts	2
0x034	8-bit signed DC Millivolts	1
0x035	16-bit signed DC Millivolts	2
0x036	8-bit unsigned AC Millivolts	1
0x037	16-bit unsigned AC Millivolts	2
0x038	8-bit signed DC Microvolts	1
0x039	16-bit signed DC Microvolt	2
Current Types (0x040-0x04F)		
0x040	8-bit signed Amps	1
0x041	16-bit signed Amps	2
0x042	8-bit unsigned Milliamps	1

Sensor Interface Identifier (hex)	Description	Read Data Payload Length (bytes)
0x043	16-bit unsigned Milliamps	2
0x044	8-bit signed Microamps	1
0x045	16-bit signed Microamps	2
Temperature Types (0x050-0x05F)		
0x050	16-bit signed Temperature, in Fahrenheit	2
0x051	8-bit signed Temperature, in Celsius	1
0x052	16-bit signed Temperature, in Celsius	2
0x053	16-bit unsigned Temperature, in Kelvin	2
Pressure Types (0x060-0x06F)		
0x060	16-bit unsigned Pressure, in Millibars	2
0x061	8-bit unsigned Pressure, in psi	1
0x062	16-bit unsigned Pressure, in psi	2
Power Types (0x70-0x07F)		
0x070	8-bit unsigned Power, in Watts	1
0x071	16-bit unsigned Power, in Watts	2
0x072	8-bit unsigned Power, in Volt Amperes	1
0x073	16-bit unsigned Power, in Volt Amperes	2
Position Location Types (0x080-0x08F)		
0x080	Position Data, Lat and Long, in 10^(-16) integer coordinates	8
Digital Types (0x090-0x09F)		
0x090	Digital I/O Data	1
Application Diagnostic Types (0x0A0-0x0AF)		
0x0A0	8-bit rACM Exception ID followed by a 32-bit Exception Timestamp followed by 16-bit rACM Exception Data	10
0x0A1	32-bit Node Indicator Timestamp followed by a 8-bit Node Indicator ID	5
Application Specific (i.e., reserved) Data (0xF00 – 0xFFF)		
0xF00	Application Specific 1-byte data field	1
0xF01	Application Specific 2-byte data field	2
0xF02	Application Specific 3-byte data field	3
0xF03	Application Specific 4-byte data field	4
0xF04	Application Specific 5-byte data field	5
0xF05	Application Specific 6-byte data field	6
0xF06	Application Specific 7-byte data field	7
0xF07	Application Specific 8-byte data field	8

NOTE: The rACM application reports 32-bit pulse counts using this format (i.e., Sensor Interface Identifier of 0x12).

9.2.8 Serial Passthrough Message

This message format is the same for both the downlink and uplink directions – it uses a variable length format to convey a contiguous ASCII character string. Its intended use is for applications that require the network node to simply serve as a "tunnel" for low-rate character bytes between the backend and the rACM application serial port.

Table 24. Serial Passthrough Message Format

Message Byte Number	Description	Valid Values
0	Header: Op Code	7
1-2	Number of serial data bytes	1 to 461 (constrained by max PDU size) – NOTE: As with all multi-byte fields, this multi-byte field uses little endian format (e.g., 0x0100 equates to a value of 1).
3-x	Serialized character stream	

9.3 Host Serial Interface

The Host UART interface is responsible for transmitting serialized data over the standard UART interface. The following functionality is supported by this interface:

- Key Value Pair Configuration (see Table 18 for the supporting key-value pairs on rACM)
- Real-time rACM Application Event Logging
- rACM Exception Log Retrieval
- rACM Alarm Log Retrieval
- rACM application firmware upgrade
- ORW Node Firmware upgrade
- ORW Node Monitor Interface (for network debugging)
- ORW Node security key provisioning
- ORW Node AP Channel list provisioning

10 rACM Internal Interfaces

10.1 MPL Event Queue FIFO

The MPL Event Queue is a simple circular buffer implementation that acts as the input stimuli of the various state engines that make up the Main Processing Loop (MPL). It also serves as the only method of performing inter-task communication from the interrupt thread to the process thread.

The MPL event queue is implemented as a simple circular buffer of event entries. It is sized for the maximum number of backlogged events that can be generated by the rACM process and interrupt threads (currently sized at 16). Each event is processed by the MPL to completion before the entry is flushed from the FIFO and the next entry is executed. There is no concept of high priority events on the rACM application.

The Event Queue FIFO has the following implementation features:

- Each Queue object is a single 16-bit enumerated value identifying the sensor event (see event tables in section 8.3). No other data is passed into the event queue. This keeps the queue entries to a fixed 2-byte size and minimizes the amount of "protected" data to be managed by the MPL event handlers.
- The Event Queue contains a head pointer where the next entry is to be processed by the MPL state engines.
- The Event Queue contains a tail pointer where the next entry is to be inserted into the Event FIFO (by either process or interrupt threads).
- All pointer updates in the process thread MUST be done with system interrupts disabled. This is required since the Event Queue is the only software resource in the rACM updated by both the Process and Interrupt Threads.
- The Event Queue is considered "empty" when the tail and head pointers are equal. This triggers the Power Manager to configure the Core for sleep (either deep or light sleep).

10.2 Rx UART FIFO

Serial data, as it is received by the Kinetis UART block, also requires a process safe method to move serialized bytes from an interrupt context to the Main Processing Loop (MPL) for subsequent processing by the appropriate endpoint handler.

There are two supported serial streams on the ACM application:

- **Host Interface serial stream** – For all Host-Specific and Node Endpoint commands are processed (e.g., On-Ramp Wireless Node Configuration and Control, rACM Provisioning commands, etc.). The HOST_CMN software library is responsible for performing the necessary stream parsing of all incoming bytes.
- **Application-specific serial stream** – Reserved for a serial interface with any external device (e.g., NMEA GPS protocol commands, Sensus Sensor Specification protocol, Modbus

protocol commands). The rACM application software is responsible for performing the necessary stream parsing of all incoming bytes.

NOTE: The rACM application currently does NOT perform this type of functionality (i.e., the application UART is by default powered down).

The UART FIFO is also implemented as a simple circular buffer of character bytes. It is sized for the maximum protocol message size that is expected for each serial link. Each iterative pass through the MPL results in a call to the UART foreground processing handler that uses the following implementation scheme.

- During UART device initialization, the application is responsible for logging a callback function which performs all stream parsing of incoming bytes during the MPL iterative pass.
- The Rx UART Interrupt always updates bytes into the head pointer of the UART circular buffer.
- Stream parsing of any outstanding Rx bytes in the UART FIFO starts at the tail pointer.
- The registered UART callback is responsible for freeing a number of bytes from the tail pointer of the UART circular buffer when it has determined that the bytes are “free.”
- When the UART circular tail pointer is the same as the head pointer, there are no pending UART characters to process.

PART 3: Appendices

The following appendices provide additional information for Part 1: Quick Start and Part 2: rACM Software Components and cover the following:

- Operating examples
- Application configuration examples
- Over-the-Air (OTA) payload examples
- Example sequence flow charts
- Abbreviations and terms

Appendix A Operating Examples

This chapter provides example to illustrate the sequence of steps to:

- Confirm rACM power-up device status using *host_app_ctrl.py*
- Change rACM operating mode to disabled host sleep using *ctrl.py*
- Configure the rACM's AP list using *ulp_node_monitor.py*
- Verify the rACM joins the network using *ulp_node_monitor.py*
- Log host operation using *host_app_logger.py*

A.1 Get Device Status

After a successful power-up signed by the 2-blink LED sequence (see section 4.1), the device status (see section 5.1.1) is read with the following command line example:

```
python host_app_ctrl.py  -d /dev/ttyUSB0 GET_DEVICE_STATUS
```

NOTES:

1. For Windows OS replace /dev/ttyUSB0 with the appropriate COMx parameter.
2. Following a power cycle, the first attempt to connect to the rACM may timeout depending on the state of the serial connection. In this instance, it may be necessary to rerun the command a second time.

A.2 Enable Host Network Operations

Following a successful power-up, the following occurs:

- The host defaults to SLEEP_OVERRIDE
- The host-to-node interface defaults to PASS-THROUGH

This configuration allows direct access to the host and node but prevents any host-initiated OTA network operations.

To enable host-initiated OTA network operations while preventing the host from sleeping, the following command lines are used:

```
python ctrl.py -d /dev/ttyUSB0 HOST_OPERATING_MODE 1
python ctrl.py -d /dev/ttyUSB0 HOST_RESET_NODE 0x2D653723
```

NOTE: Preventing the host from sleeping is desirable during host initiation/node configuration and test because it eliminates the need to wake the host with the magnetic Reed switch before issuing commands. Additional power consumption results from the host operating continuously. Care should be taken not to deploy a battery-powered host with SLEEP_OVERRIDE enabled.

A.3 Configure Host Application

The host application can be configured for a number of run-time settings to tailor the rACM platform for different application types, interfaces, and run-time requirements (see Appendix B for some examples). The end-user has different options to consider when configuring the host application:

- Configure each parameter individually, as required, using the *host_app_ctrl.py* script and the list of command options as described in section 5.1. The examples in Appendix B use this approach.
- Configure host application parameters in "one shot" using the SET_FACTORY_CONFIG and SET_FIELD_CONFIG serial command with the aid of a text file that contains all of the necessary key options for defining the rACM host application run-time settings.

Expanding on the latter option, there are two types of "one-shot" configuration utilities:

- **FACTORY_CONFIG:** Configuration settings that can be determined during factory provisioning of the application (e.g., the node type, serial ID, calibration parameters). The rACM default factory configuration data has already been presented in section 7.12. The factory configuration data can be set through the use of the SET_FACTORY_CONFIG host serial command using the *host_specific/host_app_factory_cfg.txt* file as its command line argument. The factory configuration text file contains key file descriptors for each element of the factory configuration data in rACM NVM. Each descriptor element in said file is commented and lists all of the possible options/ranges.
- **FIELD_CONFIG:** Field Configuration is made up of all of the key-value pairs (see Table 3) and supports OTA as well as local serial port updates. All of the field configuration pairs can be set through use of the SET_FIELD_CONFIG host serial command using the *host_specific/host_app_field_cfg.txt* file as its command line argument. The field configuration tests contain key file descriptors for each key-value pair in rACM NVM. Each descriptor element in said file is commented and lists all of the possible options/ranges. Alternatively, the *host_specific/host_app_field_cfg_short.txt* file can be used. All of the comments have been removed to quickly identify and configure each key-value pair descriptor.

As an example usage scenario, we need to change the following rACM defaults for a desired mode of operation:

- The rACM platform under test is populated with the dNode option and has been issued with a serial number ID of 16.
- The rACM platform under test is battery powered and needs to support the SLEEP application profile (although it can remain unsecured).
- APP_INTF1 is unused. Its data will not be reported OTA.
- APP_INTF2 is used as a digital alarm input. Not only is its input state reported at the pre-UI, it also needs to be configured as an asynchronous alarm when its input state goes from low-to-high.
- Otherwise, all of the other rACM run-time configurations can remain at their default settings.

The following configuration sequence would be followed:

1. The rACM under test would be placed into a known shutdown state using the following command:

   ```
   python host_app_ctrl.py –d /dev/ttyUSB0 RESET_DEVICE
   ```

2. Using a visual editor, the following key file elements are set in *host_specific/host_app_factory_cfg.txt*:

   ```
   NODE_TYPE = 2

   DEVICE_ID = 16
   ```

3. The updated factory configuration is set on the rACM host application using the following command from the *python_tools* subdirectory:

   ```
   python host_app_ctrl.py –d /dev/ttyUSB0 SET_FACTORY_CONFIG
   ../host_specific/host_app_factory_cfg.txt
   ```

4. The updated factory configuration can be confirmed by using the following command:

   python host_app_ctrl.py –d /dev/ttyUSB0 GET_FACTORY_CONFIG

5. Using a visual editor, the following key file elements are set in *host_specific/host_app_field_cfg.txt*:

   ```
   APPLICATION_PROFILE = UNSECURED_SLEEPY

   APP_INTF1_OTA_DATA_REPORTING = OFF

   APP_INTF2_MODE = DIG_INPUT

   APP_INTF2_OTA_DATA_REPORTING = ON

   DIG_INPUT2_EDGE_INT = INT_ON

   DIG_INPUT2_ALARM_TYPE = ALARM_HIGH
   ```

6. The updated field configuration set on the rACM host application using the following command from the *python_tools* subdirectory:

   ```
   python host_app_ctrl.py –d /dev/ttyUSB0 SET_FIELD_CONFIG
   ../host_specific/host_app_field_cfg.txt
   ```

7. The updated field configuration can be confirmed using the following command:

   ```
   python host_app_ctrl.py –d /dev/ttyUSB0 GET_DEVICE_STATUS
   ```

8. Reset the rACM platform. The rACM platform will apply the new factory/field configuration parameters as it comes out of reset.

A.4 Configure AP List and Join the Network

Node Monitor is a tool with a graphical user interface that is used to input the AP channel, re-use code, and monitor the rACM as it joins a network AP.

1. To start up the Node Monitor, type the following command at the command line:

   ```
   python ulp_node_monitor.py  -d /dev/ttyUSB0
   ```

2. Click the ***Connect Test PC to eNode*** button located in the center of the screen under the ***eNode Communication*** tab.
3. Change ***Logger Control*** (located at the top and center of the screen) from "unknown**"** to **"**frame stats verbose**.**"
4. Open the ***Flash Config Params*** page by selecting the tab (bottom center).
5. Click the ***Read Configuration from Flash*** button (top center).
6. Verify/edit the following fields:
   - **System ID:** Verify that this value matches the value displayed on the EMS Gateway page.
   - **Channel & Reuse Code:** Start at Candidate #0, fill in both the channel and re-use code for the deployed AP, and leave the unused channels set to 255.
7. Click the ***Write Configuration to Flash*** button (top center) and confirm that the response, "Config File written to flash," is written to the log window (top).
8. Open the ***ORW Graphs*** page by selecting the tab (bottom left).
9. Force a rejoin by changing the ***Over the Air Link*** (upper left) to "OFF" and then to "ON**.**"
10. Verify ***System State*** (upper center) transitions from "SCANNING" to "TRACK" to "JOINED" over the next few minutes. During the process, the following occurs:
    - Ten green bars will briefly be seen in the "Finger Energy" and "Finger Passing AFCs" graphs
    - The "RSSI" and "Spreading Factor" graphs will update accordingly
11. To close the Node Monitor, click on the window frame "X."

    NOTE: ***DO NOT*** use the ***Disconnect Test PC from eNode*** button on the ***Node Communication*** page to exit the Node Monitor. Disconnecting the Test PC from the Node can disable the host-to-node interface leaving the rACM in a non-operational state.

Node Monitor can be used at any time to monitor the OTA network status of the rACM. If the rACM host is not operating in the SLEEP_OVERRIDE mode, it may be necessary to wake the host using the magnetic Reed switch before modifying fields using the Node Monitor.

A.5 Host Logger

The host application logger (see section 5.2) parses the host-to-node traffic and is invoked using the following command line:

```
python –u host_app_logger.py  -d /dev/ttyUSB0 | tee racm_host_log.txt
```

NOTES:

1. The –u option and the UNIX "tee" command echo the logger output to the screen and to the specified file.

2. Depending on the state of the host, several minutes can elapse before any status is logged. A quick test of both the logger and the state of the rACM can be run by doing the following:

   a. Swipe the magnetic wand (see section 4.3) across the Reed switch

   b. Verify that both the logger (and later the EMS/On-Ramp Total Reach web pages; see section 4.4), log the resulting OTA alarm.

   Alarm hysteresis limits the rate of back-to-back OTA alarm generation.

A.6 Useful Commands

Additional command line examples are listed in the following table.

Table 25. Useful Commands

Command	Description
python host_app_ctrl.py -d /dev/ttyUSB0 HOST_VERSION	Get host version
python host_app_ctrl.py -d /dev/ttyUSB0 RESET_DEVICE	Reset rACM to factory defaults
python host_app_ctrl.py -d /dev/ttyUSB0 SET_UART_TIMEOUT 30	Set Wand Timeout to 30 seconds
python host_app_ctrl.py -d /dev/ttyUSB0 SET_READ_INTERVAL READ_BLOCK1 15MIN ASYNC ASYNC	Set Read Interval to 15 minutes (enum 0) starting when the rACM joins the network (async)
python host_app_ctrl.py -d /dev/ttyUSB0 SET_READ_INTERVAL READ_BLOCK1 1HR 45 0	Set Read Interval to 1 hour (enum 2) at 45 minutes past the hour
python host_app_ctrl.py -d /dev/ttyUSB0 SLEEP_OVERRIDE OFF	Disable SLEEP_OVERRIDE and put host into normal sleep mode
python sw_upgrade.py -d /dev/ttyUSB0 micronode.bin	Upgrade microNode Firmware
python host_app_ctrl.py -d /dev/ttyUSB0 HOST_SW_UPGRADE host_app_k20.bin	Upgrade Host Firmware
python host_app_ctrl.py -d /dev/ttyUSB0 DUMP_EXCEPTIONS	Dump exception buffer
Python host_app_ctrl.py –d /dev/ttyUSB0 GET_FACTORY_CONFIG	Dump factory configuration parameters

Appendix B Application Configuration Examples

This chapter provides example configurations settings for hypothetical applications that can leverage the rACM application interface configuration framework (without any firmware changes). The application examples included in this appendix are:

- A battery-powered Flow Meter with a Tamper Detect alarm mechanism.
- A battery-powered Voltage Test Point sensor, which monitors, reports, and generates alarms for two separate analog 0-3V inputs.
- A powered Thermopile Sensor, which monitors and generates alarm for a single analog 0-3V input (representing temperature). In addition, it supports 5 relay outputs that the backend can toggle on-demand.

All of the provided examples will list the necessary key-value configuration steps using the Host Interface serial commands (see section 9.1). It is important to note that all of these configurations can ALSO be set by the backend through use of the Configuration Write Message (see section 9.2.3) through use of the appropriately set key-value pair.

NOTE: These are example scenarios - it is assumed that all of the hypothetical interface connections have been worked out by an application engineer and are electrically "sound."

B.1 Battery-Powered Flow Meter Example

In this example, the rACM reference platform needs to interface with a Flow Meter sensor with the following high-level requirements:

- The flow meter primary unit of measurement is "pulse-counts" generated by a Hall Effect relay switch which represents flow volume (e.g, 1 count = cubic meter)
- The flow meter and rACM platform are located within an enclosure. When the enclosure lid is closed, a tamper-detect reed switch will be to a digital high output (3.3V). When the enclosure lid is open, the reed switch will go back to a digital low (0V).
- The rACM is required to take 8-daily measurements per day in 3 hour intervals starting at 12AM GMT. Because the device is battery powered, the Node's Uplink Interval is once-per-day.
- The rACM is required generate an alarm to the backend should the enclosure lid be opened.

To realize these requirements, the first step is determining which application interfaces to use for the Hall Effect switch and for the Tamper Detect input. Referring to Table 4, we assign the following:

- APP_INTF1 is the only I/O capable of supporting the Kinetis Low-Power Timer block which is ideally suited as a pulse-counter. FUNC_MODE0 must be selected for this interface for the application to leverage the necessary hardware/software resources.

- APP_INTF2 is selected for the Tamper Detect input. Because the device is battery-powered, we can confirm that the edge-trigger interrupt used to detect input state changes is deep sleep capable.
- APP_INTF3 – APP_INTF5 are unused. These can be left as passive digital inputs. Alternatively, the port can be effectively disable be setting it to one of the non-mapped functional modes. In either case, the input should be left “floating”.

The second step is to determine which interfaces to enable OTA reporting for:

- From the stated requirements, it is obvious that all flow meter pulse count reads need to be reported to the backend. We ensure that OTA reporting for APP_INTF1 is ENABLED.
- Even though the Tamper Detect is an asynchronous alarm, it is desired that we report the digital input state of the enclosure when sending the buffered read data for the pulse count. As a result, we ensure that OTA reporting for APP_INTF2 is ENABLED.
- APP_INTF3 – APP_INTF5 are unused and, as a result OTA reporting for these interfaces is DISABLED.

Next, the read interval blocks are configured and mapped to the applicable interfaces:

- From the stated requirements, we configure READ_BLOCK1 for the 3-Hour intervals synchronized at 0:00GMT.
- READ_BLOCK2 is not used. We ensure that block is disabled by setting its interval to NONE.
- APP_INTF1, the pulse counter, is the only interface that will read the current count and save the result to NVM at the desired 3-hour interval. We make sure that it is mapped to READ_BLOCK1.
- APP_INTF2 – APP_INTF5 do not required buffered reads. We make sure that they are all mapped to NONE.

Poll interval blocks are not used for the stated requirements. The Tamper Detect is capable of generating alarms asynchronously, so there is no need to poll this interface. To make sure no processing overhead is spent doing polls, we ensure the following:

- POLL_BLOCK1 – POLL_BLOCK3 are not used. They are all set to NONE.
- APP_INTF1 – APP_INTF5 do not use polling. Poll mapping for all of these interfaces are set to NONE.

Referring to the interface configuration framework in section 7.11, we turn our attention to the configuration methods that are specific to the interface type. The Low-Power Timer does not require any additional configuration parameters for it to operate as a pulse counter. However, the other application interfaces, configured as digital inputs, will require the following:

- APP_INTF2, from the stated requirements, is required to generate an interrupt when the line state changes from high → low. However, internal pull is not used, so we keep it disabled.
- APP_INTF3 – APP_INTF5 are passive. Interrupts, alarm level, and internal pull are all kept disabled.

The final step is to configure the application profile:

- From the stated requirements we are battery-operated and there are no stated security issues - we set the application profile to UNSECURED_SLEEPY.

To summarize, the following Host Interface commands are used to set the configuration parameters discussed in this example:

Table 26. Example Flow Meter Configuration Commands

Command	Description
python host_app_ctrl.py -d COM1 SET_APP_INTF_MODE APP_INTF1 FUNC_MODE0	Set APP_INTF1 to use the LPT_BLOCK
python host_app_ctrl.py -d COM1 SET_APP_INTF_MODE APP_INTF2 DIG_INPUT	Set APP_INTF2 to DIG_INPUT
python host_app_ctrl.py -d COM1 SET_APP_INTF_MODE APP_INTF3 DIG_INPUT	Set APP_INTF3 to DIG_INPUT (repeat for APP_INTF4-5)
python host_app_ctrl.py -d COM1 SET_APP_INTF_OTA_DATA_REPORT APP_INTF1 ON	Enable OTA reporting on APP_INTF1
python host_app_ctrl.py -d COM1 SET_APP_INTF_OTA_DATA_REPORT APP_INTF2 OFF	Disable OTA reporting on APP_INTF2 (repeat for APP_INTF3-8)
python host_app_ctrl.py -d COM1 SET_READ_INTERVAL READ_BLOCK1 3HR 0 0	Set READ_BLOCK1 for 3-hour read intervals synchronized from 0:00 GMT
python host_app_ctrl.py -d COM1 SET_READ_INTERVAL READ_BLOCK2 NONE 0 0	Disable READ_BLOCK2
python host_app_ctrl.py –d COM1 SET_APP_INTF_READ_MAPPING APP_INTF1 READ_BLOCK1	Map APP_INTF1 to use READ_BLOCK1
python host_app_ctrl.py –d COM1 SET_APP_INTF_READ_MAPPING APP_INTF2 NONE	Disable Read Mapping on APP_INTF2 (repeat for APP_INTF3-8)
python host_app_ctrl.py -d COM1 SET_POLL_INTERVAL POLL_BLOCK1 NONE 0 0	Disable POLL_BLOCK1 (repeat for POLL_BLOCK2-3)
python host_app_ctrl.py –d COM1 SET_APP_INTF_POLL_MAPPING APP_INTF1 NONE	Disable Poll Mapping on APP_INTF1 (repeat for APP_INTF2-8)
python host_app_ctrl.py –d COM1 SET_APP_INTF_DIG_IN_CFG APP_INTF2 NONE ALARM_LOW INT_ON	Enable DIG_INPUT interrupts on APP_INTF2 and alarm on a digital LOW state.
python host_app_ctrl.py –d COM1 SET_APP_INTF_DIG_IN_CFG APP_INTF3 NONE DISABLED INT_OFF	Disable interrupts/alarms on APP_INTF3 (repeat for APP_INTF4-5)
python host_app_ctrl.py –d COM1 SET_APPLICATION_PROFILE UNSECURED_SLEEPY	Set the application profile

B.2 Battery-Powered Voltage Test Point Example

In this example, the rACM reference platform needs to monitor two analog 0-3V input voltages with the following high-level requirements:

- The primary analog input must be able to read on 1-hour increments synchronized at the top of the hour and report measured data OTA at the 24-Hour UI. In addition, if any read exceeds 2.5V, it will also generate an asynchronous alarm that must be sent immediately.

- The secondary analog input most be polled every 5 minutes, but the measurements do not need to be buffered. However, if the secondary analog input indicates a measurement that is outside a nominal operating parameter of 0.75V to 1.25 V, then it too will generate an asynchronous alarm to the backend.
- Both analog input measurements require 10 samples averaged across a 167msec window to help filter out a 60 Hz analog distortion present on the inputs.

To realize these requirements, the first step is determining which application interfaces to use for the analog inputs – referring to Table 4, we assign the following:

- APP_INTF6, the dedicated single-ended ADC input, is chosen as the primary analog input
- APP_INTF5 is also capable of being configured as a single-ended ADC input and we choose this interface as the secondary analog input. FUNC_MODE0 must be selected for this interface for the application to leverage the necessary hardware/software resources.
- APP_INTF1 – APP_INTF4 are unused – these can be left as passive digital inputs. Alternatively, the port can be effectively disable be setting it to one of the non-mapped functional modes. In either case, the input should be left “floating”.

The second step is to determine which interfaces to enable OTA reporting for:

- From the stated requirements, we must report all buffered voltage reads on the primary analog input to the backend. We ensure that OTA reporting for APP_INTF6 is ENABLED.
- Even though the secondary analog input does not buffer any voltage testpoint reads, it is desired that we report the most ADC analog read when sending the buffered read data for the primary analog input. As a result, we ensure that OTA reporting for APP_INTF5 is also ENABLED.
- APP_INTF1 – APP_INTF4 are unused and, as a result OTA reporting for these interfaces is DISABLED.

Next, the read interval blocks are configured and mapped to the applicable interfaces:

- From the stated requirements, we configure READ_BLOCK1 for the 1-Hour read intervals synchronized at 0:00GMT.
- READ_BLOCK2 is not used – we ensure that block is disabled by setting its interval to NONE.
- APP_INTF6, the primary analog input, is the only interface that will read and buffer measurement data into NVM at the desired 1-hour interval. We make sure that it is mapped to READ_BLOCK1.
- APP_INTF1 – APP_INTF5 do not required buffered reads. We make sure that they are all mapped to NONE.

Next, the poll interval blocks are configured and mapped to the applicable interfaces:

- From the stated requirements, we configure POLL_BLOCK1 for the 5-minute poll intervals. Time synchronization is not needed, so we configure the poll to start ASYNC.
- POLL_BLOCK2 – POLL_BLOCK3 are not used –they are both set to NONE.
- APP_INTF5, the secondary analog input, is the only interface that will poll at the desired 5-minute interval. We make sure that it is mapped to POLL_BLOCK1.

- APP_INTF1 – APP_INTF4 and APP_INTF6 do not use polling – poll mapping for all of these interfaces are set to NONE.

Referring to the interface configuration framework in section 7.11, we turn our attention to the configuration methods that are specific to the interface type. The two analog interfaces must be configured for sampling rate and alarm threshold. The remaining interfaces are configured as passive digital inputs:

- APP_INTF1 – APP_INTF4 are passive digital inputs – interrupts, alarm level, and internal pull are all kept disabled.
- Per stated requirements for the primary analog interface, the ADC_BLOCK 1is configured to measure 10 subsamples across a 167msec window.
- Per stated requirements for the primary analog interface, the ADC_BLOCK1 is configured to generate to a HIGH_THRESH alarm type. The Upper_Hi threshold is set to 2500mV and the Upper_Lo threshold is set to 2200mV (when the alarm condition is cleared). The Lower_Hi/Lo thresholds are unused in this alarm type and are set to zero.
- Per stated requirements for the secondary analog interface, the ADC_BLOCK2 is configured to also measure 10 subsamples across a 167msec window.
- Per stated requirements for the secondary analog interface, the ADC_BLOCK2 is configured to generate a OUT_OF_BOUNDS alarm type, the Upper_Hi/Lo thresholds are set to 1250mV/1200mV and the Lower_Hi/Lo thresholds are set to 800mV/750mV.

The final step is to configure the application profile:

- From the stated requirements we are battery-operated and there are no stated security issues - we set the application profile to UNSECURED_SLEEPY.

To summarize, the following Host Interface commands are used to set the configuration parameters discussed in this example:

Table 27. Example Voltage Testpoint Configuration Commands

Command	Description
python host_app_ctrl.py -d COM1 SET_APP_INTF_MODE APP_INTF1 DIG_INPUT	Set APP_INTF1 to DIG_INPUT (repeat for APP_INTF2-4)
python host_app_ctrl.py -d COM1 SET_APP_INTF_MODE APP_INTF5 FUNC_MODE0	Set APP_INTF5 to use the ADC2_BLOCK
python host_app_ctrl.py -d COM1 SET_APP_INTF_OTA_DATA_REPORT APP_INTF6 ON	Enable OTA reporting on APP_INTF6
python host_app_ctrl.py -d COM1 SET_APP_INTF_OTA_DATA_REPORT APP_INTF5 ON	Enable OTA reporting on APP_INTF5
python host_app_ctrl.py -d COM1 SET_APP_INTF_OTA_DATA_REPORT APP_INTF1 OFF	Disable OTA reporting on APP_INTF1 (repeat for APP_INTF2-4 and APP_INTF7-8)
python host_app_ctrl.py -d COM1 SET_READ_INTERVAL READ_BLOCK1 1HR 0 0	Set READ_BLOCK1 for 1-hour read intervals synchronized from 0:00 GMT
python host_app_ctrl.py -d COM1 SET_READ_INTERVAL READ_BLOCK2 NONE 0 0	Disable READ_BLOCK2

Command	Description
python host_app_ctrl.py –d COM1 SET_APP_INTF_READ_MAPPING APP_INTF6 READ_BLOCK1	Map APP_INTF6 to use READ_BLOCK1
python host_app_ctrl.py –d COM1 SET_APP_INTF_READ_MAPPING APP_INTF1 NONE	Disable Read Mapping on APP_INTF1 (repeat for APP_INTF2-5 and APP_INTF7-8)
python host_app_ctrl.py -d COM1 SET_POLL_INTERVAL POLL_BLOCK1 5MIN ASYNC ASYNC	Set POLL_BLOCK1 for unsynchronized, 5-minute poll intervals
python host_app_ctrl.py -d COM1 SET_POLL_INTERVAL POLL_BLOCK2 NONE 0 0	Disable POLL_BLOCK2 (repeat for POLL_BLOCK3)
python host_app_ctrl.py –d COM1 SET_APP_INTF_POLL_MAPPING APP_INTF5 POLL_BLOCK1	Map APP_INTF5 to use POLL_BLOCK1
python host_app_ctrl.py –d COM1 SET_APP_INTF_POLL_MAPPING APP_INTF1 NONE	Disable Poll Mapping on APP_INTF1 (repeat for APP_INTF1-4 and APP_INTF6-8)
python host_app_ctrl.py –d COM1 SET_APP_INTF_DIG_IN_CFG APP_INTF1 NONE DISABLED INT_OFF	Disable interrupts/alarms on APP_INTF1 (repeat for APP_INTF2-4)
python host_app_ctrl.py –d COM1 SET_ADC_SAMPLE_CALC_PARAMS ADC_BLOCK1 167 10	Set the ADC_BLOCK1 sampling parameters
python host_app_ctrl.py –d COM1 SET_ADC_ALARM_TYPE ADC_BLOCK1 HIGH_THRESH	Set the ADC_BLOCK1 alarm type to HIGH_THRESH
python host_app_ctrl.py –d COM1 SET_ADC_ALARM_UPPER_THRESH ADC_BLOCK1 2500 2200	Set the ADC_BLOCK1 upper alarm thresholds.
python host_app_ctrl.py –d COM1 SET_ADC_ALARM_LOWER_THRESH ADC_BLOCK1 0 0	Set the ADC_BLOCK1 lower alarm thresholds.
python host_app_ctrl.py –d COM1 SET_ADC_SAMPLE_CALC_PARAMS ADC_BLOCK2 167 10	Set the ADC_BLOCK2 sampling parameters
python host_app_ctrl.py –d COM1 SET_ADC_ALARM_TYPE ADC_BLOCK2 OUT_OF_BOUNDS	Set the ADC_BLOCK2 alarm type to OUT_OF_BOUNDS
python host_app_ctrl.py –d COM1 SET_ADC_ALARM_UPPER_THRESH ADC_BLOCK2 1250 1200	Set the ADC_BLOCK2 upper alarm thresholds.
python host_app_ctrl.py –d COM1 SET_ADC_ALARM_LOWER_THRESH ADC_BLOCK2 800 750	Set the ADC_BLOCK2 lower alarm thresholds.
python host_app_ctrl.py –d COM1 SET_APPLICATION_PROFILE UNSECURED_SLEEPY	Set the application profile

B.3 Powered Thermopile Sensor/Remote Relay Example

In this example, the rACM reference platform needs to act as a powered remote monitoring platform that polls ambient temperature through use of an analog 0-3V input (which the back-end converts to temperature). In addition, the rACM unit needs to be able, on command from the backend, toggle up to 5 relay outputs used to turn on/off remote power.

- The primary analog input must be read on 1-hour increments synchronized at the top of the hour and report measured data OTA at the 24-Hour UI. In addition, if any read exceeds 2V, it will also generate an asynchronous alarm that must be sent immediately.
- In addition to 1-hour reads, must be polled every 5 minutes - but these measurements do not need to be buffered. However, the same alarm generation scheme must also be used for these accelerate measurements.
- All five of the configurable application interfaces must be configured as digital outputs and held low. An operator at the backend can initiate the Relay Toggle Request (see section 9.2.3), to toggle a pulse on the specified digital output to turn on/off remote equipment (usually in response to the asynchronous alarm generated by the sensor).
- The analog input measurements require 10 samples averaged across a 167msec window to help filter out a 60Hz analog distortion present on the input.

To realize these requirements, the first step is determining which application interfaces to use for the analog inputs – referring to Table 4, we assign the following:

- APP_INTF6, the dedicated single-ended ADC input, is chosen as the thermopile input
- APP_INTF1 - APP_INTF5 are all configured as digital outputs to implement the specified relay toggle scheme.

The second step is to determine which interfaces to enable OTA reporting for:

- From the stated requirements, we must report all buffered voltage reads on the thermopile sensor input to the backend. We ensure that OTA reporting for APP_INTF6 is ENABLED.
- APP_INTF1 – APP_INTF5, configured as relay toggles, do not need to send their state information to the backend. As a result OTA reporting for these interfaces is DISABLED.

Next, the read interval blocks are configured and mapped to the applicable interfaces:

- From the stated requirements, we configure READ_BLOCK1 for the 1-Hour read intervals synchronized at 0:00GMT.
- READ_BLOCK2 is not used – we ensure that block is disabled by setting its interval to NONE.
- APP_INTF6, the thermopile sensor input, is the only interface that will read and buffer measurement data into NVM at the desired 1-hour interval. We make sure that it is mapped to READ_BLOCK1.
- APP_INTF1 – APP_INTF5 do not required buffered reads. We make sure that they are all mapped to NONE.

Next, the poll interval blocks are configured and mapped to the applicable interfaces:

- From the stated requirements, we configure POLL_BLOCK1 for the 5-minute poll intervals. Time synchronization is not needed, so we configure the poll to start ASYNC.
- POLL_BLOCK2 – POLL_BLOCK3 are not used –they are both set to NONE.
- APP_INTF6, the thermopile sensor input currently mapped to READ_BLOCK1, can concurrently also be mapped to the 5-minute polling interval block. As a result, we make sure that it is mapped to POLL_BLOCK1 as well.

- APP_INTF1 – APP_INTF5 do not use polling – poll mapping for all of these interfaces are set to NONE.

Referring to the interface configuration framework in section 7.11, we turn our attention to the configuration methods that are specific to the interface type. The thermopile sensor interface must be configured for sampling rate and alarm threshold. The remaining interfaces are configured as digital outputs with a high driver strength:

- APP_INTF1 – APP_INTF5 are all digital outputs that are held LOW with DSE_HIGH.
- Per stated requirements for the thermopile sensor interface, the ADC_BLOCK 1is configured to measure 10 subsamples across a 167msec window.
- Per stated requirements for the thermopile sensor interface, the ADC_BLOCK1 is configured to generate to a HIGH_THRESH alarm type. The Upper_Hi threshold is set to 2000mV and the Upper_Lo threshold is set to 1800mV (when the alarm condition is cleared). The Lower_Hi/Lo thresholds are unused in this alarm type and are set to zero.

The final step is to configure the application profile:

- From the stated requirements we are battery-operated and there are no stated security issues - we set the application profile to UNSECURED_POWERED.

To summarize, the following Host Interface commands are used to set the configuration parameters discussed in this example:

Table 28. Example Voltage Testpoint Configuration Commands

Command	Description
python host_app_ctrl.py -d COM1 SET_APP_INTF_MODE APP_INTF1 DIG_OUTPUT	Set APP_INTF1 to DIG_OUTPUT (repeat for APP_INTF2-5)
python host_app_ctrl.py -d COM1 SET_APP_INTF_OTA_DATA_REPORT APP_INTF6 ON	Enable OTA reporting on APP_INTF6
python host_app_ctrl.py -d COM1 SET_APP_INTF_OTA_DATA_REPORT APP_INTF1 OFF	Disable OTA reporting on APP_INTF1 (repeat for APP_INTF2-5 and APP_INTF7-8)
python host_app_ctrl.py -d COM1 SET_READ_INTERVAL READ_BLOCK1 1HR 0 0	Set READ_BLOCK1 for 1-hour read intervals synchronized from 0:00 GMT
python host_app_ctrl.py -d COM1 SET_READ_INTERVAL READ_BLOCK2 NONE 0 0	Disable READ_BLOCK2
python host_app_ctrl.py –d COM1 SET_APP_INTF_READ_MAPPING APP_INTF6 READ_BLOCK1	Map APP_INTF6 to use READ_BLOCK1
python host_app_ctrl.py –d COM1 SET_APP_INTF_READ_MAPPING APP_INTF1 NONE	Disable Read Mapping on APP_INTF1 (repeat for APP_INTF2-5 and APP_INTF7-8)
python host_app_ctrl.py -d COM1 SET_POLL_INTERVAL POLL_BLOCK1 5MIN ASYNC ASYNC	Set POLL_BLOCK1 for unsynchronized, 5-minute poll intervals
python host_app_ctrl.py -d COM1 SET_POLL_INTERVAL POLL_BLOCK2 NONE 0 0	Disable POLL_BLOCK2 (repeat for POLL_BLOCK3)
python host_app_ctrl.py –d COM1 SET_APP_INTF_POLL_MAPPING APP_INTF6 POLL_BLOCK1	Map APP_INTF6 to use POLL_BLOCK1
python host_app_ctrl.py –d COM1 SET_APP_INTF_POLL_MAPPING APP_INTF1 NONE	Disable Poll Mapping on APP_INTF1 (repeat for APP_INTF1-5 and APP_INTF6-8)

Command	Description
python host_app_ctrl.py –d COM1 SET_APP_INTF_DIG_OUT_CFG APP_INTF1 ODE_OFF DSE_HIGH OUTPUT_LOW	Set APP_INTF1 digital output for an digital low output with high drive strength
python host_app_ctrl.py –d COM1 SET_ADC_SAMPLE_CALC_PARAMS ADC_BLOCK1 167 10	Set the ADC_BLOCK1 sampling parameters
python host_app_ctrl.py –d COM1 SET_ADC_ALARM_TYPE ADC_BLOCK1 HIGH_THRESH	Set the ADC_BLOCK1 alarm type to HIGH_THRESH
python host_app_ctrl.py –d COM1 SET_ADC_ALARM_UPPER_THRESH ADC_BLOCK1 2000 1800	Set the ADC_BLOCK1 upper alarm thresholds.
python host_app_ctrl.py –d COM1 SET_ADC_ALARM_LOWER_THRESH ADC_BLOCK1 0 0	Set the ADC_BLOCK1 lower alarm thresholds.
python host_app_ctrl.py –d COM1 SET_APPLICATION_PROFILE UNSECURED_POWERED	Set the application profile

Appendix C OTA Payload Examples

The chapter is provided as a guide to parsing Uplink OTA payload packets as they are received at the backend. The payload examples included in this appendix are:

- Configuration Write (Read Interval)
- Configuration Response (auto-generated by Network Discovery Connect feature)
- Sensor Data History payload for an example Pulse Counter.
- Sensor Data History payload for an example CPMon Cathodic Testpoint.
- Sensor Data History payload for an example CPMon Current Loop Testpoint.
- Sensor Data History payload for an example CPMon Cathodic Rectifier.
- Asynchronous Alarm payload

NOTES:

1. The Sensor Data History packs message payload fields using Little-Endian byte ordering (i.e., least significant byte first). This is an especially important consideration for multi-byte bit packed fields (e.g., the bit-packed time stamp).
2. All OTA packets used in this example have been captured by an operational rACM unit joined to an On-Ramp Wireless field-engineering test network. Using the On-Ramp Total View application, the raw hex data has been captured for each of the use-cases.
3. For On-Ramp Wireless System Release 1.4, all OTA packets are padded to an 8-byte boundary. Padded bytes are conveniently discarded/ignored in the following parsing examples presented in this appendix.

C.1 Configuration Write Example

In this example, a rACM unit was powered cycled. After the acquiring/joining the network, the Downlink tab in the On-Ramp Total View application is used to generate a Configuration Write OTA packet to change the Read Interval (ORWID 7, see Table 18). The following screen capture from the On-Ramp Total View application provides an example.

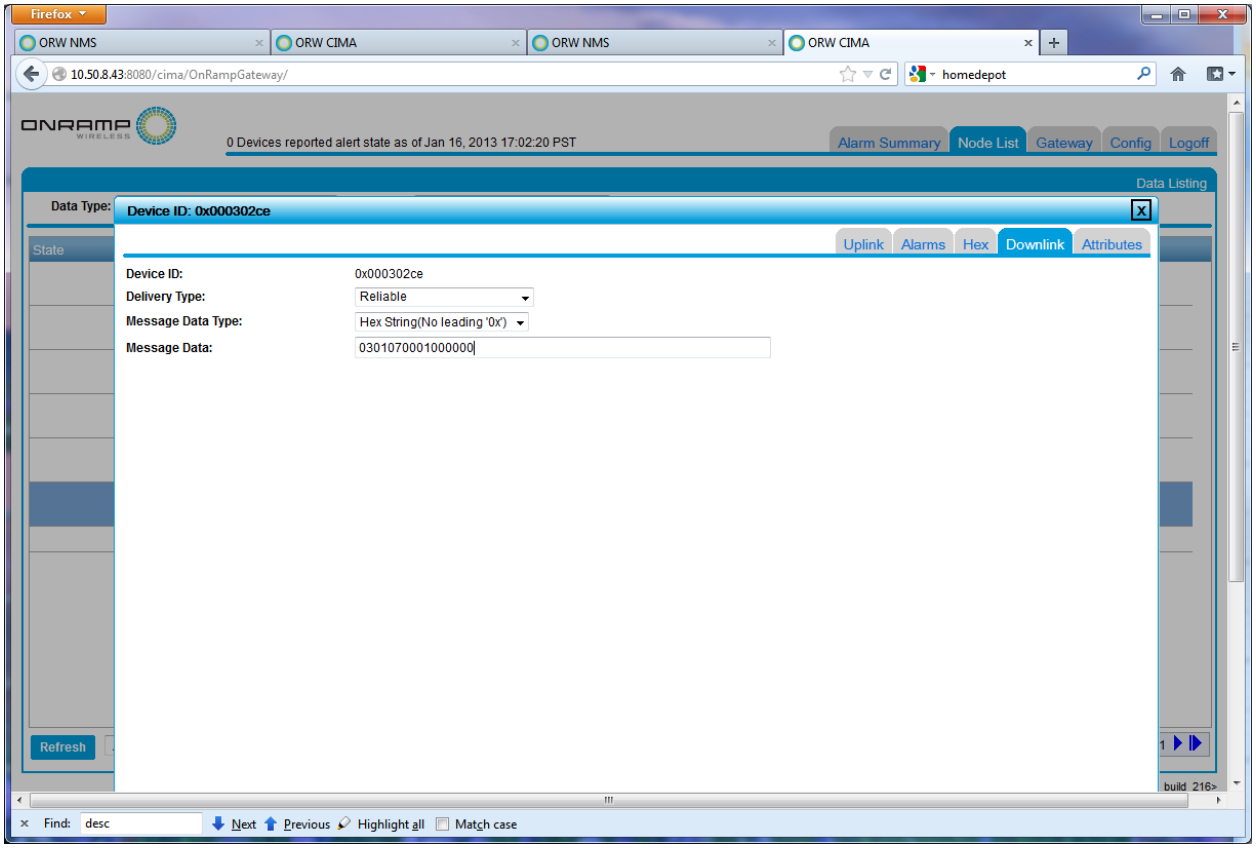
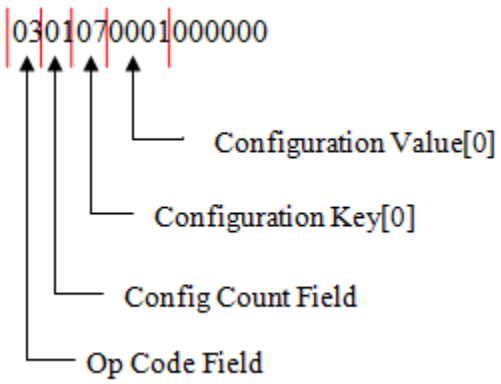


Figure 11. On-Ramp Total View Configuration Write Example

Expanding the highlighted Configuration Write OTA packet, the raw hex data for the Configuration Write OTA payload (see section 9.2.3) is broken down into the following data stream:



Taking into consideration the big-endian byte ordering and the Configuration Write Message format (Table 17 of section 9.2.3), the Configuration Write message can be deconstructed into the following fields:

- **Op Code –** 0x03 (Configuration Write Request DL Message)
- **Config Count –** 0x01 (1 Configuration Key-Value Pair)
- **Config Key[0] –** 0x07 (ORW ID#7: Read Interval)
- **Config Value[0] –** 0x0001 (Reads every 30 minutes with zero offset)

NOTE: Configuration write value fields are entered into the On-Ramp Total View Downlink generation box in big-endian format while Uplink fields use little-endian byte ordering.

C.2 Configuration Response Example

In this example, a rACM unit was powered cycled. After the acquiring/joining the network, the network discovery connect feature auto-generated to the Configuration Response OTA packet as the following screen capture from the On-Ramp Total View application shows:

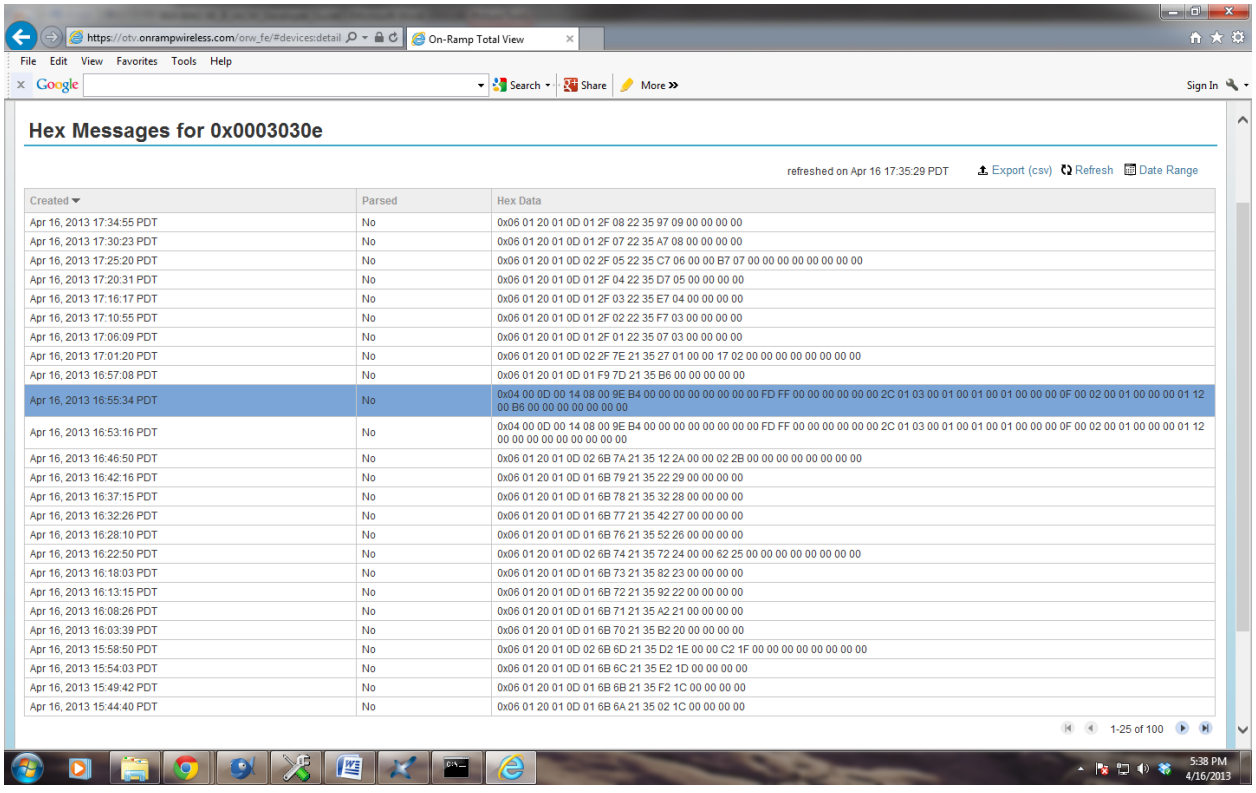
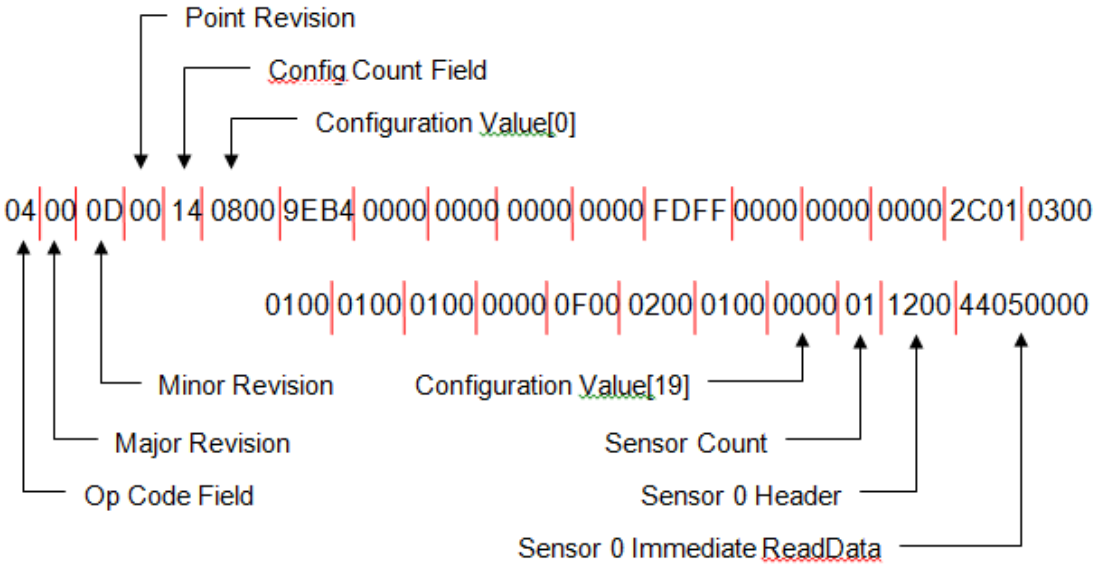


Figure 12. On-Ramp Total View Configuration Response Example

Expanding the highlighted OTA packet, the raw hex data for the Configuration Response OTA payload (see section 9.2.5) is broken down into the following data stream:



Taking into consideration the little-endian byte ordering and the Configuration Response Message format (Table 19 of section 9.2.5), the Configuration Response message can be deconstructed into the following fields:

- **Op Code –** 0x04 (Configuration Read Response UL Message)
- **Major Revision –** 0x00 (from app/version.h)
- **Minor Revision –** 0x0D (from app/version.h)
- **Point Revision –** 0x00 (from app/version.h)
- **Config Count –** 0x14 (20 Configuration Key-Value Pairs)
- **Config Value[0] –** 0x0008 (ORW ID#1: UART Timeout = 8 seconds)
- **Config Value[1] –** 0xB49E (ORW ID#2: UART Passcode[15:0] = 0xB49E)
- **Config Value[2] –** 0x0000 (ORW ID#3: UART Passcode[31:16] = 0x0000)
- **Config Value[3] –** 0x0000 (ORW ID#4: UART Passcode[47:32] = 0x0000)
- **Config Value[4] –** 0x0000 (ORW ID#5: UART Passcode[63:48] = 0x0000)
- **Config Value[5] –** 0x0000 (ORW ID#6: UART Password Disable Threshold = disabled)
- **Config Value[6] –** 0xFFFD (ORW ID#7: Read Interval = Reads every 4 minutes - async)
- **Config Value[7] –** 0x0000 (ORW ID#8: Low Battery Alarm Threshold = disabled)
- **Config Value[8] –** 0x0000 (ORW ID#9: Device ID[15:0] = 0x0000)
- **Config Value[9] –** 0x0000 (ORW ID#10: Device ID[31:16] = 0x0000)
- **Config Value[10] –** 0x012C (ORW ID#11: Alarm Hysteresis = 300 seconds)
- **Config Value[11] –** 0x0003 (ORW ID#12: POR Default Host Interface State = App Override)
- **Config Value[12] –** 0x0001 (ORW ID#13: POR Sleep Override State = On)

- **Config Value[13] –** 0x0001 (ORW ID#14: Network Discovery Connect = On)
- **Config Value[14] –** 0x0001 (ORW ID#15: Provision Quick Start = On)
- **Config Value[15] –** 0x0000 (ORW ID#16: RHT Support = Off)
- **Config Value[16] –** 0x000F (ORW ID#17: Poll Interval = None)
- **Config Value[17] –** 0x0002 (ORW ID#18: External Switch Function = OTA Config Generation)
- **Config Value[18] –** 0x0001 (ORW ID#19: Application Profile = UNSECURED_POWERED)
- **Config Value[19] –** 0x0000 (ORW ID#20: Application Serial Function = NONE)
- **Sensor Count –** 0x0001 (single sensor interface)
- **Sensor 0 Header –** 0x0056 (32-bit pulse absolute count type on App Interface 1)
- **Sensor 0 Immediate Read Data** – 0x00000544 (1346 “counts”)

C.3 Pulse Counter Data Example

In this example, a rACM unit, configured to monitor a pulse counter digital input, was powered cycled and allowed to acquire/join the network. Once joined, the unit entered steady-state operations taking pulse measurements every 15-minutes and generating OTA read records at a 2-hour uplink interval as the following screen capture from the On-Ramp Total View application demonstrates:

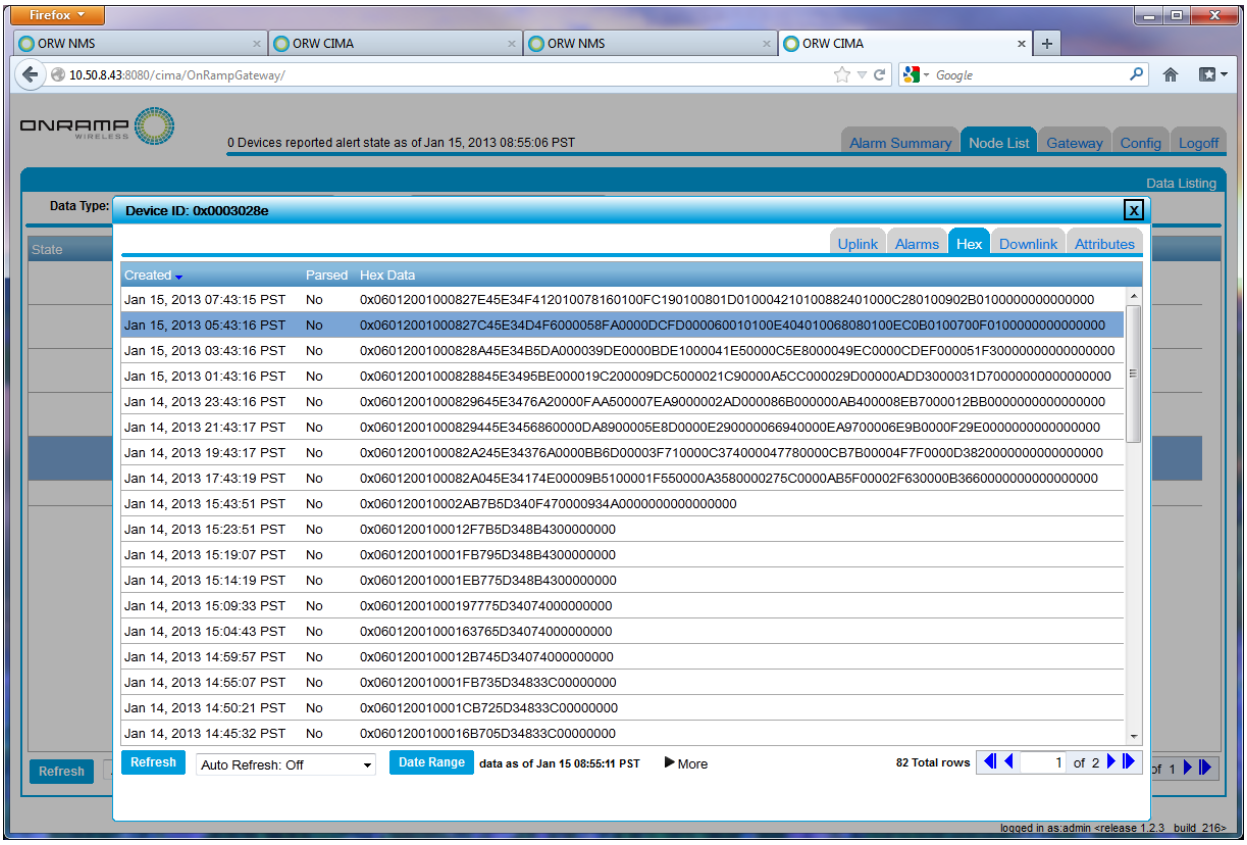
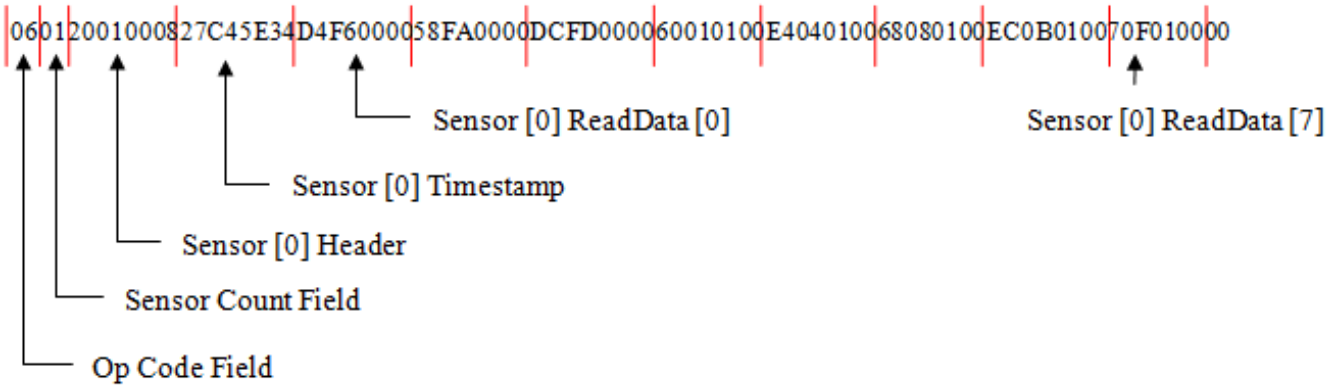


Figure 13. On-Ramp Total View Pulse Count Sensor Data Example

Expanding the highlighted OTA packet, the raw hex data for the Sensor Data OTA payload (see section 9.1) is broken down into the following data stream:



Taking into consideration the little-endian byte ordering and the Sensor Data Message format (Table 21 of section 9.2.7), the Sensor Data OTA payload can be deconstructed into the following fields:

- **Op Code –** 0x06 (Configuration Read Response UL Message)
- **Sensor Count –** 0x01 (A single sensor record)
- **Sensor[0] Header** - 0x08000120 (bit packed Header field)
  - Sensor Interface ID (see Table 22) - 0x0 (Sensor Index)
  - Sensor Interface Type (see Table 23) - 0x012 (32-bit unsigned pulse counts)
  - Read Interval – 0x00 (15-minute reads)
  - Read Count - 0x08 (8 chained read records)
- **Sensor[0] Timestamp** - 0x345EC427 (bit packed GMT timestamp for first read record):
  - Seconds - 0x27 (39 sec)
  - Minutes – 0x10 (16 min)
  - Hour – 0x0C (12 hour)
  - Day – 0x0F (15th day)
  - Month – 0x01 (January)
  - Year Offset from 2000 – 0x0D (2013)
- **Sensor[0] ReadData[0]** - 0x0000F6D4 (63188 pulses)
- **Sensor[0] ReadData[1]** – 0x0000FA58 (64088 pulses)
- **Sensor[0] ReadData[2]** – 0x0000FDDC (64988 pulses)
- **Sensor[0] ReadData[3]** – 0x00010160 (65888 pulses)
- **Sensor[0] ReadData[4]** – 0x000104E4 (66788 pulses)
- **Sensor[0] ReadData[5]** – 0x00010868 (67688 pulses)

- **Sensor[0] ReadData[6]** – 0x00010BEC (68588 pulses)
- **Sensor[0] ReadData[7]** – 0x00010F70 (69488 pulses)

Extrapolating further, the deconstructed sensor data reported to the backend consists of a chained record of 8 pulse count measurements spaced 15-minutes apart starting at Jan 15th, 2013 12:16:39 GMT.

C.4 Voltage Sensor Data Example (CPMon Only)

In this example, a rACM unit with a CPMON daughter card, configured to monitor a Cathodic Testpoint (using rms averaging), was powered cycled and allowed to acquire/join the network. Once joined, the unit entered steady-state operations taking voltage measurements every 30-seconds (test build read rate for a 900mV input) and generating OTA read records at a 5-minute uplink interval as the following screen capture from the On-Ramp Total View application demonstrates:

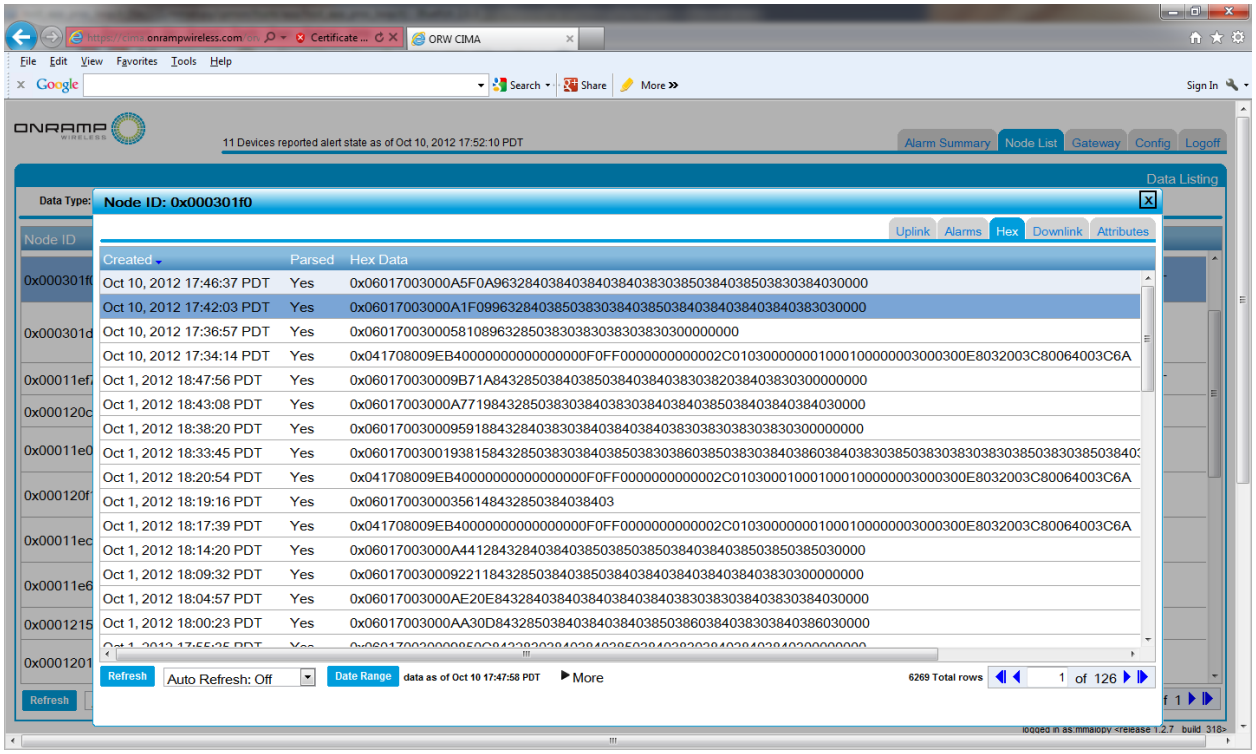
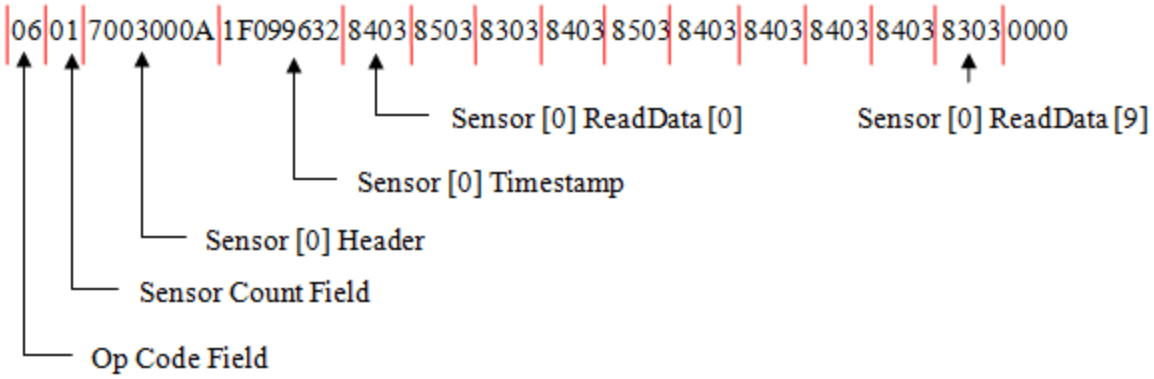


Figure 14. On-Ramp Total View Voltage Sensor Data Example

Expanding the highlighted OTA packet, the raw hex data for the Sensor Data OTA payload (see section 9.1) is broken down into the following data stream:



Taking into consideration the little-endian byte ordering and the Sensor Data Message format (Table 21 of section 9.2.7), the Sensor Data OTA payload can be deconstructed into the following fields:

- **Op Code –** 0x06 (Configuration Read Response UL Message)
- **Sensor Count –** 0x01 (A single sensor record)
- **Sensor[0] Header** - 0x0A000370 (bit packed Header field)
  - Sensor Interface ID (see Table 22) - 0x0 (Cathodic Monitor Voltage Testpoint)
  - Sensor Interface Type (see Table 23) - 0x037 (16-bit unsigned AC Millivolts)
  - Read Interval – 0x00 (30-seconds for test build – for normal builds, 15-minute reads)
  - Read Count - 0x0A (10 chained read records)
- **Sensor[0] Timestamp** - 0x3296091F (bit packed GMT timestamp for first read record):
  - Seconds - 0x1F (31 sec)
  - Minutes – 0x24 (36 min)
  - Hour – 0x00 (0 hour)
  - Day – 0x0B (11th day)
  - Month – 0xA (October)
  - Year Offset from 2000 – 0x0C (2012)
- **Sensor[0] ReadData[0]** - 0x0384 (900mV rms)
- **Sensor[0] ReadData[1]** – 0x0385 (901mV rms)
- **Sensor[0] ReadData[2]** – 0x0383 (899mV rms)
- **Sensor[0] ReadData[3]** – 0x0384 (900mV rms)
- **Sensor[0] ReadData[4]** – 0x0385 (901mV rms)
- **Sensor[0] ReadData[5]** – 0x0384 (900mV rms)
- **Sensor[0] ReadData[6]** – 0x0384 (900mV rms)
- **Sensor[0] ReadData[7]** – 0x0384 (900mV rms)

- **Sensor[0] ReadData[8]** – 0x0384 (900mV rms)
- **Sensor[0] ReadData[9]** – 0x0383 (899mV rms)

Extrapolating further, the deconstructed sensor data reported to the backend consists of a chained record of 10 Vrms measurements spaced 30-seconds apart (15 minutes on a deployed unit) starting at Oct 11th, 2012 00:36:31GMT.

C.5 Current Loop Sensor Data Example (CPMon Only)

In this example, a rACM unit with a CPMON daughter card, configured to monitor a 4-20mA Current Loop testpoint, was powered cycled and allowed to acquire/join the network. Once joined, the unit entered steady-state operations taking current measurements every 30-seconds (test build read rate for a 12mA input) and generating OTA read records at a 5-minute uplink interval as the following screen capture from the On-Ramp Total View application shows:

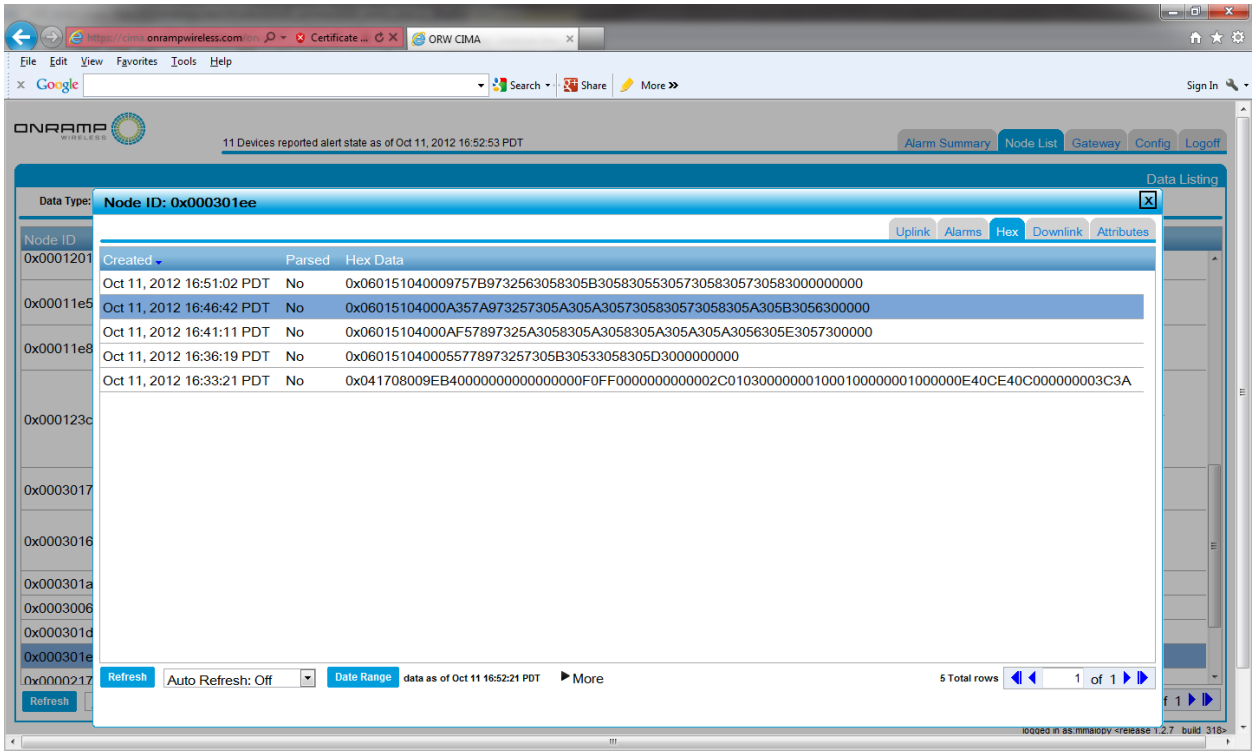
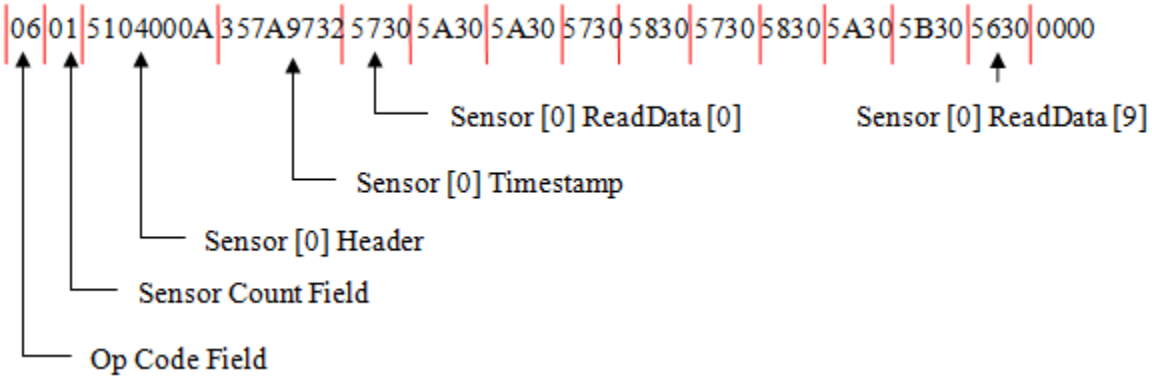


Figure 15. On-Ramp Total View Current Loop Data Example

Expanding the highlighted OTA packet, the raw hex data for the Sensor Data OTA payload (see section 9.1) is broken down into the following data stream:



Taking into consideration the little-endian byte ordering and the Sensor Data Message format (Table 21 of section 9.2.7), the Sensor Data OTA payload can be deconstructed into the following fields:

- **Op Code –** 0x06 (Configuration Read Response UL Message)
- **Sensor Count –** 0x01 (A single sensor record)
- **Sensor[0] Header** - 0x0A000451 (bit packed Header field)
  - Sensor Interface ID (see Table 22) - 0x1 (Cathodic Monitor Current Loop Testpoint)
  - Sensor Interface Type (see Table 23) - 0x045 (16-bit signed Micro-Amps)
  - Read Interval – 0x00 (30-seconds for test build – for normal builds, 15-minute reads)
  - Read Count - 0x0A (10 chained read records)
- **Sensor[0] Timestamp** - 0x32977A35 (bit packed GMT timestamp for first read record):
  - Seconds - 0x35 (53 sec)
  - Minutes – 0x28 (40 min)
  - Hour – 0x17 (23 hour)
  - Day – 0x0B (11th day)
  - Month – 0xA (October)
  - Year Offset from 2000 – 0x0C (2012)
- **Sensor[0] ReadData[0]** - 0x3057 (12375uA)
- **Sensor[0] ReadData[1]** – 0x305A (12379uA)
- **Sensor[0] ReadData[2]** – 0x305A (12379uA)
- **Sensor[0] ReadData[3]** – 0x3057 (12375uA)
- **Sensor[0] ReadData[4]** – 0x3058 (12376uA)
- **Sensor[0] ReadData[5]** – 0x3057 (12375uA)
- **Sensor[0] ReadData[6]** – 0x3058 (12376uA)
- **Sensor[0] ReadData[7]** – 0x305A (12379uA)

- **Sensor[0] ReadData[8]** – 0x305B (12380uA)
- **Sensor[0] ReadData[9]** – 0x3056 (12374uA)

Extrapolating further, the deconstructed sensor data reported to the backend consists of a chained record of 10 Vrms measurements spaced 30-seconds apart (15 minutes on a deployed unit) starting at Oct 11th, 2012 23:28:35GMT.

C.6 Cathodic Rectifier Sensor Data Example (CPMon Only)

In this example, a rACM unit with a CPMON daughter card, configured as a Cathodic Rectifier monitoring device, was powered cycled and allowed to acquire/join the network. Once joined, the unit entered steady-state operations taking voltage measurements every 30-seconds on both the primary analog input (monitoring a 20mVrms input) and the secondary rectifier shunt input (a 3mVrms input). OTA read records are generated for a 5-minute uplink interval as the following screen capture from the On-Ramp Total View application demonstrates:

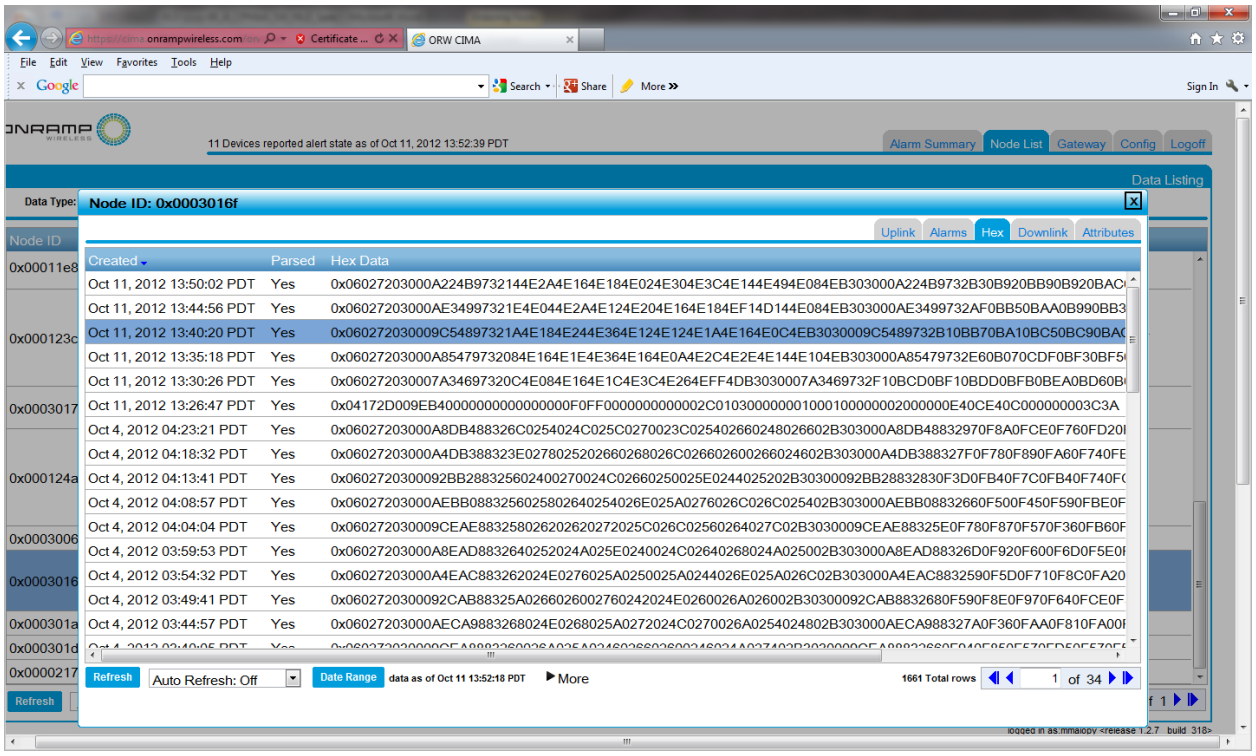
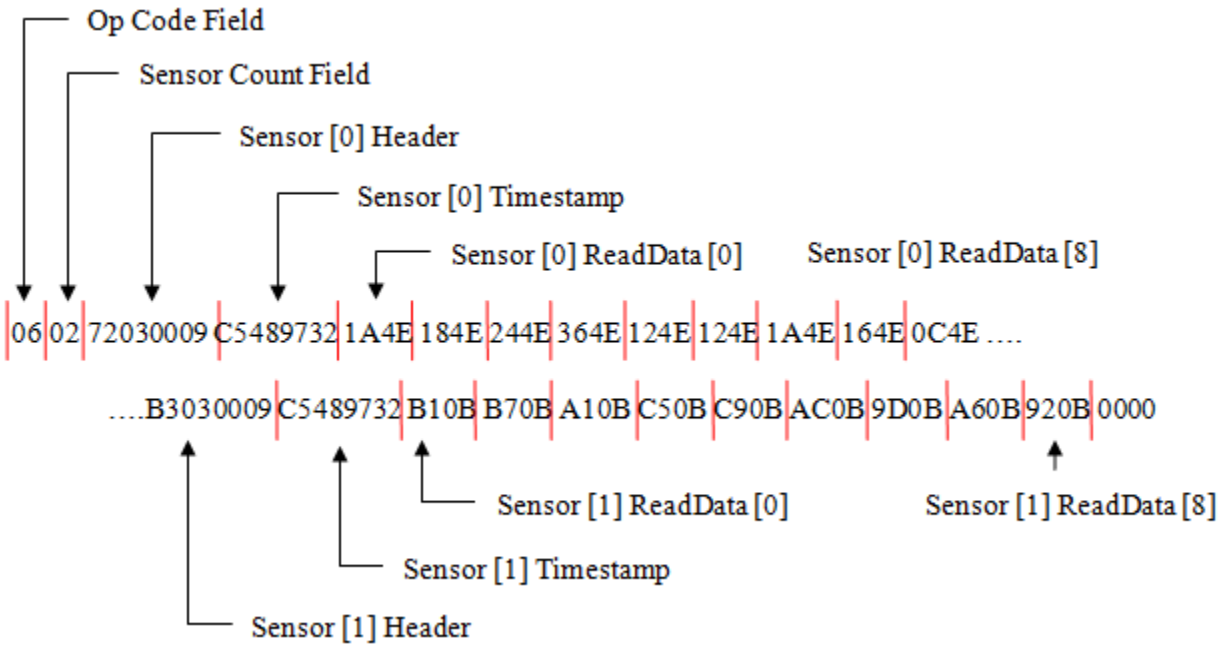


Figure 16. On-Ramp Total View Cathodic Rectifier Data Example

Expanding the highlighted OTA packet, the raw hex data for the Sensor Data OTA payload (see section 9.1) is broken down into the following data stream:



Taking into consideration the little-endian byte ordering and the Sensor Data Message format (Table 21 of section 9.2.7), the Sensor Data OTA payload can be deconstructed into the following fields:

- **Op Code –** 0x06 (Configuration Read Response UL Message)
- **Sensor Count –** 0x02 (Two chained sensor records)
- **Sensor[0] Header** - 0x09000372 (bit packed Header field)
  - Sensor Interface ID (see Table 22) - 0x2 (Cathodic Rectifier Testpoint)
  - Sensor Interface Type (see Table 23) - 0x037 (16-bit unsigned AC Millivolts)
  - Read Interval – 0x00 (30-seconds for test build – for normal builds, 15-minute reads)
  - Read Count - 0x09 (9 chained read records)
- **Sensor[0] Timestamp** - 0x329748C5 (bit packed GMT timestamp for first read record):
  - Seconds - 0x05 (5 sec)
  - Minutes – 0x23 (35 min)
  - Hour – 0x14 (20 hour)
  - Day – 0x0B (11th day)
  - Month – 0xA (October)
  - Year Offset from 2000 – 0x0C (2012)
- **Sensor[0] ReadData[0]** - 0x4E1A (19994mV rms)
- **Sensor[0] ReadData[1]** – 0x4E18 (19990mV rms)
- **Sensor[0] ReadData[2]** – 0x4E24 (20004mV rms)

- **Sensor[0] ReadData[3]** – 0x4E36 (20022mV rms)
- **Sensor[0] ReadData[4]** – 0x4E12 (19984mV rms)
- **Sensor[0] ReadData[5]** – 0x4E12 (19984mV rms)
- **Sensor[0] ReadData[6]** – 0x4E1A (19994mV rms)
- **Sensor[0] ReadData[7]** – 0x4E16 (19988mV rms)
- **Sensor[0] ReadData[8]** – 0x4E0C (19980mV rms)
- **Sensor[1] Header** - 0x090003B3 (bit packed Header field)
  - Sensor Interface ID (see Table 22) - 0x3 (Cathodic Rectifier Shunt)
  - Sensor Interface Type (see Table 23) - 0x03B (16-bit unsigned AC Micro-Volts)
  - Read Interval – 0x00 (30-seconds for test build – for normal builds, 15-minute reads)
  - Read Count - 0x09 (9 chained read records)
- **Sensor[1] Timestamp** - 0x329748C5 (bit packed GMT timestamp for first read record):
  - Seconds - 0x05 (5 sec)
  - Minutes – 0x23 (35 min)
  - Hour – 0x14 (20 hour)
  - Day – 0x0B (11th day)
  - Month – 0xA (October)
  - Year Offset from 2000 – 0x0C (2012)
- **Sensor[1] ReadData[0]** - 0x0BB1 (2993uV rms)
- **Sensor[1] ReadData[1]** – 0x0BB7 (2999uV rms)
- **Sensor[1] ReadData[2]** – 0x0BA1 (2977uV rms)
- **Sensor[1] ReadData[3]** – 0x0BC5 (3013uV rms)
- **Sensor[1] ReadData[4]** – 0x0BC9 (3017uV rms)
- **Sensor[1] ReadData[5]** – 0x0BAC (2988uV rms)
- **Sensor[1] ReadData[6]** – 0x0B9D (2973uV rms)
- **Sensor[1] ReadData[7]** – 0x0BA6 (2982uV rms)
- **Sensor[1] ReadData[8]** – 0x0B92 (2962uV rms)

Extrapolating further, the deconstructed sensor data reported to the backend consists of a chained record of two analog sensor read sequences. The first sensor read sequence is for the primary Rectifier circuit and consists of 9 mVrms measurements spaced 30-seconds apart (15 minutes on a deployed unit) starting at Oct 11th, 2012 20:35:05GMT. The second read sequence is for the secondary Rectifier Shunt circuit and also consists of 9 uVrms measurements spaced 30 seconds apart at the same start time.

C.7 Asynchronous Alarm Example (CPMon Only)

In this example, a rACM unit with CPMON daughter card, is configured to monitor a Vrms analog input with the following alarm configuration parameters (as reported in the Configuration Response Message generated as part of the Network Discovery Connect process):

- **Config Value[17] –** 0x0003 (ORW ID#18: Analog Alarm Type = Out-of-Bounds)
- **Config Value[18] –** 0x03E8 (ORW ID#19: Analog Upper Alarm Threshold Hi = 1000)
- **Config Value[19] –** 0x0320 (ORW ID#20: Analog Upper Alarm Threshold Lo = 800)
- **Config Value[20] –** 0x00C8 (ORW ID#21: Analog Lower Alarm Threshold Hi = 200)
- **Config Value[21] –** 0x0064 (ORW ID#22: Analog Lower Alarm Threshold Lo = 100)

While in steady-state operation the input voltage is initially set at 900mV and the CPMon is operating in the Mid Hysteresis Region for purposes of alarm boundary checks (i.e., below Upper Alarm Threshold Hi and above Lower Alarm Threshold Lo). The input was then increased from 900mV to 1300mV exceeding the Upper Alarm Threshold Hi level and causing an asynchronous alarm event to the backend as the following screen capture from the On-Ramp Total View application shows:

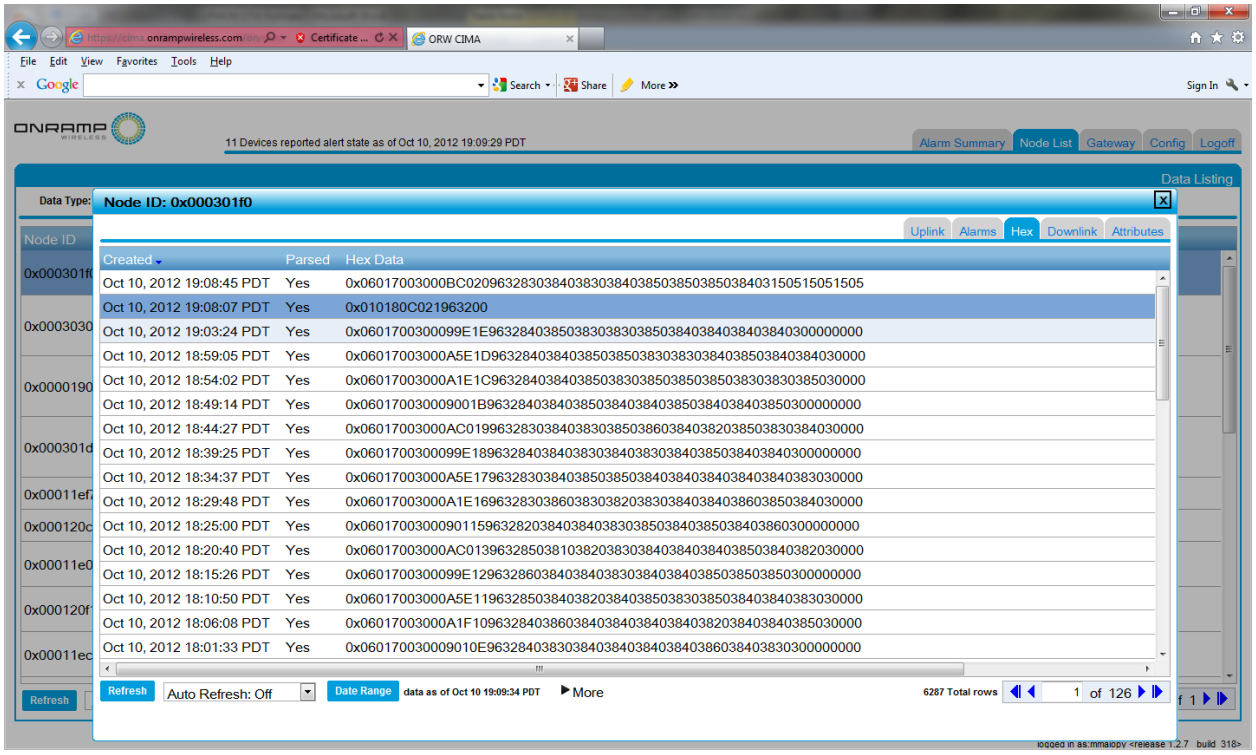
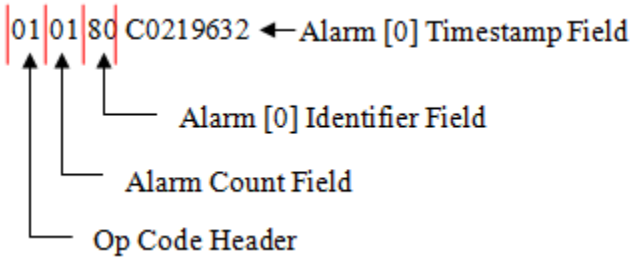


Figure 17. On-Ramp Total View Asynchronous Alarm Example

Expanding the highlighted OTA packet, the raw hex data for the Alarm OTA payload (see section 9.1) is broken down into the following data stream:



Taking into consideration the little-endian byte ordering and the Alarm Message format (see section 9.2.1), the Alarm OTA payload can be deconstructed into the following fields:

- **Op Code –** 0x01 (Alarm UL Message)
- **Alarm Count –** 0x01 (A single alarm record)
- **Alarm[0] Identifier –** 0x80 (bit packed alarm field)
  - Alarm Type – 0x00 (Analog Sensor Alarm)
  - Alarm State – 0x1 (Alarm Set)
- **Alarm[0] Timestamp** – 0x329621C0 (bit packed GMT timestamp for first read record):
  - Seconds – 0x00 (0 sec)
  - Minutes – 0x07 (7 min)
  - Hour – 0x02 (2 hour)
  - Day – 0x0B (11th day)
  - Month – 0xA (October)

Extrapolating further, the deconstructed alarm reported to the backend consists of an alarm indicator indicating that the Out-Of-Bounds alarm condition is in effect starting at Oct 11th, 2012 02:07:00GMT. This can be correlated with the Sensor Data packet received over the same time period.

C.8 OTA Passthrough Example

In this example, a rACM unit configured for OTA Serial Passthrough mode, has received a 4-byte character string, '1234' over the Application UART while the device was in the network JOINED state. The main processing loop received the TX_SERIAL_PASSTHROUGH event and sent the following packet to the backend as the following screen capture from the On-Ramp Total View application shows:

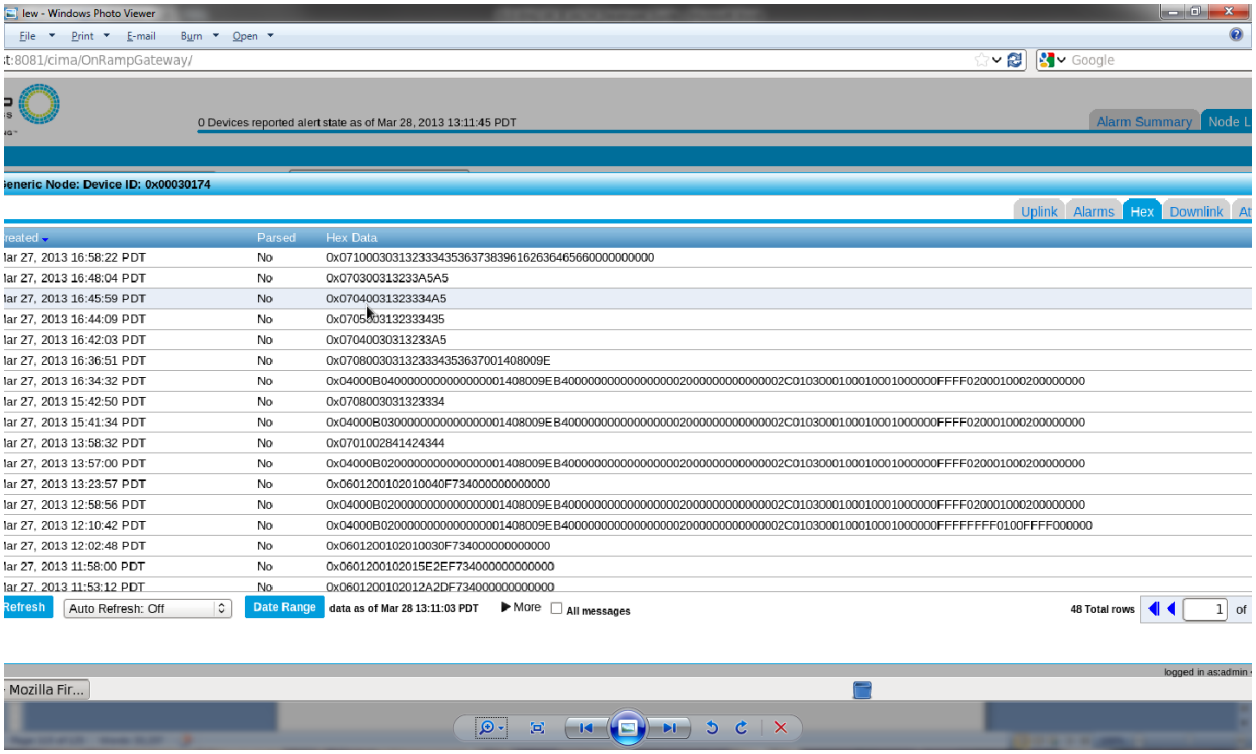
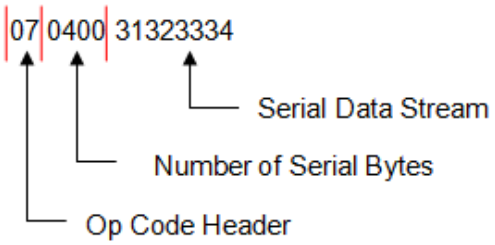


Figure 18. On-Ramp Total View Serial Passthrough Example

Expanding the highlighted OTA packet, the raw hex data for the Serial Passthrough OTA payload (see section 9.1) is broken down into the following data stream:



Taking into consideration the little-endian byte ordering and the Alarm Message format (see section 9.2.1), the Alarm OTA payload can be deconstructed into the following fields:

- **Op Code –** 0x07 (Serial Passthrough Message)
- **Number of Serial Bytes –** 0x0004 (4-byte serial character stream)
- **Serial Data Stream** – 0x31323334

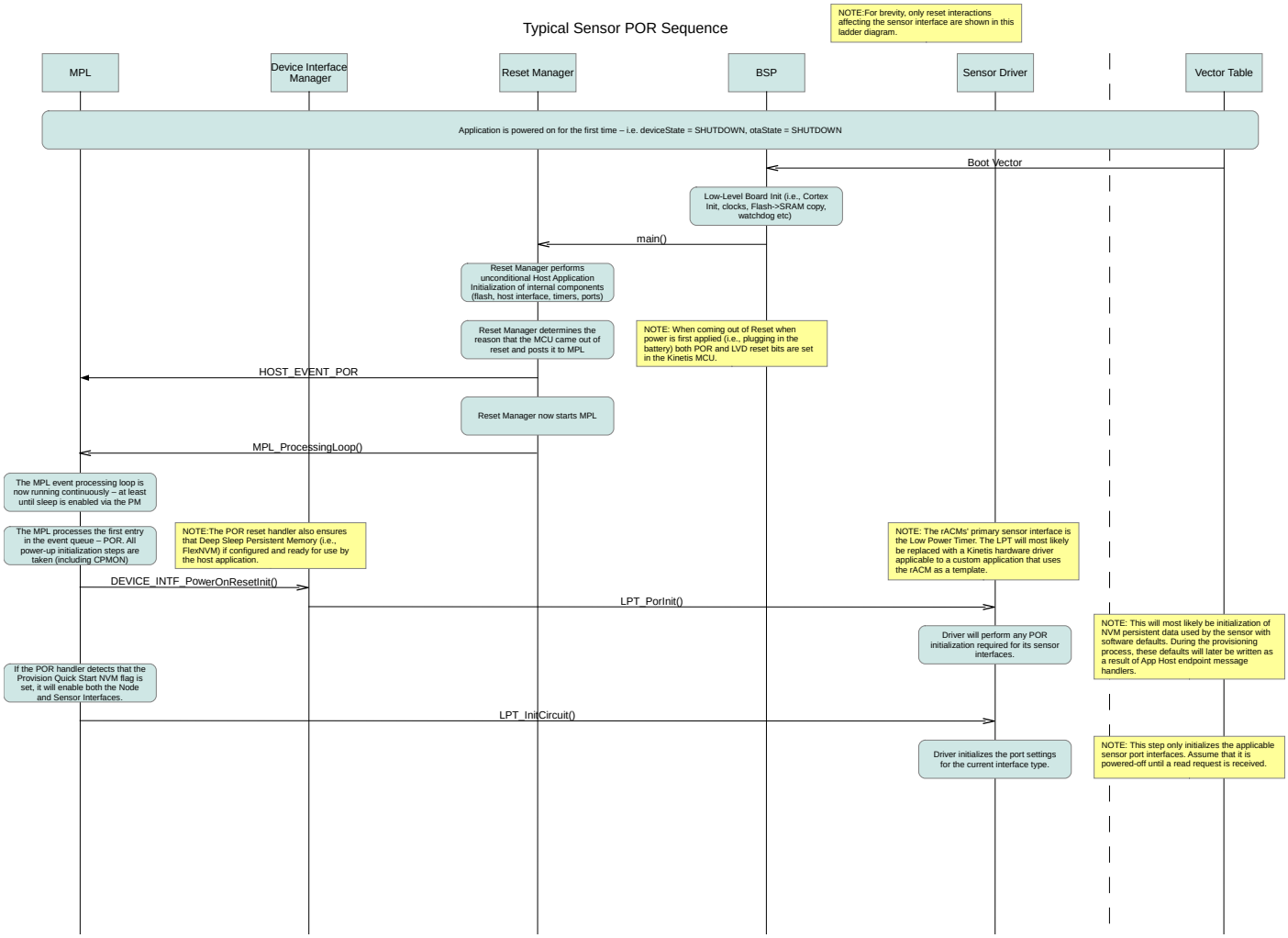
- **Data Byte[0]** – 0x31 or ascii character ‘1’
- **Data Byte[1]** – 0x32 or ascii character ‘2’
- **Data Byte[2]** – 0x33 or ascii character ‘3’
- **Data Byte[3]** – 0x34 or ascii character ‘4’

Appendix D Example Sequence Flow Charts

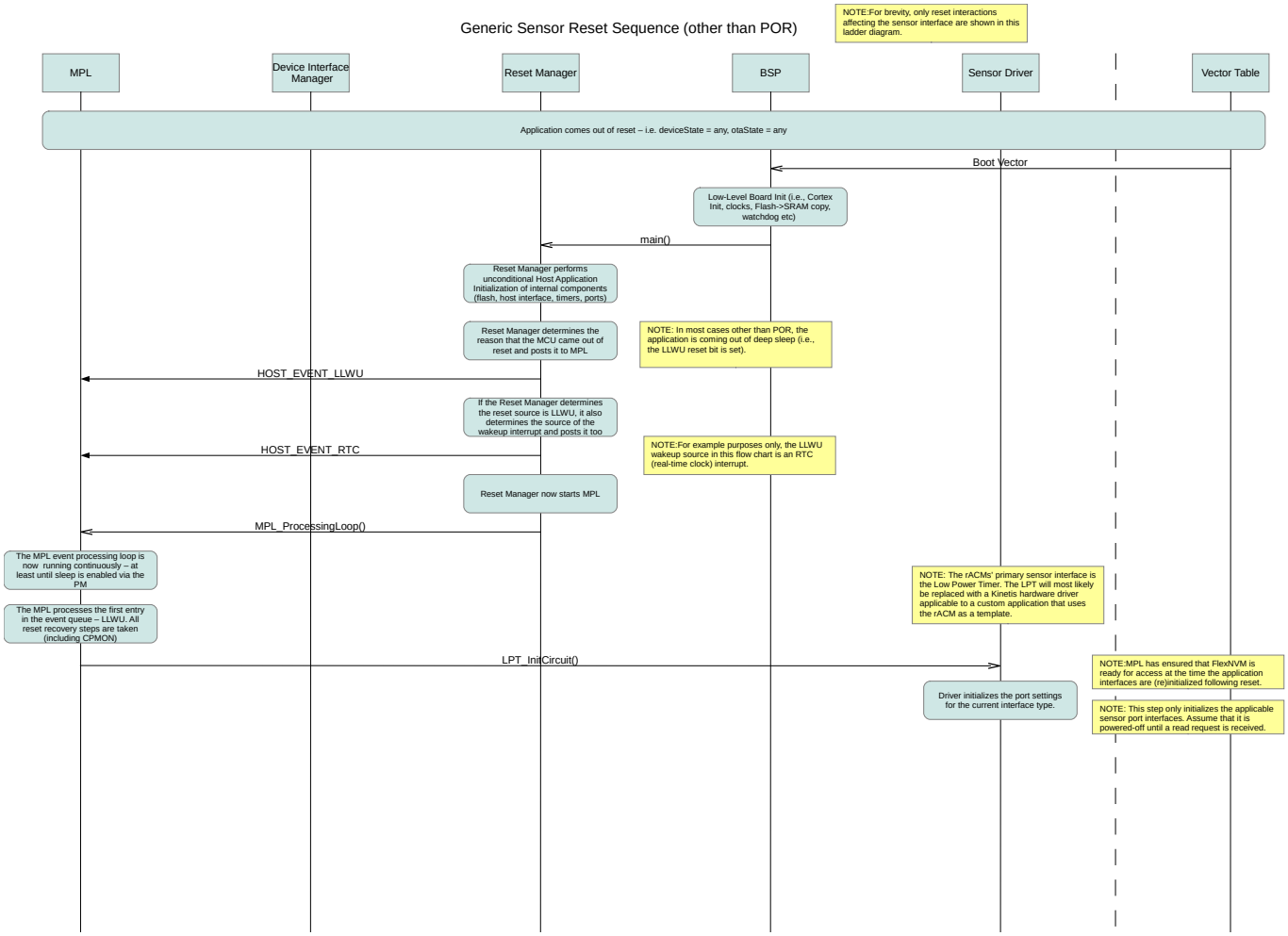
This appendix illustrates the interactions between the Main Processing Loop (MPL), an example set of generic sensors drivers, an example set of Cathodic Protect Monitor (CPMon) Analog Interface Drivers, and the hardware blocks that are required for the various analog interface control sequences.

NOTE: The CPMon analog interface drivers require a separate hardware daughter card and are not provided with the rACM. This appendix provides flow details for supporting additional sensors.

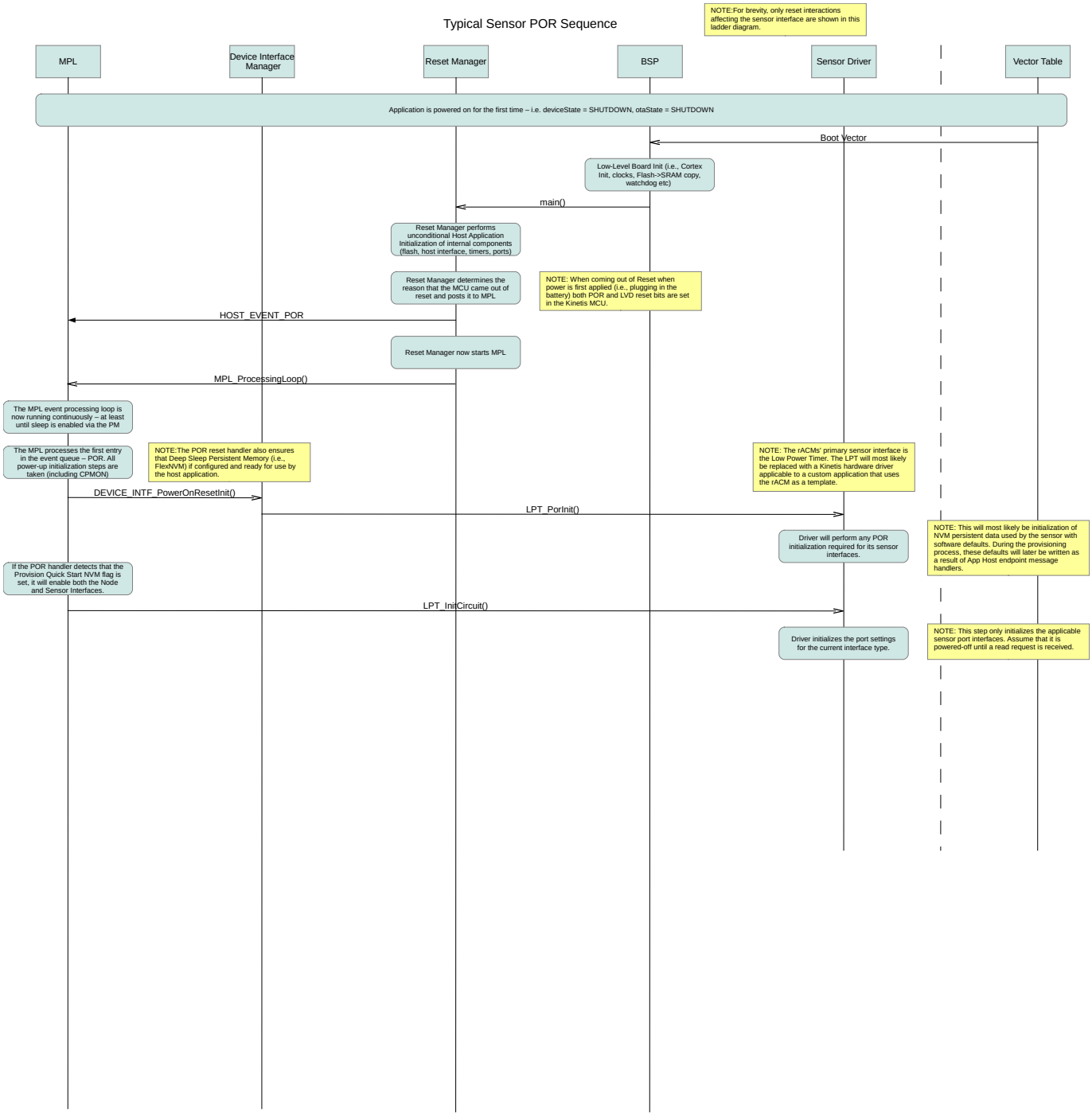
D.1 Generic Sensor POR Sequence



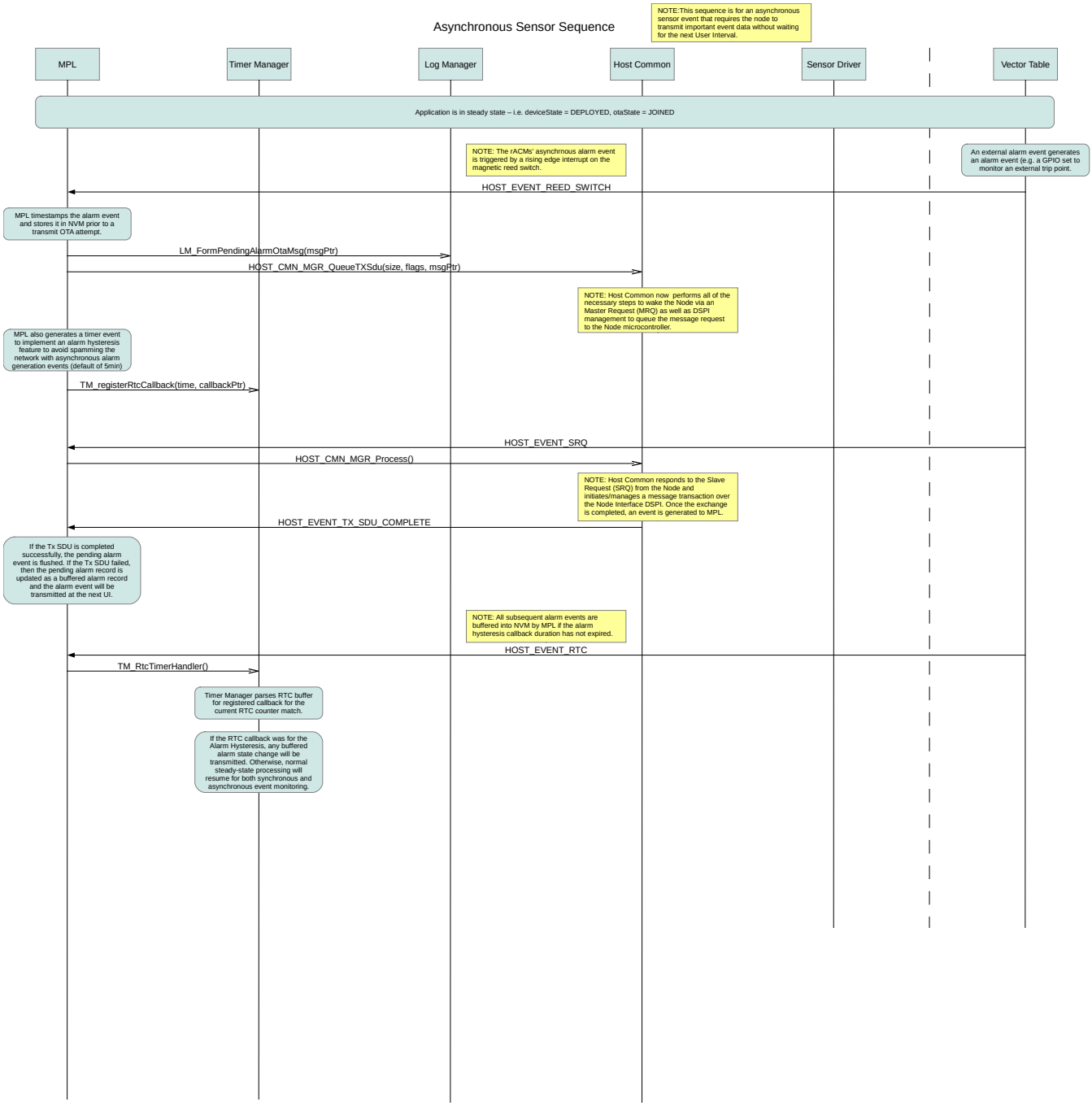
D.2 Generic Sensor Reset (non-POR) Sequence



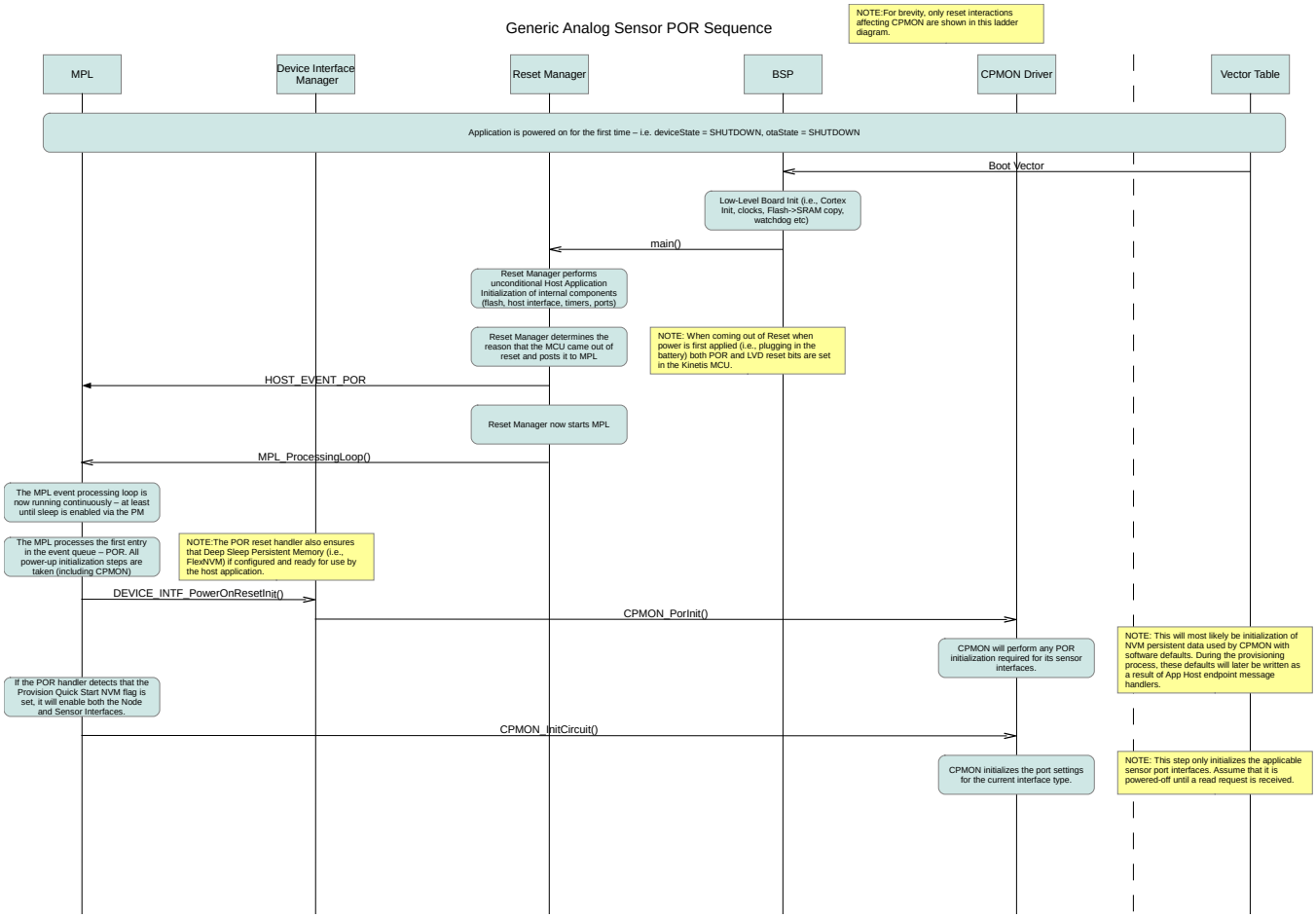
D.3 Synchronous Sensor Sequence



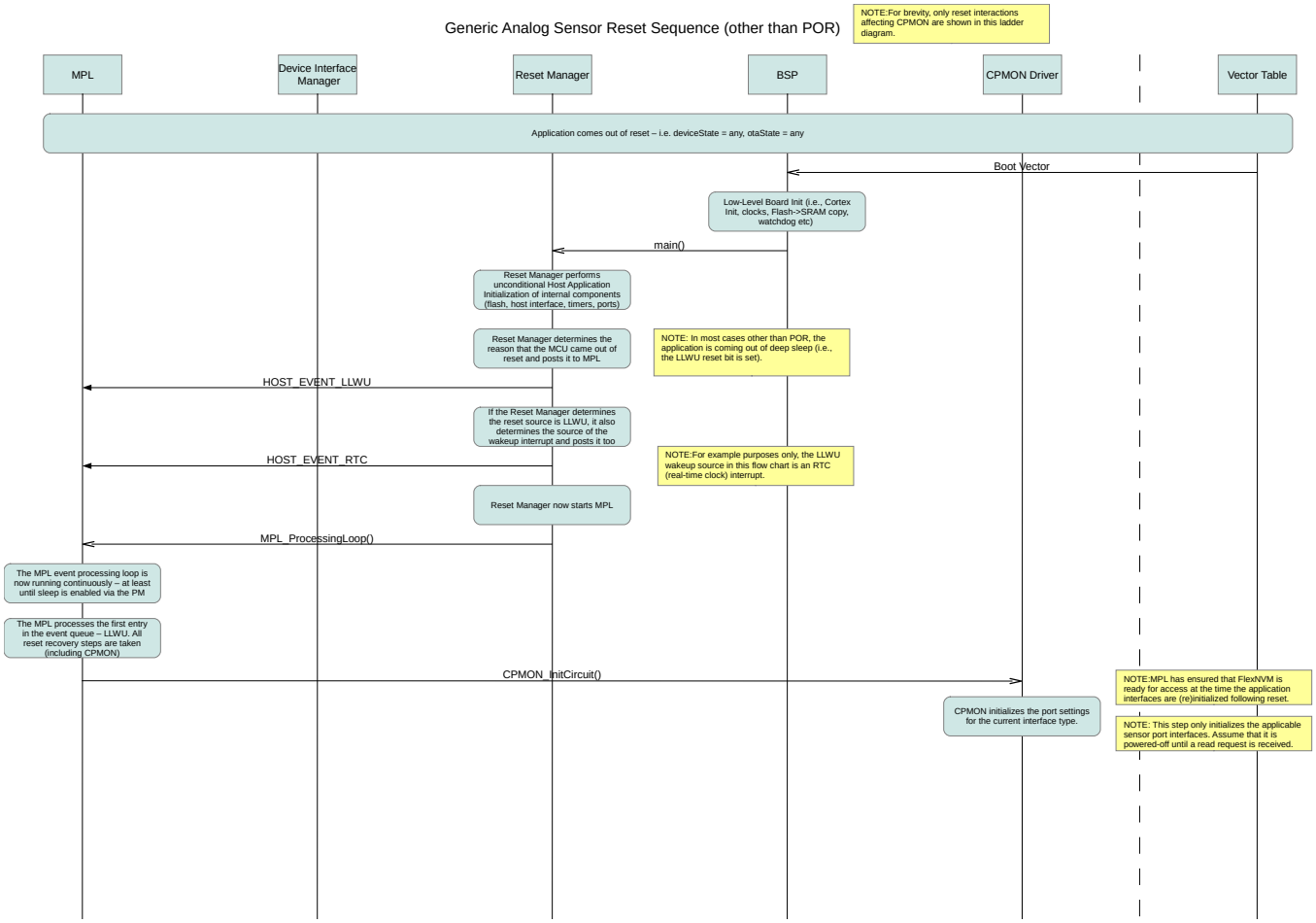
D.4 Asynchronous Sensor Sequence



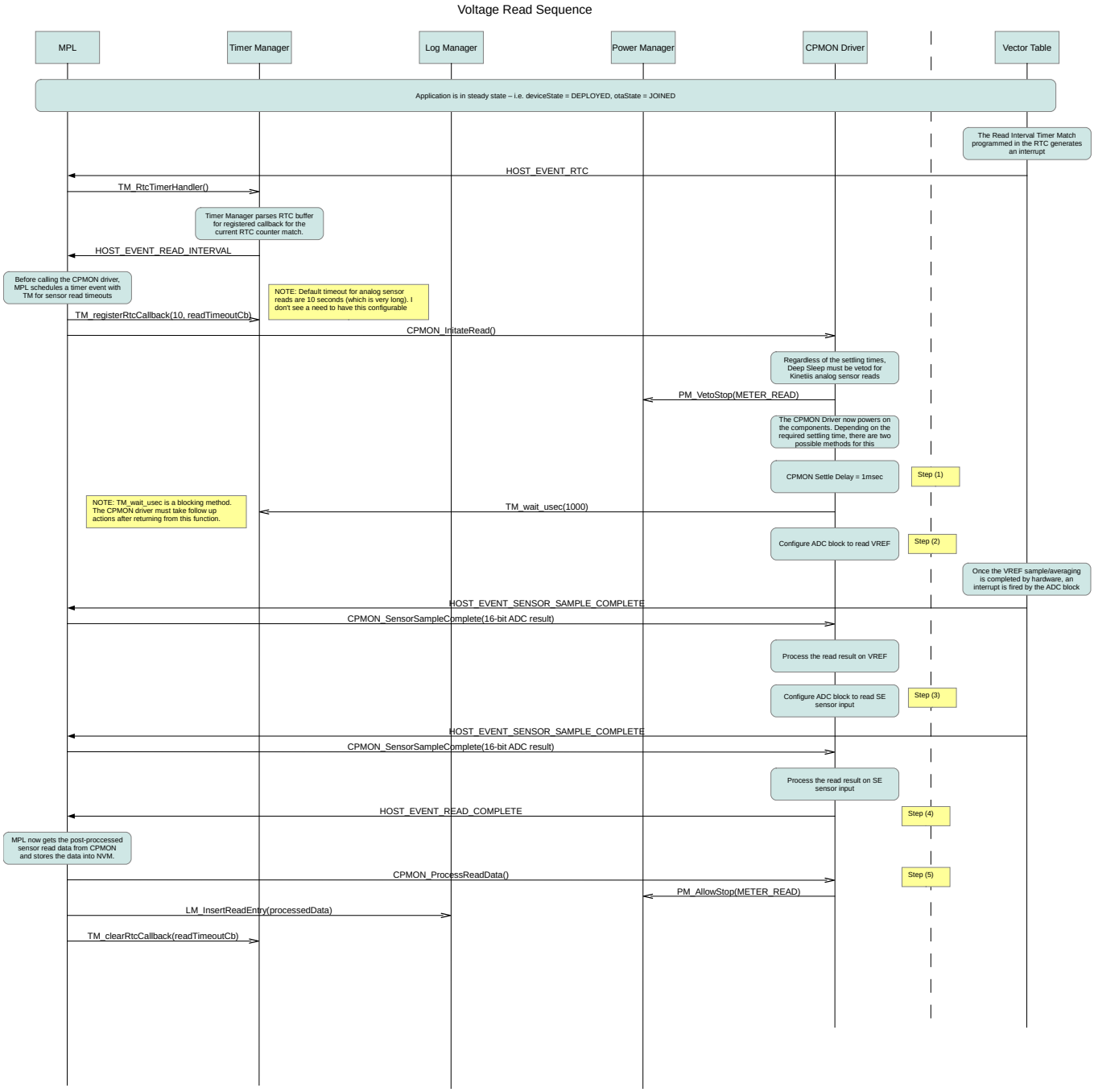
D.5 Analog Sensor POR Sequence (CPMON Only)



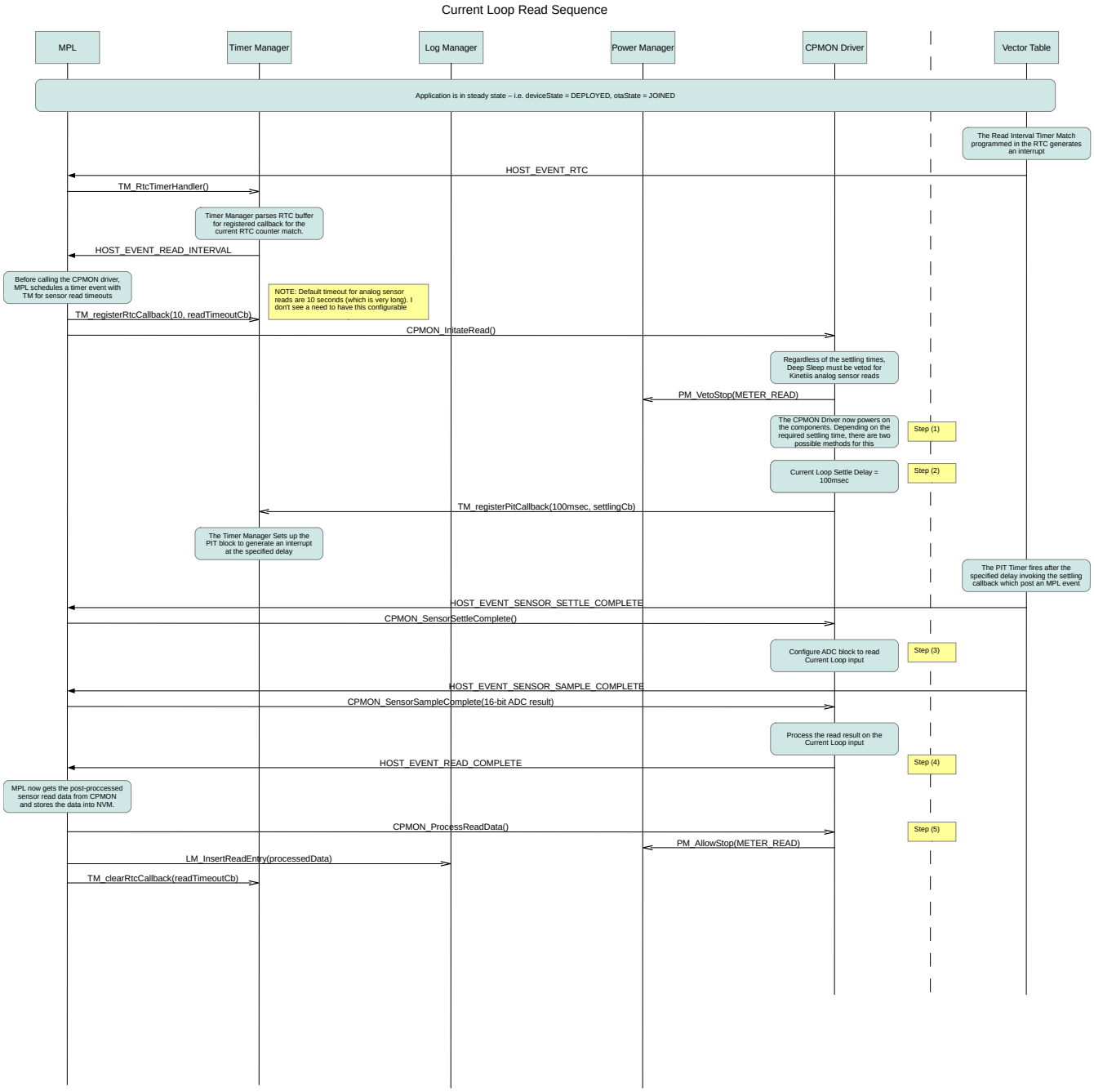
D.6 Analog Sensor Reset (non-POR) Sequence (CPMON Only)



D.7 Voltage Testpoint Read Sequence (CPMON Only)

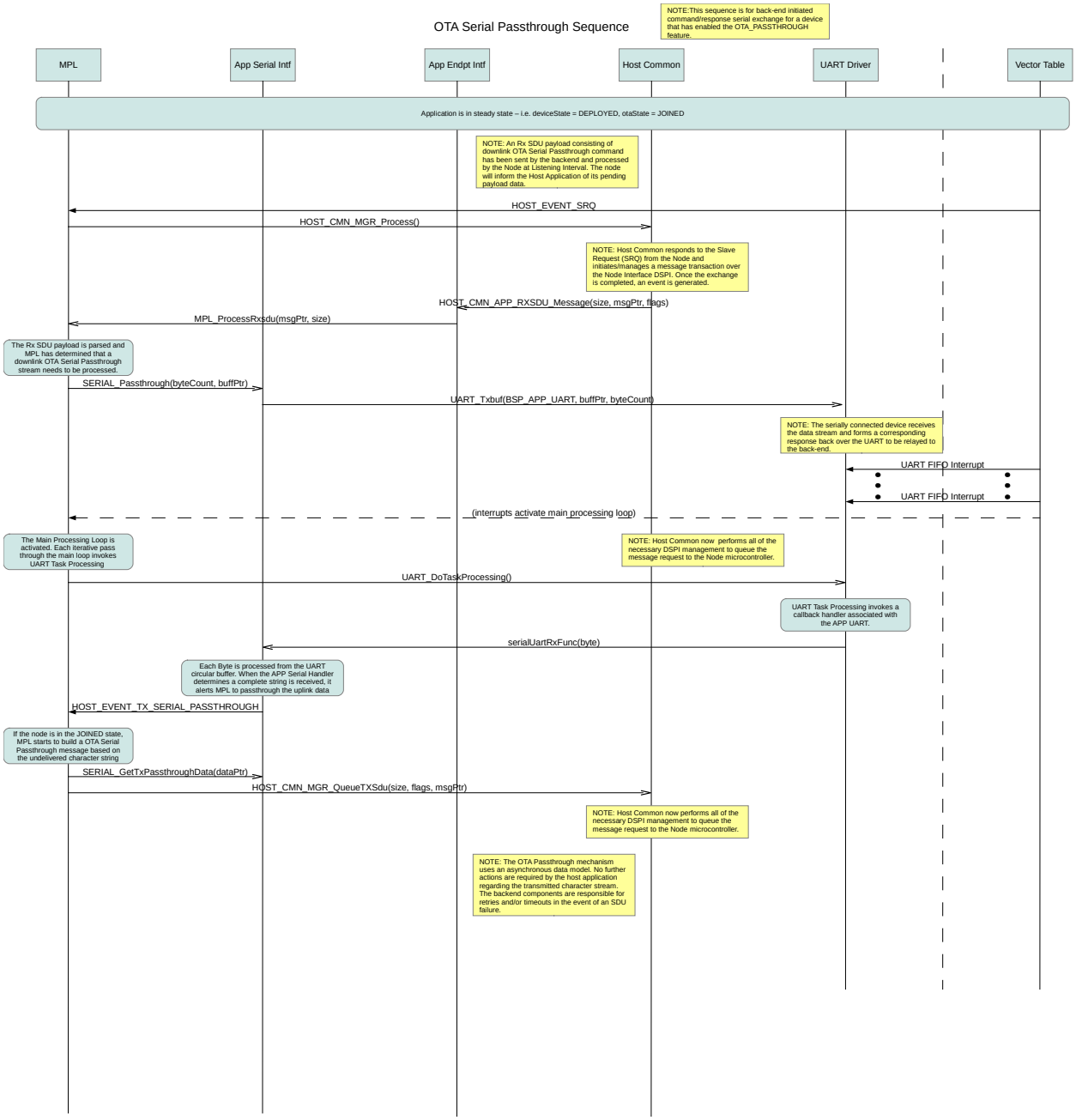


D.8 Current Loop Read Sequence (CPMON Only)



D.9 OTA Serial Passthrough Sequence

The following sequence example demonstrates a command-response serial exchange initiated by the backend to a serially-connected sensor device to the rACM:



Appendix E Schematics

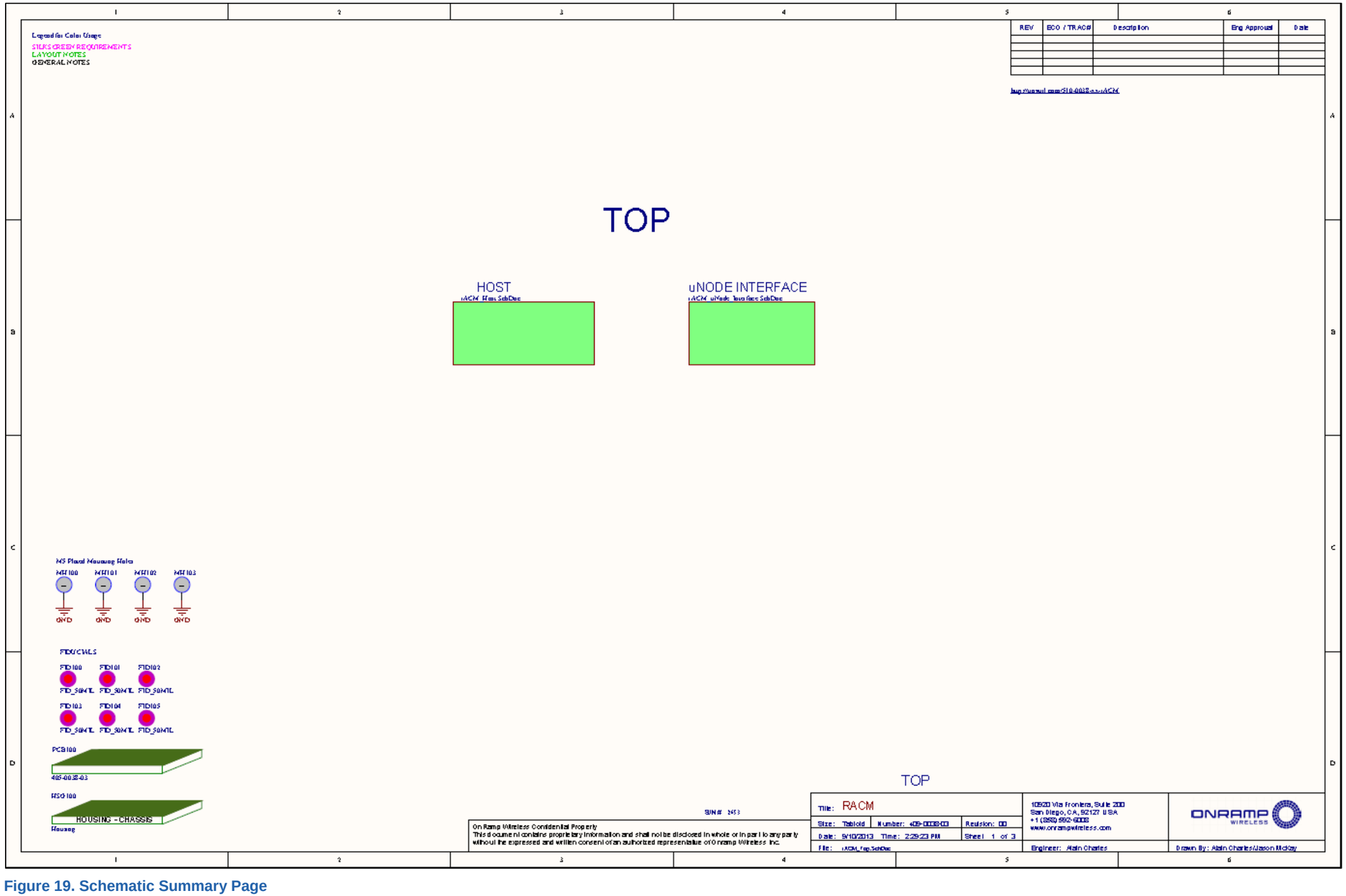


Figure 19. Schematic Summary Page

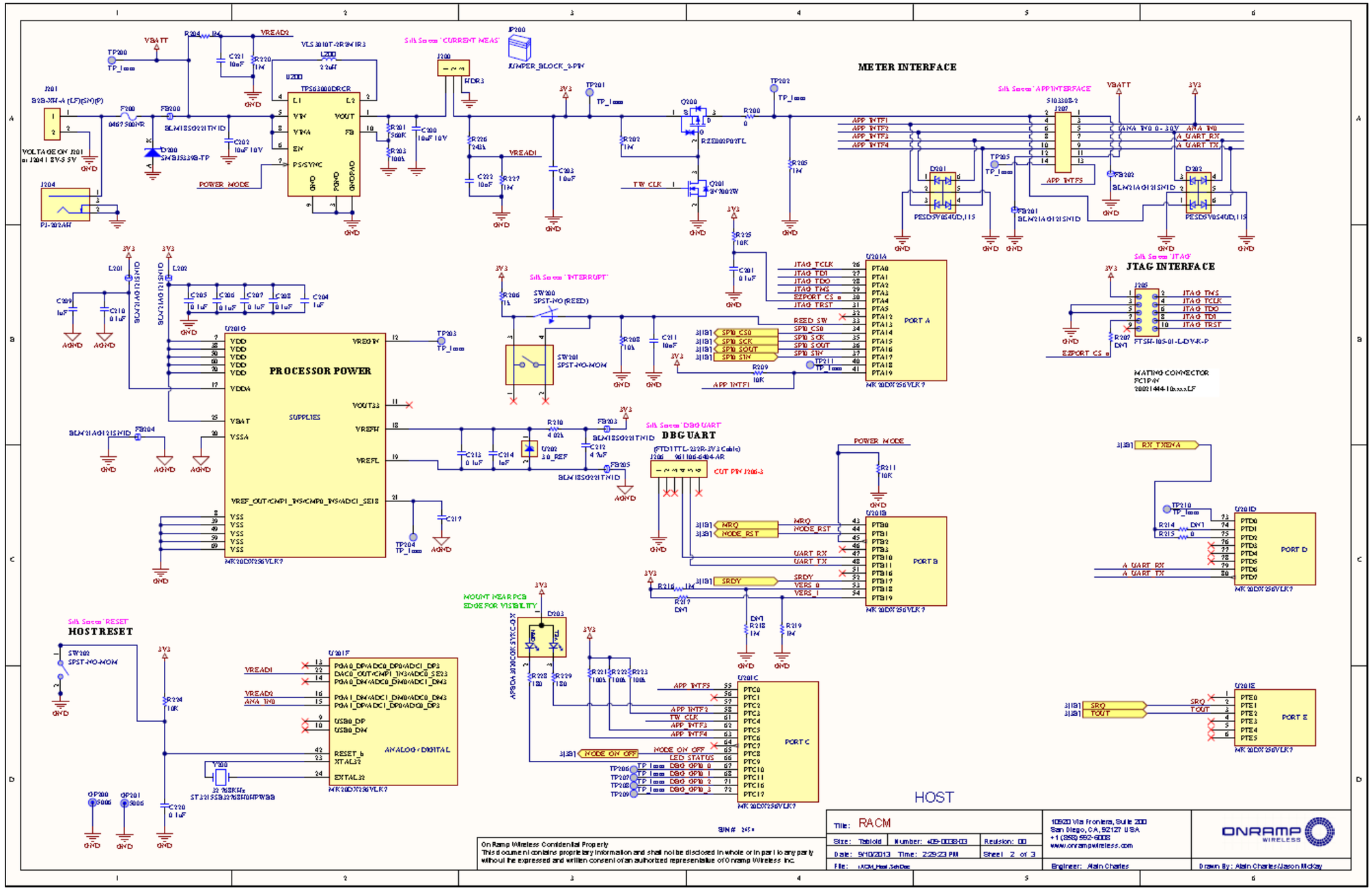


Figure 20. Host Components

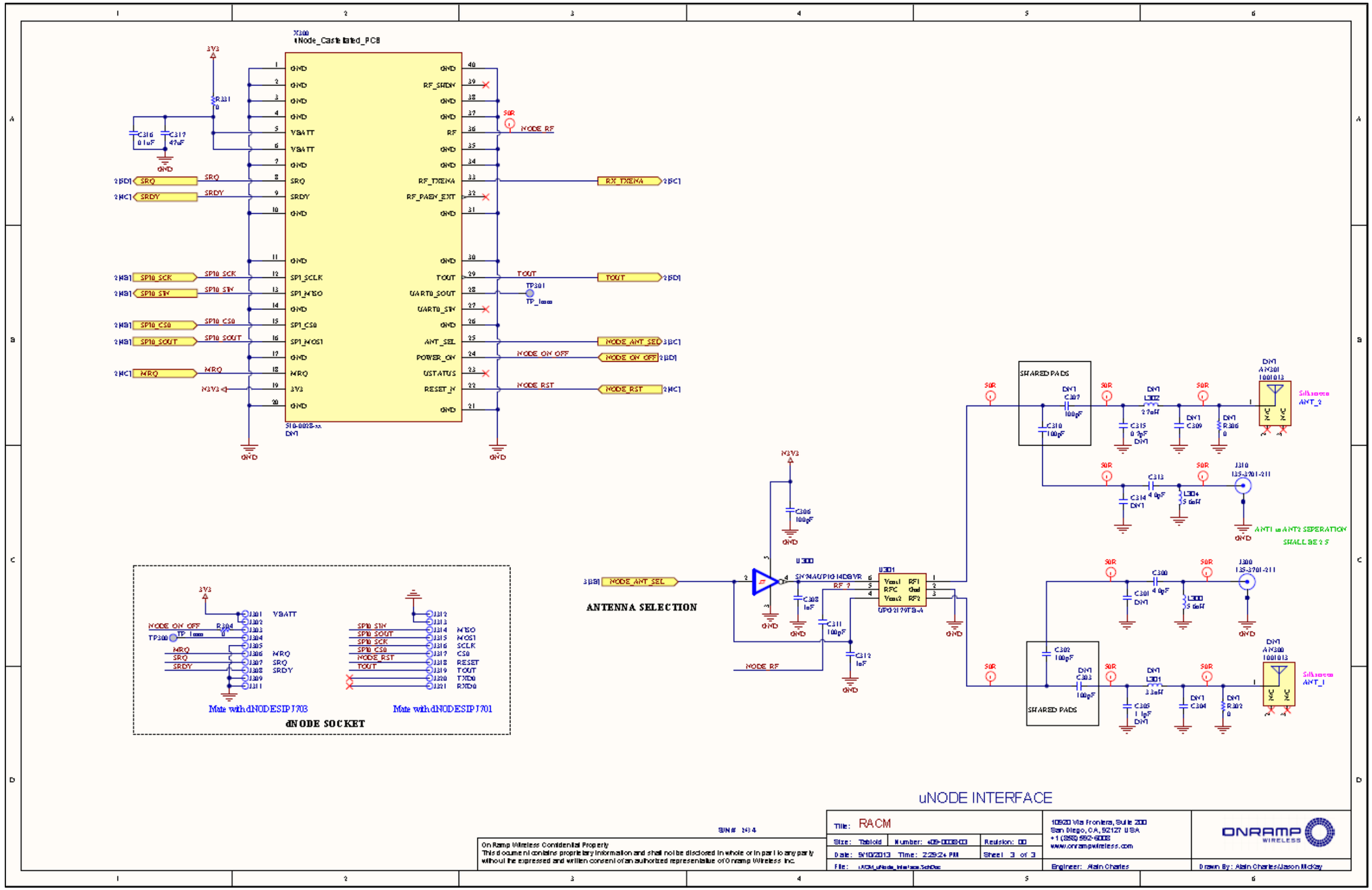


Figure 21. microNode Interface

Appendix F Bill of Materials

	A	B	C	D	E	F	G	H	I	J	K	L
1	PARENT	ORW	Item	Quantity	Designator	Population	Value	Mfr Part Description	Manufacturer	Mfr Part Number	Supplier	Supplier Part Number
2	510-0038-03	624-0010-00	1	2	C200, C202		10uF 10V	CAP CER 10uF ±10% 10V X7R 0805 (2012 Metric)	Murata Electronics North America	GRM21BR71A106KE51L	Digikey	490-3905-6-ND
3	510-0038-03	628-0016-00	2	9	C201, C205, C206, C207, C208, C210, C213, C220, C316		0.1uF	CAP CER 0.1uF ±10% 10V X5R 0402 (1005 Metric)	Murata Electronics North America	GRM155R61A104KA01D	Digikey	490-1318-6-ND
4	510-0038-03	628-0053-00	3	1	C203		1.0uF	CAP CER 1.0uF ±10% 50V X5R 0805 (2012 Metric)	Taiyo Yuden	UMK212BJ105KG-T	Digikey	587-2229-6-ND
5	510-0038-03	628-0021-00	4	2	C204, C209		1uF	CAP CER 1.0uF ±10% 35V X5R 0603 (1608 Metric)	Taiyo Yuden	GMK107BJ105KA-T	Digikey	587-1437-6-ND
6	510-0038-03	624-0030-00	5	3	C211, C221, C222		10nF	CAP CER 10000pF ±10% 25V X7R 0402 (1005 Metric)	Murata Electronics North America	GRM155R71E103KA01D	Digikey	490-1312-1-ND
7	510-0038-03	624-0013-00	6	1	C212		4.7uF	CAP CER 4.7uF ±10% 10V X7R 0805 (2012 Metric)	Taiyo Yuden	LMK212B7475KG-T	Digikey	587-1442-6-ND
8	510-0038-03	624-0002-00	7	3	C214, C308, C312		1nF	CAP CER 1000pF ±10% 50V X7R 0402 (1005 Metric)	Murata Electronics North America	GRM155R71H102KA01	Digikey	490-1303-6-ND
9	510-0038-03	620-0009-00	8	5	C217, C302, C306, C310, C311		100pF	CAP CER 100pF ±5% 50V NPO 0402 (1005 Metric)	Murata Electronics North America	GRM1555C1H101JZ01D	Digikey	490-3458-1-ND
10	510-0038-03	620-0065-00	9	2	C300, C313		4.0pF	CAP CER 4.0pF ±0.25pF 50V NPO 0402 (1005 Metric)	Murata Electronics North America	GJM1555C1H4R0CB01D	Digikey	490-3098-6-ND
11	510-0038-03	628-0054-00	10	1	C317		47uF	CAP CER 47uF ±20% 6.3V X5R 0805 (2012 Metric)	Taiyo Yuden	JMK212BJ476MG-T	Digikey	587-1779-6-ND
12	510-0038-03	710-0067-00	11	1	D200		SMBJ5339B-TP	DIODE ZENER 5.6V 5W DO214AA	Micro Commercial Co	SMBJ5339B-TP	Digikey	SMBJ5339B-TPMSDKR-ND
13	510-0038-03	710-0047-00	12	2	D201, D202		PESD5V0S4UD,115	DIODE TVS ARRAY ESD 5V 6-TSOP	NXP Semiconductors	PESD5V0S4UD,115	Digikey	568-4267-6-ND
14	510-0038-03	700-0018-00	13	1	D203		APBDA3020CGKSYKC-GX	LED RA SMD GRN-YLW 570NM, 590NM	Kingbright Corp	APBDA3020CGKSYKC-GX	Digikey	754-1572-2-ND
15	510-0038-03	698-0044-00	14	1	F200		0467.500NR	FUSE FAST 32V 500MA 0603	Littelfuse Inc	0467.500NR	Digikey	F1391DKR-ND
16	510-0038-03	650-0008-00	15	3	FB200, FB203, FB205		BLM18SG221TN1D	FERRITE CHIP 220 OHM 2500MA 0603	Murata Electronics North America	BLM18SG221TN1D	Digikey	490-5225-6-ND
17	510-0038-03	650-0004-00	16	5	FB201, FB202, FB204, L201, L202		BLM21AG121SN1D	FERRITE CHIP 120 OHM 800mA 0805	Murata Electronics North America	BLM21AG121SN1D	Digikey	490-1038-1-ND
18	510-0038-03	694-0001-00	17	2	GP200, GP201		5006	TP Test Point Compact .063 Black Thru-Hole	Keystone Electronics	5006	Digikey	5006K-ND
19	510-0038-03	671-0005-00	18	1	J200		HDR3	CONN HEADER 1x3 100mil	Samtec	TSW-103-07-T-S	Samtec	TSW-103-07-T-S
20	510-0038-03	671-0054-00	19	1	J201		B2B-XH-A (LF)(SN)(P)	CONN HEADER XH TOP 2POS 2.5MM	JST Sales America Inc	B2B-XH-A (LF)(SN)(P)	Digikey	455-2247-ND
21	510-0038-03	673-0001-00	20	1	J204		PJ-202AH	CONN POWER JACK 24V, 5A, 2.1X5.5mm	CUI	PJ-202AH	Digikey	CP-202AH-ND
22	510-0038-03	670-0023-00	21	1	J205		FTSH-105-01-L-DV-K-P	CONN HEADER 10POS DUAL (Pitch 1.27mm) 0.05",3.05mm Leads, 0.25um Gold SMD	Samtec	FTSH-105-01-L-DV-K-P	Samtec	FTSH-105-01-L-DV-K-P
23	510-0038-03	671-0049-00	22	1	J206		961106-6404-AR	CONN HEADER VERT SGL 6POS 100mil GOLD	3M	961106-6404-AR	Digikey	3M9451-ND
24	510-0038-03	671-0055-00	23	1	J207		5103308-2	CONN HEADER LOW PROFILE MALE STR 14POS GOLD	TE Connectivity	5103308-2	Digikey	A33161-ND
25	510-0038-03	206-0003-00	24	2	J300, J310		135-3701-211	CONN MMCX JACK STR PCB 68mil GOLD	Emerson Network	135-3701-211	Digikey	J598-ND
26	510-0038-03	671-0034-00	25	20	J301, J302, J303, J304, J305, J306, J307, J308, J309, J311, J312, J313, J314, J315, J316, J317, J318, J319, J320, J321		0305-2-15-15-47-27-10-0	CONN RECEPT PIN .025-.037" .155"	Mill-Max Manufacturing Corp.	0305-2-15-15-47-27-10-0	Digikey	ED90333-ND
27	510-0038-03	671-0059-00	26	1	JP200		JUMPER_BLOCK_2-PIN	Conn Shunt Socket 2 POS 2.54mm ST	Samtec	SNT-100-BK-G	Samtec	SNT-100-BK-G
28	510-0038-03	654-0001-00	27	1	L200		2.2uH	INDUCTOR 2.2uH ±20% 1.3A SMD	TDK Corporation	VLS3010T-2R2M1R3	Digikey	445-4254-6-ND
29	510-0038-03	654-0067-00	28	2	L300, L304		5.6nH	INDUCTOR 5.6nH ±0.1nH 140mA 0402 (1005 Metric)	Murata	LQP15MN5N6B02D	Mouser	81-LQP15MN5N6B02D
30	510-0038-03	405-0038-03	29	1	PCB100		405-0038-03	PCB, BOARD - UN-POPULATED	OnRamp Wireless	405-0038-03	OnRamp Wireless	405-0038-03
31	510-0038-03	720-0006-00	30	1	Q200		RZE002P02TL	MOSFET P-CH 20V 200MA EMT3	Rohm Semiconductor	RZE002P02TL	Digikey	RZE002P02TLCT-ND
32	510-0038-03	720-0001-00	31	1	Q201		2N7002W	FET N-TYPE MOSFET N-CH 60V 115mA SOT-323	Fairchild	2N7002W	Digikey	2N7002WDKR-ND
33	510-0038-03	600-0209-00	32	1	R200		0	RES 0.0 Ohms 1/8W 0805 (2012 Metric)	Panasonic Electronic Components	ERJ-6GEY0R00V	Digikey	P0.0ADKR-ND
34	510-0038-03	600-0070-00	33	1	R201		560K	RES 560k Ohms ±1% 1/16W 0402 (1005 Metric)	Yageo	RC0402FR-07560KL	Digikey	311-560KLRDKR-ND
35	510-0038-03	600-0090-00	34	7	R202, R204, R205, R216, R219, R220, R227		1M	RES 1M Ohms ±1% 1/16W 0402 (1005 Metric)	Vishay Dale	CRCW04021M00FKED	Digikey	541-1.00MLDKR-ND
36	510-0038-03	600-0141-00	35	4	R203, R221, R222, R223		100k	RES 100k Ohms ±1% 1/10W 0402 (1005 Metric)	Panasonic Electronic Components	ERJ-2RKF1003X	Digikey	P100KLDKR-ND
37	510-0038-03	600-0003-00	36	1	R206		1k	RES 1k Ohms ±1% 1/16W 0402 (1005 Metric)	Rohm Semiconductor	MCR01MZPF1001	Digikey	RHM1.00KLDKR-ND
38	510-0038-03	600-0079-00	37	5	R208, R209, R211, R224, R225		10k	RES 10k Ohms ±1% 1/16W 0402 (1005 Metric)	Rohm Semiconductor	MCR01MZPF1002	Digikey	RHM10.0KLDKR-ND
39	510-0038-03	600-0122-00	38	1	R210		4.02k	RES 4.02k Ohms ±1% 1/10W 0402 (1005 Metric)	Panasonic Electronic Components	ERJ-2RKF4021X	Digikey	P4.02KLDKR-ND
40	510-0038-03	600-0040-00	39	2	R215, R304		0	RES 0.0 Ohms 1/10W 0402 (1005 Metric)	Panasonic Electronic Components	ERJ-2GE0R00X	Digikey	P0.0JDKR-ND
41	510-0038-03	600-0265-00	40	1	R226		243k	RES 243k Ohms ±1% 1/10W 0402 (1005 Metric)	Panasonic Electronic Components	ERJ-2RKF2433X	Digikey	P243KLDKR-ND
42	510-0038-03	600-0241-00	41	2	R228, R229		180	RES 180 Ohms ±1% 1/16W 0402 (1005 Metric)	Vishay Dale	CRCW0402180RFKED	Digikey	541-180LDKR-ND

Figure 22. Bill of Materials (Page 1)

	A	B	C	D	E	F	G	H	I	J	K	L
1	PARENT	ORW	Item	Quantity	Designator	Population	Value	Mfr Part Description	Manufacturer	Mfr Part Number	Supplier	Supplier Part Number
43	510-0038-03	600-0137-00	42	1	R331		0	RES 0.0 Ohms 1/2W 1210 (3225 Metric)	Vishay Dale	CRCW12100000Z0EA	Digikey	541-0.0VDKR-ND
44	510-0038-03	680-0003-00	43	1	SW200		SPST-NO (REED)	SWITCH REED SPST 15-20AT SMD	Hamlin	59170-1-T-00-D	Digikey	HE551CT-ND
45	510-0038-03	674-0016-00	44	2	SW201, SW202		SPST-NO-MOM	SWITCH TACTILE SPST-NO 0.02A 15V	Panasonic Electronic Components	EVQ-P2402W	Digikey	P11081SDKR-ND
46	510-0038-03	760-0004-00	45	1	U200		TPS63000DRCR	IC REG BUCK BOOST SYNC ADJ 10SON	Texas Instruments	TPS63000DRCR	Digikey	296-19641-6-ND
47	510-0038-03	742-0030-00	46	1	U201		MK20DX256VLK7	IC MCU, 72MHz, 256K, USB, -40C/+150C, 80 LQFP	Freescale Semiconductor	MK20DX256VLK7	Avnet	MK20DX256VLK7
48	510-0038-03	760-0071-00	47	1	U202		3.0_REF	IC VREF SHUNT PREC 3V SOT-23-3	Maxim Integrated	MAX6009BEUR+T	Digikey	MAX6009BEUR+TCT-ND
49	510-0038-03	740-0125-00	48	1	U300		SN74AUP1G14DBVR	IC SNGL SCHMITT TRIG SOT23-5	Texas Instruments	SN74AUP1G14DBVR	Digikey	296-18213-6-ND
50	510-0038-03	740-0063-00	49	1	U301		UPG2179TB-A	IC SWITCH SPDT 6-SMINI	CEL	UPG2179TB-E4-A	Digikey	UPG2179TB-E4-ADKR-ND
51	510-0038-03	730-0051-00	50	1	Y200		32.768KHz	CLK XTAL 32.768KHz 12pF, -40°C to 85°C SMD	Kyocera	ST3215SB32768H0HPWBB	Kyocera	ST3215SB32768H0HPWBB
52	510-0038-03	698-0041-00	51	2	AN300, AN301	DNI	1001013	RF ANTENNA 2.4GHz 3dBi SMD	Ethertronics	1001013	Ethertronics	1001013
53	510-0038-03	620-0021-00	52	4	C301, C304, C309, C314	DNI	0.5pF	CAP CER 0.50pF ±0.1pF 50V C0H 0402 (1005 Metric)	Taiyo Yuden	UVK105CH0R5BW-F	Digikey	587-1002-6-ND
54	510-0038-03	620-0009-00	53	2	C303, C307	DNI	100pF	CAP CER 100pF ±5% 50V NPO 0402 (1005 Metric)	Murata Electronics North America	GRM1555C1H101JZ01D	Digikey	490-3458-1-ND
55	510-0038-03	620-0058-00	54	1	C305	DNI	1.1pF	CAP CER 1.1pF ±0.1pF 50V NPO 0402 (1005 Metric)	Murata	GJM1555C1H1R1BB01D	Mouser	81-GJM1555C1H1R1BB01
56	510-0038-03	620-0027-00	55	1	C315	DNI	0.7pF	CAP CER 0.7pF ±0.1pF 50V NPO 0402 (1005 Metric)	Murata	GJM1555C1HR70BB01D	Mouser	81-GJM1555C1HR70BB1D
57	510-0038-03	654-0060-00	56	1	L301	DNI	3.3nH	INDUCTOR 3.3nH ±0.1nH 190mA 0402 (1005 Metric)	Murata Electronics North America	LQP15MN3N3B02D	Digikey	490-1130-6-ND
58	510-0038-03	654-0010-00	57	1	L302	DNI	2.7nH	INDUCTOR 2.7nH ±0.1nH 220mA 0402 (1005 Metric)	Murata Electronics North America	LQP15MN2N7B02D	Digikey	490-1129-6-ND
59	510-0038-03	600-0040-00	58	1	R207	DNI	0	RES 0.0 Ohms 1/10W 0402 (1005 Metric)	Panasonic Electronic Components	ERJ-2GE0R00X	Digikey	P0.0JDKR-ND
60	510-0038-03	600-0090-00	59	3	R214, R217, R218	DNI	1M	RES 1M Ohms ±1% 1/16W 0402 (1005 Metric)	Vishay Dale	CRCW04021M00FKED	Digikey	541-1.00MLDKR-ND
61	510-0038-03	600-0013-00	60	2	R302, R306	DNI	0	RES 0.0 Ohms 1/16W 0402 (1005 Metric)	Yageo	RC0402JR-070RL	Digikey	311-0.0JRDKR-ND
62	510-0038-03	510-0028-xx	61	1	X300	DNI	510-0028-xx	uNode Module	OnRamp Wireless	510-0028-xx	OnRamp Wireless	510-0028-xx

Figure 23. Bill of Materials (Page 2)

Appendix G Abbreviations and Terms

Abbreviation/Term	Definition
ACM	Application Communication Module
ADC	Analog-to-Digital Conversion
AP	Access Point. The On-Ramp Total Reach network component geographically deployed over a territory.
AWIC	Asynchronous Wake Interrupt Controller
BSP	Board Support Package
CIMA	Critical Infrastructure Monitoring Application. The network component that passes data from the Gateway to the associated upstream databases. This product name is transitioning to On-Ramp Total View.
Dashboard	Web page view of the aggregated end-device monitoring data.
DIM	Device Interface Manager. A set of drivers on the ACM that manages all sensor application interfaces on the application host.
EEE	Enhanced EEPROM. A mode of operation on the Kinetis FlexNVM package that improves wear-leveling and endurance cycles on data flash.
EMS	Element Management System. The network component that provides a concise view of controls and alarms on the On-Ramp Total Reach network.
FIFO	First-In, First Out. A software queue implementation type.
FMC	Flash Memory Controller. A Kinetis module responsible for managing Program Flash, Data Flash, and/or FlexNVM.
FTFL	Flash memory module
FTM	Flex Timer Module. A timer block on the Kinetis MCU.
GPIO	General Purpose Input/Output. A pin on the Kinetis MCU.
GW	Gateway. The network appliance that provides a single entry point into the back office for the On-Ramp Total Reach Network. A gateway talks upstream to the EMS and On-Ramp Total View software applications. It talks downstream to multiple APs.
HAL	Hardware Abstraction Layer. A software functional block on the ACM used to implement drivers for all node interfaces.
HLD	High Level Design
ICD	Interface Control Document
IRQ	Interrupt Request
IM	Image Manager. A software functional block on the ACM used to manage firmware updates to program flash.
LM	Log Manager. A software functional block on the ACM used to manage log entries in data flash.
LLWU	Low Leakage Wake Up. A system block on the Kinetis MCU allowing the processor to wake up from MCU deep sleep.
LPTMR	Low Power Timer
MCU	Microcontroller Unit
microNode	A second generation, small form factor, wireless network module developed by On-Ramp Wireless that works in combination with various devices and sensors and communicates data to an Access Point. Also referred to as uNode.
MPL	Main Processing Loop. A software functional block on the ACM used to manage all sensor operations.

Abbreviation/Term	Definition
MSP	Main Stack Pointer
Node	The generic term used interchangeably with an end point On-Ramp Wireless device.
NVIC	Nested Vector Interrupt Controller. A system block on the Kinetis MCU used to enable/disable interrupts.
NVM	Non-Volatile Memory. In the context of the Kinetis, memory that is persistent across the different Low Leakage power modes.
On-Ramp Total View	The network component that passes data from the Gateway to the associated upstream databases. Formerly known as CIMA.
On-Ramp Total Reach	The On-Ramp Wireless' proprietary wireless communication technology and network.
ORW	On-Ramp Wireless
OTA	Over-the-Air
OTV	On-Ramp Total View
PM	Power Manager. A software functional block on the rACM used to manage power mode transitions.
PMC	Power Management Controller. A system block on the Kinetis MCU used to manage power savings states on the MCU.
POR	Power-on Reset
PWM	Pulse Width Modulation. A waveform with a configurable duty cycle.
rACM	reference Application Communication Module
RHT	Reliable Host Transfer. An optional retry protocol used on all serial data transfers over the Host Interface.
RM	Reset Manager. A software functional block on the ACM used to manage all transitions out of system reset.
RTC	Real Time Clock. A timer block on the Kinetis MCU that clocked by the low-power 32 K crystal.
RTOS	Real Time Operating System
Rx	Receive/Receiver
SDU	Service Data Unit
TM	Timer Manager. A software functional block on the ACM used to manage timer resources.
Tx	Transmit/Transmitter
UART	Universal Asynchronous Receiver/Transmitter
UI	Uplink Interval. The scheduled uplink interval where the rACM periodically sends accumulated sensor read data.
UNIL	Universal Node Interface Library. The On-Ramp Wireless software library containing all of the necessary software interfaces to communicate with the On-Ramp Wireless Node.
uNode	Shorthand for microNode. The microNode is a second generation, small form factor, wireless network module developed by On-Ramp Wireless that works in combination with various devices and sensors and communicates data to an Access Point.
UTC	Universal Time Coordinated. The time standard maintained on the On-Ramp wireless network.
VM	■ Vector Manager. A software functional block on the ACM used to manage all hardware interrupts. or ■ Virtual Machine. A way to distribute the RACM release, Python environment, and tools.