TI Designs：TIDEP0084
低于 1GHz 传感器到云工业物联网 (IoT) 网关参考设计

说明

此参考设计展示如何通过低于 1GHz 的远距离无线网络将传感器连接到云，适用于楼宇控制和资产跟踪等工业环境。此设计搭载了 TI Sitara™AM335x 处理器和 SimpleLink™超低功耗 (ULP) 低于 1GHz 的 CC1310 和 CC1350 器件。此参考设计预先集成了 SimpleLink CC13x0 软件开发套件 (SDK)（TI SimpleLink MCU 平台组成部分）的 TI 15.4-Stack 器件，在 TI 低功耗有线 MCU 和无线 MCU 中实现了一致的软件体验。此参考设计还包括 TI Linux®处理器 SDK。TI 设计网络与 stackArmor 合作，支持通过云应用服务实现云连接和传感器节点数据可视化。

资源

TIDEP0084	设计文件夹
CC1310	产品文件夹
CC1350	产品文件夹
AM335x	产品概述

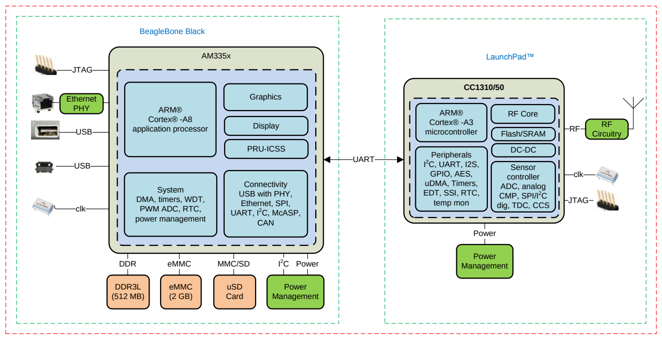
TI E2E™ Community　咨询我们的 E2E 专家

特性

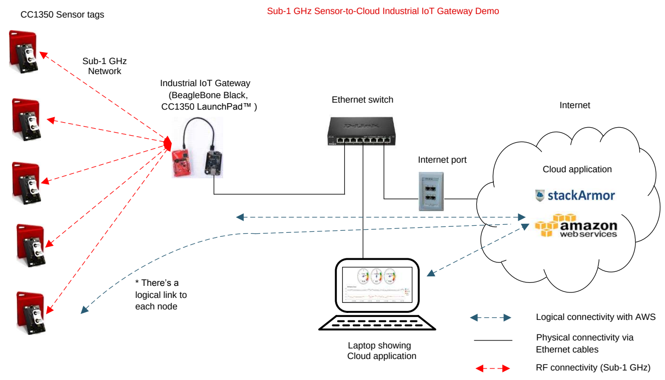
- 大型网络到云连接可实现远距离应用，支持长达 1km 的视线
- 采用 TI 15.4-Stack 且符合 IEEE 802.15.4e/g 标准的低于 1GHz 解决方案
- 基于成熟的硬件设计，具有开箱即用的现成演示软件，有助于加快推向市场的步伐
- TI Linux 处理器 SDK 提供跨多个 Sitara 处理器（例如 AM437x 和 AM57x）的可扩展性
- 支持星形网络
- 超低功耗传感器节点

应用

- 楼宇安全网关
- 门窗传感器网络
- HVAC 网关
- 资产管理和跟踪


BeagleBone Black
AM335x
JTAG
Ethernet PHY
USB
USB
clk
ARM® Cortex® -A8 application processor
Graphics
Display
PRU-ICSS
System DMA, timers, WDT, PWM ADC, RTC, power management
Connectivity USB with PHY, Ethernet, SPI, UART, I²C, McASP, CAN
DDR
eMMC
MMC/SD
I²C
Power
DDR3L (512 MB)
eMMC (2 GB)
uSD Card
Power Management
UART
LaunchPad™
CC1310/50
ARM® Cortex® -A3 microcontroller
RF Core
Flash/SRAM
DC-DC
Peripherals I²C, UART, I2S, GPIO, AES, uDMA, Timers, EDT, SSI, RTC, temp mon
Sensor controller ADC, analog CMP, SPI/I²C dig, TDC, CCS
RF
RF Circuitry
clk
JTAG
Power
Power Management

Copyright © 2016, Texas Instruments Incorporated


CC1350 Sensor tags
Sub-1 GHz Sensor-to-Cloud Industrial IoT Gateway Demo
Sub-1 GHz Network
Industrial IoT Gateway (BeagleBone Black, CC1350 LaunchPad™)
Ethernet switch
Internet
Internet port
Cloud application
stackArmor
amazon web services
* There's a logical link to each node
Laptop showing Cloud application
Logical connectivity with AWS
Physical connectivity via Ethernet cables
RF connectivity (Sub-1 GHz)

该 TI 参考设计末尾的重要声明表述了授权使用、知识产权问题和其他重要的免责声明和信息。

1 System Description

This reference design provides a reference for creating an industrial IoT gateway that is capable of connecting a network of wireless sensors to an enterprise cloud provider. In this reference design, a long-range, low-power wireless network, made up of Sub-1 GHz CC1310 or CC1350 devices (both are supported) that run the TI 15.4-Stack-based application, is connected to the AWS IoT cloud. An online dashboard is provided that allows the user to visualize the real-time sensor data as well as send actuation commands from anywhere in the world using an Internet-connected device with a web browser.

This reference design provides a list of required hardware, schematics, and foundational software to quickly begin your Internet of things (IoT) product development. Amazon Web Services™ (AWS) IoT service, which is brokered by stackArmor, was chosen as the cloud service provider for the demonstration. The software design, however, is architected to be flexible to enable other cloud service providers.

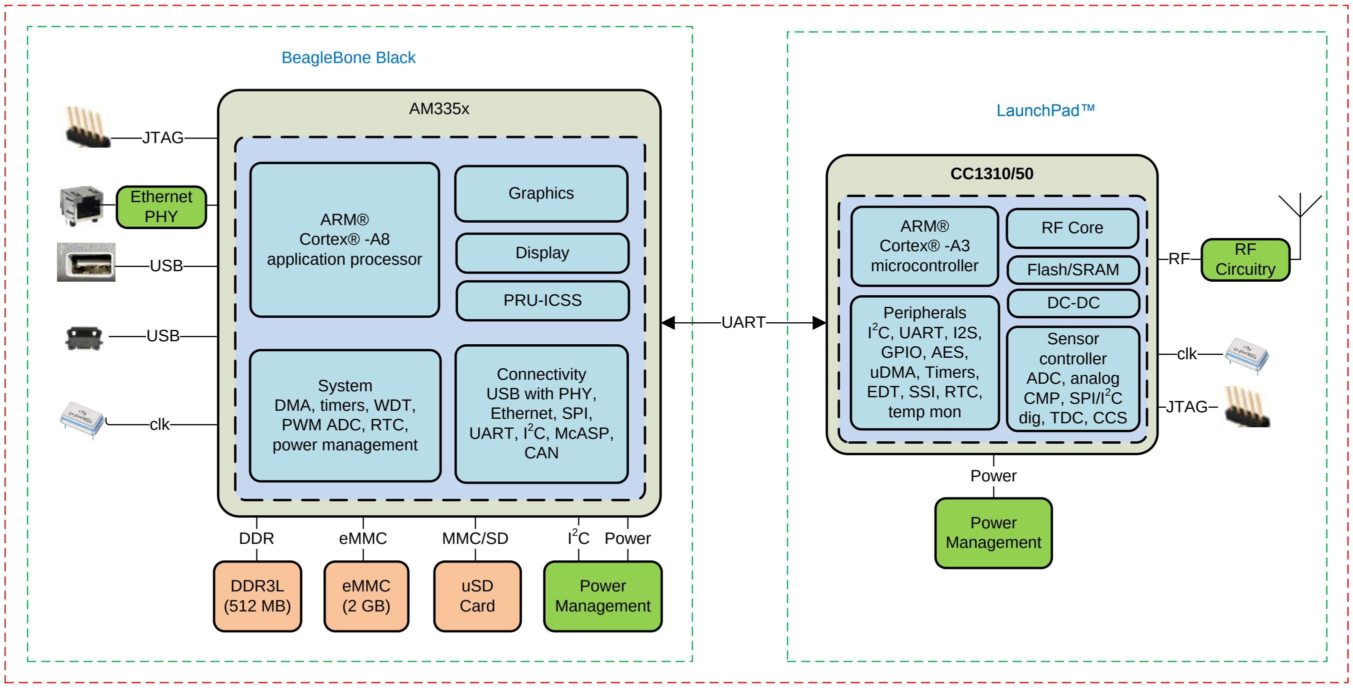
This reference design enables IoT in numerous applications, such as building security gateways, door and window sensor networks, asset management and tracking, and other IoT-enabled home and industrial automation applications.

The connection between the wireless sensor network and the cloud is made possible by TI's Sitara AM335x device on the BeagleBone Black development platform. On one side, the AM335x is connected to a Sub-1 GHz device acting as the central node in the wireless network, and on the other side, the device is connected to the AWS IoT cloud using Ethernet. These two connections allow the AM335x device to act as a gateway to get the sensor messages from the wireless network to the cloud and, also, to get the actuation requests from the cloud dashboard sent back to the wireless network.

Due to the long-range and low-power capabilities of the Sub-1 GHz sensors, this reference design is useful for any type of application that would benefit from distributed sensing. This reference design provides a blueprint that gives the ability to visualize or actuate tens or hundreds of sensors while only needing one gateway device, TI's Sitara AM335x, to be connected to the Internet.

2 System Overview

2.1 Block Diagram



BeagleBone Black
AM335x
JTAG
Ethernet PHY
USB
USB
clk
ARM® Cortex® -A8 application processor
Graphics
Display
PRU-ICSS
System DMA, timers, WDT, PWM ADC, RTC, power management
Connectivity USB with PHY, Ethernet, SPI, UART, I2C, McASP, CAN
DDR
eMMC
MMC/SD
I2C
Power
DDR3L (512 MB)
eMMC (2 GB)
uSD Card
Power Management
UART
LaunchPad™
CC1310/50
ARM® Cortex® -A3 microcontroller
RF Core
Flash/SRAM
DC-DC
Peripherals I2C, UART, I2S, GPIO, AES, uDMA, Timers, EDT, SSI, RTC, temp mon
Sensor controller ADC, analog CMP, SPI/I2C dig, TDC, CCS
RF
RF Circuitry
clk
JTAG
Power
Power Management

图 1. IoT Gateway Reference Design Block Diagram

2.1.1 Software Block Diagram

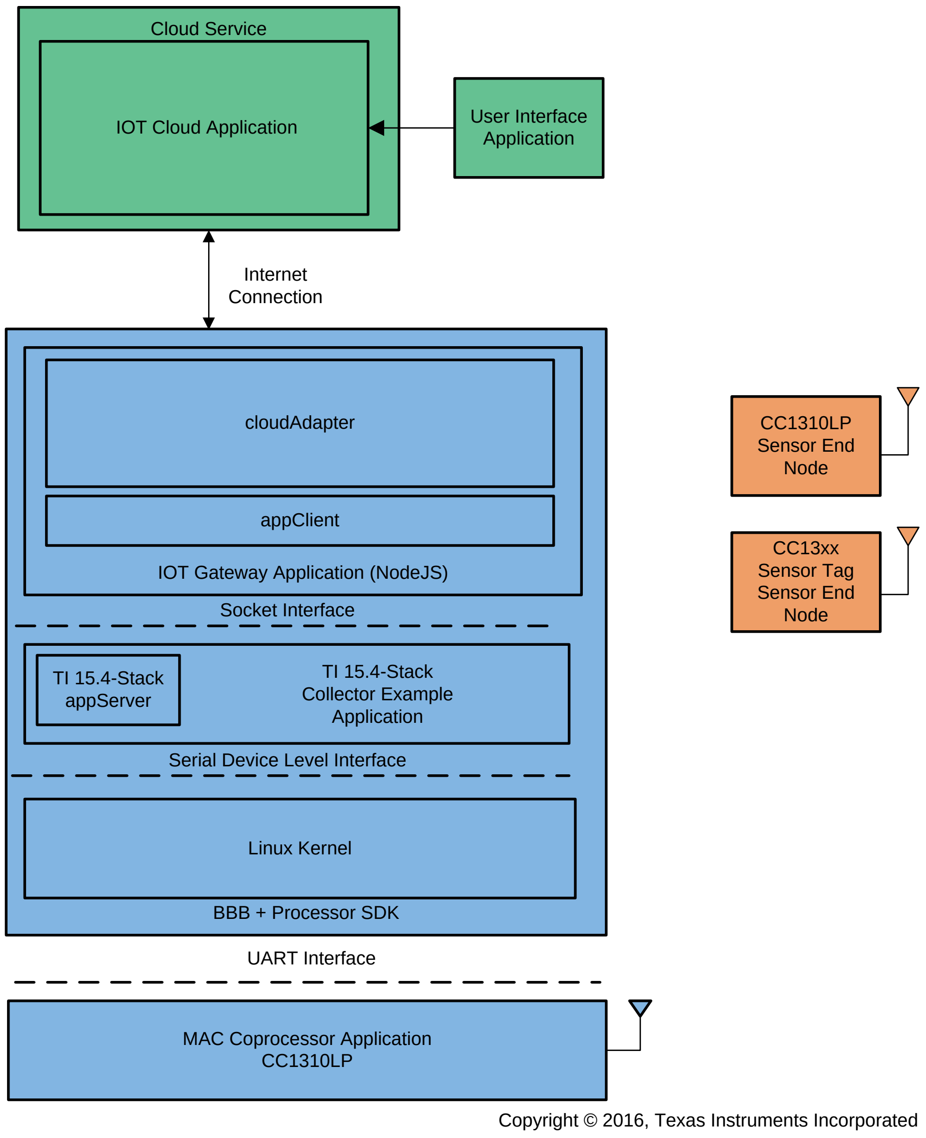

Cloud Service
IOT Cloud Application
User Interface Application
Internet Connection
cloudAdapter
appClient
IOT Gateway Application (NodeJS)
Socket Interface
TI 15.4-Stack appServer
TI 15.4-Stack Collector Example Application
Serial Device Level Interface
Linux Kernel
BBB + Processor SDK
UART Interface
MAC Coprocessor Application CC1310LP
CC1310LP Sensor End Node
CC13xx Sensor Tag Sensor End Node
Copyright © 2016, Texas Instruments Incorporated

图 2. TI 15.4-Stack Sensor-to-Cloud Reference Design Software Block Diagram

The following is a high-level description of each module in the software block diagram:

- **User Interface Application**: This application presents the network information, device information, and provides ability to control network behavior to the end user.
- **IOT Cloud Application**: This application runs on the AWS cloud, which communicates with the IoT gateway application. The interface of the IoT cloud application with the cloud server is described in 节 3.1.2.10 and 节 3.1.2.11.
- **IoT Gateway Application**: This application runs on the BeagleBone Black board. The application interfaces on one side with the cloud service to enable cloud connectivity and on the other side to the Linux collector application to interface with the TI 15.4-Stack based network. The interface between the IoT gateway and the cloud service is described in 节 3.1.2.10.
  - **cloudAdapter**: This application provides the cloud service provider specific functionality and is described in 节 3.1.2.11. Users can take the current interface, which is designed as an extensible framework, and quickly modify the interface to add their own functionality for their end product development.
  - **appClient**: This application interfaces with the Linux collector application over the socket interface to enable connection with the TI-15.4 Stack network. The interface is described in 节 3.1.2.9.
- **TI 15.4-Stack Linux Collector Example Application**: This application implements an example application that starts the network, allows new devices to join the network, configures the joining devices on how often to report the sensor data, configures how often to poll for buffered messages in case of non-beacon and frequency-hopping mode of network operation for sleepy network devices, and tracks connected devices to determine if they are active or inactive on the network. This determination is achieved by the collector periodically sending tracking request messages and awaiting corresponding tracking response messages.
  - **TI 15.4-Stack appServer**: The collector application also opens up a socket server to talk to the *iot-gateway* application. The communication over the socket is implemented through a serialization protocol known as Protocol Buffers or ProtoBuf. Protocol Buffers are a language-independent and platform-neutral way of serializing structured data for efficient data communication. The proto files are defined in the /example/collector/ directory for the collector application. For the iot-gateway application, the protocol file is at /example/iot-gateway/appClient/protofiles. The definitions in these proto files should match for successful communication. For details on protocol buffers, see Protocol Buffers. The interface between the collector application and the iot-gateway application is described in 节 3.1.2.9.
- **MAC CoP Application**: The MAC coprocessor application runs on the CC1310 or CC1350 LaunchPad™, which provides a UART-based interface from TI 15.4-Stack sensor to cloud IoT gateway SDK.
- **CC1310 LaunchPad Sensor End Node**: The sensor example application from TI 15.4-Stack and runs on the CC1310 LaunchPad.
- **CC13xx SensorTag**: The CC13xx SensorTag runs the sensor example application ported from the TI 15.4-Stack out-of-box sensor example applications, which enable support of the CC1350 SensorTag platform and integrate support for various sensors on the SensorTag platform.

2.2 Highlighted Products

This section highlights key hardware devices and software components used in the reference design.

2.2.1 AM335x

The AM335x processors, based on the ARM® Cortex®-A8 core, are enhanced with image, graphics processing, peripherals, and industrial interface options, such as EtherCAT® and PROFIBUS®.

These devices support high-level operating systems (HLOS) such as Linux, which is available free of charge from TI. The AM335x processors contain the subsystems shown in 图 3. The microprocessor unit (MPU) subsystem is based on the ARM Cortex-A8 core and the PowerVR SGX™ graphics accelerator subsystem provides 3D graphics acceleration to support display and gaming effects.

The PRU-ICSS is separate from the ARM core, which allows independent operation and clocking for greater efficiency and flexibility. The programmable real-time unit subsystem and industrial communication subsystem (PRU-ICSS) enables additional peripheral interfaces and real-time protocols such as EtherCAT, PROFINET®, EtherNet/IP, PROFIBUS, Ethernet Powerlink, Sercos, and others.

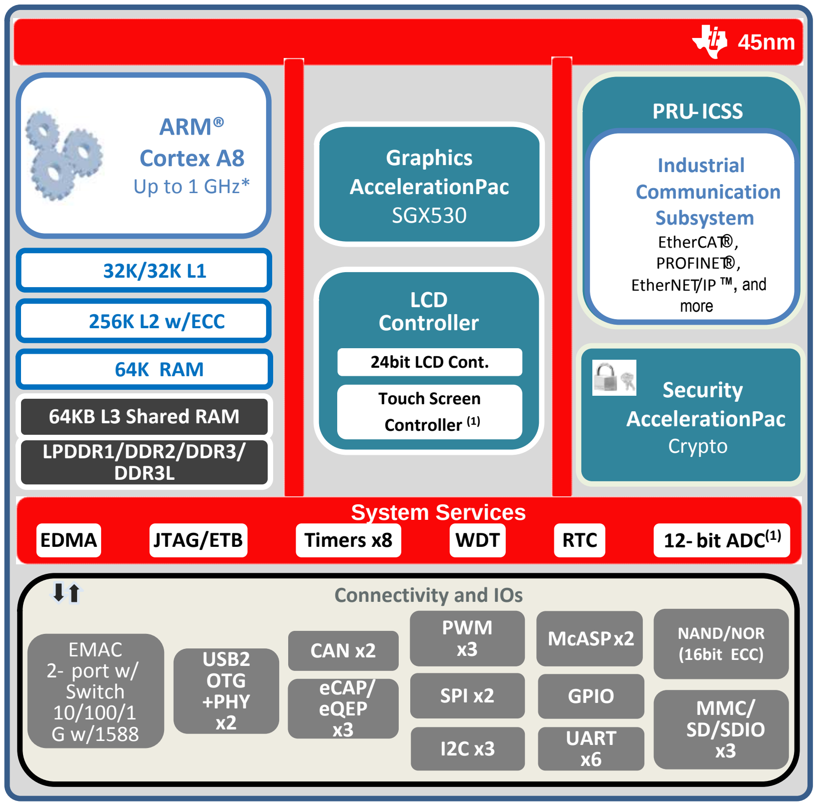


图 3. AM335x Block Diagram

Additionally, the programmable nature of the PRU-ICSS, along with its access to pins, events, and all system-on-chip (SoC) resources, provides flexibility in implementing fast, real-time responses, specialized data handling operations, custom peripheral interfaces, and in offloading tasks from the other processor cores of the SoC.

2.2.2 SimpleLink™ Ultra-Low-Power CC1310 or CC1350

The CC1350 is a member of the CC26xx and CC13xx family of cost-effective, ultra-low-power, 2.4-GHz and Sub-1 GHz RF devices. Very-low active RF and microcontroller (MCU) current consumption, in addition to flexible low-power modes, provide excellent battery lifetime and allow long-range operation on small coin-cell batteries and in energy-harvesting applications.

The CC1350 is the first device in the CC13xx and CC26xx family of cost-effective, ultra-low-power wireless MCUs capable of handling both Sub-1 GHz and 2.4 GHz RF frequencies. The CC1350 device combines a flexible, very-low-power RF transceiver with a powerful 48-MHz Cortex-M3 MCU in a platform supporting multiple physical layers and RF standards. A dedicated radio controller (Cortex-M0) handles low-level RF protocol commands that are stored in ROM or RAM, thus, ensuring ultra-low power and flexibility to handle both Sub-1 GHz protocols and 2.4-GHz protocols [for example *Bluetooth*® low energy (BLE)]. This enables the combination of a Sub-1 GHz communication solution that offers the best possible RF range together with a BLE smartphone connection that enables great user experience through a phone application. The Sub-1 GHz-only device in this family is the CC1310.

The CC1350 device is a highly-integrated, true single-chip solution, which incorporates a complete RF system and an on-chip DC/DC converter.

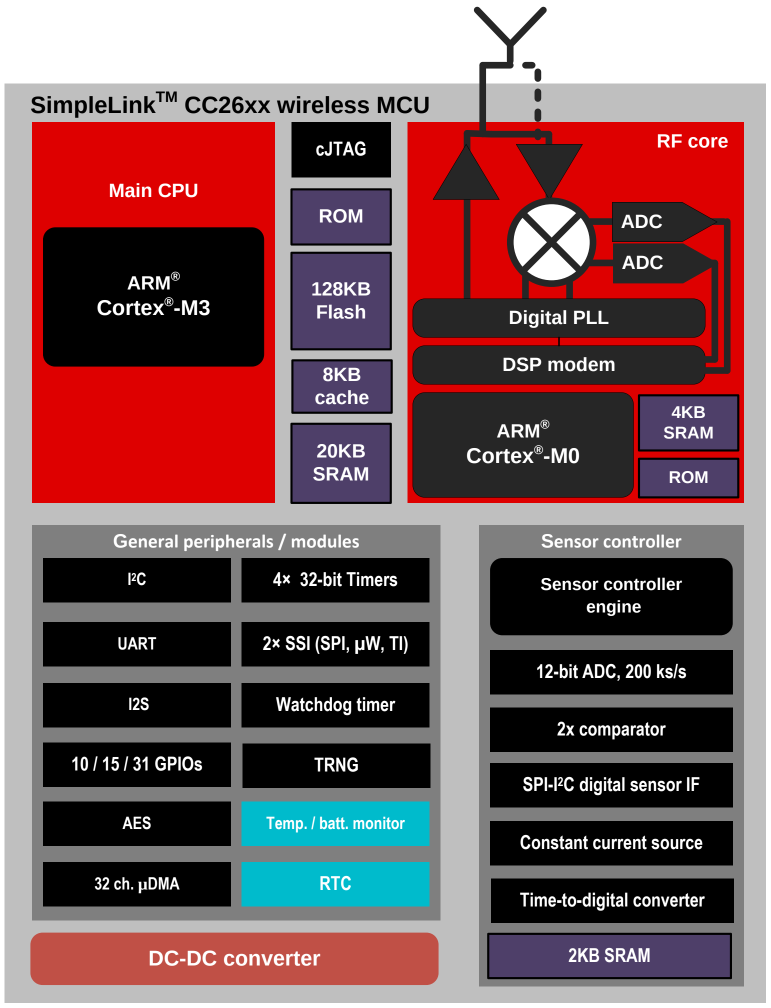


图 4. CC1350 Block Diagram

Sensors can be handled in a very low-power manner by a dedicated autonomous ultra-low-power MCU that can be configured to handle analog and digital sensors; thus, the main MCU (Cortex-M3) can maximize sleep time.

2.2.3 TI 15.4-Stack

TI 15.4-Stack is an IEEE802.15.4e/g-based software stack part of the SimpleLink CC13x0 SDK supporting a Star network topology for Sub-1 GHz applications. TI 15.4-Stack software runs on TI's SimpleLink Sub-1 GHz CC1310 or CC1350 wireless MCU. TI 15-4 Stack offers several key benefits such as longer range in FCC band and better protection against in-band interference by implementing frequency hopping. The SDK also provides customers an accelerated time to market with a complete end-to-end, node-to-gateway solution. TI 15.4-Stack is supported on the industry's lowest-power SimpleLink Sub-1 GHz wireless MCU platform.

This release is available royalty-free to customers using TI's CC1310 or CC1350 wireless MCU and also runs on TI's SimpleLink Sub-1 GHz CC1310 or CC1350 wireless MCU LaunchPad development kit. This release is available royalty-free to customers using TI's CC1310 or CC1350 wireless MCU and also runs on TI's SimpleLink Sub-1 GHz CC1310 or CC1350 wireless MCU LaunchPad development kit.

Features:

- IEEE 802.15.4e/g standards-based solution
- Frequency hopping
- Medium access with CSMA/CA
- Built in acknowledgment and retries
- Network and device management (joining, commissioning, service discovery)
- Security feature through AES -128 encryption and integrity check
- Supported on SimpleLink Sub-1 GHz CC1310 wireless MCU
- Star topology: Point-to-point, one-to-many, and data concentrator
- Synchronous (beacon) and asynchronous (non-beacon) modes
- Designed for 915-MHz FCC, 863-MHz ETSI, and 433-MHz China bands
- SimpleLink long range mode for all supported frequency bands
- Support for SimpleLink CC1190
- Bluetooth low energy beacon advertisement support
- Sensor-to-web example application
- Easy application development guided through sample applications showcasing the stack configuration and APIs
- Coprocessor mode for adding connectivity to any MCU or MPU, with Linux host middleware and console application

For more details and to get the TI 15.4-Stack software, download the *SimpleLink CC13x0 SDK* [1], which includes the TI 15.4-Stack.

2.2.4 TI Processor Linux® SDK for AM335x

The TI processor SDK is a unified software platform for TI embedded processors, which provides easy setup and fast out-of-the-box access to benchmarks and demonstrations. All releases of the processor SDK are consistent across TI's broad portfolio, which allows developers to seamlessly reuse and migrate software across devices. Developing scalable platform solutions has never been easier with the processor SDK and TI's embedded processor solutions.

TI processor Linux SDK highlights:

- Long-term stable (LTS) mainline Linux kernel support
- U-Boot bootloader support
- Linaro GNU compiler collection (GCC) tool chains
- Yocto Project™ OE Core compatible file systems

For more details and to get the processor SDK, see *AM335x Processor SDK* [2].

3 Hardware, Software, Testing Requirements and Test Results

3.1 Required Hardware and Software

This section provides details on required hardware and software to be able to run the out-of-box TI 15.4-Stack sensor-to-cloud reference design software application. Developers can then quickly use the out-of-box application as a framework to develop end products.

3.1.1 Required Hardware

The following hardware is required to get the out-of-box application running and to develop applications.

- A CC1310 or CC1350 LaunchPad to run the MAC coprocessor application
- One or more CC1310 or CC1350 LaunchPads or CC1350 SensorTags to run the TI 15.4-Stack sensor application to create one or more Sub-1 GHz network devices
- An AM335x-based BeagleBone Black board
- An 8-GB microSD card (the TI processor SDK image requires at least 8 GB of space)
- A 5-V power supply for the BeagleBone Black
- An Ethernet cable to connect the BeagleBone Black to the Internet
- An LCD BoosterPack (optional if using the CC1310 or CC1350 LaunchPad as the Sub-1 GHz network device)
- A means to configure and set up the BeagleBone Black microSD card (Windows® or Linux machine)
- A PC to host and run the web browser used to view the web application
- A standard Ethernet router required for internet connectivity to the BeagleBone Black and the host computer or tablet to view the web-application to monitor and control the sensor nodes in the network
- A USB cable to connect the BeagleBone Black with the CC1310 or CC1350 LaunchPad

注: The out-of-box application is demonstrated using a USB cable to connect the AM335x-based BeagleBone Black with the CC1310 or CC1350 LaunchPad. The reference design includes design files for a hardware adapter board that connects the BeagleBone Black with the CC1310 or CC1350 LaunchPad the way an end product should. The adapter board is not available for purchase but customers can either build their own using the design files provided, or they can jump straight to their own form factor design using the adapter board design files as a reference for how to connect the AM335x and CC1310 or CC1350 devices for an end product. When designing an end product, customers must also keep in mind that certificates must be stored in secure memory; therefore, a trusted platform module (TPM) or other means of having secure storage must be included in the end-product design.

图 5 shows the hardware setup to run the demonstration.

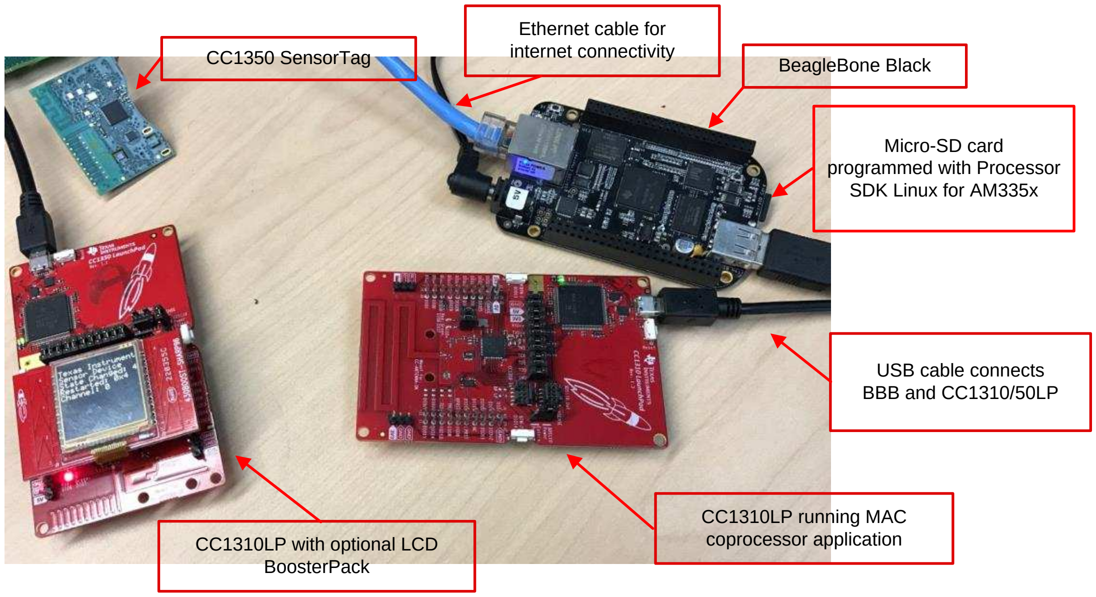

CC1350 SensorTag
Ethernet cable for internet connectivity
BeagleBone Black
Micro-SD card programmed with Processor SDK Linux for AM335x
USB cable connects BBB and CC1310/50LP
CC1310LP running MAC coprocessor application
CC1310LP with optional LCD BoosterPack

图 5. Demonstration Hardware Setup

3.1.2 Required Software

With the required hardware, perform the following steps to replicate the software portion of the demonstration:

1. Boot the Linux kernel and file system from the Linux Processor SDK on the BeagleBone Black.
2. Copy the provided Sub-1 GHz IoT gateway demonstration reference design software to the BeagleBone Black.
3. Program a CC1310 or CC1350 LaunchPad with the provided MAC coprocessor application.
4. Program the remaining CC1310 or CC1350 LaunchPad and CC1350 SensorTags with the provided sensor application.

The following sections in this chapter detail these instructions. For the purposes of this design guide, it is assumed that a Windows host machine is being used.

3.1.2.1 BeagleBone Black SD Card

Program the SD card with the Linux processor SDK image using the following steps:

1. Download the prebuilt TI Linux processor SDK SD card image am335x-evm-linux-xx.xx.xx.xx.img.zip from http://software-dl.ti.com/processor-sdk-linux/esd/AM335X/latest/index_FDS.html (where xx.xx.xx.xx is the version number of the latest Linux Processor SDK).
2. To program the microSD memory card, see the instructions at *Processor SDK Linux Creating an SD Card with Windows* [4].

3.1.2.2 Booting BeagleBone Black

Boot the BeagleBone Black from the microSD card using the following steps:

1. Disconnect power and unplug the USB cable from the BeagleBone Black board.
2. Insert the microSD card into the BeagleBone Black (see 图 6).
3. Press and hold the boot switch (S2).
   - Important: The boot switch is detected only at initial power on.
4. Provide power to the BeagleBone Black (1.5 A, 5 V).
5. Wait a few seconds then release the boot switch. In about 5 to 15 seconds, the LEDs begin to blink.

3.1.2.2.1 Configuring BeagleBone Black With Wireless Connectivity Cape (Wi-Fi Optional)

A few extra steps must be taken after booting the BeagleBone Black for the first time to enable use of the Element14 Wireless Connectivity Cape.

1. Clone the Sensor To Cloud repository to the BeagleBone Black
2. On the BeagleBone run the *setup_beaglebone.sh* script. This script will prompt for various setting and configure the BeagleBone with the correct pins upon reboot. After reboot the BeagleBone broadcasts a Wi-Fi network (SSID and password are configured using the setup_beaglebone script).
3. For more information on configuring the Wireless Cape can be found at *Using the WL18xx Cape with BeagleBone Black*.

注: The first boot from a freshly-formatted SD card takes about one to two minutes longer. During this extended time, the BeagleBone Black Linux distribution performs some one-time-only steps.

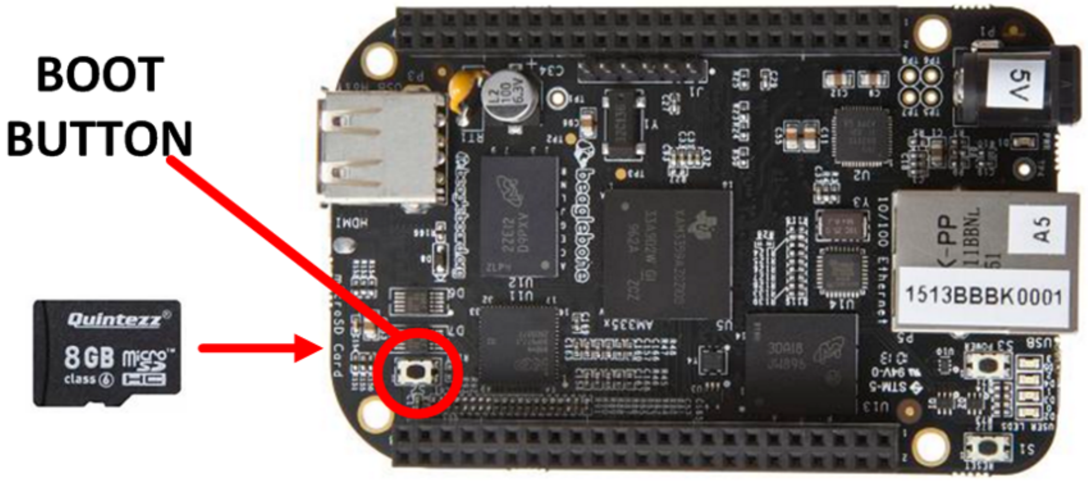
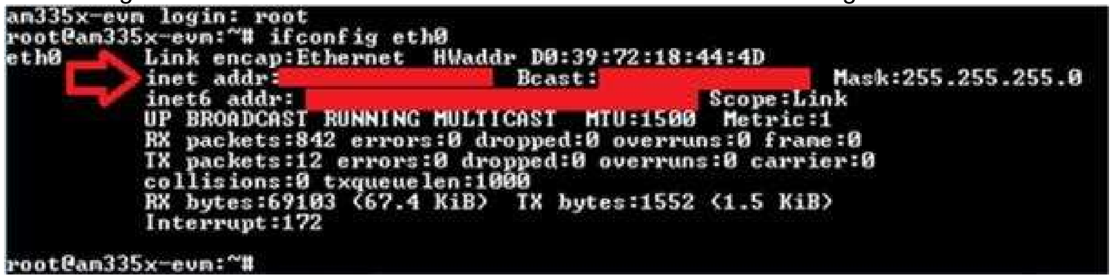


图 6. Boot BeagleBone Black From SD Card

3.1.2.3 Determining BeagleBone Black Network Address

In order to transfer files to the BeagleBone Black using its network interface, it is necessary to find its network address (IP address). There are two methods to determine the IP address of the BeagleBone Black:

- Method 1: Use the FTDI cable to connect through the serial header on the BeagleBone Black, and use the ifconfig command to determine the IP address allocated to the BeagleBone Black.



```
am335x-evm login: root
root@am335x-evm:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr D0:39:72:18:44:4D
          inet addr:[redacted]  Bcast:[redacted]  Mask:255.255.255.0
          inet6 addr:[redacted]  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:842 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:69103 (67.4 KiB)  TX bytes:1552 (1.5 KiB)
          Interrupt:172

root@am335x-evm:~#
```

- Method 2: Most routers include a built-in web server to configure the device (see 表 1).
  – Connect the BeagleBone Black to the router.
  – Boot the BeagleBone Black.
  – Find the DHCP client page to determine the IP address of the BeagleBone Black. Some examples follow. The generic name for this feature is the DHCP client table.

注: Troubleshooting—the DHCP IP address is often determined by the order in which the devices boot. If the user's laptop booted first, it may receive address: xx.xx.xx.100. The BeagleBone Black boots second and receives the address: xx.xx.xx.101; however, on the next use or if another device is attached (for example, a cell phone or tablet), the resulting boot order may change, and therefore, the IP address might change.

表 **1. Commercial Routers**

BRAND	EXAMPLE LINK
LINKSYS™	http://www.linksys.com/us/support-article?articleNum=139502
NETGEAR®	http://documentation.netgear.com/fvs336g/enu/202-10257-01/FVS336G_RM-11-07.html
BELKIN™	http://www.belkin.com/pyramid/AdvancedInfo/F5D8235-4/Advance/reserveIP.htm

3.1.2.4 Get Sub-1 GHz IoT Gateway Demonstration Software

The Sub-1 GHz sensor to cloud Industrial IoT gateway reference design demonstration software is located on a Git repository found at https://git.ti.com/apps/tidep0084. Clone the repository to the host machine and copy it over to the BeagleBone Black using secure copy (SCP).

- On the Windows® host machine:
  – cd C:\path\to\desired\clone\directory\
  – git clone git://git.ti.com/apps/tidep0084.git tidep0084
  – Use WinSCP, Teraterm, or FileZilla to copy the *tidep0084* directory to the BeagleBone Black using the network address found earlier.

3.1.2.5 Logging in to BeagleBone Black Using Secure Shell (SSH) Protocol

Putty or TeraTerm can be used (along with the IP address found in 节 3.1.2.3) to connect to the BeagleBone Black using SSH. The user name is *root*, and there is no password. Once connected, the root user will be logged into the board and the Linux console prompt will appear.

3.1.2.6 Programming CC1310 or CC1350 LaunchPad™

To run the example application users must first program one CC1310 or CC1350 LaunchPad with the MAC CoP hex file and the other LaunchPads with the sensor example application hex file. In this design guide, the Flash Programmer 2 tool running on a Windows machine is used. Developers can also use the Serial Flash Programmer tool, described in the *CC13x0 SimpleLink TI 15.4-Stack 2.x.x Linux Developer's Guide* [5] to program the desired hex image onto the CC1310 or CC1350 LaunchPad.

注: It is easy to confuse the sensor and CoP devices. Be sure to label the devices as they are programmed.

To program the LaunchPad or SensorTag, follow these steps:

1. Download and install the SmartRF Flash Programmer 2.
2. Program CC1310 or CC1350 LaunchPad 1 – this device runs the CoP example application.
   a. Label this device collector. The LCD BoosterPack is not supported in the CoP application.
   b. If using the CC1310LP as MAC coprocessor: From a Windows PC, use the SmartRF Flash Programmer 2 to program a CC1310 LaunchPad MAC CoP with the coprocessor_cc1310_lp.hex file located here:
   {demo software clone directory}/firmware/CC1310_LAUNCHXL/coprocessor_cc1310_lp.hex
   c. If using the CC1350LP as MAC coprocessor: From a Windows PC, use the SmartRF Flash Programmer 2 to program a CC1350 LaunchPad MAC CoP with the coprocessor_cc1350_lp.hex file located here:
   {demo software clone directory}/firmware/CC1350_LAUNCHXL/coprocessor_cc1350_lp.hex
3. Program CC1310 or CC1350 LaunchPad 2 or SensorTag – this device runs the sensor example application.
   a. Label this device sensor. Optional: connect the LCD BoosterPack to this LaunchPad.
   b. To program CC1310 LaunchPad: From a Windows PC, use the SmartRF Flash Programmer 2 to program the hex file sensor_cc13x0lp.hex file located here:
   {demo software clone directory}/firmware/ CC1310_LAUNCHXL/sensor_cc13x0lp.hex
   c. To program CC1350 LaunchPad: From a Windows PC, use the SmartRF Flash Programmer 2 to program the hex file sensor_cc13x0lp.hex file located here:
   {demo software clone directory}/firmware/ CC1350_LAUNCHXL/sensor_cc13x0lp.hex
   d. To program CC1350 SensorTag: From a Windows PC, use the SmartRF Flash Programmer 2 to program the hex file sensor_cc13x0stk.hex file located here:
   {demo software clone directory}/firmware/ CC1350_SensorTag/sensor_cc13x0stk.hex

---

注: Important—the default hex files are built for 915-MHz, 863-MHz, and 433-MHz bands of operation at 50 kbps. To rebuild the hex files for other bands (for example, SimpleLink Long Range mode), see the *CC13x0 SimpleLink TI 15.4-Stack 2.x.x Embedded Developer's Guide* [6] or *TI 15.4-Stack CC13x0 SimpleLink Embedded Applications Quick Start Guide* [7]. See the *CC13x0 SimpleLink TI 15.4-Stack 2.x.x Linux Developer's Guide* [5], specifically the Example Collector Application configuration section, to change the Linux example application.

To change the band of operation of the CoP, configure collector.cfg. For more information on configuring the CoP, refer to *CC13x0 SimpleLink TI 15.4-Stack 2.x.x Linux Developer's Guide* [5].

For porting the out-of-box TI 15.4-Stack sensor application, which is supported on the LaunchPad platform to the CC1350 SensorTag platform, see the *TI 15.4-Stack Wiki* [3].

Troubleshooting—If the devices (sensor or CoP) get mixed up, use the Flash Programmer 2 tool to verify the flash content. Uncheck the ERASE option, uncheck the PROGRAM option, and enable the VERIFY option along with the read-back feature, to double-check or double-verify the flash operation.

3.1.2.7 Running the Demonstration

With the required hardware and software together, the demonstration can be completed. At this point, the following assumptions are made:

- The BeagleBone Black is booted from the SD card using the latest kernel and file system from the Linux processor SDK.
- The BeagleBone Black is powered up, and the user is logged in using SSH and can send commands on the Linux console.
- The Git repository containing the demonstration software was copied to the file system of the BeagleBone Black.
- The coprocessor LaunchPad has been programmed with the coprocessor firmware.
- The remaining LaunchPads and SensorTags have been programmed with the sensor example application.

If any of these assumptions are not true at this point, return to the previous corresponding sections in this chapter.

3.1.2.7.1 Connecting CC1310 or CC1350 LaunchPad™ Coprocessor

Plug the CC1310 or CC1350 LaunchPad running the coprocessor application into the BeagleBone Black using the USB cable. In 图 7, the USB connection on the right side of the image is connected to the CC1310 or CC1350 LaunchPad coprocessor.

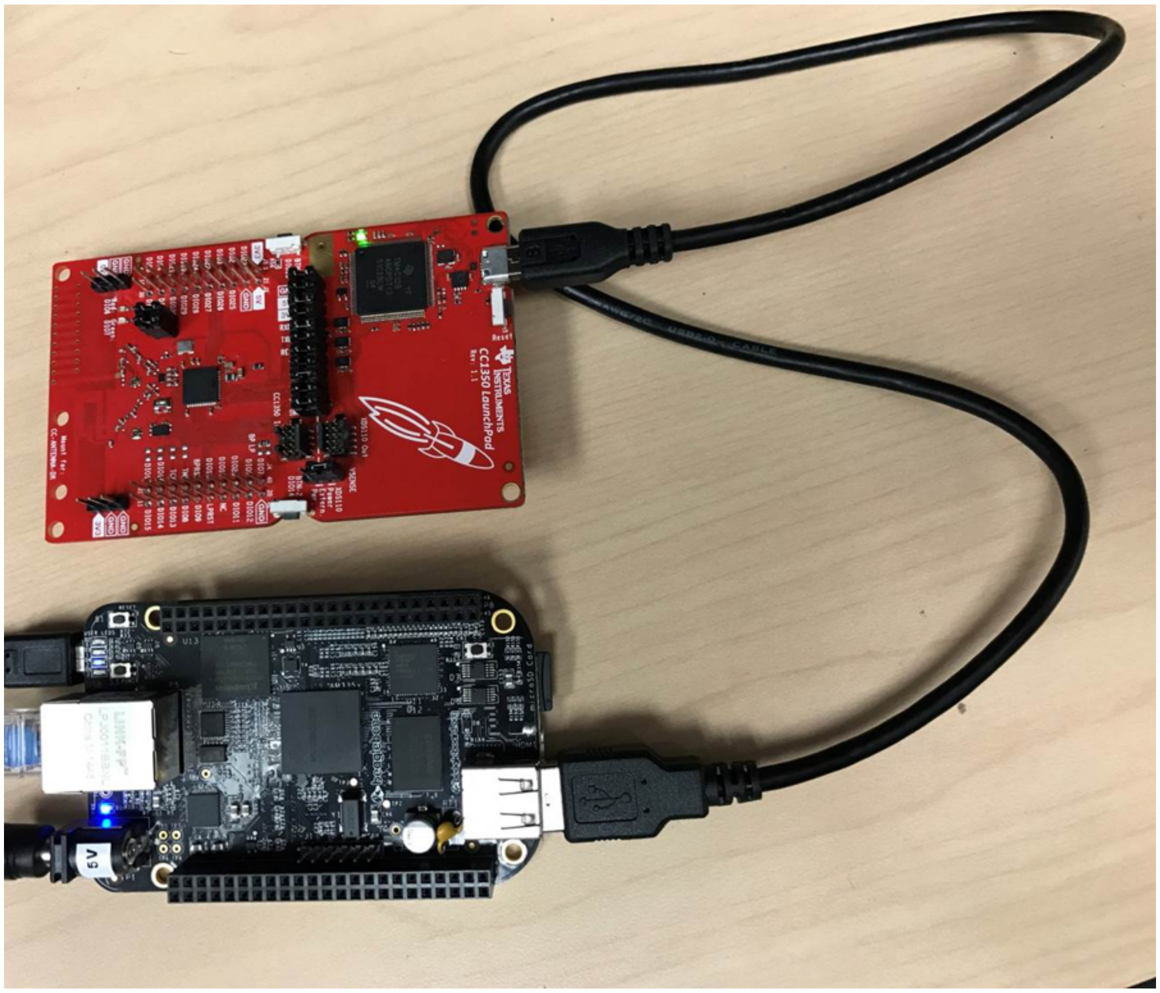
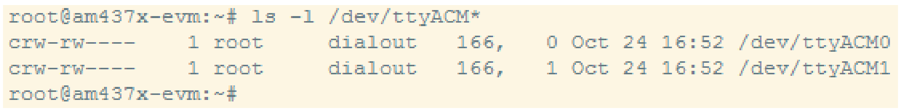


图 7. Coprocessor LaunchPad™ Connected to BeagleBone Black

Once this connection is made, type

*ls -l /dev/ttyACM**

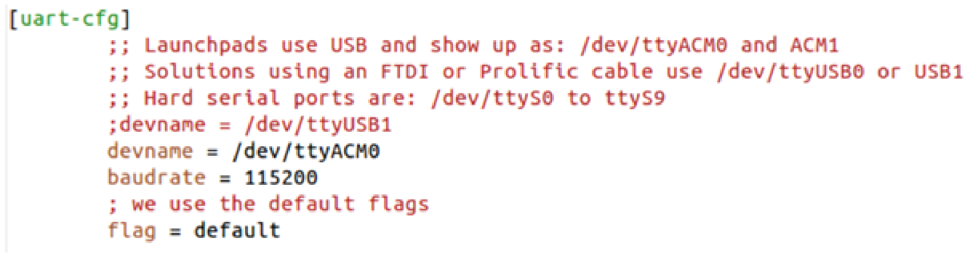
at the BeagleBone Black console. There are two ttyACM devices that correspond to the serial ports from the CC1310 or CC1350 LaunchPad (similar to 图 8).



```
root@am437x-evm:~# ls -l /dev/ttyACM*
crw-rw----    1 root     dialout   166,   0 Oct 24 16:52 /dev/ttyACM0
crw-rw----    1 root     dialout   166,   1 Oct 24 16:52 /dev/ttyACM1
root@am437x-evm:~#
```

图 8. /dev/ttyACM0 Device Check

Make sure /dev/ttyACM0 shows up in the list. This is the UART connection between the BeagleBone Black and the CC1310 or CC1350 LaunchPad device over which all of the sensor and network information is transferred. Open the {demo software directory}/prebuilt/bin/collector.cfg file and double check that the [uart-cfg] section for the collector application is pointing to the correct device, as shown in 图 9.



```
[uart-cfg]
        ;; Launchpads use USB and show up as: /dev/ttyACM0 and ACM1
        ;; Solutions using an FTDI or Prolific cable use /dev/ttyUSB0 or USB1
        ;; Hard serial ports are: /dev/ttyS0 to ttyS9
        ;devname = /dev/ttyUSB1
        devname = /dev/ttyACM0
        baudrate = 115200
        ; we use the default flags
        flag = default
```

图 9. UART Configuration

3.1.2.7.2 AWS Certificates and Configuration From stackArmor

To connect the IoT Gateway to the AWS IoT service, the gateway needs authentication certificates provisioned by AWS. For these certificates, as well as a unique AWS URL, see the stackArmor web page[8]. Return to this guide once obtaining the following:

- certificate.pem.crt
- private.pem.key
- public.pem.key
- root-CA.crt
- a URL to the AWS host that should be used

Use SCP to copy the four files into the {demo software directory}/prebuilt/iot-gateway/cloudAdapter/certs/ directory on the BeagleBone Black’s file system.

Open the {demo software directory}/prebuilt/iot-gateway/cloudAdapter/awsConfig.json file and do the following:

- **certDir** - Make sure that the *certDir* parameter is set to the correct path to the certs directory where the four files were copied.
- **host** - Set the *host* parameter equal to the URL that was provided by stackArmor.
- **region** - Make sure that the *region* parameter matches the region portion of the URL. It should be something similar to *us-east-1*.
- **clientId** - The *clientId* parameter must be changed to a unique string. Only one connection to the AWS

cloud from a specific *clientId* is allowed. If the same *clientId* is used by more than one device connecting to the AWS cloud, connectivity issues can occur as the connection may timeout or be refused.

3.1.2.7.3 Starting the Application

The {demo software directory}/prebuilt/ directory has a simple shell script called *run_demo.sh*. Type the following at the BeagleBone Black console to run the IoT Gateway application:

cd {demo software directory}/prebuilt/

chmod +x bin/bbb_collector

bash run_demo.sh

The shell script will start all of the necessary programs in order to get the full demonstration application up and running. The script will also print the URL to the IoT dashboard to the console. Navigate to the URL from the console output using the web browser on the host machine.

3.1.2.7.3.1 Common Issues

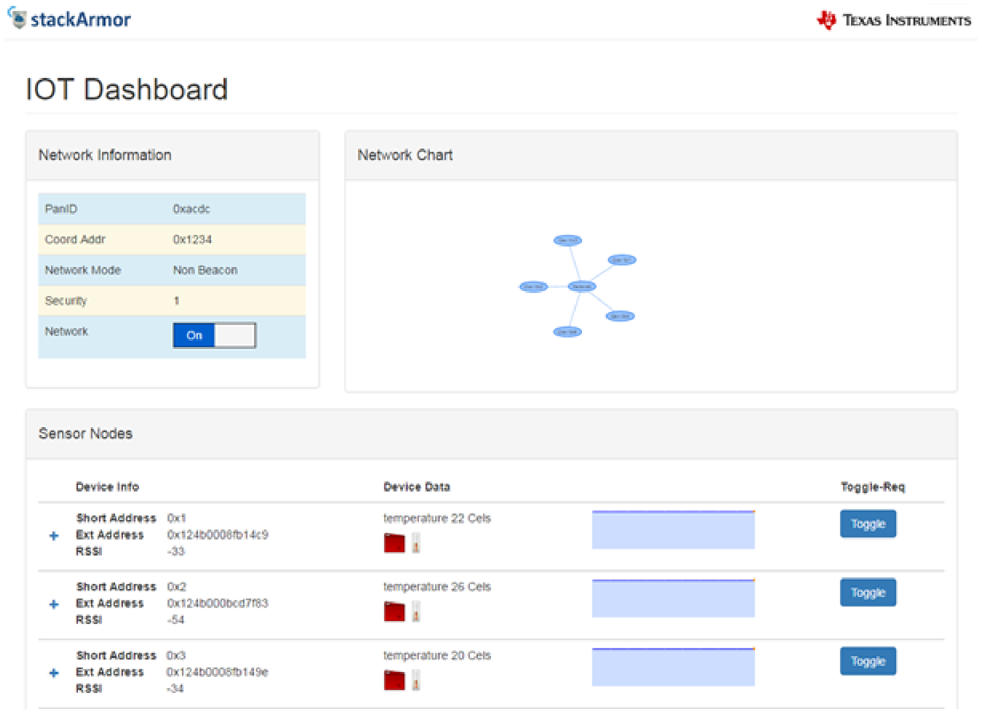
The following is a list of common issues that might be seen while starting or running the application:

- **Error: Rcvd Error on the socket connection with AppServer (ECONNREFUSED 127.0.0.1:5000)**
  This error occurs when the AppClient (which is started by the IoT Gateway) is not able to make a local socket connection with the AppServer (which is started by the bbb_collector). Make sure that the bbb_collector application is up and running before starting the IoT Gateway. The run_demo.sh script in the prebuilt directory gives an example on how to start the demonstration in the correct order. This script starts up the bbb_collector application and then starts the IoT Gateway.
- **Error: getaddrinfo ENOTFOUND <your unique AWS URL> (AWS Cloud Adapter error)**
  This error can happen if the BeagleBone Black is behind a firewall and cannot connect to the servers at the AWS URL. This issue can be resolved by using a mobile hotspot to connect the BeagleBone Black to the Internet or possibly by configuring the local network settings to allow the BeagleBone Black to access outside servers.
- **Error: certificate is not yet valid (AWS Cloud Adapter error)**
  This issue can occur if the date and time on the BeagleBone Black are set incorrectly to a time before the AWS certificates were generated by stackArmor. Setting the date of the BeagleBone Black to the current date and time should resolve this issue.
- **Removing Sensor Nodes from the Sub-1 GHz Wireless Network**
  The current demonstration does not provide a method in the user interface to remove sensor nodes from the Sub-1 GHz wireless network. The bbb_collector application uses a file named *nv-simulation.bin* (that can be found in the *prebuilt/bin/* directory) to save the information of the sensor nodes that have connected to the Sub-1 GHz wireless network. Delete the *nv-simulation.bin* file and restart the demonstration in order to remove sensor nodes. This process also means the remaining sensor nodes must reconnect to the Sub-1 GHz wireless network before they will show up in the user interface again.
- **Error: Cannot find module *moduleName* (or any other Node-JS error)**
  Cloning the TIDEP0084 Git repository to a Windows machine and then copying it to the BeagleBone Black might produce Node-JS errors when starting the demonstration. These errors appear to be caused by the line endings in the repository getting changed by Windows before being copied to the

BeagleBone Black. To correct this issue, the TIDEP0084 Git repository can be cloned directly to the BeagleBone Black. To accomplish this, make sure your BeagleBone Black has an internet connection and then run the following command from the terminal:
git clone git://git.ti.com/apps/tidep0084.git.

3.1.2.7.4 IoT Dashboard Web Page—Open Network for New Device Joins

图 10 shows the IoT dashboard provided by stackArmor. Navigate to the dashboard by following the URL provided by the console output in the previous step. Initially, the application starts with no devices present (not shown), the network is closed to new devices joining (not shown), and the network will not accept new devices.



IOT Dashboard

Network Information

PanID	0xacdc
Coord Addr	0x1234
Network Mode	Non Beacon
Security	1
Network	On

Sensor Nodes

	Device Info		Device Data	Toggle-Req
+	Short Address Ext Address RSSI	0x1 0x124b0008fb14c9 -33	temperature 22 Cels	Toggle
+	Short Address Ext Address RSSI	0x2 0x124b000bcd7f83 -54	temperature 26 Cels	Toggle
+	Short Address Ext Address RSSI	0x3 0x124b0008fb149e -34	temperature 20 Cels	Toggle

图 **10. IoT Dashboard**

3.1.2.7.5 Joining the Sensor Devices to the Network

At start up, the collector example application initially has the network closed; therefore, sensor devices cannot join. To open the network, switch the *On/Off* button on the web browser to the *On* state. Within a few seconds (time depends on the polling interval and other configuration settings), the sensor joins the network. When the device joins the network, the red LED turns on. If the sensor LaunchPad has an LCD module, the device indicates the current state on the LCD. See 图 11.

- State 1 = Not joined
- State 3 = Joined
- State 4 = Restored
- State 5 = Orphan condition

More details can be found in the *CC13x0 SimpleLink TI 15.4-Stack 2.x.x Embedded Developer's Guide* [6].

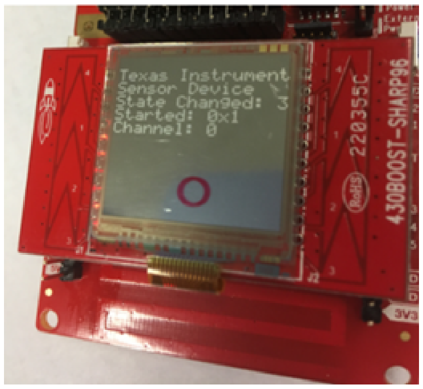

```
Texas Instrument
Sensor Device
State Changed: 3
Started: 0x1
Channel: 0
```

图 11. Sensor LaunchPad™ State Change LCD

3.1.2.7.6 Data Communication

After the new device appears, initially only the short and extended addresses appear. The data fields will not show any data as none have been reported yet.

Sensor Data Reports:
After about one minute data appears on the screen (the exact interval is configured in the collector application using a #define value), see the *CC13x0 SimpleLink TI 15.4-Stack 2.x.x Embedded Developer's Guide* [6] or the Linux example collector source code for more details. After this time, the sensor nodes periodically report the sensor data.

Actuation:
Clicking on the toggle LED button sends a message to the sensor module to toggle the LED. There may be a slight delay (a few seconds) in toggle operation on the desired sensor LaunchPad. This delay is because the sensor nodes are in sleep mode and only wake up periodically to get the command buffered on the collector.

See 图 10 for an example of the IoT dashboard with multiple sensors and reported data.

3.1.2.8 Interactive GUI

TI's Sensor-to-Cloud design now includes a web interface that can be used to setup, connect to a network, and launch a cloud connected gateway. Wireless functionality is enabled with the use of Element14 wireless cape, which allows the user to easily get a gateway up and running without knowing Linux, networking, or terminal commands. Refer to the *TI 15.4-Stack CC13x0 SimpleLink Embedded Applications Quick Start Guide*[7] for more information on getting started with the Sensor-to-Cloud web interface.

3.1.2.9 IoT-Gateway and Collector Application Interface API

The purpose of this section is to provide a description of the application programming interface between the TI 15.4-Stack Linux collector example application and the IoT gateway application. The collector example application implements an appsrv module, which opens up a server socket to which a client application can connect. The interface allows management and data interface to the client application, connecting to the socket server, to monitor and control the TI 15.4-Stack-based network. Management functionalities include the ability to open and close the network for new device joins, whereas the data interface allows sending and receiving data to and from the network devices. It is easy to add new APIs or modify the current implementation.

This API is defined at a specific interface level, which is a TCP socket pipe.

API commands and parameters are serialized over the socket interface using Google® protocol buffers (protobuf; for details, see Protocol Buffers). The names of some parameters in this API section do not reflect the true name of the parameter, which is used by the application using that API, as those parameters are automatically generated by the protobuf engine.

For transport using a TCP socket, the protobuf-packed packets are preceded by a 4-byte header, containing the following fields (in this order):

1. **len** – 16-bit number that specifies the actual length (in bytes) of the protobuf-packed packet
2. **Subsystem** – 1 byte: specifies the subsystem to or from which the packet is sent or received. The value '10' is reserved for TI 15.4-Stack application server interface.
3. **cmd_id** – 1 byte: The command ID of the actual command being sent. This value is also available inside the packed packet. The actual command ID numbers are provided in the protobuf definition files that are part of the TI 15.4-Stack Linux SDK (collector example application and the gateway example application). When using command IDs in code, always use the defined names (never hardcode the command ID numbers), as the numbers may change between releases.

3.1.2.9.1 Management Interface

3.1.2.9.1.1 APPSRV_SET_JOIN_PERMIT_REQ

3.1.2.9.1.1.1 Description

Allows client application to enable or disable network for join for new devices.

3.1.2.9.1.1.2 Parameter List

表 **2. APPSRV_SET_JOIN_PERMIT_REQ Parameter List**

PARAMETER	TYPE	DESCRIPTION
Duration	INT32	Duration for join permit to be turned on in milliseconds: • 0 sets the join permit pff, and 0xFFFFFFFF sets the join permit on indefinitely. • Any other non-zero value sets the join permit on for that duration.

3.1.2.9.1.2 APPSRV_SET_JOIN_PERMIT_CNF

3.1.2.9.1.2.1 Description

The application server notifies the client of the result of processing of permit join request message.

3.1.2.9.1.2.2 Parameter List

表 3. APPSRV_SET_JOIN_PERMIT_CNF Parameter List

PARAMETER	TYPE	DESCRIPTION
Status	INT32	0 if success

3.1.2.9.1.3 APPSRV_NWK_INFO_IND

3.1.2.9.1.3.1 Description

The application server notifies the client of the network information when a network is formed using this API.

3.1.2.9.1.3.2 Parameter List

表 4. APPSRV_NWK_INFO_IND Parameter List

PARAMETER	TYPE	DESCRIPTION	
Fh	UINT32	True if network is frequency hopping	
channel	UINT32	Channel number used, if non-frequency hopping network configuration	
panID	UINT32	The 16-bit PAN identifier of the network	
shortAddress	UINT32	The 16-bit short address of the PAN coordinator	
extAddress	INT64	The 64-bit IEEE extended address of the PAN coordinator device	
securityEnabled	INT32	*true* if security enabled, *false* otherwise	

nwkMode	ENUM	Network operation mode
		BEACON_ENABLED = 1
		NON_BEACON = 2
		FREQUENCY_HOPPING = 3	

state	ENUM	PAN coordinator state values	
		STATE	**VALUE**
		Initialized waiting for user to start	1
		Starting coordinator	2
		Restoring coordinator (from NV)	3
		Started	4
		Restored	5
		Joining allowed for new devices	6
		Joining not allowed for new devices	7

3.1.2.9.1.4 APPSRV_GET_NWK_INFO_REQ

3.1.2.9.1.4.1 Description

The application server's client can use this API to get the current network information

3.1.2.9.1.4.2 Parameter List

There is no parameter in the command message.

3.1.2.9.1.5 APPSRV_GET_NWK_INFO_CNF

3.1.2.9.1.5.1 Description

The application server sends the current network information as a response to the get network information request from the client using this API.

3.1.2.9.1.5.2 Parameter List

表 5. APPSRV_GET_NWK_INFO_CNF Parameter List

PARAMETER	TYPE	DESCRIPTION	
Status	INT32	0 if success	
Fh	UINT32	True if network is frequency hopping (optional)	
channel	UINT32	Channel number used, if non-frequency hopping network configuration (optional)	
panID	UINT32	The 16-bit PAN identifier of the network (optional)	
shortAddress	UINT32	The 16-bit short address of the PAN coordinator (optional)	
extAddress	INT64	The 64-bit IEEE extended address of the PAN coordinator device (optional)	
securityEnabled	INT32	*true* if security enabled, *false* otherwise (optional)	

nwkMode	ENUM	Network operation mode (optional)
		BEACON_ENABLED = 1
		NON_BEACON = 2
		FREQUENCY_HOPPING = 3	

state	ENUM	PAN coordinator state values (optional)	
		STATE	**VALUE**
		Initialized waiting for user to start	1
		Starting coordinator	2
		Restoring coordinator (from NV)	3
		Started	4
		Restored	5
		Joining allowed for new devices	6
		Joining not allowed for new devices	7

3.1.2.9.1.6 APPSRV_GET_DEVICE_ARRAY_REQ

3.1.2.9.1.6.1 Description

The application client requests the current list of connected device using this API.

3.1.2.9.1.6.2 Parameter List

There is no parameter in the command message.

3.1.2.9.1.7 APPSRV_GET_DEVICE_ARRAY_CNF

3.1.2.9.1.7.1 Description

The application server sends the current list of connected device as a response to the get device array request message using this API.

3.1.2.9.1.7.2 Parameter List

表 **6. APPSRV_GET_DEVICE_ARRAY_CNF Parameter List**

PARAMETER	TYPE	DESCRIPTION
Status	INT32	0 if success
devInfo	Csf_deviceInformation	Multiple entries of this structure element. Number of entries is equal to the number of connected devices in the network.
panID	UINT32	The 16-bit PAN identifier of the network
shortAddress	UINT32	The 16-bit short address of the network device
extAddress	INT64	The 64-bit IEEE extended address of the network device
panCoord	UINT32	True if the device is PAN coordinator
ffd	UINT32	True if the device is a full function device
mainsPower	UINT32	True if the device is mains powered
rxOnWhenIdle	UINT32	True if the device's RX is on when the device is idle
security	UINT32	True if the device is capable of sending and receiving secured frames
allocAddr	UINT32	True if allocation of a short address in the associate procedure is needed.

3.1.2.9.1.8 APPSRV_DEVICE_JOINED_IND

3.1.2.9.1.8.1 Description

The application server informs the client of a new device join in the network using this API.

3.1.2.9.1.8.2 Parameter List

表 **7. APPSRV_DEVICE_JOINED_IND Parameter List**

PARAMETER	TYPE	DESCRIPTION
panID	UINT32	The 16-bit PAN identifier of the network
shortAddress	UINT32	The 16-bit short address of the network device
extAddress	INT64	The 64-bit IEEE extended address of the network device
panCoord	UINT32	True if the device is PAN coordinator
ffd	UINT32	True if the device is a full function device
mainsPower	UINT32	True if the device is mains powered
rxOnWhenIdle	UINT32	True if the device's RX is on when the device is idle
security	UINT32	True if the device is capable of sending and receiving secured frames
allocAddr	UINT32	True if allocation of a short address in the associate procedure is needed.

3.1.2.9.1.9 APPSRV_DEVICE_NOTACTIVE_UPDATE_IND

3.1.2.9.1.9.1 Description

The application server informs the client of an inactive device using this API.

3.1.2.9.1.9.2 Parameter List

表 8. APPSRV_DEVICE_NOTACTIVE_UPDATE_IND Parameter List

PARAMETER	TYPE	DESCRIPTION
panID	UINT32	The 16-bit PAN identifier of the network
shortAddress	UINT32	The 16-bit short address of the network device
extAddress	INT64	The 64-bit IEEE extended address of the network device
timeout	UINT32	True if not active because of tracking timeout. meaning that the device didn't respond to the tracking request within the timeout period.

3.1.2.9.1.10 APPSRV_COLLECTOR_STATE_CNG_IND

3.1.2.9.1.10.1 Description

The application server informs the client of change in the state of the collector application using this API.

3.1.2.9.1.10.2 Parameter List

表 9. APPSRV_COLLECTOR_STATE_CNG_IND Parameter List

PARAMETER	TYPE	DESCRIPTION	
		STATE	VALUE
state	ENUM	Initialized waiting for user to start	1
		Starting coordinator	2
		Restoring coordinator (from NV)	3
		Started	4
		Restored	5
		Joining allowed for new devices	6
		Joining not allowed for new devices	7

3.1.2.9.2 Data Interface

3.1.2.9.2.1 APPSRV_DEVICE_DATA_RX_IND

3.1.2.9.2.1.1 Description

The application server informs the client of receipt of sensor data from a network device using this API.

3.1.2.9.2.1.2 Parameter List

表 10. APPSRV_DEVICE_DATA_RX_IND Parameter List

PARAMETER	TYPE	DESCRIPTION
srcAddr	UINT32	The 16-bit PAN identifier of the network
Rssi	SINT32	RSSI of the message received
sDataMsg	Smsgs_sensorMsg	Received sensor message (optional)
sConfigMsg	Smsgs_configRspMsg	Received config response message (optional)

表 11. Smsgs_sensorMsg

PARAMETER	TYPE	DESCRIPTION		
cmdId	ENUM	Sensor message command ID		
		COMMAND ID	**DESCRIPTION**	**VALUE**
		Smsgs_cmdIds_configReq	Configuration message, sent from the collector to the sensor	1
		Smsgs_cmdIds_configRsp	Configuration response message, sent from the sensor to the collector	2
		Smsgs_cmdIds_trackingReq	Tracking request message, sent from the collector to the sensor	3
		Smsgs_cmdIds_trackingRsp	Tracking response message, sent from the sensor to the collector	4
		Smsgs_cmdIds_sensorData	Sensor data message, sent from the sensor to the collector	5
		Smsgs_cmdIds_toggleLedReq	Toggle LED message, sent from the collector to the sensor	6
		Smsgs_cmdIds_toggleLedRsp	Smsgs_cmdIds_toggleLedRsp	7
Framecontrol	UINT32	Frame control field states what data fields are included in reported sensor data, each value is a bit mask value so that they can be combined (OR'd together) in a control field. When sent over-the-air in a message this field is 2 bytes.		
		PARAMETER	**DESCRIPTION**	**VALUE**
		Smsgs_dataFields_tempSensor	Bit mask for temperature sensor	0x0001
		Smsgs_dataFields_lightSensor	Bit mask for light sensor	0x0002
		Smsgs_dataFields_humiditySensor	Bit mask for humidity sensor	0x0004
		Smsgs_dataFields_msgStats	Bit mask for stats message	0x0008
		Smsgs_dataFields_configSettings	Bit mask for configuration settings	0x0010
		Smsgs_dataFields_pressureSensor	Bit mask for pressure sensor	0x0020
		Smsgs_dataFields_toggleSettings	Bit mask for toggle settings	0x0030

表 11. Smsgs_sensorMsg (continued)

PARAMETER	TYPE	DESCRIPTION		
tempSensor	Smsgs_tempSensorField	Lists the reported temperature sensor data (optional) Smsgs_tempSensorField:		
		PARAMETER	**TYPE**	**DESCRIPTION**
		ambienceTemp	UINT32	Ambience chip temperature - each value represents a 0.01° C, so a value of 2475 represents 24.75° C.
		objectTemp	UINT32	Object temperature - each value represents a 0.01° C, so a value of 2475 represents 24.75° C.
lightSensor	Smsgs_lightSensorField	Lists the reported light sensor data (optional) Smsgs_lightSensorField:		
		PARAMETER	**TYPE**	**DESCRIPTION**
		rawData	UINT32	Raw sensor data read out of the OPT2001 light sensor
humiditySensor	Smsgs_humiditySensorField	Lists the reported humidity sensor data (optional) Smsgs_humiditySensorField:		
		PARAMETER	**TYPE**	**DESCRIPTION**
		temp	UINT32	Raw temperature sensor data
		humidity	UINT32	Raw humidity sensor data
configSettings	Smsgs_configSettingsField	Lists the reported configuration settings (optional) Smsgs_configSettingsField:		
		PARAMETER	**TYPE**	**DESCRIPTION**
		reportingInterval	UINT32	Reporting interval - in milliseconds, how often to report sensor data to the pan-coordinator, 0 means reporting is off
		pollingInterval	UINT32	Polling interval - in milliseconds (32 bits) - If the sensor device is a sleep device, this states how often the device polls its parent for data. This field is 0 if the device does not sleep.
pressureSensor	Smsgs_pressureSensorField	Lists the reported pressure sensor data (optional) Smsgs_pressureSensorField:		
		PARAMETER	**TYPE**	**DESCRIPTION**
		tempValue	UINT32	Temperature value
		pressureValue	UINT32	Pressure value

表 12. Smsgs_configRspMsg

PARAMETER	TYPE	DESCRIPTION
cmdId	Smsgs_cmdIds	Sensor message command ID
Status	Smsgs_statusValues	Status of the processing of the request message
Framecontrol	Smsgs_dataFields	Bit mask of Smsgs_dataFields
treportingInterval	UINT32	Sensor data reporting interval
pollingInterval	UINT32	Polling interval if the device is a sleepy device

3.1.2.9.2.2 APPSRV_TX_DATA_REQ

3.1.2.9.2.2.1 Description

The application client uses this to send data to a network device.

3.1.2.9.2.2.2 Parameter List

表 13. APPSRV_TX_DATA_REQ Parameter List

PARAMETER	TYPE	DESCRIPTION		
msgId	Smsgs_cmdIds	Sensor message command ID		
panID	UINT32	The 16-bit PAN identifier of the network		
shortAddress	UINT32	The 16-bit short address of the network device		
extAddress	INT64	The 64-bit IEEE extended address of the network device		
configReqMsg	Smsgs_configReqMsg	Configuration request message parameters (optional)		
		PARAMETER	**TYPE**	**DESCRIPTION**
		cmdId	Smsgs_cmdIds	The value will be *Smsgs_cmdIds_configReq* (= 1).
		frameControl	UINT32	Frame control field states what data fields are included in reported sensor data, each value is a bit mask value so that they can be combined (OR'd together) in a control field. When sent over the air in a message this field is 2 bytes.
		reportingInterval	UINT32	Sensor data reporting interval
		pollingInterval	UINT32	Polling interval if the device is a sleepy device
toggleLedReq	Smsgs_toggleLedReqMsg	Toggle led request message parameters (optional)		

3.1.2.9.2.3 APPSRV_TX_DATA_CNF

3.1.2.9.2.3.1 Description

Thepplication server informs the client of result of the transmit data request

3.1.2.9.2.3.2 Parameter List

表 14. APPSRV_TX_DATA_CNF Parameter List

PARAMETER	TYPE	DESCRIPTION
Status	INT32	0 if success

3.1.2.10 TI IoT Gateway to Cloud Interface

The purpose of this section is to provide a description of the message types and expected data flows that will be shared between the TI IoT gateway and an IoT cloud server. The interface is designed to be flexible to support multiple cloud vendors. For this purpose, the Sub-1 GHz wireless network and node information will be exchanged between the gateway and the cloud using the long-established JavaScript object notation (JSON) format. Additionally, IPSO alliance smart object definitions will be used to define sensors (and their data) that are connected to each node in the wireless networks.

3.1.2.10.1 Message Types

To fully specify the Sub-1 GHz wireless network information, as well as the Sub-1 GHz sensors and their data, two distinct message types have been defined for the IoT gateway to update the cloud. In order to allow the cloud to send messages back to the TI IoT gateway, two additional message types are defined that allow the cloud to update the wireless network state and also send actuation messages to specific devices in the network.

3.1.2.10.1.1 Network Information Message Type (From TI IoT Gateway to Cloud)

This message type presents information about the wireless network, its current state, and a list of devices that are connected to the network. As shown later in this document, this will be the first message type sent after the network is initialized, and it contains all the information necessary to prepare for receiving sensor data from devices. This message type contains the following fields:

- **name**: begins as the short address of the network but allows for the cloud to provide a more specific name
- **channels**: list of channels that the wireless network is operating on
- **pan_id**: the 16-bit PAN identifier of the network
- **short_addr**: the 16-bit short address of the pan-coordinator
- **ext_addr**: the 64-bit IEEE extended address of the pan-coordinator device
- **security_enabled**: *yes* if security enabled, *no* otherwise
- **mode**: network operation mode (beacon, non-beacon, frequency hopping)
- **state**: PAN coordinator state values (waiting, starting, restoring, started, open, closed)
- **devices**: list of wireless nodes in the network
  - **name**: begins as the short address of the device but allows cloud to update
  - **active:** whether or not the wireless node is active
  - **rssi:** received signal strength indicator of the last message received
  - **last_reported:** timestamp of the last message received
  - **short_addr**: the 16-bit short address of the pan-coordinator
  - **ext_addr**: the 64-bit IEEE extended address of the PAN coordinator device
  - **topic**: the topic that the device will send its sensor data updates to
  - **smart_objects**: list of IPSO alliance smart objects (sensors) attached to this device
    - **object ID description**: type of sensor (as defined in the IPSO standard); can be multiple types of sensors connected to each device ('temperature' for example)
      - **instance ID**: the instance ID for the parent object type; can be multiple sensors of the same

type (is usually '0' and counts up with each instance added

- **resource ID description list**: sensor data name value pairs (for example, *sensorValue*: 32.5, *units*:*Celsius, dInState: true*, and so forth); these resources match what is specified for the given object ID in the IPSO standard

3.1.2.10.1.2 Device Information Message Type (From TI IoT Gateway to Cloud)

This message type provides information about the wireless device as well as the latest data for all of the sensors connected to the device. This message type will be sent when a device reports sensor data or switches between an active or inactive state. The following fields are contained in this message type:

- **active**: whether or not the wireless node is active
- **short_addr:** the 16-bit short address of the device
- **ext_addr**: the 64-bit IEEE extended address of the PAN coordinator device
- **rssi**: received signal strength indicator of the last message received
- **last_reported:** timestamp of the last message received
- **smart_objects**: list of the IPSO alliance smart objects connected to this wireless device
  - **object ID description**: type of sensor (as defined in the IPSO standard); can be multiple types of sensors connected to each device
    - **instance ID**: the instance ID for the parent object type; can be multiple sensors of the same type
    - **resource ID description list**: sensor data name value pairs (for example, *sensorValue*: 32.5, *units*:*Celsius*, and so forth); these resources match what is specified for the given object ID in the IPSO standard

3.1.2.10.1.3 Update Network State Message Type (From Cloud to TI IoT Gateway)

In the current implementation of the TI IoT gateway, this message type is intended to be able to open or close the wireless network to new devices joining. The cloud’s front end user interface can allow a user to click a button to open or close the network and then generate this message type and send it to the TI IoT gateway. The gateway will then notify the network on whether it needs to open or close to new device joins. This message type only includes the desired state of the network and should be sent to the same topic that the cloud is receiving the network information messages from. The following field is all that is required:

- **state**: should be set to either *open* or *closed*

3.1.2.10.1.4 Device Actuation Message Type (From Cloud to TI IoT Gateway)

This message type is added to allow the cloud to send actuation messages to specific devices in the wireless network. The current implementation only supports toggling an LED on the wireless device’s board. The device actuation message should be sent to the topic of the device as given in the devices list of the network information message. The following field is the only requirement for this message:

- **toggleLED**: should be set to *true*

3.1.2.10.2 Data Flows

This section of the document specifies the expected data flow when different events occur within the wireless network and also when the cloud must send configuration or commands to the TI IoT gateway.

3.1.2.10.2.1 Network Information Sent to the Cloud

The following items are the list of events that can occur on the TI IoT gateway that will cause a network information message type to be sent to the cloud. A description is given with each event, and the end of this section describes the expected behavior from the cloud upon receipt of this type of message.

- **Network Startup**
  This is the initial event in the TI IoT gateway. The TI IoT gateway will aggregate the information about the wireless network as well as the list of connected devices and their sensor types. The TI IoT gateway will then make a connection to the cloud and will send the aggregated data encapsulated in the network information message type.
- **Network Information Update**
  This event can occur if any of the information about the wireless network changes. For example, if the network operation mode of the wireless network was changed, the TI IoT gateway would once again aggregate all the information needed (network information and device list) and send the network information message type to the cloud.
- **Network State Change**
  This event occurs if the state of the wireless network changes. For example, if the network state changes from open to closed, the TI IoT gateway will send a network information message type to the cloud.
- **Device Joins the Wireless Network**
  When a new device joins the network, after the network is up and running, this event will occur. In this case, the TI IoT gateway will add the new device and its information to the devices list within the network information message type and then send the updated information to the cloud

Expected Cloud Behavior
It is expected that the cloud will be prepared for the network startup event and will be able to receive the network information message type (using a wildcard and then filtering or by having prior knowledge about the destination or topic of the message). Once the cloud receives the network information message, the wireless network information (PANID, security, mode, and so forth) can be displayed to users and the device list information (topic, object list, and so forth) can be used to prepare itself to receive and display device and sensor data.

3.1.2.10.2.2 Device Information Sent to the Cloud

The following is the list of events that will cause the TI IoT gateway to send a device information message type to the cloud. A description is given with each event and the end of this section describes the expected behavior from the cloud upon receipt of this type of message.

- **Device Becomes Inactive**
  This event occurs when the TI IoT gateway detects that one of the devices in the connected devices list has stopped sending sensor data updates. The TI IoT gateway will update the *active* field and send a device information message type to the cloud for the inactive device.
- **Device Reports Sensor Data**
  Each time a sensor on a connected device reports sensor data this event occurs. The TI IoT gateway updates the IPSO alliance smart object list in the device for each sensor and then sends a device information message type to the cloud.

Expected Cloud Behavior
It is expected that the Cloud will be alert on each topic given in the connected devices list from the network information message. When one of the two events occur in this section, the TI IoT gateway will send the device information message to the topic (corresponding to the device being update) that the cloud should be listening on or subscribed to. When the device information message arrives at the cloud, the cloud should display the latest device information and sensor data to users.

3.1.2.10.2.3 Update Network State Message Sent to the TI IoT Gateway

This message is used to open or close the wireless network to new devices joining. This should be an option provided to users in the front end user interface that the cloud presents. When the user decides to update the network state, the cloud should send an update network state message type to the TI IoT gateway on the same topic that the network information messages are arriving on.

Expected TI IoT Gateway Behavior
The TI IoT Gateway will receive the update network state message and will generate the correct command (either open or close) to the wireless network. This command should, in turn, cause a network state change event (from 节 3.1.2.10.2.1) that will send a network information message back to the cloud, which can confirm the successful completion of the update network state command.

3.1.2.10.2.4 Device Actuation Message Sent to the TI IoT Gateway

This method is used to toggle the LED on the board of the connected devices. This is meant to be a proof-of-concept on the current device setup and will change for customer use-case specific actuations. A toggle LED button for each device will be provided to users of the cloud's front end interface. When the toggle LED button is clicked, the cloud should send a device actuation message to the TI IoT gateway on the same topic that the device information messages are arriving on.

Expected TI IoT Gateway Behavior
The TI IoT Gateway will generate a toggle LED command and send it to the device corresponding to the topic that the device actuation message was received on. This will cause the LED to toggle. Because the state of the LED is not captured in the device information message type, there will be no feedback to the cloud that the LED actually toggled.

3.1.2.11 AWS IoT

节 3.1.2.10 of this document was generic to any cloud host or vendor. This section will give specific implementation details when using AWS IoT as the Cloud vendor. The numbering and header names of this section will be the same as 节 3.1.2.10, but additional information specific to the AWS IoT implementation is added here.

3.1.2.11.1 Message Types

The message types from 节 3.1.2.10.1 will remain the same for messages traveling in both directions. However, the message payload sent to and from the AWS cloud will be wrapped with some additional information specific to the use of the Amazon® thing shadow interface.

3.1.2.11.1.1 Network Information Message Type (From TI IoT Gateway to the Cloud)

The network information message type will be sent to the AWS cloud using a thing name that includes the extended address of the wireless collector node that is attached to the TI IoT gateway. For example, the thing name could be *ti_iot_0x124b000a27dda1_network* and would use the base thing shadow topic of *$aws/things/ti_iot_0x124b000a27dda1_network/shadow* where 0x124b000a27dda1 is the extended address of the collector node. Because all of the initial information needed to describe a network will be sent to this thing shadow, the thing name will either require to be known beforehand *or* the cloud front end must subscribe to an MQTT wildcard topic and then filter on the *_network* keyword in order to receive the initial message containing the network Information.

The message payload shown in 节 3.1.2.10.1.1 will remain the same, but the message will be in JSON format and will be encapsulated in a *state* JSON object to comply with the Amazon thing shadow interface. Further, all messages sent from the TI IoT gateway toward the AWS cloud will be sent to the *state.reported* property of the thing shadow document. Sending data to the *state.reported* property is how the AWS cloud receives and stores the latest state of the thing.

The only other note that should be made on this message type is that the *topic* property given in each connected device will be the base topic for the thing device shadow in the following format:
"topic" : *"$aws/things/ti_iot_0x124b000a27dda1_0x124b000a27d849/shadow"*
where the thing name is *ti_iot_0x124b000a27dda1_0x124b000a27d849*. This name is comprised of first the wireless network's extended address (0x124b000a27dda1) and second the extended address of the device connected to the wireless network (0x124b000a27d849). This naming convention guarantees a distinct thing name and also makes it easy to determine the wireless network that the device is connected to.

3.1.2.11.1.2 Device Information Message Type (From TI IoT Gateway to the Cloud)

The device information message type messages will be sent to the thing shadows or MQTT topics that are provided in the devices list from the network information message type. It is recommended that things are registered (or MQTT topics are subscribed to) for each device once the network Information message is received. This registration will allow the cloud front end to receive all sensor and information updates from all devices.

Similar to the network information message type above, the device information message type will be sent in JSON format and will be wrapped in a *state* JSON object to comply with the Amazon thing shadow interface. Once again, when this message type is sent from the TI IoT Gateway to the AWS Cloud, the message will be sent to the *state.reported* property of the thing shadow document.

3.1.2.11.1.3 Update Network State Message Type (From Cloud to TI IoT Gateway)

The update network state message type messages will be sent to the same thing shadow or MQTT topic that the cloud receives network information messages on. This message type will also be sent in JSON format and will also be wrapped in a *state* JSON object to comply with the Amazon Thing Shadow API.

The major difference is that this message (*from* the Cloud *to* the TI IoT Gateway) will send its data to the *state.desired* property of the thing shadow. This is the method that the Amazon thing shadow provides to request a thing to make a state or property change. In this case, the only currently supported wireless network change that can be requested is to either open or close the network for new device joins. The following is an example of the message in JSON format that should be sent to the thing shadow:
{ “state” : { “desired” : { “state” : “open” } } }

3.1.2.11.1.4 Device Actuation Message Type (From Cloud to TI IoT Gateway)

The device actuation message type messages will be sent to the same thing shadow or MQTT topic that the Cloud receives device information messages on. This message type will also be sent in JSON format and will also be wrapped in a *state* JSON object to comply with the Amazon thing shadow API.

Similar to the update network state messages, this message type will also be sent to the *state.desired* property of the thing shadow. The only currently supported actuation message that can be sent is to request that the wireless device toggle an onboard LED. The following JSON object is the only currently supported device actuation message type that should be sent from the AWS cloud to the TI IoT gateway: { “state” : { “desired” : { “toggleLED” : “true” } } }

3.1.2.11.2 Data Flows

The data flows for the AWS cloud remain the same as in 节 3.1.2.10.2. The only AWS specific information is that the data is being sent to thing shadows and that the messages are wrapped in either *state.reported* or *state.desired* JSON objects as described in 节 3.1.2.11.1.

3.1.2.12 IBM Cloud

节 3.1.2.10 of this document was generic to any cloud host or vendor. This section will give specific implementation details when using IBM® Watson IoT™ as the cloud vendor. The numbering and header names will be the same as 节 3.1.2.10.

3.1.2.12.1 Message Types

The message types from 节 3.1.2.10.1 will remain the same for messages traveling in both directions. However, the message payload sent to and from the AWS cloud will be wrapped with some additional information specific to the use of the Amazon thing shadow interface.

3.1.2.12.1.1 Network Information Message Type (From TI IoT Gateway to Cloud)

The network information message type will be sent to the IBM cloud by publishing the *nwkUpdate* gateway event. This publishes an MQTT topic with the gateway’s device type and device ID as identifiers. This means the application running on IBM cloud must subscribe to this particular gateways *nwkUpdate* event to receive this publication. The message payload shown in 节 3.1.2.10.1.1 will remain the same, but will be in JSON format.

3.1.2.12.1.2 Device Information Message Type (From TI IoT Gateway to Cloud)

The device information message type will be sent to the IBM cloud by publishing the *deviceUpdate* gateway event. This publishes an MQTT topic with the gateway’s device type and id. This means the application running on IBM cloud must subscribe to this particular gateways *deviceUpdate* event to receive this publication. The message payload shown in 节 3.1.2.10.1.1 will remain the same, but will be in JSON format.

3.1.2.12.1.3 Update Network State Message Type (From Cloud to TI IoT Gateway)

The update network state message type messages will be sent to the same MQTT topic that the cloud receives network information messages on. This message will also be sent in JSON format. The gateway will receive the message as part of the “command” event. This event will include a command name and payload. The command name for this message will be *nwkUpdate* and the payload will be a JSON object. The payload JSON object specifies the new network state in an action field like so {action: “open”}.

3.1.2.12.1.4 Device Actuation Message Type (From Cloud to TI IoT Gateway)

The device actuation message type message will be sent to the same MQTT topic that the cloud receives device information messages on. This gateway will receive this message through a “command” event. This event includes command name and payload data parameters. The command name for this message type is *deviceUpdate* and the payload is a JSON object. The JSON object specifies the target device in a dstAddr field like so {dstAddr: ‘0x0001’}.

3.1.2.12.2 Data Flows

The data flows for IBM cloud remain the same as in 节 3.1.2.11.1.

3.1.2.12.3 IBM Cloud Application

In order to use IBM as the cloud provider, an application must be created and configured on the cloud. There is an example application *ibm-frontend* that can be used located in the examples directory of the sensor to cloud repository. This example application can be uploaded using the *cf* command-line utility provided by IBM. For a step by step walkthrough of setting up this application please refer to the Sensor To Cloud Quickstart guide.

3.1.2.13 IBM Quickstart

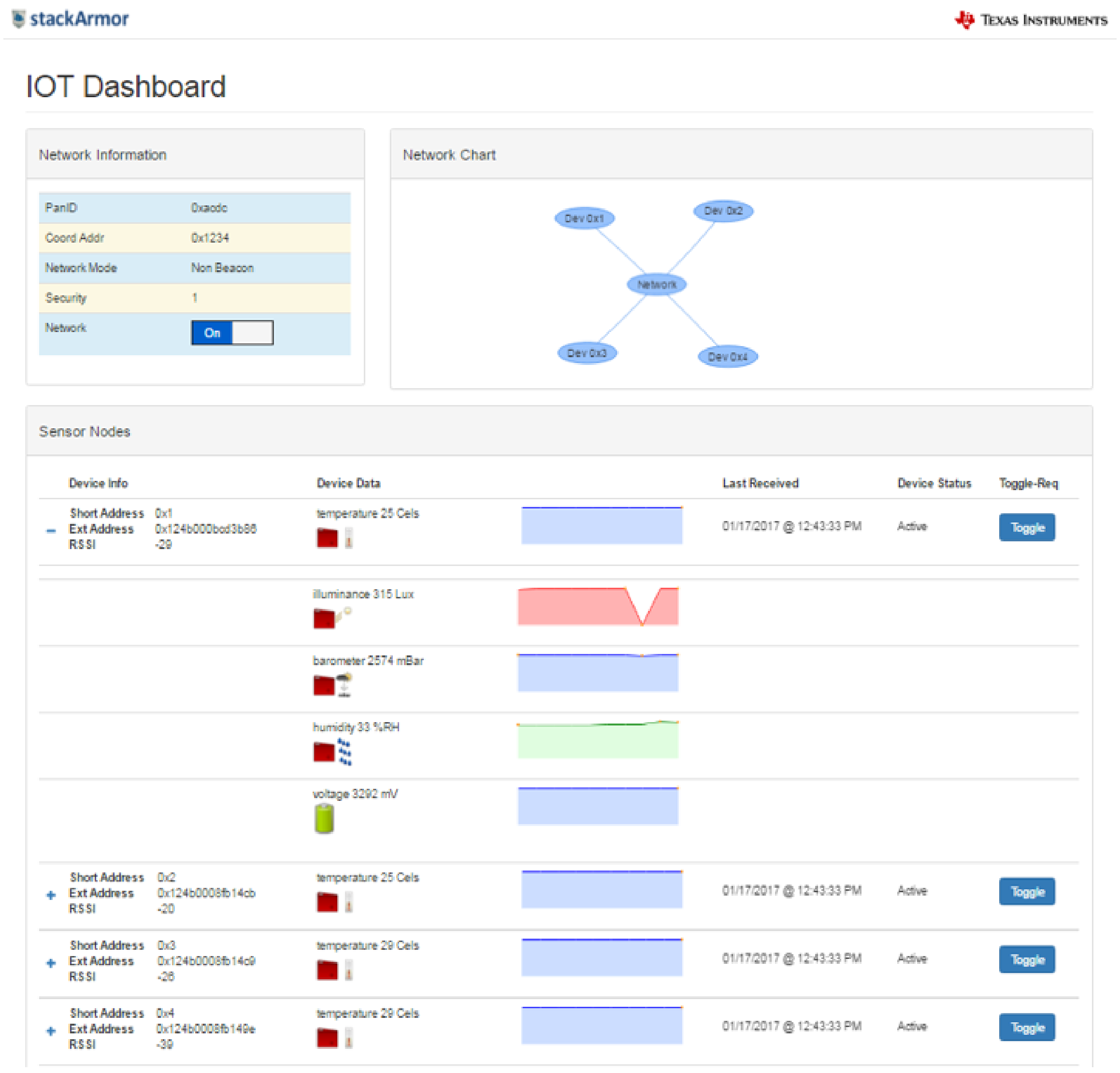
TI's Sensor To Cloud supports the IBM Quickstart cloud platform as a way to quickly see sensor data and network metadata on the cloud without any cloud-side overhead. This gives a good idea as to the types of messages being sent to the cloud application. The gateway reports two types of messages on the IBM Quickstart dashboard nwkUpdates and deviceUpdates. A nwkUpdate is emitted when something in the network changes, such as a device joining or leaving. A deviceUpdate is reported whenever a sensor node reports data to the collector. Using the IBM Quickstart platform is a great way to get started and get a feel for how data is sent to the cloud. More information on IBM Quickstart can be found here.

3.2 Testing and Results

During the development process of this reference design, the full hardware and software portions described in earlier sections were used for testing. Multiple CC1310 and CC1350 sensor nodes and a BeagleBone Black (connected to a CC1310 coprocessor) were used to verify the IoT gateway functionality with the AWS cloud enabled by stackArmor. The culmination of this reference design can be visualized by the IoT dashboard described in 节 3.2.1.

3.2.1 IoT Dashboard

图 12 shows an example of the IoT Dashboard being displayed on the web interface. Observe that the current network information is shown, the network chart displays the number of connected devices, and that the sensor nodes section shows the device and current sensor information for all the devices in the network.



PanID	0xacdc
Coord Addr	0x1234
Network Mode	Non Beacon
Security	1
Network	On

	Device Info		Device Data	Last Received	Device Status	Toggle-Req
–	Short Address Ext Address RSSI	0x1 0x124b000bcd3b86 -29	temperature 25 Cels	01/17/2017 @ 12:43:33 PM	Active	Toggle
			illuminance 315 Lux			
			barometer 2574 mBar			
			humidity 33 %RH			
			voltage 3292 mV			
+	Short Address Ext Address RSSI	0x2 0x124b0008fb14cb -20	temperature 25 Cels	01/17/2017 @ 12:43:33 PM	Active	Toggle
+	Short Address Ext Address RSSI	0x3 0x124b0008fb14c9 -26	temperature 29 Cels	01/17/2017 @ 12:43:33 PM	Active	Toggle
+	Short Address Ext Address RSSI	0x4 0x124b0008fb149e -39	temperature 29 Cels	01/17/2017 @ 12:43:33 PM	Active	Toggle

图 12. IoT Dashboard

4 Design Files

This reference design showcases the connectivity between AM335x and CC13x0 devices. The AM335x acts as a gateway processor and CC13x0 as communication node.

The AM335x-based BeagleBone Black is used as a platform for gateway processor, and the CC13x0-based LaunchPad acts as communication node. The schematic for this reference design shows how to map one UART port and backdoor signals from BeagleBone Black to the LaunchPad.

For software flexibility, the schematic also maps various SPI, I²C, and GPIOs from BeagleBone Black to the LaunchPad; however, for this IoT gateway reference design only one UART port and backdoor signals are valid.

4.1 Schematics

To download the schematics for each board, see the design files at TIDEP0084.

4.2 Bill of Materials

To download the bill of materials (BOM), see the design files at TIDEP0084.

4.3 PCB Layout Recommendations

4.3.1 Layout Prints

To download the layout prints, see the design files at TIDEP0084.

4.4 Altium Project

To download the Altium project files, see the design files at TIDEP0084.

4.5 Gerber Files

To download the Gerber files, see the design files at TIDEP0084.

4.6 Assembly Drawings

To download the assembly drawings, see the design files at TIDEP0084.

5 Software Files

To download the software files for this reference design, please see the link at https://git.ti.com/apps/tidep0084.

6 Related Documentation

1. Texas Instruments, *TI-15.4 Stack: IEEE802.15.4e/g Standard Based Star Networking Software Development Kit (SDK)*
2. Texas Instruments, *Processor SDK for AM335x Sitara™ Processors - Linux and TI-RTOS support*
3. Texas Instruments, *TI 15.4-Stack Wiki*
4. Texas Instruments, *Processor SDK Linux Creating an SD Card with Windows*
5. Texas Instruments, *CC13x0 SimpleLink TI 15.4-Stack 2.x.x Linux Developer's Guide*
6. Texas Instruments, *CC13x0 SimpleLink TI 15.4-Stack 2.x.x Embedded Developer's Guide*
7. Texas Instruments, *TI 15.4-Stack CC13x0 SimpleLink Embedded Applications Quick Start Guide*
8. stackArmor, *Industrial IoT Gateway Demonstration Request Form*

6.1 商标

Sitara, SimpleLink, LaunchPad are trademarks of Texas Instruments Incorporated.
ARM, Cortex are registered trademarks of ARM Limited.
Amazon Web Services is a trademark of Amazon Web Services, Inc..
Amazon is a registered trademark of Amazon Web Services, Inc..
LINKSYS, BELKIN are trademarks of Belkin International, Incorporated.
Bluetooth is a registered trademark of Bluetooth SIG, Incorporated.
EtherCAT is a registered trademark of EtherCAT Technology Group.
Google is a registered trademark of Google, Inc..
Watson IoT is a trademark of IBM Corporation.
IBM is a registered trademark of IBM Corporation.
PowerVR SGX is a trademark of Imagination Technologies Limited.
Linux is a registered trademark of Linux Foundation.
Windows is a registered trademark of Microsoft Corporation.
NETGEAR is a registered trademark of NETGEAR, Incorporated.
PROFIBUS, PROFINET are registered trademarks of PROFIBUS and PROFINET International (PI).
Yocto Project is a trademark of The Linux Foundation.
All other trademarks are the property of their respective owners.

7 About the Author

SUYASH JAIN is an Applications Engineer at Texas Instruments, where he is responsible for supporting customers designing low power wireless systems. Suyash earned his Master of Science in Electrical Engineering (MSEE) from Texas Tech University in Lubbock, TX.

JASON REEDER is a Software Applications Engineer at Texas Instruments, where he is responsible for supporting customers using Linux on TI's Sitara family of devices. Jason earned his Bachelor of Science in Computer Software Engineering and his Master of Science in Electrical Engineering at the University of Florida in Gainesville, FL.

AMRIT MUNDRA is a part of the Systems Team in Catalog Processors BU. He has been with TI for 13 years and has worked on multiple IP's and SoC's. He is the security architect for Keystone3 and security lead for Catalog BU. Amrit also is system lead for IoT EE initiative in BU.

BROCK ALLEN is an Applications Engineer at Texas Instruments. He is responsible for supporting customers designing low power and long range wireless systems using a Sub-1 Ghz radio. Brock earned his Bachelor of Science in Computer Engineering from Virginia Polytechnic Institute and State University (Virginia Tech).

修订版本 A 历史记录

注：之前版本的页码可能与当前版本有所不同。

Changes from Original (November 2016) to A Revision **Page**

- 已添加 *Cannot find module* error to *Section 2.3.3.1 Common Issues*. 19

修订版本 B 历史记录

Changes from A Revision (April 2017) to B Revision **Page**

- 已更改 调整了各部分的顺序，以适应当前设计指南模板 1
- 已添加 *and 433-MHz China bands* to tenth bulleted list item. 9
- 已添加 the following list items: *SimpleLink long range mode for all supported frequency bands*, *Support for SimpleLink CC1190*, and *Bluetooth low energy beacon advertisement support* 9
- 已添加 节 3.1.2.2.1 12
- 已更改 location of Note and 图 6 from 节 3.1.2.2 to 节 3.1.2.2.1 12
- 已添加 *, 863-MHz, and 433-MHz bands of operation at 50 kbps.* 15
- 已更改 *To rebuild the hex files for other bands (for example, 868 MHz ETSI band)* to *To rebuild the hex files for other bands (for example, SimpleLink Long Range mode),* 15
- 已添加 节 3.1.2.8 23
- 已添加 the following bullet items: ***active:*** *whether or not the wireless node is active*, ***rssi:*** *received signal strength indicator of the last message received*, ***last_reported:*** *timestamp of the last message received*, ***smart_objects***: *list of IPSO alliance smart objects (sensors) attached to this device*, ***object ID description***: *type of sensor (as defined in the IPSO standard); can be multiple types of sensors connected to each device ('temperature' for example)*, ***instance ID***: *the instance ID for the parent object type; can be multiple sensors of the same type (is usually '0' and counts up with each instance added*, and *resource ID description list*: *sensor data name value pairs (for example, sensorValue*: *32.5, units*:*Celsius, dlnState: true, and so forth); these resources match what is specified for the given object ID in the IPSO standard* 32
- 已添加 the following bullet items: ***short_addr:*** *the 16-bit short address of the device* and ***last_reported:*** *timestamp of the last message received* 34
- 已添加 节 3.1.2.12 and all child subsections 38
- 已添加 节 3.1.2.13 39
- 已添加 About the Author for Brock Allen 42