Design Guide: TIDC-01005

Battery-Powered, Smart-Lock Reference Design With Cloud Connectivity Using SimpleLink™ Wi-Fi®

Description

Wireless, battery-powered smart locks are changing the way customers control access to homes and buildings. The desire to eliminate the need for physical keys, control a lock from anywhere, and get notifications when an entry point is locked or unlocked is driving the demand for smart locks with Internet connectivity. This design demonstrates how SimpleLink™ Wi-Fi® enables the development of battery-powered smart locks with integrated Wi-Fi for direct Internet connectivity.

Resources

TIDC-01005	Design Folder
CC3220	Product Folder
CC2640R2F	Product Folder
DRV8837	Product Folder
CC3220S-LAUNCHXL	Tool Folder
LAUNCHXL-CC2640R2	Tool Folder
DRV8837EVM	Tool Folder
BOOSTXL-SENSORS	Tool Folder

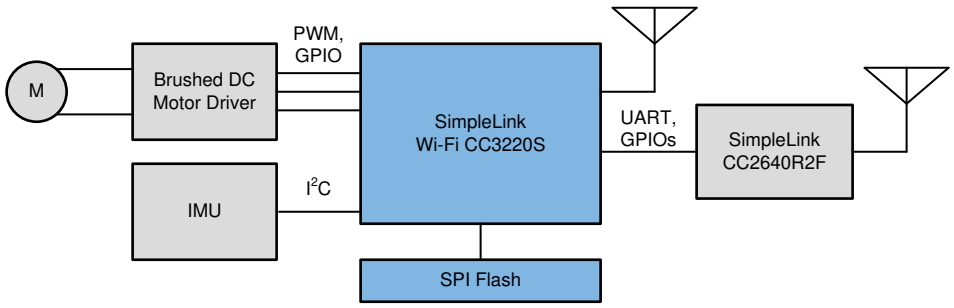
Ask our TI E2E™ support experts

Features

- 1.26 Years of Estimated Battery Life, Using Four AA Batteries While Always Connected
- Direct Connection to Cloud, to Enable Remote Monitoring and Control
- Integrated Wi-Fi Radio, Network Processor, and MCU, for Optimized Design With Low-Power Consumption
- Multiple Wi-Fi Provisioning Methods – BLE, AP, and SmartConfig™
- Wide Range of Built-in Security Features Necessary for Secure, Cloud-Connected Applications
  - Secure Boot of MCU Image
  - Internal HTTPS Server for AP Provisioning
  - MQTT Over TLS With Eclipse Internet of Things (IoT) Broker
  - Secure Over the Air (OTA) Using Dropbox™
  - 128-Bit, Unique Device ID
- Motor Driver With Integrated FETs, Enabling Easy Design and Small Solution Size With Minimal Discretes

Applications

- Electronic Smart Lock
- Door Keypads and Readers



1 System Description

This reference design demonstrates how to create a battery-powered, electronic smart lock with integrated Wi-Fi. The design demonstrates how the SimpleLink Wi-Fi CC3220S wireless MCU (SoC) can be used as the main system controller and network processor to create a highly integrated design. The TIDC-01005 reference design combines the CC3220S with a DRV8837, 1.8-A, low-voltage, brushed DC motor driver, to form the core of a Wi-Fi enabled electronic smart lock design. The design also features the SimpleLink Bluetooth® low energy CC2640R2F wireless MCU to demonstrate Wi-Fi provisioning over BLE. The design leverages LaunchPad™ Development kits and the DRV8837EVM, which makes it easy to reproduce and evaluate. The software for the TIDC-01005 is based on the SimpleLink SDK to enable maximum portability within the SimpleLink Platform.

The TIDC-01005 can be connected to a Wi-Fi access point (AP) using either the SimpleLink SDK Explorer or SimpleLink Wi-Fi Starter Pro mobile application. When connected to the AP, the design establishes a secure connection to the cloud, which lets the user remotely control and monitor the state of the lock system. Integrating Wi-Fi into the lock eliminates the need for the user to have a wireless bridge to connect the system to their Wi-Fi network and IoT ecosystem. In addition to demonstrating remote access to the lock system, the TIDC-01005 design also demonstrates how the CC3220S can be used to implement a fast OTA software update for the lock, while demonstrating the use of multiple CC3220S security enablers.

The two main design considerations for the TIDC-01005 were low-power capability and integrated security features. The ability to support a Wi-Fi connection with very low-power is critical for Wi-Fi-enabled electronic smart locks, because they are often battery powered. Additionally, many security features must be supported by an electronic smart-lock design for it to be robust and function as a reliable component in an access control system. The TIDC-01005 demonstrates how to achieve very low-current consumption with the CC3220S when creating a system that is always connected to the network. The design also demonstrates the following key security features:

- Secure boot of application code
- AP provisioning using WPA2 authentication and HTTPS
- Wi-Fi provisioning over BLE, with secure simple pairing
- Secure socket connections to the cloud for messaging and OTA updates
- OTA update using multiple security enablers:
  - File integrity check
  - Signature verification
  - Failsafe files
  - Bundle protection

Many of the features demonstrated in the TIDC-01005 application are based on the networking application libraries available in the SimpleLink SDK, including SNTP, MQTT, and HTTP. These protocols and the features they are used to implement in the software are applicable to many other building automation systems that integrate Wi-Fi connectivity. In particular, multiple aspects of the TIDC-01005 design may also apply to door keypad and reader designs.

1.1 Key System Specifications

Table 1-1. Key System Specifications

PARAMETER	SPECIFICATIONS	DETAILS
Input power source	Demo: 5-V USB connection to LaunchPad	Section 3.1.1.5 and Section 3.2.5
	Calculations: Based on theoretical energy capacity of four AA alkaline batteries	
Sensor type	Inertial measurement unit (IMU)	Section 2.1
Average current consumption	Wi-Fi and MCU: 377 µA	Section 3.2.2
	Bluetooth low energy and MCU: 40 µA	
	Motor: 42 µA	
Lock or unlock events	24 per day	Section 3.2.5
Motor type	Brushed DC	Section 2.3
User interface	Mobile device with Wi-Fi interface (for example, smartphone, tablet, or PC)	N/A
Theoretical estimate of battery life	1.26 years	Section 3.2.5

2 System Overview

2.1 Block Diagram

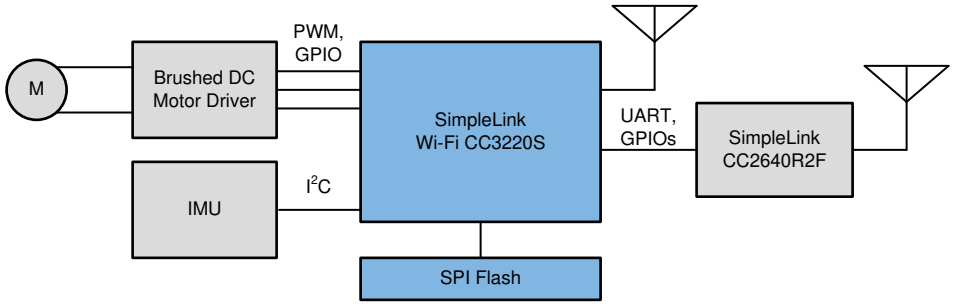


Figure 2-1. TIDC-01005 Block Diagram

The TIDC-01005 reference design demonstrates a highly-integrated, low-power solution for the host MCU, Wi-Fi connectivity subsystem, and motor driver subsystem of an electronic smart lock design. The TIDC-01005 also demonstrates a solution for provisioning Wi-Fi over BLE and a sensor interface to an IMU.

2.2 Design Considerations

It is important for the system to be as power efficient as possible, to provide a good user experience and keep maintenance costs low for battery-powered smart locks or door keypads and readers. This consideration can often be a challenge when adding new, wireless, connectivity interfaces, such as Wi-Fi, BLE, or Sub-1GHz, to existing battery-powered systems. The addition of wireless connectivity interfaces can also bring new concerns related to maintaining the security of the system. For these reasons, low-power capability and integrated security features were both considered as top priorities for the devices used in this design. The CC3220 SimpleLink Wi-Fi and IoT SoC enables the development of electronic smart locks, door keypads, and door readers with built-in Wi-Fi connectivity through its architecture, designed for low-power applications and a wide-range of integrated security solutions.

In addition to the low-power capability and security features of the CC3220 SoC, the portability of the solution was also an important consideration. Portability is important because the connectivity solution that is best suited for one of these systems depends on the target use-case and market. Electronic smart locks, door keypads, and door readers can be found in both commercial and residential buildings, each of which provide different connectivity challenges. To accommodate the need for a flexible design and enable maximum reuse across different technologies, the software in the design is based on the SimpleLink Software Development Kit (SDK). The SimpleLink SDK includes a common RTOS interface as well as a common set of hardware drivers and networking services, for the various wired and wireless connectivity solutions in the SimpleLink Platform. All of the networking services used in the Wi-Fi doorlock demo are built on the SlNetSock library, which enables the key networking functions to be ported to different networking stacks (for example, the SimpleLink Wi-Fi driver or Ethernet NDK).

2.3 Highlighted Products

The three TI products highlighted in this reference design are the CC3220S SimpleLink Wi-Fi and IoT, Single-Chip Wireless MCU Solution, the CC2640R2F SimpleLink BLE Wireless MCU, and the DRV8837, a 1.8-A, low-voltage brushed DC motor driver.

2.3.1 CC3220

The CC3220x device is part of the SimpleLink MCU platform, which consists of Wi-Fi, low-energy, Sub-1 GHz, and host MCUs that all share a common, easy-to-use development environment, with a single-core SDK and rich tool set. A one-time integration of the SimpleLink platform enables users to add any combination of the devices in the portfolio into their design, allowing 100% code reuse when their design requirements change. For more information, visit the SimpleLink Solutions page on ti.com.

Start a IoT design with a Wi-Fi CERTIFIED, single-chip MCU, SoC, with built-in Wi-Fi connectivity. Created for the IoT, the SimpleLink CC3220x device family from TI is a single-chip solution, integrating two physically-separated, on-chip MCUs. The design includes an application processor, the Arm Cortex-M4 MCU, with a user-dedicated 256KB of RAM and an optional 1MB of XIP flash, as well as a network processor MCU, to run all Wi-Fi and Internet logical layers. This ROM-based subsystem includes an 802.11b/g/n radio, baseband, and MAC, with a powerful crypto engine for fast, secure, internet connections with 256-bit encryption.

The CC3220x wireless MCU family is a part of the second generation of the Internet-on-a-chip family of solutions from TI. This generation introduces new features and capabilities that further simplify the connectivity of devices to the Internet. The new capabilities including the following:

- IPv6
- Enhanced Wi-Fi provisioning
- Enhanced power consumption
- Enhanced file system security (supported only by the CC3220S and CC3220SF devices)
- Wi-Fi AP connection with up to four stations
- More concurrently opened BSD sockets: up to 16 BSD sockets, of which 6 are secure
- HTTPS support RESTful API support
- Asymmetric keys crypto library

The CC3220x wireless MCU family supports the following modes: station, AP, and Wi-Fi Direct. The device also supports WPA2 personal and enterprise security, WPA2 + PMF, and WPA3. This subsystem includes embedded TCP/IP and TLS/SSL stacks, the HTTP server, and multiple Internet protocols. The device supports a variety of Wi-Fi provisioning methods including HTTP based on AP mode, SmartConfig Technology, and WPS2.0.

The power-management subsystem includes integrated DC-DC converters that support a wide range of supply voltages. This subsystem enables low-power consumption modes for extended battery life, such as low-power deep sleep, hibernate with RTC (consuming only 4.5 µA), and shutdown mode (consuming only 1 µA).

The device includes a wide variety of peripherals, including a fast parallel-camera interface, I2S, SD, UART, SPI, I^2C, and 4-channel ADCs.

The SimpleLink CC3220x family of devices is available in three different variants: CC3220R, CC3220S, and CC3220SF. The CC3220R and CC3220S devices include 256KB of application-dedicated, embedded RAM for code and data, ROM with an external serial-flash bootloader, and peripheral drivers. The CC3220SF device includes an application-dedicated 1MB of XIP flash and 256KB of RAM for code and data, ROM with an external serial-flash bootloader, and peripheral drivers. The CC3220S and CC3220SF device options have additional security features, such as encrypted and authenticated file systems, user IP encryption and authentication, secured boot (authentication and integrity validation of the application image at flash and boot time), and more.

The CC3220x device family is a complete platform solution including software, sample applications, tools, user and programming guides, reference designs, and the E2E online community. The device family is also part of the SimpleLink MCU portfolio and supports the SimpleLink developers ecosystem.

2.3.2 CC2640R2F

The CC2640R2F device is a wireless MCU targeting Bluetooth 4.2 and Bluetooth 5 low-energy applications. The device is a member of the SimpleLink ultra-low power CC26xx family of cost-effective, 2.4-GHz RF devices. Ultra-low active RF and MCU current and low-power mode current consumption provide excellent battery lifetime and allow for operation on small, coin-cell batteries and in energy-harvesting applications. The SimpleLink BLE CC2640R2F device contains a 32-bit Arm Cortex-M3 core, which runs at 48 MHz, as the main processor and a rich peripheral feature set that includes a unique ultra-low power sensor controller. This sensor controller is ideal for interfacing external sensors and for collecting analog and digital data autonomously, while the rest of the system is in sleep mode. Thus, the CC2640R2F device is great for a wide range of applications where long battery lifetime, small form factor, and ease of use are important. The power and clock management and radio systems of the CC2640R2F wireless MCU require specific configuration and handling by the software to operate correctly, which has been implemented in the TI-RTOS. TI recommends using this software framework for all application development on the device. The complete TI-RTOS and device drivers are offered in the source code, free of charge from www.ti.com. Bluetooth low energy controller and host libraries are embedded in the ROM and run partly on an Arm Cortex-M0 processor. This architecture improves overall system performance and power consumption and frees up significant amounts of flash memory for the application. The Bluetooth stack is available, free of charge from www.ti.com.

2.3.3 DRV8837

The DRV883x family of devices provides an integrated motor-driver solution for cameras, consumer products, toys, and other low-voltage or battery-powered, motion control applications. The device can drive one DC motor or other devices like solenoids. The output driver block consists of N-channel power MOSFETs configured as an H-bridge, to drive the motor winding. An internal charge pump generates the needed gate-drive voltages.

The DRV883x family of devices can supply up to 1.8 A of output current. The device operates on a motor power-supply voltage from 0 to 11 V, and a device power-supply voltage of 1.8 V to 7 V. The DRV8837 device has a PWM (IN1-IN2) input interface, and the DRV8838 device has a PH-EN input interface. Both interfaces are compatible with industry-standard devices. Internal shutdown functions are provided for overcurrent protection, short-circuit protection, undervoltage lockout, and overtemperature.

For all available packages, see the orderable addendum at the end of the DRV883x Low-Voltage H-Bridge Driver data sheet.

2.4 System Design Theory

2.4.1 CC3220S to CC2640R2F Interface

The CC2640R2F is used in the TIDC-01005 to enable the system to be provisioned to a Wi-Fi network over a Bluetooth low energy connection. In the design, the CC3220S acts as the main system controller (simple application processor or SAP) and the CC2640R2F acts as a BLE network processor (simple network processor or SNP). Using the SAP and SNP solution simplifies the design, because it abstracts away the BLE protocol and allows the BLE functionality to be implemented by sending predefined commands to the CC2640R2F over the Unified Network Processor Interface (NPI) from TI.

The NPI supports either UART or SPI as the serial communication protocol, but only UART is implemented for the TIDC-01005. In addition to using the standard TX and RX data lines of UART, the NPI also uses three additional signals between the processors, for the host to reset the SNP and to indicate when each device is ready to send or receive data. The three additional signals are called RESET, Master Ready (MRDY), and Slave Ready (SRDY).

For more information on the SAP and SNP solution, see the SimpleAP and SNP wiki page. For more information on the NPI, see the Unified Network Processor Interface wiki page.

2.4.2 CC3220S to DRV8837 Interface

The brushed DC-motor driver, DRV8837, is used to provide a controlled current source to the motor. In this design, the CC3220S uses a three-pin interface consisting of two general-purpose input output (GPIO) signals and one PWM signal, to control the DRV8837. Figure 2-2 shows a simplified diagram of the interface between the CC3220 and DRV8837, where the Controller represents the CC3220 MCU.

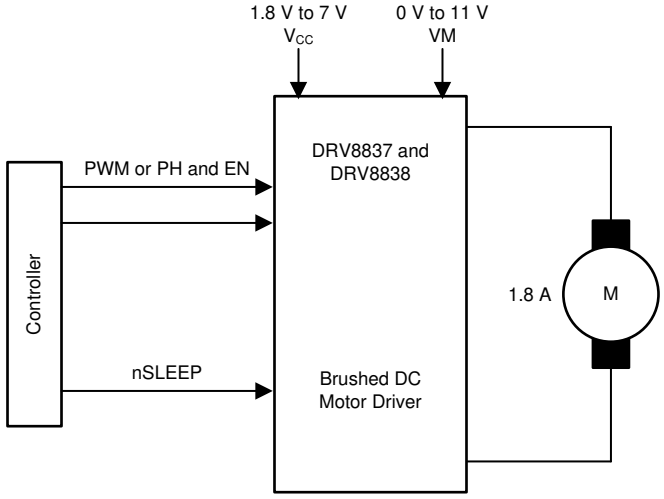


Figure 2-2. DRV8837 Simplified Diagram

The DRV8837 device behaves according to a PWM input scheme called the IN-IN interface. For this design, the control interface has been configured such that one GPIO from the CC3220 is used to control IN1, and a PWM is used to control IN2 (see Table 2-1). For a full description of the device logic, see the DRV883x Low-Voltage H-Bridge Driver data sheet.

Table 2-1. DRV8837 Device Logic

nSLEEP	IN1	IN2	FUNCTION (DC MOTOR)
0	X	X	Coast
1	0	0	Coast
1	0	1	Reverse
1	1	0	Forward
1	1	1	Brake

In the software application for the TIDC-01005, P16 and P17 of the CC3220 are connected to IN1 and IN2 of the DRV8837, respectively. P16 is configured as a standard GPIO, and P17 is configured as a PWM output. P18 of the CC3220 is configured as a GPIO and connected to nSLEEP on the DRV8837. When turning the motor forward, P18 (nSLEEP) and P16 (IN1) are both set to a logic high and the duty cycle of the PWM output on P17 (IN2) is fixed to drive the motor at a constant speed. By duty cycling P17 with P16 high, the motor switches between the Forward drive and Brake modes, to turn forward at a controlled speed. When driving the motor in reverse, P18 (nSLEEP) is set to a logic high, and P16 (IN1) is kept as logic low, while the PWM output on P17 is set to a fixed duty cycle. By duty cycling P17 with P16 low, the motor switches between the Reverse and Coast functions, which causes the motor to turn backward at a controlled speed.

2.4.3 Software Architecture

The software provided for the TIDC-01005 is a multithreaded application based on the SimpleLink CC3220 SDK, SimpleLink SDK BLE Plugin, and the Sensor and Actuator Plugin. The TIDC-01005 software uses TI-RTOS to support multithreading and TI-Drivers as the primary interface for managing hardware peripherals (for example, communications peripherals, timers, and digital I/Os). Figure 2-3 shows an overall block diagram for the software.

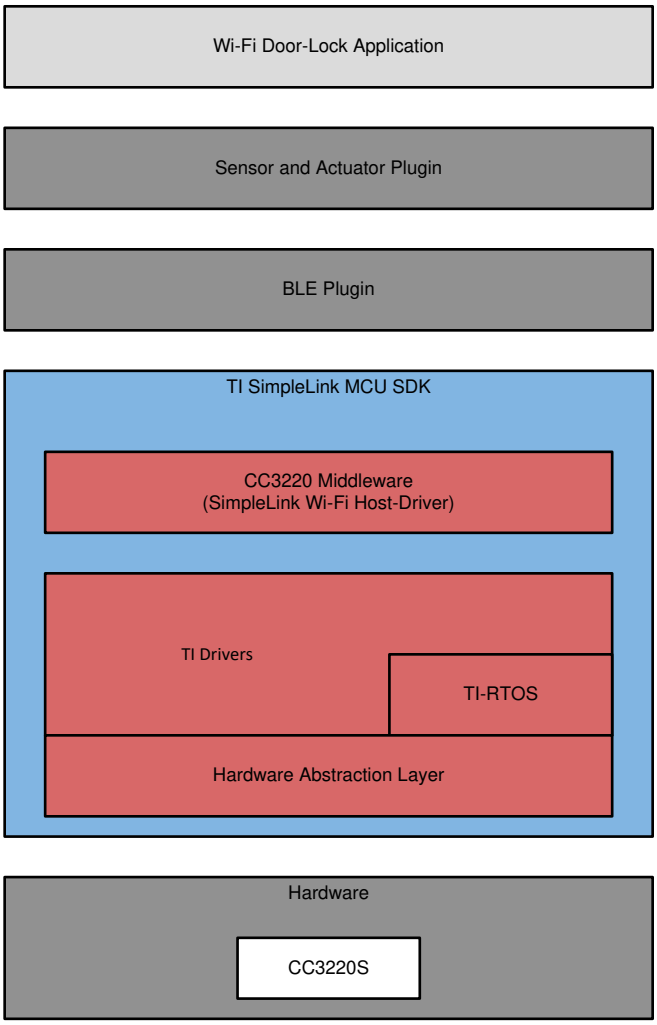


Figure 2-3. Wi-Fi, Door-Lock Software Block Diagram

The software for the TIDC-01005 demonstrates multiple features that are supported in a Wi-Fi, electronic, smart-lock design including controlling a motor driver to actuate a lock, reading sensor data, managing a Wi-Fi connection (including provisioning), sending and receiving messages through the cloud, and performing OTA software updates. Each of these features are handled by different software modules in the application, and each module consists of one or more TI-RTOS tasks. In addition to tasks dedicated for each feature, the software includes an overall Wi-Fi door-lock module that is responsible for spawning all other threads and a control module that handles system-level requests such as resets. Dividing the software into separate modules and dedicating independent tasks to each feature helps make the design more modular, so it can be easily modified for different applications.

Table 2-2 lists a description of each software module and the associated source files.

Table 2-2. Wi-Fi Door-Lock Software Modules

MODULE	SOURCE FILES	DESCRIPTION
Wi-Fi Door Lock Application Module	wifi_doorlock_app.h/.c	Starts Wi-Fi door-lock application by initializing system and spawning threads
Control Module	control_task.h/.c	Handles system-level requests such as system resets
Wi-Fi Connection Management	network_if.h/.c	Connects CC3220 to an AP based on profiles, or handles events to start provisioning
Wi-Fi Provisioning	provisioning_task.h/.c	Starts or stops provisioning process and executes state machine to handle provisioning events
MQTT Client	mqtt_client_task.h/.c and mqtt_client_cbs.h/.c	Runs MQTT client, receives incoming MQTT messages, publishes MQTT messages to signal device state, and triggers events based on MQTT messages
Wi-Fi OTA	cloud_ota_task.h/.c	Runs OTA state machine
Motor Control	motor_driver_if.h/.c	Drives motor based on MQTT messages received from the cloud
Sensor	bmi160_support.h/.c	Configures and reads sensor data from IMU when enabled

Figure 2-4 shows the overall software flow used to initialize the system and spawn all of the software tasks. The details of each of the spawned tasks are described in the following sections.

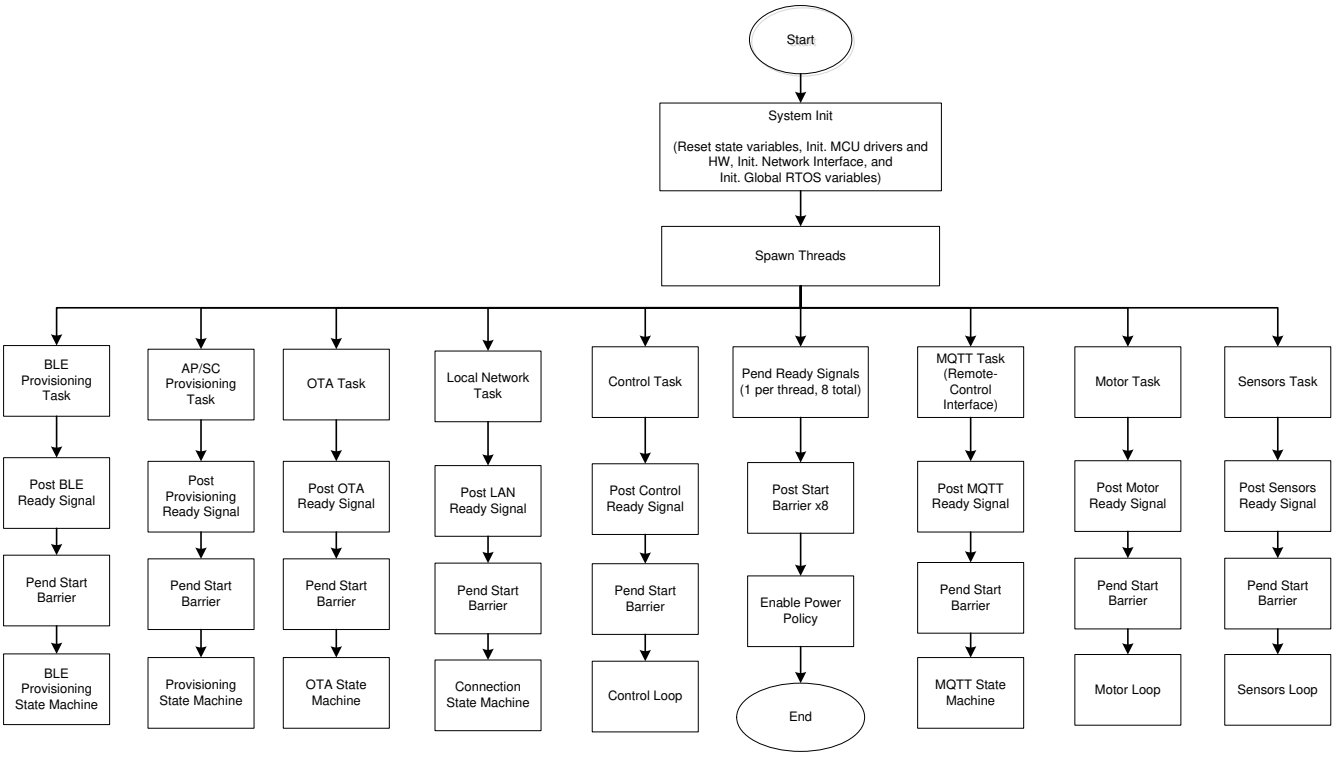


Figure 2-4. Overall Software Flow Diagram

2.4.4 Network Connection Management

Connection to the local network (CC3220 to AP) is handled independently from the provisioning task in the TIDC-01005 software application. The reason the connection is handled separately is because the system does not need to be provisioned every time it tries to connect to a network. A profile stored in the external serial flash of the CC3220S is used to connect the system to a network during most connection attempts. Additionally, connection management was implemented as a separate thread, because provisioning does not need to be automatically triggered if the system fails to automatically connect to a network based on a stored profile.

When the network connection thread runs, it first checks to see if a stored profile exists on the serial flash. If a profile does not exist, the system assumes that an automatic connection attempt will fail and, therefore, immediately enters one of the enabled provisioning modes. If a profile exists, the thread waits a predefined amount of time to see if the system automatically connects to an AP based on the stored profile. If the connection is successful, the connection thread goes to sleep until a disconnection event occurs. If the device does not connect to an AP based on the stored profile, the system waits a predefined amount of time before attempting to connect again. The reason the system attempts to connect again after a set amount of time is because it is possible that the AP described by the stored profile exists, but was inactive or out of range during the connection attempt.

The network connection task is also responsible for starting provisioning. When the system is first started by a user or reset to factory defaults, no profile exists on the serial flash. Therefore, either AP, SmartConfig, or BLE provisioning must run before the system can be used. The provisioning process is triggered by the network connection task when there is no profile.

Figure 2-5 shows the network-connection management state machine.

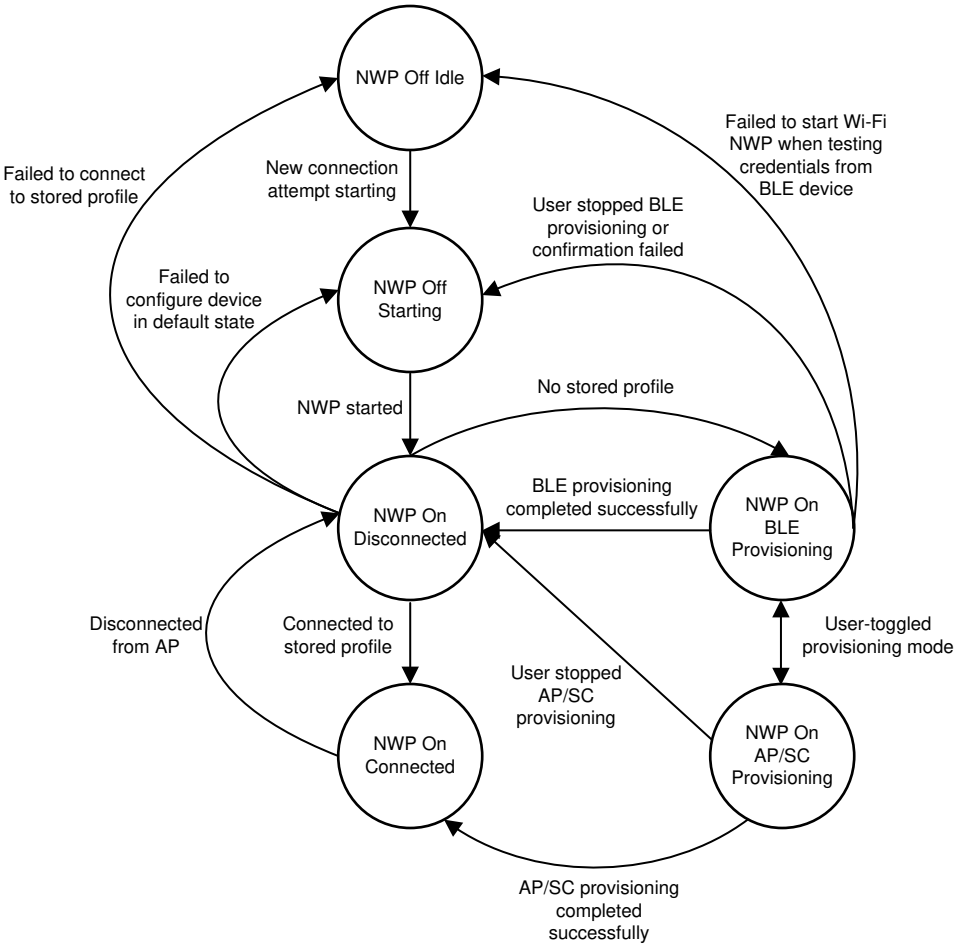


Figure 2-5. Network-Connection Management-Thread Flow Diagram

2.4.5 Provisioning

The TIDC-01005 is a headless system, like most electronic smart locks. A headless system is one that lacks a traditional user-interface (for example, keyboard, display, and so on) and must, therefore, obtain the SSID and password for the network it connects to using alternate methods. The TIDC-01005 supports multiple provisioning methods, including AP provisioning, SmartConfig from TI, and provisioning over BLE. The CC3220 can run AP provisioning and SmartConfig simultaneously, which saves the user from having to manually switch between the modes. Because the TIDC-01005 lacks a hardware-based coexistence mechanism, BLE provisioning is not enabled during AP and SmartConfig provisioning, and a user must manually switch between the methods.

2.4.5.1 AP Provisioning and SmartConfig™

AP provisioning is the most common provisioning method. During AP provisioning, an unprovisioned Wi-Fi device temporarily operates as an AP. This method allows a device, such as a smartphone, tablet, or PC, to connect to the AP over Wi-Fi and transmit the information for the desired network connection directly to the device. The CC3220 receives the network credentials through a Restful API based on its internal HTTP server.

It is important to secure the connection between the device sending the network credentials and the CC3220 during AP provisioning. To protect the network credentials and ensure a user does not unknowingly transmit them to an unwanted device, the TIDC-01005 uses WPA2 authentication when a station connects to the CC3220 AP. The TIDC-01005 is also configured to direct incoming connections to a secured HTTP port and establish a TLS session over which the credentials are passed.

SmartConfig is a proprietary provisioning method from TI that uses a smartphone or tablet to broadcast network credentials to a TI Wi-Fi device. The unprovisioned device can scan for SmartConfig broadcasts while operating in station mode or AP mode.

The CC3220S integrates the code needed to run AP and SmartConfig provisioning in the network processor firmware. This saves the user from having to dedicate a large portion of the application memory for implementing provisioning. Instead, the application must only handle the following:

- Sending a command to the network processor to start the provisioning process
- Receiving asynchronous events during provisioning
- Stopping provisioning when complete

To manage the provisioning process within an application, it is important to understand the internal provisioning flow used by the network processor. When provisioning is first started by the application, the CC3220 device enters the configuration state where it waits to receive the parameters of the network. After receiving the parameters, the device then attempts to connect to the network and provides feedback to the user upon success. A diagram of the provisioning process flow is in the CC3120, CC3220 SimpleLink Wi-Fi Internet-on-a chip Solution Device Provisioning application report.

In the wifi_doorlock example code, the AP provisioning and SmartConfig process is handled by a dedicated provisioning task. The task remains idle until the provisioning state machine is started by the network connection task. When the provisioning state machine is started, the provisioning task puts the network processor in the provisioning state and handles asynchronous events until the system successfully connects to the network. Whenever the application is restarted, the system always attempts to connect to a network based on a stored profile and skip the provisioning process if possible.

Figure 2-6 shows a diagram of the provisioning state machine implemented by the TIDC-01005 software.

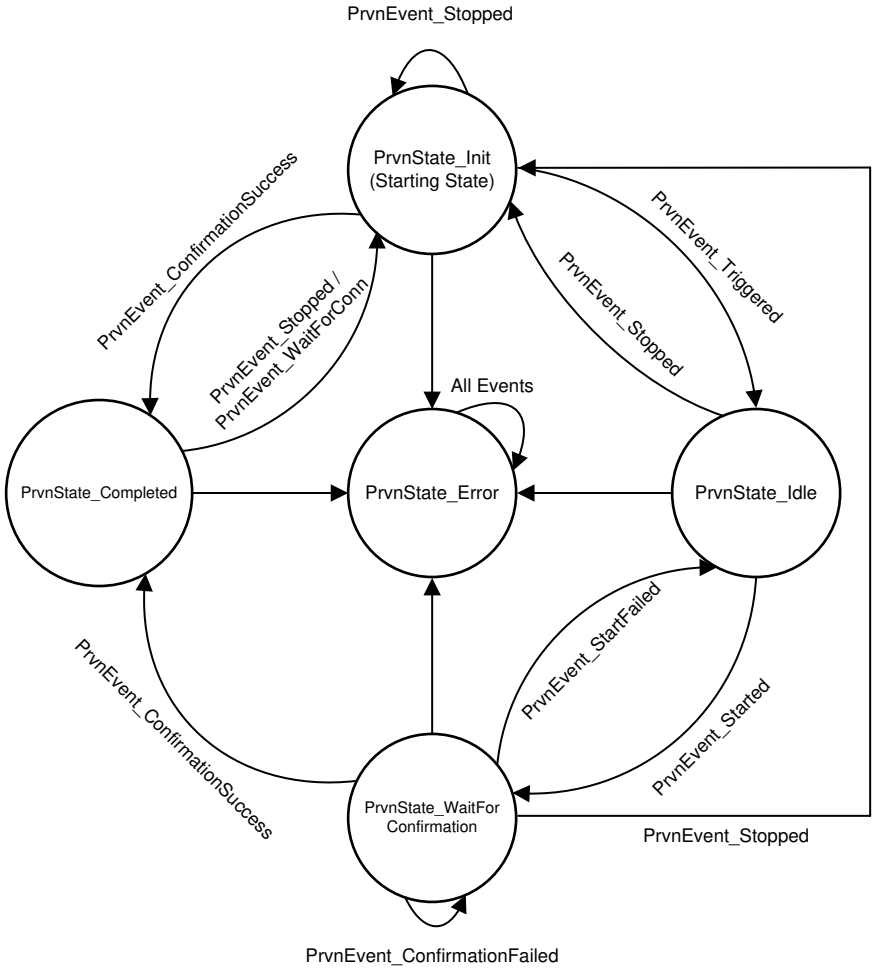


Figure 2-6. Provisioning Task State Machine

2.4.5.2 Wi-Fi Provisioning Over BLE

Wi-Fi provisioning over BLE is another method of provisioning a Wi-Fi based system to a new Wi-Fi network. The main benefit of provisioning a system over BLE is that it can provide a more consistent and streamlined user experience when provisioning from a smartphone or tablet. On some mobile operating systems, AP provisioning requires a user to do the following:

1. Open a settings application to disconnect the mobile device from the Wi-Fi network it is currently using.
2. Connect to the AP created by the device being provisioned.
3. Enter the credentials for the original Wi-Fi network.
4. Switch back to the original Wi-Fi network to confirm that provisioning succeeded.

When provisioning over BLE, the process can be completed without the user ever having to change the Wi-Fi network to which the mobile device is connected. Therefore, the entire provisioning process can be carried out within a single mobile application. The process for provisioning the system over BLE is as follows:

1. BLE advertising starts.
2. A peer connects to a BLE device and authenticates with a passkey.
3. The user writes the SSID, Security Key, and Device Name to corresponding BLE characteristics.
4. The user writes 0x01 to the Provisioning Start characteristic, to signal that a new profile must be tested.
5. CC3220S disconnects the BLE, reads the values written to the characteristics, starts the Wi-Fi network processor, and attempts to connect to the profile.
6. CC3220S waits to obtain an IP address and then disconnects the Wi-Fi and stops the Wi-Fi network processor.
7. The BLE advertising restarts.
8. The mobile device reconnects to the system over BLE and reads the Provisioning Status characteristic (success or fail).
9. If the process succeeds, the CC3220S adds the Wi-Fi network information as a stored profile, exits the provisioning state, and then connects to the Wi-Fi network.

Even though the overall process implemented by the system may take more steps, the user experienced is simplified, because the user only needs to take an action for Steps 2, 3, and 4.

The TIDC-01005 software application implements provisioning over BLE using a modified version of the BLE provisioning example from the SimpleLink SDK Bluetooth Plugin. In the TIDC-01005 software, a BLE thread is dedicated to handling communication with the CC2640R2F, while the system is being provisioned. Because there is no hardware coexistence mechanism for Wi-Fi and BLE in the system, the software application turns off the Wi-Fi network processor while the BLE interface is active and stops the BLE activity while the Wi-Fi interface is active. To keep the Wi-Fi and BLE activity separate, all of the Wi-Fi activity (such as testing received credentials) is handled inside the network interface task instead of the BLE provisioning task.

Figure 2-7 shows the state machine implemented by the task that is dedicated to provisioning over BLE.

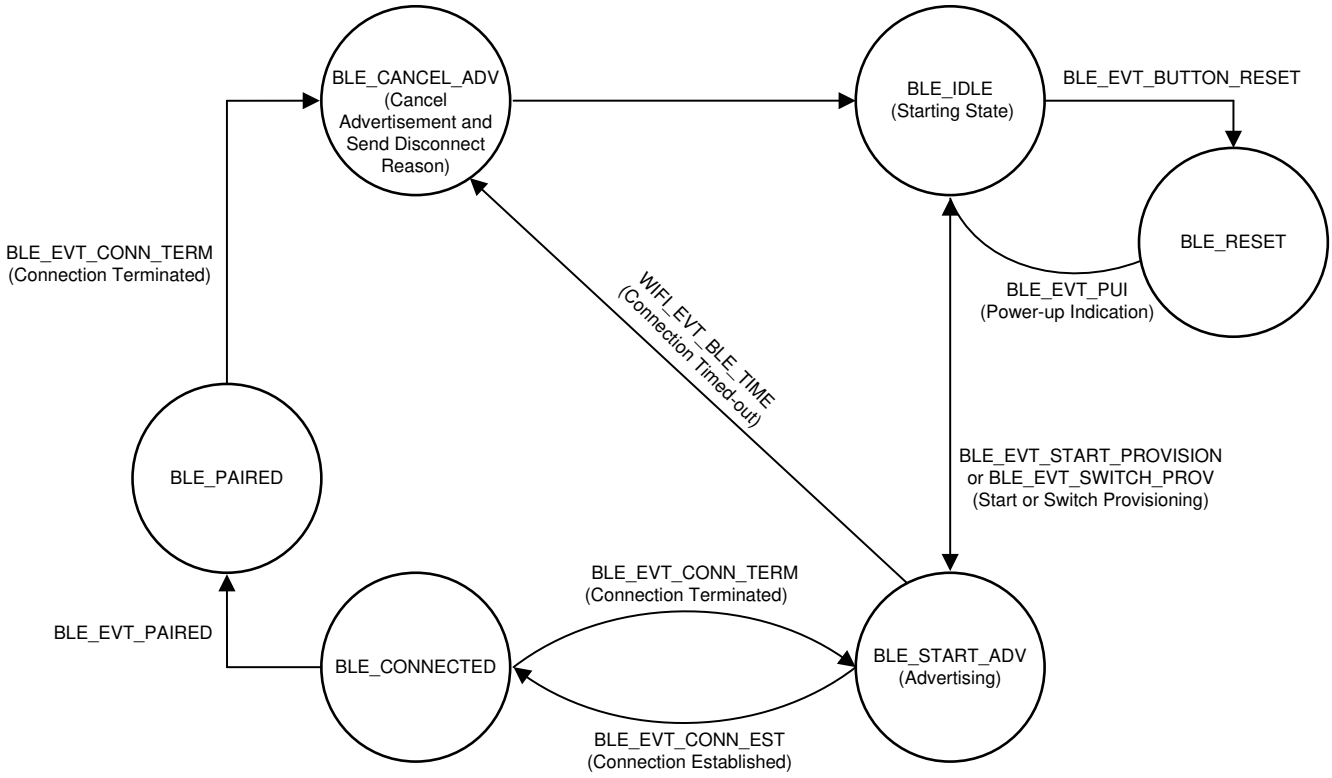


Figure 2-7. BLE Provisioning State Machine

2.4.6 Sending and Receiving Messages Through Cloud

After the TIDC-01005 reference design is successfully connected to a network with Internet access, the device uses the Message Queue Telemetry Transport (MQTT) protocol to send messages to and receive messages from the cloud. Sending and receiving messages through the cloud makes it possible for the system to be controlled and monitored from anywhere, as long as a user has Internet access. To control the system, a user simply sends a message to the system through the cloud server from another client, such as a smartphone, tablet, or PC. When the cloud server receives a message destined for a specific end system, the server forwards the message onto that system. The TIDC-01005 system is also designed to push status updates to clients that request them through the cloud server.

In the TIDC-01005 reference design, the lock can be controlled (locked or unlocked) by sending messages to the CC3220 through the cloud. The state of the lock (locked or unlocked) can be monitored by polling the cloud for an update on the lock state. By default, the software included with the TIDC-01005 uses an open Eclipse IoT MQTT development broker to handle messaging through the cloud. However, the cloud broker must be updated by the developer to a production broker when creating an actual Wi-Fi electronic smart lock design based on the TIDC-01005.

2.4.6.1 Message Queue Telemetry Transport Protocol

MQTT is a light-weight, machine-to-machine, connectivity protocol. MQTT is based on the publish/subscribe messaging model, and it is designed to be used on the top of the TCP/IP protocol. The key benefits of MQTT are that it requires a small memory footprint and low network bandwidth. MQTT also provides fast response time and ease of scalability for applications with a low-power requirement. These features make it an ideal communication protocol for battery-powered, embedded connectivity solutions such as electronic smart locks and door keypads and readers.

2.4.6.2 MQTT Client Implementation

The TIDC-01005 leverages the MQTT client library from the SimpleLink Wi-Fi CC3220 device SDK, to communicate with the cloud. When building an application based on the MQTT library, two software threads

are responsible for implementing the MQTT client functionality: an overall MQTT Thread and an MQTT Client Thread. The overall MQTT Thread configures the parameters of the MQTT connection, establishes the connection with the MQTT broker, spawns the MQTT Client Thread, and then handles all messages received through the MQTT client callbacks. The MQTT Client Thread runs a task that is implemented in the MQTT library and is dedicated to receiving incoming messages from the MQTT connection, then passing the messages to the MQTT client callback.

In the TIDC-01005, the MQTT client task is implemented as a state machine that waits for the system to be connected to a local network and then attempts to connect to an MQTT broker. When the system is connected to the local network and the broker, the MQTT task runs the main loop to handle received messages. The main loop of the MQTT Thread (MQTT Client Connected state) is intended to be customized by the developer, based on the application needs. Figure 2-8 shows a diagram of the MQTT client task state machine.

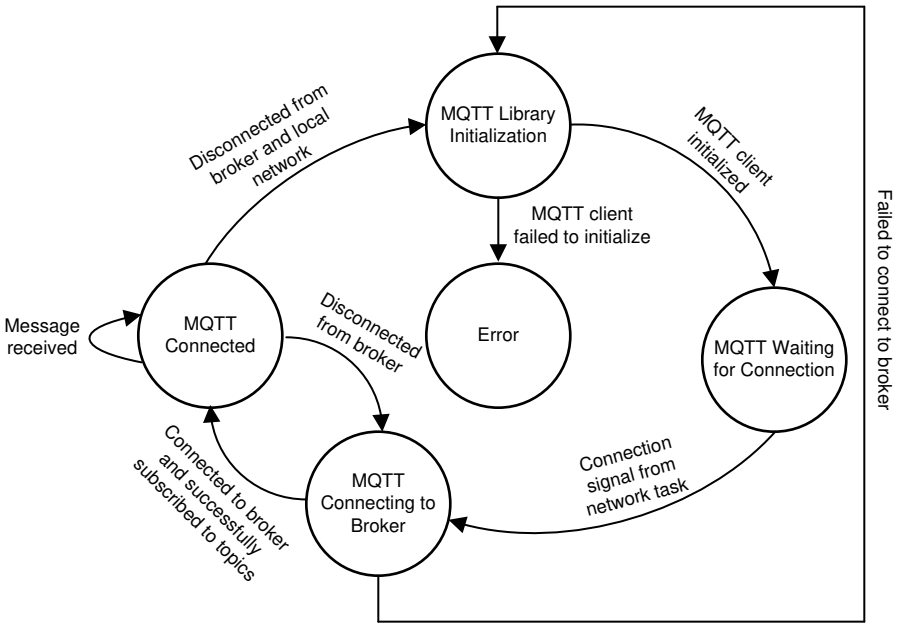


Figure 2-8. MQTT Thread Main Loop

When the system successfully enters the MQTT Connected state, the MQTT client task in the TIDC-01005 acts as an interface, for the user to access the electronic smart lock through the cloud. In the TIDC-01005, the client task uses multiple MQTT topics and messages to enable remote control of the system.

The messages received by the client are handled according to the decision tree shown in Figure 2-9.

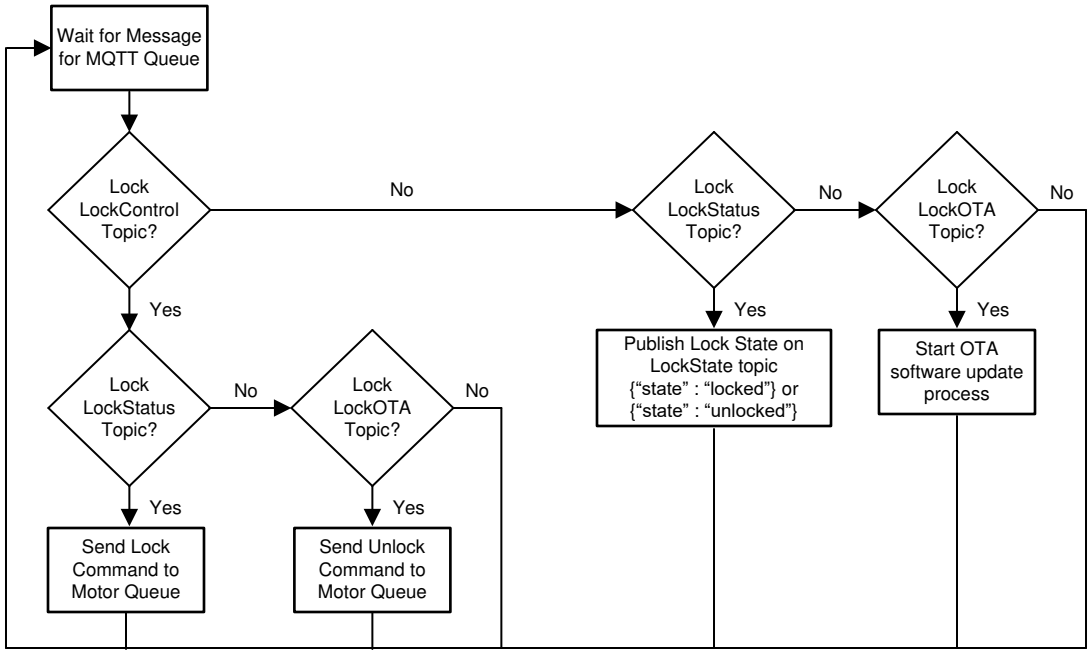


Figure 2-9. MQTT Message Handling

2.4.7 Over-the-Air Updates

An OTA update refers to a software update that is wirelessly transferred and installed on a system. Wi-Fi-enabled systems can receive software updates directly from cloud servers, which makes it easier for vendors to distribute software updates to a large number of devices deployed across different regions and in various locations.

It is important for all products that connect to peers through the Internet to support software updates, because they are necessary to ensure that components of the software that enable security features stay up-to-date over the lifetime of a product. For example, the files needed to support TLS authentication and connect with a peer through the Internet may need to be updated over time. In addition to supporting the security life-cycle of a product, software updates also let developers enable new features for end customers after the product is sold.

The SimpleLink Wi-Fi SDK includes an OTA library that can be used to quickly implement OTA updates in an application. The library includes support for both Dropbox and GitHub APIs, but the libray can also be modified to support alternate content delivery networks (CDN). The TIDC-01005 demonstrates an OTA update solution for a Wi-Fi electronic smart lock using the OTA library with the Dropbox API.

The OTA process in the wifi_doorlock application is implemented in a software task that executes its own state machine. When the OTA task starts, it prevents the user from triggering an OTA update until the system has successfully connected to a local network and acquired an IP address. When connected, the task enters an idle state where it remains until an OTA update is triggered by the user. The user can trigger an OTA update by using MQTT to publish a message to the CC3220 device on the LockOTA topic. When started, the OTA task enters the OTA Run State, where the task downloads the components of the software update and writes the update to the file system until the update is complete. After all the components are downloaded and have been stored on the external serial flash, the OTA task sends a reset request to the control task, to trigger the system to reboot and run with the updated software.

Figure 2-10 shows a diagram of the state machine implemented by the OTA task.

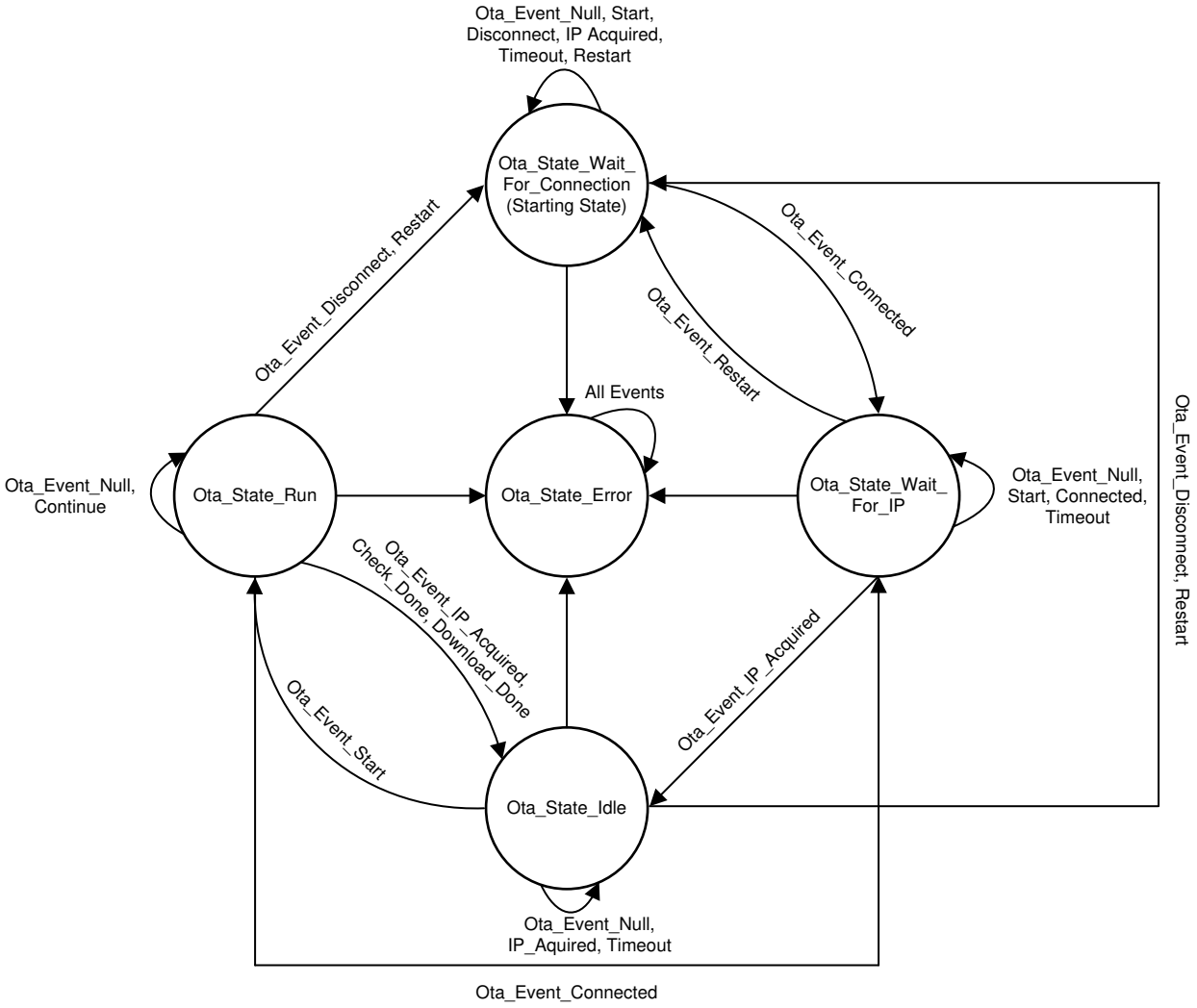


Figure 2-10. OTA Task State Machine

2.4.7.1 HyperText Transfer Protocol

The OTA library used in this example implements a simple HyperText Transfer Protocol (HTTP) client, which supports the Dropbox API and can be used to download the software update. The source for the simple HTTP client implementation is in the OtaHttpClient.h/.c files of the OTA library. For additional details on the implementation of the cloud OTA solution used in this design and the SimpleLink OTA library, see the SimpleLink Academy Training on Wi-Fi OTA.

2.4.8 Security Enablers

IoT products and systems, such as Wi-Fi-enabled electronic smart locks, may hold or exchange information that is sensitive or private, which is why security is an important design consideration. Private information may include passwords, keys, credentials, configurations, personal information, vendor intellectual property (IP), and more. In the case of an electronic smart lock, the security features supported are especially important, because locks are intended to provide security to buildings.

The TIDC-01005 demonstrates a few key security features for Wi-Fi-enabled electronic smart locks.

- Secure boot
- Secure sockets
- OTA update using digital signatures, failsafe files, and bundle protection
- File system security

2.4.8.1 Secure Boot

Secure boot refers to the process of verifying software that is loaded from external memory into on-chip memory on an embedded processor. The CC3220S has a secure boot mechanism built into its ROM bootloader, to help prevent the device from loading code from its external serial flash that has not been signed with an acceptable digital signature. During the secure boot process, the CC3220S validates the integrity and authenticity of the runtime binary (MCU image). The UniFlash ImageCreator tool for CC3120 and CC3220 devices can be used to generate images with digital signatures that can be programmed to the external serial flash used by the CC3220S. Secure boot is an important feature for electronic smart locks, because it can enable developers to prevent the system from being compromised by malicious code, which could cause the system to stop functioning properly or expose the system to unauthorized access.

2.4.8.2 Secure Sockets

The SimpleLink Wi-Fi CC3x20 devices include embedded, standard-compliant, secure transport layer (TLS/SSL) stacks, which are network protocols that involve cryptographic paradigms designed to provide communications security over a TCP/IP connection. Using secure sockets can help protect information sent to peers through the Internet. Embedding the TLS/SSL stacks in the device helps simplify the code needed to create secure socket connections and helps keep the MCU memory free for the user application. The TIDC-01005 uses secure socket connections during AP provisioning (HTTPS server), when communicating with the Eclipse IoT broker to monitor and control the lock (MQTT client over TLS), and while performing OTA software updates through Dropbox (HTTPS client).

2.4.8.2.1 Hardware Accelerators

As described in the previous section, secure sockets rely on many cryptographic algorithms. In the SimpleLink Wi-Fi CC3x20 devices, hardware accelerators are used to offload many of the intense arithmetic calculations involved in these cryptographic algorithms. Offloading the operations to the hardware accelerators helps keep the network processor available to carry out other tasks and speeds up TLS/SSL connection times.

To find out more about the hardware cryptography accelerator, see the CC3220 Technical Reference Manual. For more information on the supported cryptographic algorithms, see the CC3120, CC3220 Network Processor Programmer's Guide.

2.4.8.2.2 Simple Network Time Protocol

When creating a secure socket connection, the server sends a message to the client containing the certificate chain of the server. The client must determine if the certificate chain is valid for the connection, which includes verifying that the current date and time is within the validity period of the server certificate. The TIDC-01005 uses a Simple Network Time Protocol (SNTP) implementation to automatically retrieve the current network date and time from an NTP server and set the internal RTC of the CC3220S. The date and time of the RTC are then used to verify the server certificates are valid when using secure sockets.

2.4.8.3 File System Security

SimpleLink Wi-Fi CC3x20 devices require an externally-attached, nonvolatile memory (NVM) in the form of a serial flash (SFLASH) device. Data stored on the SFLASH is organized in a file system, which can be accessed by the application MCU through an API built into the SimpleLink Wi-Fi host driver. The CC3220S includes multiple, built-in, file-system security enablers to help developers protect information stored on the file system. Some of the key file-system security enablers demonstrated in the TIDC-01005 include the following:

- Encryption – Files are stored and encrypted using a standard, AES-128 encryption
- Cloning protection – The entire file system content is readable only by the device instance which first booted the image, because the image is encrypted with a device-unique key.
- Access control – File access may be restricted to the rightful owner, per access type
- Integrity verification – The integrity of the file system structure is verified to check whether the file system was manipulated.
- Recovery mechanism – The file system can be restored to the initial programmed configuration.

Certain system files are considered special, and have security enablers applied by default. For example, the service pack and the trusted root-certificate catalog are system files that must be created as secure signed files on all devices. For the CC3220S, the runtime binary must also be created as a secure-signed file. In addition to these special files, the private key used by the internal HTTPS server must be configured as a secure file when running the application. By default, the private key is a secure file with no signature verification and public write access in the provided UniFlash ImageCreator bundle for the TIDC-01005.

For more information on the file-system security enablers supported by the CC3220S, see the CC3120, CC3220 Wi-Fi Solution Built-In Security Features application report and the CC3120, CC3220 Network Processor Programmer's Guide.

2.4.8.3.1 Failsafe Files and Bundle Protection

The TIDC-01005 uses two additional security enablers to help protect the file system during infield software updates, such as OTA software updates. The CC3220S provides the option to create files as failsafe files, which means that two banks of memory on the external serial flash are reserved for the file when it is created. Reserving two banks of memory allows two copies of the file to be stored at the same time. During OTA updates, the new copy of a failsafe file is written to the second bank of memory alongside the current version of the file. After the download completes, the new version of a failsafe file is committed (set as the active version) only after the application verifies the file is intact.

The CC3220S also provides a feature called bundle protection, which developers can use to help keep the overall system integrity while updating a collection of failsafe files referred to as a bundle. Because software testing is typically performed on a system with a specific set of file versions, it is important for all files in an update to be applied at the same time. Bundle protection provides the developer with the ability to commit or rollback all files in a bundle together, to prevent the system from ending up with a file system containing a mixture of multiple software versions.

Using failsafe files and the bundle-protection feature is important for electronic smart lock designs that implement OTA updates, because it helps developers maintain the system integrity and ensure the lock remains functional at all times. The OTA update demonstrated by the default TIDC-01005 application uses bundle protection for all files in the update.

2.4.9 Low-Power Consumption

TI-Drivers include a power manager that puts the MCU subsystem of the CC3220S into low-power modes when the system is idle. In the TIDC-01005, the MCU enters LPDS any time the idle task is executed. The network processor subsystem of the CC3220S uses a policy called Long Sleep Interval (LSI) to increase the amount of

time the Wi-Fi NWP spends in a low-power mode between AP beacons and broadcasts, which can significantly reduce the average power consumption. The maximum sleep interval allowed by default in the TIDC-01005 software is 500 milliseconds, but the actual amount of time the Wi-Fi NWP spends in LPDS depends on the parameters of the AP (for example, beacon interval and DTIM value). The SNP application for the CC2640R2F is designed to include a built-in power management scheme, to minimize power consumption when the network processor interface is not active. The DRV8837 also supports a low-power sleep mode, which the device enters when the CC3220S drives the nSLEEP pin low. The TIDC-01005 is designed to keep the DRV8837 in sleep mode at all times except for when a lock or unlock command is received by the MQTT client and the motor is being driven.

Multiple measurements of the TIDC-01005 power consumption were made while testing the system and are described in Section 3.

3 Hardware, Software, Testing Requirements, and Test Results

3.1 Required Hardware and Software

The TIDC-01005 is based on the CC3220S-LAUNCHXL, LAUNCHXL-CC2640R2, DRV8837EVM, and BOOSTXL-SENSORS evaluation modules (EVMs). Section 3.1.1 provides a description of how the EVMs can be used together, including the necessary hardware modifications for running the demo. This reference design is based on an existing software application for the CC2640R2F and includes a software reference for the CC3220S, which can be used together to evaluate the system. Section 3.1.2 provides a guide on how to use the software. Specifically, Section 3.1.2 describes how to build the application for the CC2640R2F as well as how to use the CC3220S application.

3.1.1 Hardware

The following hardware is required to evaluate the TIDC-01005:

- CC3220S-LAUNCHXL
- LAUNCHXL-CC2640R2F
- BOOSTXL-SENSORS
- DRV8837EVM
- 6-V, brushed DC motor
- 5x jumper wires
- Two-pin jumper
- Micro USB cable (included with LaunchPad)

The following sections describes how the EVMs must be configured (including necessary hardware modifications) and connected together to run the TIDC-01005 demo.

3.1.1.1 CC3220S LaunchPad™ Development Kit

To run the demo, the CC3220S LaunchPad kit must be configured with the jumpers highlighted in Figure 3-1 placed. The placement of jumpers on the SOP header depends on whether the software is being debugged or run standalone from the external serial flash.

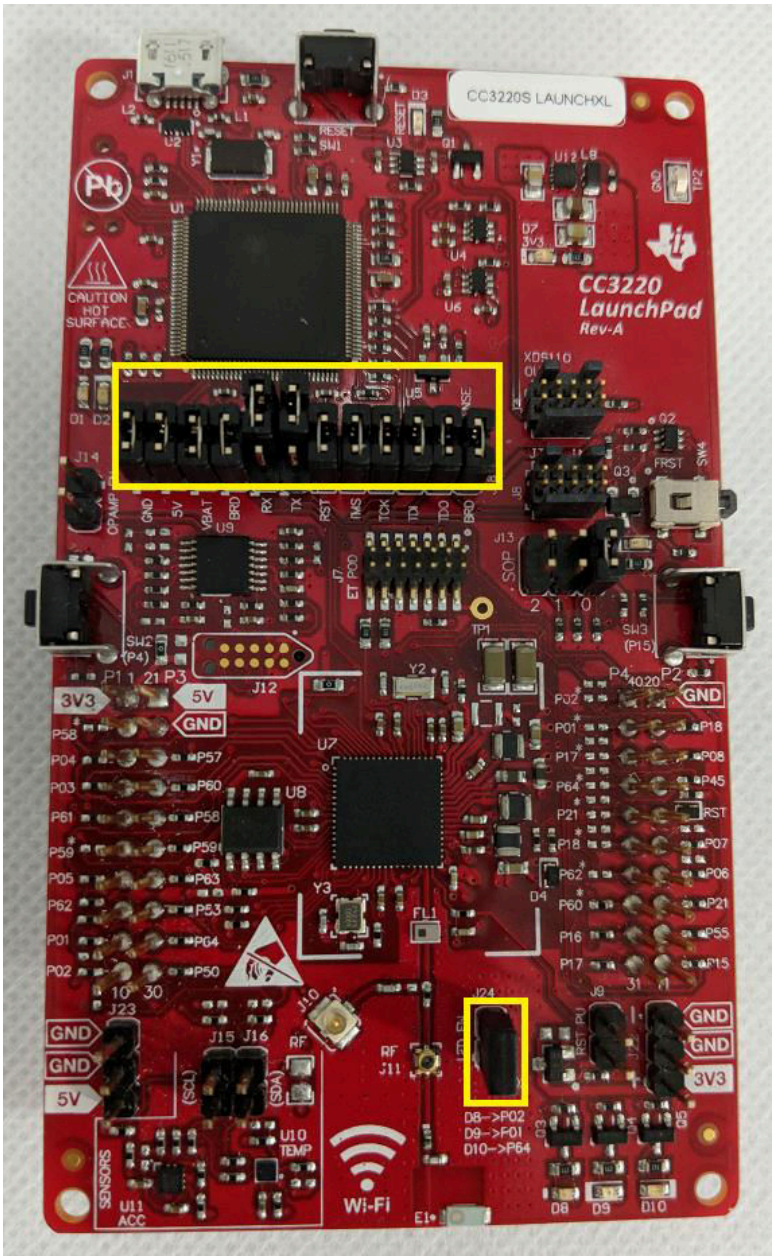


Figure 3-1. CC3220S-LAUNCHXL Jumper Configuration

The CC3220S-LAUNCHXL must be modified when measuring the power consumption of the system. Section 3.2 describes the modification, because the modification is not necessary to run the software application and demonstrate the system.

3.1.1.2 CC2640R2F LaunchPad™ Development Kit

All jumpers must be removed from the CC2640R2F LaunchPad kit before connecting it to the CC3220S LaunchPad kit to run the demo. However, the steps for programming the LAUNCHXL-CC2640R2 described in Section 3.1.2 must be completed before removing all jumpers and assembling the hardware. Figure 3-2 shows the LAUNCHXL-CC2640R2 when programmed.

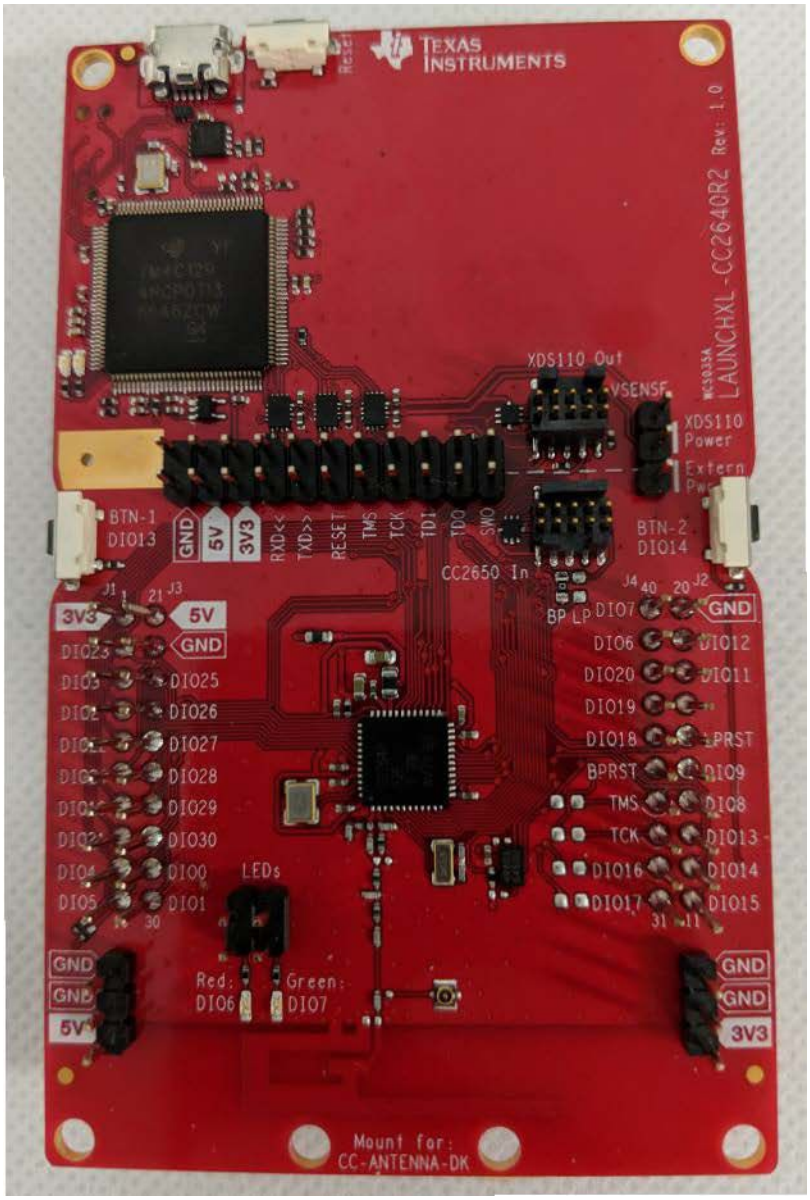


Figure 3-2. CC2640R2F LaunchPad Kit Configuration

The default pin mapping of the SNP images for the CC2640R2F, provided in the BLE plugin, lead to pin conflicts with the CC3220S LaunchPad kit and the sensor BoosterPack plugin module. To allow all three EVMs to be connected together with minimal hardware modifications, the user must rebuild the SNP application (simple_np) from the SimpleLink CC2640R2F SDK, with an updated pin mapping before programming the board. The process for updating the pin mapping of the CC2640R2F is described in Section 3.1.2, but it results in the following connections between the CC3220S and the CC2640R2F (see Table 3-1).

Table 3-1. CC3220S and CC2640R2F Pin Mapping

CC3220S	CC2640R2F
Pin 3 (UART0 TX)	DIO 2 (UART RX)
Pin 4 (UART0 RX)	DIO 3 (UART TX)
Pin 62 (MRDY)	DIO 21 (MRDY)
Pin 8 (SRDY)	DIO 11 (SRDY)
Pin 7 (RESET)	BPRST (RESET)

All of the pins listed in Table 3-1 are connected when stacking the LaunchPads to assemble the demo, except for the RESET line. A single, 2-pin jumper can be used to connect Pin 7 of the CC3220S to the BPRST pin of the CC2640R2F when the LaunchPad kits and BoosterPack module are stacked, as described in Section 3.1.1.5.

3.1.1.3 Sensor BoosterPack™ Connections (BMI160)

Only the BMI160 sensor on the Sensor BoosterPack is used in the TIDC-01005. When the BMI160 sensor is enabled, it signals to the CC3220S that data is ready using a single interrupt pin and transfers data to the CC3220S over I^2C. Table 3-2 lists the pins used to support the interface between the CC3220S and the BMI160.

Table 3-2. CC3220S Pins Used by BMI160

CC3220S PIN	FUNCTION
Pin 1	I^2C clock
Pin 2	I^2C data
Pin 61	Interrupt

3.1.1.4 DRV8837EVM Modifications and Connections

By default, the DRV8837 on the DRV8837EVM is controlled by an onboard MSP430™ MCU, and the entire DRV8837EVM is powered through a micro-USB interface. To control the DRV8837 from the CC3220S, the following steps must be taken to modify the DRV8837EVM and connect it to the CC3220S LaunchPad.

1. Remove R3 and R4 (0-Ω resistors) from the back of the DRV8837EVM to disconnect the IN1/IN2 signals of the DRV8837 from the MSP430 (see Figure 3-3).
2. Provide power to the DRV8837 from the CC3220 LaunchPad by connecting the 5-V, GND pins of J23 on the CC3220S LaunchPad to J1 on the DRV8837EVM see Figure 3-4).
3. Connect to the IN1 and IN2 test points on the DRV8837EVM from the CC3220S LaunchPad headers corresponding to P16 and P17 (see Figure 3-4).

Figure 3-3. R3 and R4 Location on Back of DRV8837EVM

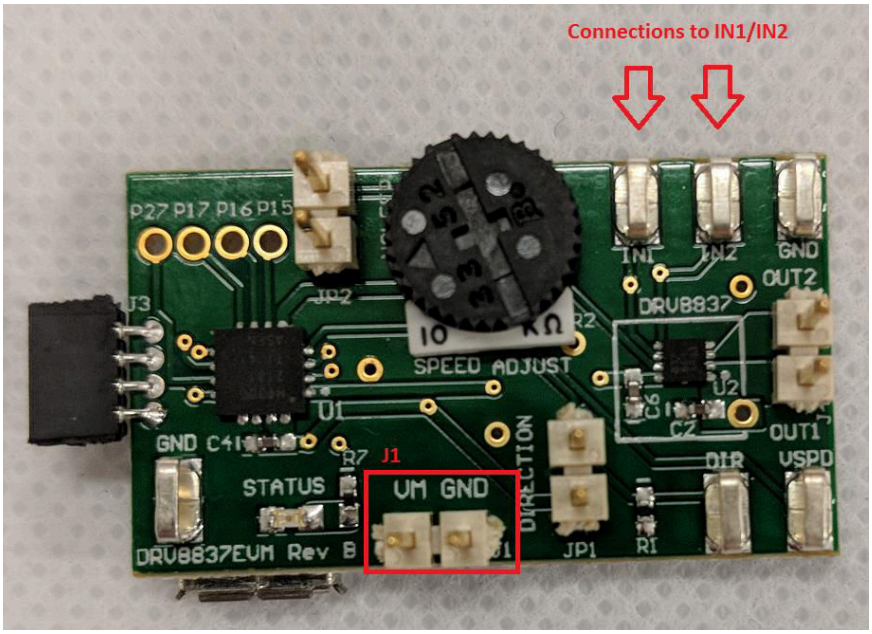


Figure 3-4. J1 and IN1/IN2 Highlighted on Front of DRV8837EVM

In this example, two GPIOs and one PWM signal are used to control the DRV8837. Table 3-3 lists the connections required between the LaunchPad and the modified DRV8837EVM when using the included reference software.

Table 3-3. CC3220 LaunchPad™ to DRV8837EVM Connection List

CC3220 LAUNCHPAD	DRV8837EVM
P18	nSleep (JP2)
P16	IN1 (TP1)
P17	IN2 (TP2)
5V	VM (J1)
GND	GND (J1)

Further modifications of the DRV8837EVM are necessary to accurately measure the power consumption of the motor. These modifications are described in Section 3.2.

3.1.1.5 Assembling EVMs

When assembling the TIDC-01005 demo, CC3220S-LAUNCHXL, LAUNCHXL-CC2640R2, and BOOSTXL-SENSORS must all be connected by stacking the 20-pin headers on the EVMs in the following order (with 1 being the top of the stack and 3 being the bottom).

1. BOOSTXL-SENSORS
2. CC3220S-LAUNCHXL
3. LAUNCHXL-CC2640R2

When the LaunchPads and BoosterPack are stacked, ensure the 2-pin jumper is placed horizontally across J4-J6 and J2-J6 on the sensor BoosterPack, to connect pin 7 of the CC3220S and BPRST of the CC2640R2F. Figure 3-5 shows how the assembly appears.

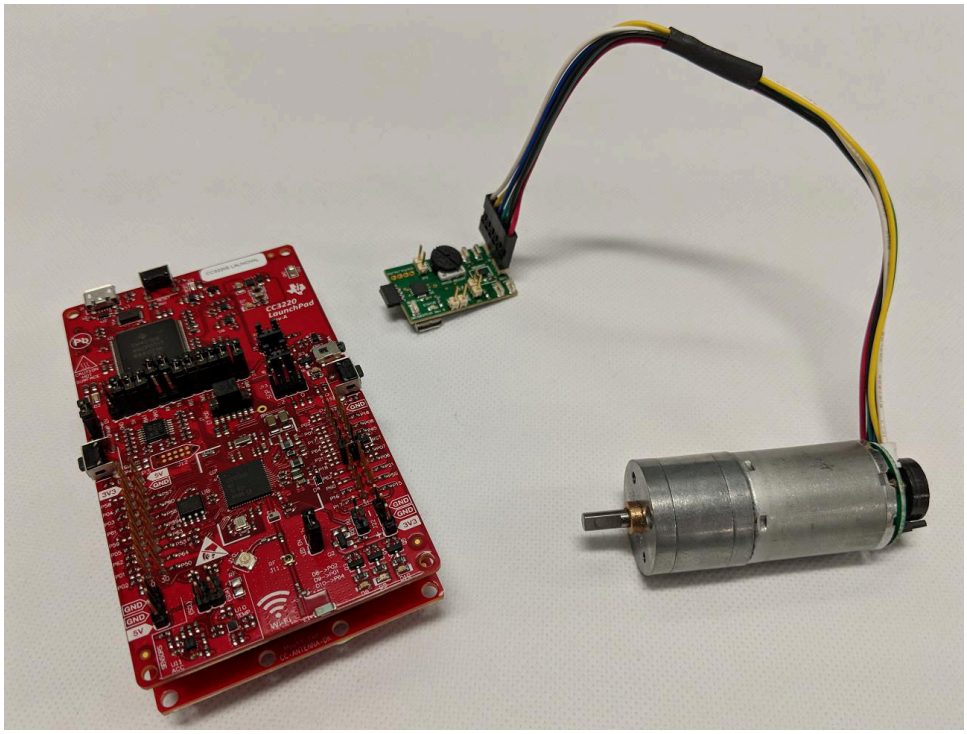


Figure 3-5. LaunchPad™ and BoosterPack™ Stackup

Last, connect the stackup to the DRV8837EVM according to Table 3-3 using the jumper wires (connections shown in Figure 3-6). The entire system can be powered by connecting the USB cable to the USB connector on the CC3220S LaunchPad.

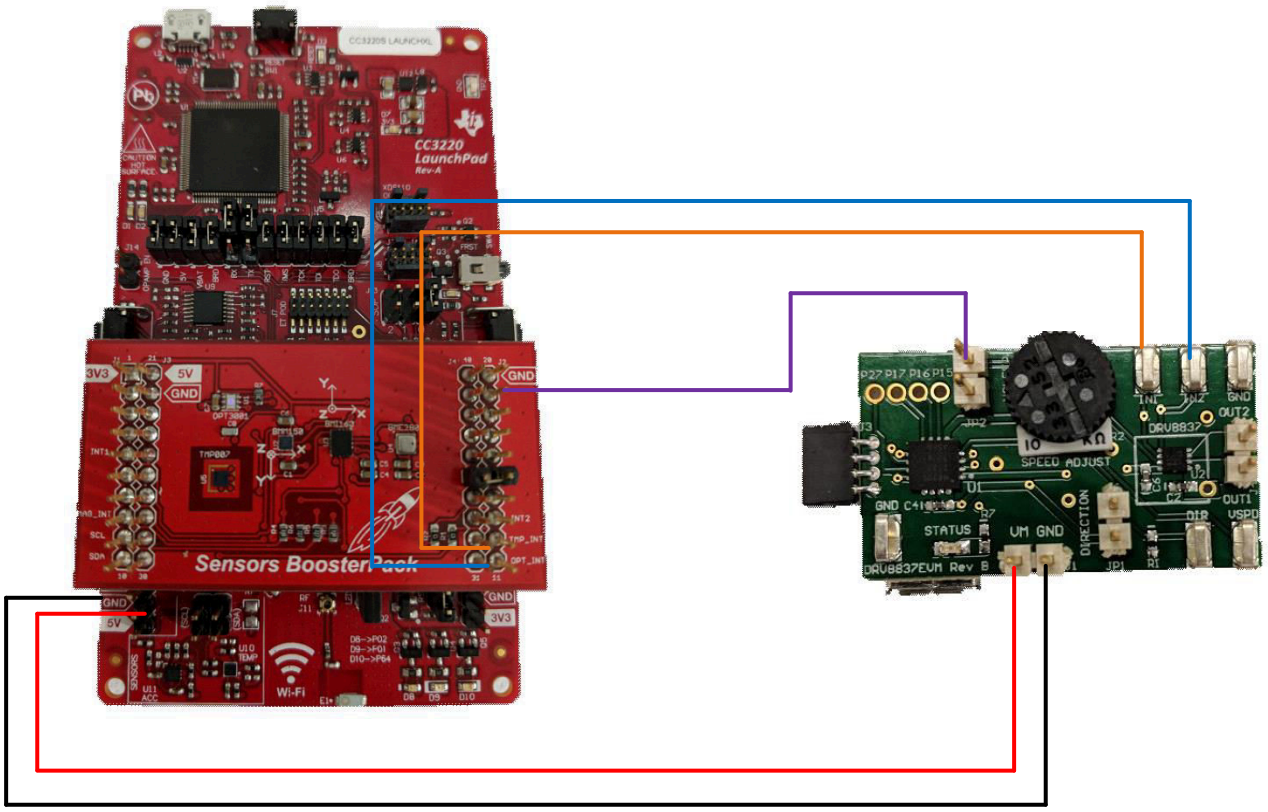


Figure 3-6. Jumper Wire Placement to Connect DRV8837EVM to System

3.1.2 Software

The software created for this reference design is based on the SimpleLink SDK and is intended to enable maximum code portability within the SimpleLink platform. By using the SimpleLink platform, developers can invest time in creating software once and then reuse the software across different MCUs with integrated wired or wireless technologies. This feature enables the design to be flexible and makes it easier to develop a set of products all based on the same core code.

The software for this reference design shows how a low-power, Wi-Fi-enabled door lock with multiple built-in security features can be created using the SimpleLink Wi-Fi Wireless MCU. Specifically, the software demonstrates how to implement the following key functions in a Wi-Fi-enabled electronic smart lock.

- Wi-Fi provisioning (AP provisioning, SmartConfig, and BLE provisioning), to connect a door lock to a local network
- SNTP, to acquire network time and set the system RTC
- MQTT over TLS, to control and monitor the door lock state over a secure server connection
- OTA software update using HTTPS with Dropbox API v2
- Motor control through the DRV8837
- Power management, to extend system battery-life

3.1.2.1 Getting Started With Software

Currently, support for editing and building the source is limited to Code Composer Studio™ (CCS) only, but the project could be moved to other IDEs and compilers, such as IAR or GCC. To use the example, download the following:

- Latest version of Code Composer Studio
- UniFlash 4.3 (with ImageCreator)
- SimpleLink Wi-Fi CC3220 SDK (v2.10.00.04)
- SimpleLink SDK Bluetooth Plugin (v1.40.00.42)
- SimpleLink SDK Sensor and Actuator Plugin (v1.20.00.02)
- SimpleLink CC2640R2 SDK (v1.50.00.58)

When the tools and SDK are installed, run the installer for the TIDC-01005 software. The installer will attempt to place the software in the CC3220 SDK by default. If the default path used by the installer does not match the actual location of the CC3220 SDK on the PC, it must be updated. Once installed, the zip file containing the software package must be extracted into the CC3220 SDK under the RTOS demos folder for the CC3220S (extracted contents should end up under <SDK Install Location>/simplelink_cc32xx_sdk_2_10_00_04/examples/rtos/CC3220S_LAUNCHXL/demos). The installed package includes the software source files along with a UniFlash project (.zip file), which provides two different options for evaluating the system.

3.1.2.1.1 Build simple_np Application and Flash CC2640R2F

Regardless of which method is used to evaluate the CC3220S application, the CC2640R2F must first be programmed with the SNP (simple_np) application. The following steps describe how to modify and build the simple_np project.

1. Import the simple_np project from the CC2640R2 SDK into a CCS workspace.
2. Right-click on the simple_np_cc2640r2lp_app project that appears in the project explorer, and select the Properties button.
3. Open the Predefined Symbols view in the Project Properties window (under Build → ARM Compiler → Predefined Symbols).
4. Ensure the following symbols are defined:
   - NPI_USE_UART
   - POWER_SAVING
5. Open the *board_cc2640r2lp.h* file used by the project (at *<SDK Install Location>\simplelink_cc2640r2_sdk_1_50_00_58\examples\rtos\CC2640R2_LAUNCHXL\blestack\simple_np\src\app\board_cc2640r2lp*).
6. Change the Board_MRDY and Board_SRDY definitions, as shown in Figure 3-7.
7. Right-click on the simple_np_cc2640r2lp_app project in the workspace, and select the Rebuild Project button.

```
150 /* MRDY/SRDY Board */
151 #define Board_MRDY                      IOID_21 //IOID_23         /* MRDY */
152 #define Board_SRDY                      IOID_11 //IOID_12         /* SRDY */
```

Figure 3-7. Updated SRDY and MRDY Pin Definitions

Rebuilding the application generates a file called *simple_np_cc2640r2lp_app.hex* in the FlashROM_StackLibrary folder of the project. Use the Flash Programmer 2 tool to program the LAUNCHXL-CC2640R2 with the simple_np_cc2640r2lp_app.hex file. After the board is programmed, all of the jumpers on the board can be removed, as described in Section 3.1.1.

3.1.2.1.2 Use Premade UniFlash ImageCreator Project

To evaluate the software out-of-box using the premade UniFlash project, do the following:

1. Download the CC3220 SDK and wifi_doorlock.zip (follow the instructions in Section 3.1.2.1 to install the wifi_doorlock demo).
2. Open UniFlash, and select CC3120/CC3220 or the CC3220 LaunchPad.
3. Select Start Image Creator (for CC3120/CC3220).
4. Choose Manage Projects.
5. Select Import Project from ZIP file.
6. Navigate to the wifi_doorlock example in the CC3220 SDK, and select the zip file with the latest version in the uniflash folder (wifi_doorlock/uniflash/wifi_doorlock_RS_tirtos_<version>.zip).
7. Locate the UniFlash project named wifi_doorlock_RS_tirtos in the list of Available Projects.
8. Select Open Project.
9. Connect to the CC3220S-LAUNCHXL by pressing the Connect button on the right-hand side.
10. Press the Generate Image button (flame symbol on the right-hand side).
11. Choose Program Image (Create & Program) at the top of the Generate Image page.

These steps result in the software application and all associated system files being written to the external serial flash of the CC3220S. The example, as provided in the UniFlash bundle, is designed to provide updates on the application state by printing debug messages to a serial terminal. To see the debug messages, open a serial terminal application on the computer that the LaunchPad is connected to, and connect to the COM port called XDS110 Class Application/User UART. Table 3-4 lists the settings for how the serial port is configured.

Table 3-4. Serial Port Settings

SETTING	VALUE
Baud rate	115200
Data	8 bit
Parity	—
Stop	1 bit
Flow control	—

When successfully connected to the serial terminal, restart the CC3220S by pressing the RESET switch at the top of the CC3220S-LAUNCHXL. The application boots from the external serial flash and starts running. Upon a restart, the application checks to see if a network profile exists in the file system of the CC3220S. If a profile exists, the system automatically connects to the AP. Otherwise, the system waits to be provisioned over a Bluetooth low energy connection. Figure 3-8 shows the messages that appear in the terminal when the system starts up and waits to be provisioned.

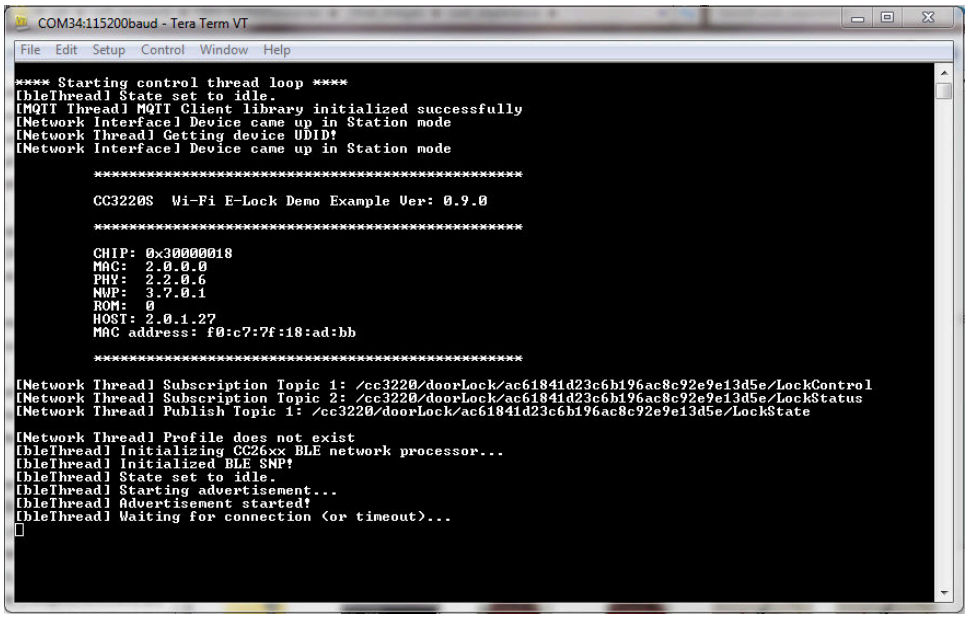


Figure 3-8. Debug Print Messages in Serial Terminal on First Application Start

To connect the device to a network and evaluate the rest of the application, see Section 3.1.2.3.1. When working with the premade UniFlash Image-Creator project, the system uses power-management by default.

3.1.2.1.3 Importing Project Source Files Into CCS

The downloaded software includes source files and a premade CCS project, which enables users to quickly start debugging, modifying, and rebuilding the code used by the design. When the software is successfully extracted into the CC3220S RTOS demos folder of the SimpleLink Wi-Fi SDK, the software can be directly imported into a CCS workspace by doing the following:

1. Right-click in the Project Explorer.
2. Select the Import → CCS Project option.
3. Navigate to the wifi_doorlock folder in the SDK (<SDK Install Location>/ simplelink_cc32xx_sdk_2_10_00_04/examples/rtos/CC3220S_LAUNCHXL/demos/wifi_doorlock).
4. Select the Ok button.
5. A project called wifi_doorlock_CC3220S_LAUNCHXL_tirtos_ccs appears in the Discovered Projects list of the Import window. Select the project and then press Finish.

When the project imports, it also imports a tirtos_build project for the CC3220S, if one does not already exist in the workspace.

Note

For the wifi_doorlock project to build successfully, both the OTA library and the SimpleLink Wi-Fi host driver must be imported into the same workspace and successfully built first.

Prior to building the wifi_doorlock_CC3220S_LAUNCHXL_tirtos_ccs project, users must also import and build the OTA library and the SimpleLink Wi-Fi host driver library projects from the SDK. To import and build the OTA library, do the following:

1. Right-click on the Project Explorer.
2. Select the Import → CCS Project option.
3. Navigate to the source/ti/net folder of the SDK.
4. Select the Ok button.
5. Select the ota library from the list of Discovered projects, then click Finish.

Before building the ota library, a couple of changes must be made to the project properties. The changes can be made by doing the following:

1. Right-click on the ota project in the workspace and select Properties.
2. Select Resource → Linked Resources from the list on the left-hand side of the CCS Project Properties window.
3. Choose the Linked Resources tab.
4. Edit all of the paths that appear as Invalid Locations, by replacing the PROJECT_LOC symbol in the path with ORIGINAL_PROJECT_ROOT.
5. Select Build → Arm Compiler → Optimization from the list on the left-hand side of the CCS Project Properties window.
6. Change the Optimization level to 2-Global Optimizations.
7. Select Build → Arm Compiler → Advanced Options → Runtime Model Options.
8. Enable the option to place each function in a separate subfunction (set to on).
9. When all the changes are made, select Apply and then Close.

To import and build the SimpleLink Wi-Fi host-driver library, do the following:

1. Right-click on the Project Explorer.
2. Select the Import → CCS Project option.
3. Navigate to the source/ti/drivers/net/wifi folder of the SDK.
4. Select the Ok button.
5. Select the simplelink project from the list of Discovered projects, then click Finish.
6. Right-click on the simplelink project when it appears in the workspace.
7. Select Build → Arm Compiler → Optimization from the list on the left-hand side of the CCS Project Properties window.
8. Change the Optimization level to 4-Whole Program Optimizations.
9. Select Build → Arm Compiler → Advanced Options → Runtime Model Options.
10. Enable the option to place each function in a separate subfunction (set to on).
11. Select Apply and then Close, to apply the change.

After making the changes to both the ota and simplelink project properties, the libraries can be rebuilt and then the wifi_doorlock_CC3220S_LAUNCHXL_tirtos_ccs project can be successfully built.

By default, the CCS application is configured without power management enabled, to let the debugger maintain a connection to the CC3220S while the wifi_doorlock application is running. Without the power policy enabled, the application can be debugged, as long as the necessary user files are first added to the CC3220 file system. Alternatively, the CCS project can be used to modify and rebuild the application, then replace the mcuimg.bin file in the provided UniFlash ImageCreator project. Section 3.1.2.2 describes how to add the necessary User Files when debugging the application using CCS.

The Wi-Fi doorlock software was created so that different key modules in the software can be disabled during development and evaluation. A set of defines at the top of the wifi_doorlock_app.h file control whether different modules are enabled or disabled. Specifically, it is often convenient to disable the provisioning process during development and instead use a statically defined set of AP credentials to connect to a test network. Setting both the APSC_PROVISIONING and BLE_PROVISIONING defines to (0) instead of (1) bypasses the provisioning process and uses statically defined network credentials located in wifi_doolock_app.h to connect the system to a network.

Note

By default, the power manager is disabled in the CCS project, to let the system remain connected to the CCS IDE while debugging. Enable the power manager by adding USE_POWER_POLICY to the list of Predefined Symbols in the wifi_doorlock_CC3220S_LAUNCHXL_tirtos_ccs project.

3.1.2.2 User Files

The Wi-Fi doorlock example application needs certain files to be present in the CC3220 file system to successfully run. These files are already included as part of the premade UniFlash ImageCreator Project, but the user must program them to the serial flash prior to debugging the application using CCS.

Table 3-5 lists the files that must be on the file system for the demo and their function.

Table 3-5. User Files Needed to Run Wi-Fi DoorLock Demo

FILE	FUNCTION
dummy-root-ca-cert	Dummy certificate chain used to verify application binary for secure boot evaluation (must be replaced by user in production)
dummy-trusted-ca-cert	
dummy-trusted-cert	
dummy_ota_vendor_cert.der	Dummy certificate used to verify the signature on each file in the OTA bundle during secure OTA update (must be replaced by user in production)
dummy-root-ca-cert	Certificate and private key for running internal HTTP server with secure connection (HTTPS). **Note** Server can use a certificate chain in PEM format in place of the single certificate (must be replaced by user in production).
dummy-root-ca-cert-key	
eclipsecert.der	Root CA of Eclipse IoT MQTT server used by the client to verify the server during TLS connection
digicertca.der	Root CA of api.dropboxapi.com used by the CC3220 to verify the Dropbox server during TLS connection

The root CA for the Eclipse server and Dropbox can be found using a website like ssltools.com or a tool such as OpenSSL or Curl. The dummy certificates and keys are located in the SimpleLink Wi-Fi SDK under *tools/cc3220_tools/certificate-playground* and *tools/cc3220_tools/ota-example-cert*.

Note

The dummy certificates and keys are to be used only for demo and development purposes. It is the responsibility of the developer to get actual certificates and keys for code signing and SSL/TLS when needed.

3.1.2.3 Run Wi-Fi® Doorlock Demo

3.1.2.3.1 Connect CC3220 to Network

When the example is run out-of-the-box, the example starts by first executing the provisioning process. BLE provisioning starts automatically, but the system can be switched to AP provisioning or SmartConfig by holding the switch on the right side of the CC3220S LaunchPad for 3 seconds.

- To provision the device using BLE provisioning, first download the SimpleLink SDK Explorer mobile application for Android or iOS. Then follow the instructions in the README.html for the ble_wifi_provisioning example of the SimpleLink SDK Bluetooth Plugin.
- To provision the device using AP provisioning or SmartConfig, download SimpleLink Starter Pro mobile application for Android or iOS. Then follow the instructions in the mobile application or see the instructions in Task 3 of the SimpleLink Academy Training for Wi-Fi Provisioning.

Note

AP provisioning for the Wi-Fi doorlock demo has one main difference from the SimpleLink Academy training – the SimpleLink AP is password protected. When prompted by SimpleLink Starter Pro, enter 1234567890 as the security key, to connect to the mysimplelink-xxxxxx device. The default password (1234567890) is only intended to keep the demo simple and must be changed when implementing a real application to maintain security.

Multiple print statements are made to the terminal during provisioning. The device prints when a new profile is added, a successful connection to the AP is made, an IP address is acquired, and when the mobile device used to provision the CC3220 confirms that provisioning succeeded.

Note

A default confirmation method is built into the AP provisioning and SmartConfig process, but the message can be changed by the developer for the final design. If a confirmation message is not received during AP provisioning or SmartConfig (even though the terminal shows that a profile was added and the device connects to the AP), the profile is still valid and stored on the device. Resetting the CC3220S-LaunchPad causes the application to restart and the device to attempt to connect to the stored profile.

The first time BLE provisioning is run, the mobile device used to provision the TIDC-01005 system must pair with the system using BLE secure-simple pairing. When the mobile device first connects to the TIDC-01005 system over BLE, the device prompts the user to enter a key. The key needed to pair with the system is displayed in the terminal application running on the PC, as shown in Figure 3-9. Successfully entering the key allows the two devices to pair.

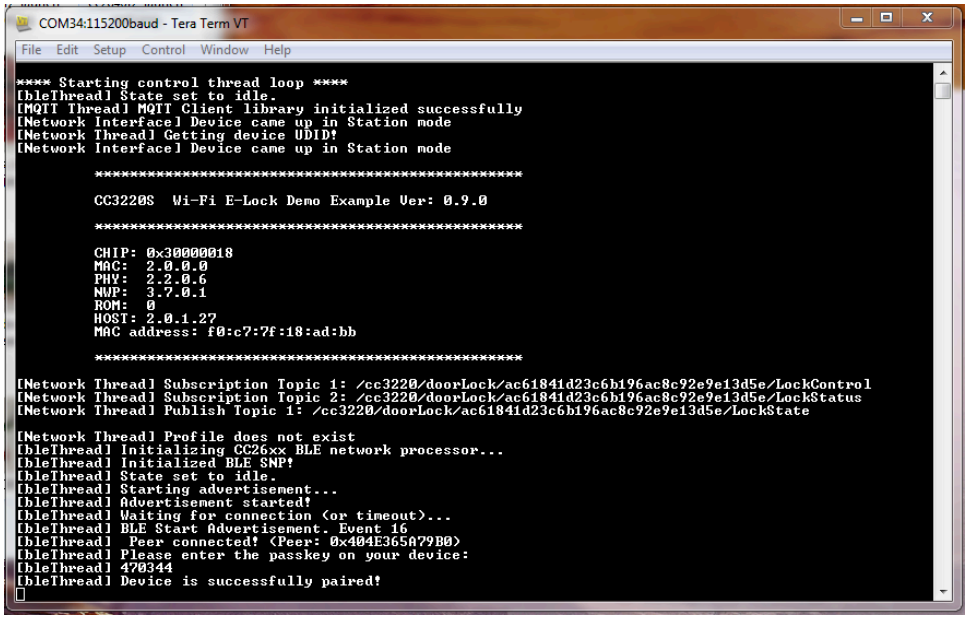


Figure 3-9. BLE Secure-Simple Pairing Key Displayed in PC Terminal

When paired, the user must enter the AP credentials in the SimpleLink SDK Explorer mobile application (see Figure 3-10) and then press the Provision button to send them to the system over BLE.

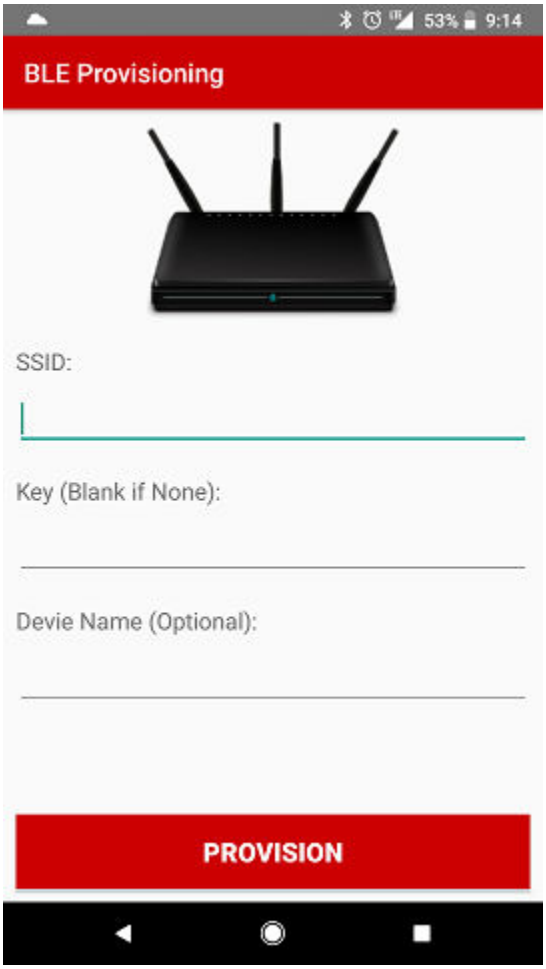


Figure 3-10. SimpleLink™ SDK Explorer GUI For BLE Provisioning

When the credentials are successfully written, the CC3220S MCU reads them from the CC2640R2 over the NPI and tests the connection. If the connection succeeds and confirmation is successfully provided to the user, a profile is added to the CC3220S file system and used for the remainder of the application.

The CC3220S automatically connects to a profile on the file system for all subsequent restarts of the application.

3.1.2.3.2 Networking Functions

After the provisioning process completes, the rest of the doorlock demo starts to run. The main networking functions used in the Wi-Fi doorlock demo are SNTP, MQTT, and HTTP. Immediately after connecting to the Internet, the CC3220 must update its RTC, based on the network time, to enable the CC3220 to later create secure network connections (SSL/TLS). When the RTC is set to the current time, the CC3220 establishes an MQTT connection with an Eclipse MQTT broker (iot.eclipse.org) to notify a remote user of its state and receive lock or unlock commands.

3.1.2.3.2.1 Get Current Date and Time (SNTP)

The CC3220S acquires the current date and time using SNTP. Connecting to the NTP server, requesting the date and time, and interpreting the timestamp is handled by the Net Utils and SNTP libraries included in the CC3220 SDK. These libraries are built to use a predefined set of NTP servers to request the date and time. The application code used to make the request and update the RTC value can be found in the *setTime()* function defined at the top of wifi_doorlock_app.c.

Note

The RTC must be set for the device to verify peer certificates are being used during their stated validity period. Without setting the RTC to the current network time, the connection to the server could fail or the device could be exposed to an insecure connection.

3.1.2.3.2.2 Send and Receive Messages (MQTT)

After the RTC is updated to the current network time, the CC3220 connects to an MQTT broker over TLS, using the MQTT library from the SimpleLink SDK. The MQTT library implements a set of functions that can be used to operate as an MQTT client without dependency on connecting to a specific server. In this example, an open eclipse broker (iot.eclipse.org).

For this demo, the CC3220 subscribes to four different topics, as follows:

1. /Broker/To/cc32xx/doorLock
2. /cc3220/doorLock/<Unique Device ID>/LockStatus
3. /cc3220/doorLock/<Unique Device ID>/LockControl
4. /cc3220/doorLock/<Unique Device ID>/LockOTA

In the topic names, <Unique Device ID> indicates the 128-bit unique device identifier (in hexadecimal format) built into each CC3220S device. This ID makes the topics used by each device unique, which can prevent devices from receiving messages intended for other systems. The actual topic names used for a specific device are printed at the top of the serial terminal when the application restarts. The application dynamically sets the topic names by reading the device ID from the network processor and printing the ID value to the topic strings before they are used in the application.

Only topics 2, 3, and 4 are currently used by the demo. Topic 2 can be used by a peer device to get the current state of the lock. The CC3220 publishes the state of the lock in JSON format in response to any message received on topic 2. To prevent the device from receiving its response to a status request, a separate topic is used by the device to publish the lock state. The CC3220 publishes the lock state on the topic: /cc3220/doorlock/<Unique Device ID>/LockState.

Table 3-6 lists the format of the JSON message sent on the LockState topic.

Table 3-6. LockState Topic Data Format

STATE	DATA
Locked	{"state": "locked"}
Unlocked	{"state": "unlocked"}

Topic 3 can be used to change the state of the lock (locked or unlocked). For example, to put the demo in the locked state, the user must publish the message *lock* to topic 3. To put the demo in the unlocked state, the user must publish the message *unlock* to topic 3. Depending on which command is received on topic 3, the motor

spins clockwise or counter-clockwise. When running the demo while connected to a serial terminal, the received command is printed to the screen along with the sensor data read while driving the motor.

Figure 3-11 shows an example of what is shown in the serial terminal.

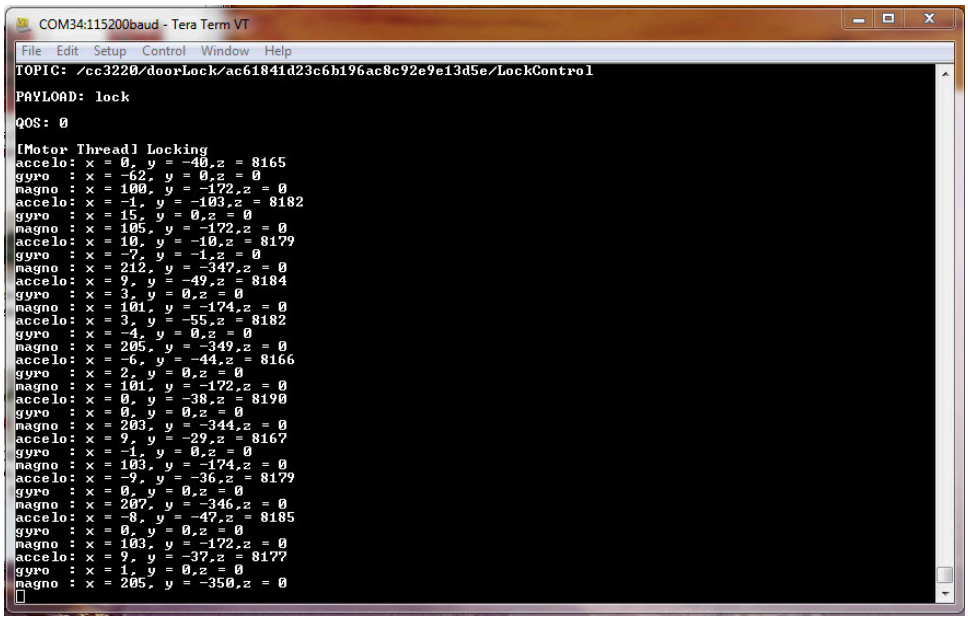


Figure 3-11. Debug Print When Lock Message Sent on LockControl Topic

Topic 4 is used to signal to the device to initiate an OTA update. When the device receives any message published on topic 4, the OTA state machine runs. For a detailed description of the OTA implementation, see the Section 2.4.7 of this document.

Any time the CC3220 receives a message on a topic that it is subscribed to or publishes to the Lock State topic, the device prints a debug message to the serial terminal. This feature can be used to monitor the system response while debugging. Any MQTT client that can connect to the Eclipse IoT broker can be used to test out the functionality of the demo using the topic names shown here. For example, the mobile applications mentioned in the Wi-Fi MQTT SimpleLink Academy Training can be used to test the behavior of a remote client that can control and monitor the state of the demo.

Establishing the MQTT connection over TLS is facilitated by the use of the secure MQTT port (8883) and includes the Root CA for the eclipse server, to enable the CC3220 to validate the server certificate chain. The Eclipse server Root CA must be added to the CC3220S file system before running the demo (already included in the provided UniFlash bundle). In the application, the path to this file on the file system and the secure MQTT port are specified in the MQTT client context struct when SECURE_CLIENT is defined at the top of wifi_doorlock_app.c.

Due to the fact that the software provided is a demo built to be used with a preconfigured broker, there are some limitations to the implementation. Mutual authentication is not supported by the demo, because there is no mechanism configured for the Eclipse broker to identify the CC3220S device. Mutual authentication can be added by the user when using an actual server for production. The demo also skips the domain name verification step and disables the option to verify the server root CA name in the certificate catalog. The certificate catalog verification step must be disabled when using the certificate playground during development, because it does not include all of the actual trusted root certificate authorities. For a production system, the production certificate catalog must be used and TI recommends enabling both certificate catalog verification and domain name verification.

Note

The root CA for the Eclipse broker will expire over time. The user must ensure the certificate loaded onto the CC3220S serial flash is the most recent root CA or the connection to the broker may fail.

3.1.2.3.2.3 Perform Software Update Using Dropbox (OTA Update)

In addition to providing the ability to remotely control and monitor the state of the demo using MQTT, the CC3220 makes it possible to perform a wireless software update for the system through the cloud. This reference design uses the OTA library provided as part of the CC3220 SimpleLink SDK to implement cloud OTA updates. Dropbox is used as the content delivery network (CDN) in this design, but the OTA implementation could be changed to support other CDNs.

When running the demo from the premade UniFlash ImageCreator project, the system uses a preconfigured Dropbox account to perform the OTA update. When the demo is connected to a network, and the user has verified the ability to communicate with the lock through the MQTT broker, a cloud-based OTA update can be triggered by publishing a message from another client on topic 4 that is subscribed to by the CC3220 (discussed in Section 3.1.2.3.2.2). Receiving a message on topic 4 tells the wifi_doorlock application to execute the RUN step of the cloud OTA state machine.

When the OTA is running, the demo establishes a secure HTTP connection to the Dropbox server and checks to see if there is an available software version with a higher version number than the one the device is currently executing. If an update is discovered, the device downloads all the components of the OTA in a .tar file and saves them to the file system in failsafe mode. When the entire update is successfully downloaded, the OTA state machine requests a system reset. The control task executes the reset and the system reboots with the new software version. If the system successfully reboots and reconnects to the network, the system commits the new software version to make it permanent for all future resets.

To run the OTA update supported by this design with a custom Dropbox account and OTA image, perform the following steps:

1. Create and configure a Dropbox developer account, as shown in Steps 1 to 10 of *Getting Started With the Dropbox™ Cloud Delivery Network (CDN)* in the CC3x20 SimpleLink Wi-Fi and Internet of Things Over-the-Air Update application report.
2. Update the otauser.h file of the OTA library, as stated in Step 10 of *Getting Started With the Dropbox™ Cloud Delivery Network (CDN)* in the CC3x20 SimpleLink Wi-Fi and Internet of Things Over-the-Air Update application report.
3. Open the OtaJson.c file in the OTA library and increase the size of MAX_METADATA_FILENAME to 150 (this allows users to support a longer path name to the update file in their Dropbox account and can be adjusted as needed).
4. Open the OtaArchive.h file in the OTA library and increase the MAX_BUNDLE_CMD_FILES define to 12, so that all of the user files in the wifi_doorlock image can be included in the bundle.
5. Open the ota.h file in the OTA library and increase OTA_BLOCK_SIZE to 11000.
6. Rebuild the OTA library and ensure the new library is linked to the wifi_doorlock application in the workspace.
7. Rebuild the wifi_doorlock application.
8. Open the premade UniFlash ImageCreator Project for the wifi_doorlock application, then replace mcuimg.bin with the wifi_doorlock application binary built in Step 4.
9. Reprogram the LaunchPad with the new application containing the custom Dropbox credentials and directory information.
10. (Optional) Make a small change to the wifi_doorlock application, so to see that the update works correctly, then rebuild. For example, try incrementing the APPLICATION_VERSION define in wifi_doorlock_app.h.
11. (Optional) Replace the mcuimg.bin file in the UniFlash project with the binary created in Step 10.
12. Press the Generate Image button in UniFlash ImageCreator.
13. Select the Create OTA button on the Generate Image page.
14. When prompted for the OTA private key file name, select Browse and choose the dummy_ota_vendor_key.der file from the SDK (*<SDK Install Location>/<SDK Version>/tools/cc32xx_tools/ota-example-cert/dummy_ota_vendor_key.der*).
15. Press the Create OTA button in the prompt when it shows the correct private key.
16. Save the generated file and load it to the Dropbox account in the folder matching the OTA_VENDOR_DIR define of the otauser.h file (for example, Apps/OTA_R2/OTA_R2_MCU/).

After all of these steps are completed and the LaunchPad is flashed with the new software, the OTA can be run by following the same steps as when evaluating the application from the premade UniFlash ImageCreator project.

3.2 Testing and Results

The performance of the TIDC-01005 was evaluated based on a combination of pass or fail tests and power measurements on key system components. The following sections present a description of the tests and the results of the tests.

3.2.1 Pass or Fail Tests

Table 3-7 shows the set of pass or fail tests that were used to evaluate the performance of the TIDC-01005. All tests were performed using the software provided with the TIDC-01005 and the hardware setup described in Section 3.1.

Table 3-7. Pass or Fail Test Results

TEST DESCRIPTION	RESULT
Provision system to a new Wi-Fi network using AP provisioning	Pass
Provision system to a new Wi-Fi network using TI SmartConfig	Pass
Provision system to a new Wi-Fi network using BLE Provisioning	Pass
System automatically connects to a stored profile out of reset/hibernate if it exists	Pass
System connects to MQTT broker over TLS with server authentication	Pass
System receives lock and unlock commands from the cloud and drives the motor appropriately	Pass
System sends lock status to the cloud when polled by the user	Pass
System software successfully updates using a cloud-based OTA software update	Pass

3.2.2 Power Measurements

Power consumption is a key design consideration for electronic smart locks, which are often battery-powered. Multiple power measurements were performed on the TIDC-01005 system and can be used to estimate the battery life of an electronic smart lock system using the featured devices. Specifically, the power consumed by the CC3220S (MCU host and Wi-Fi network processor), CC2640R2F (Bluetooth low energy wireless MCU), and the DRV8837 (low-voltage, brushed, DC motor driver) were measured, because the wireless connectivity and motor subsystems typically have the largest impact on the average system power consumption. The power measurements were performed without the sensors included by disabling the sensor thread in the software and removing the BOOSTXL-SENSORS from the test setup.

For the CC3220S, the power consumption was measured for various activities that occur during normal system operation. The power consumed while the CC3220S was being provisioned was not measured, because provisioning is only required for first-time system setup and is likely to occur only a few times over the product lifetime. During AP provisioning, the power consumption of the CC3220S is higher than during normal system operation, in which the CC3220S is configured as a station. The power consumed by the CC3220S during BLE provisioning reflects the MCU active power. Similar to provisioning, the power consumed during an OTA software update was not measured, because OTA updates typically occur only a few times over a product lifetime and are not expected to have a significant impact on the average power consumption of the system.

When the system has been provisioned and successfully connects to a Wi-Fi network, normal operation begins. During normal operation, the system attempts to keep the CC3220S MCU and the network processor in a low-power mode called LPDS, except for when the following occurs:

- AP beacon reception or other broadcast occurs
- Keep-alive message is sent to the AP
- Keep-alive message is sent to the MQTT server
- Message is received from the server
- Message is sent to the server

The power consumption of the CC3220S was measured while the system was in an idle connected state (receiving AP beacons) and when the system received or sent messages to the MQTT broker. The power consumption of the CC2640R2F was measured during both BLE provisioning and while kept idle after provisioning. The power consumed by the DRV8837 was measured while the DRV8837 was driving a motor connected to a load and while held in sleep mode.

3.2.3 Test Setup

The power consumption of the CC3220S, CC2640R2F, and the DRV8837 were independently measured using a Keysight DC Power Analyzer (Keysight N6700 Power Modules). During testing, the DC Power Analyzer was used to power each device with a constant input voltage and measure the current consumption.

3.2.3.1 CC3220S

While measuring the power consumption of the CC3220S, the CC3220S and CC2640R2F LaunchPads were not stacked, to allow the devices to be powered separately. Instead, only the signals required for the NPI were connected between the boards using jumper wires. Resistor R141 on the CC3220S LaunchPad must also be removed. R141 is used to form a pull down on the input used by one of the CC3220S LaunchPad switches (SW2 on Rev. A). The resistor must be removed when measuring the power consumption on both the CC3220S and CC2640R2F LaunchPads, because the pin used by the switch is P4, which is used by the NPI and may increase power consumption while the NPI is idle (see Figure 3-12).

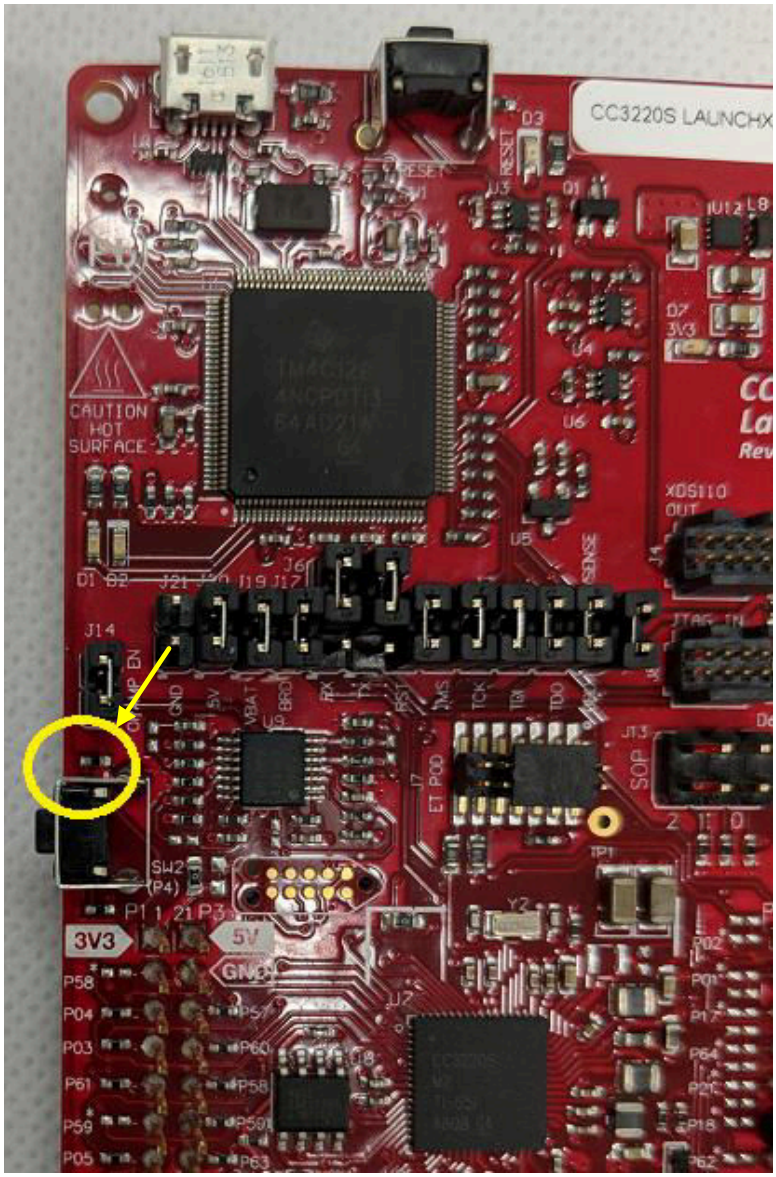


Figure 3-12. R141 Location on CC3220S LaunchPad™

While powering the CC3220S LaunchPad from the DC power analyzer, the LaunchPad was configured according to the *Battery Powering Only the CC3220 and U8 (Onboard Serial Flash)* section of the CC3220 SimpleLink™ Wi-Fi® LaunchPad™ Development Kit Hardware User's Guide, and run standalone (flashed instead of the debugger), to let the power management policy for the MCU host stay enabled.

The power analyzer was configured with an output of 3.3-V DC while powering the CC3220S LaunchPad. A Netgear AT&T hotspot was used as the AP during the tests, with no other stations connected. Three different modes of operation that occur during normal application operation were measured, as follows:

- TIDC-01005 idle connected to AP (not including AP keep alive messages)
- TIDC-01005 receiving lock status request and sending lock state to cloud
- TIDC-01005 receiving lock/unlock command from cloud and running a PWM output to control the motor driver

Note

When using the LaunchPad configuration from the *Battery Powering Only the CC3220 and U8 (Onboard Serial Flash)* section of the CC3220 SimpleLink™ Wi-Fi® LaunchPad™ Development Kit Hardware User's Guide, the back-channel UART is disconnected and debug prints are not available.

3.2.3.2 CC2640R2F

To accurately measure the power consumption of the CC2640R2F, all jumpers on the CC2640R2 LaunchPad must be removed. The Measuring Bluetooth Low Energy Power Consumption application report shows an example of the LaunchPad with all the jumpers removed. With all the jumpers removed, the CC2640R2F was powered by connecting the VDD output of the power analyzer to one of the 3V3 pins on the LaunchPad and the GND to any of the GND pins. A 3.3-V DC output was used to power the CC2640R2F LaunchPad while measuring the power consumption.

Note

When measuring power with the JTAG jumpers removed, programming and debug capabilities of the chip become unavailable.

3.2.3.3 DRV8837

To accurately measure the power consumption of the DRV8837 on the DRV8837EVM, several hardware modifications are required, in addition to the modifications described in Section 3.1.1. The complete list of hardware modifications required to accurately measure the power consumption of the DRV8837 follows (see Figure 3-13):

- Remove R3 and R4 (0-Ω resistors) from the back of the DRV8837EVM.
- Remove the LP2985 LDO voltage regulator (REG1) from the back of the DRV8837EVM.
- Remove the R1 pull down of the DIR pin on the front of the DRV8837EVM.
- Remove the MSP430G2131 MCU (U1) from the front of the DRV8837EVM.

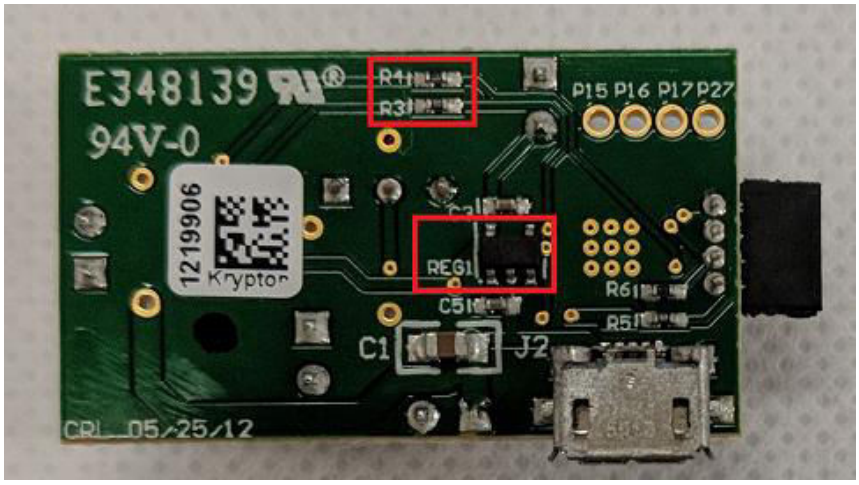


Figure 3-13. Back of DRV8837EVM With Components To Remove Highlighted

When the modifications are complete, the following connections must be made to the DRV8837EVM to perform the power measurement (see Figure 3-14).

- Connect the IN1 and IN2 test points to P16 and P17 of the CC3220 LaunchPad headers, respectively.
- Connect the top of JP2 (nSleep) to P18 of the CC3220 LaunchPad headers.
- Connect the bottom of JP2 (VCC of DRV8837) to one of the 3V3 pins on the CC3220 LaunchPad.
- Apply power across J1 (VM and GND).

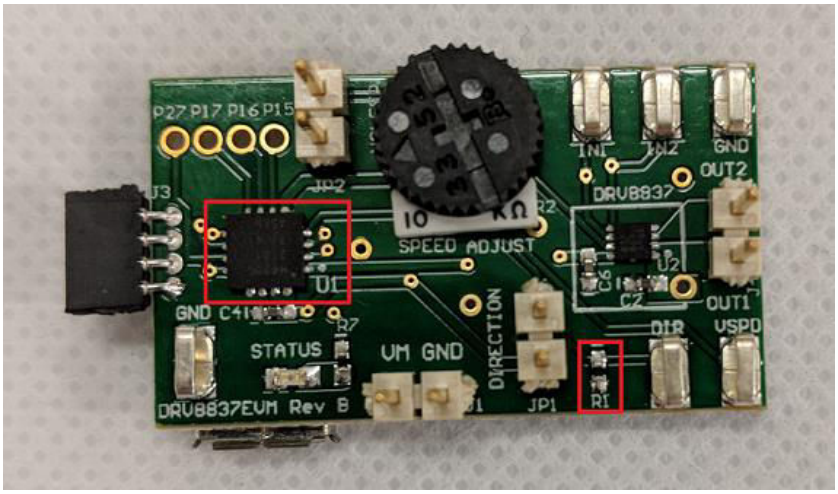


Figure 3-14. Front of DRV8837EVM With Highlighted Components To Remove

When testing the TIDC-01005, a 5-V input was used to power the DRV8837.

3.2.4 Test Results

The average current consumed by the CC3220, while the system was in the idle connected mode (connected to the AP and only receiving AP beacons or broadcast and multicast traffic) with an LSI of 400 milliseconds, was measured to be 377 µA on average, over multiple DTIM receptions. The power consumed while in the idle connected mode contributes the most to the overall average power consumption, because the system is expected to spend the majority of its active time in the idle connected state. Figure 3-15 shows a graph of the measured current.

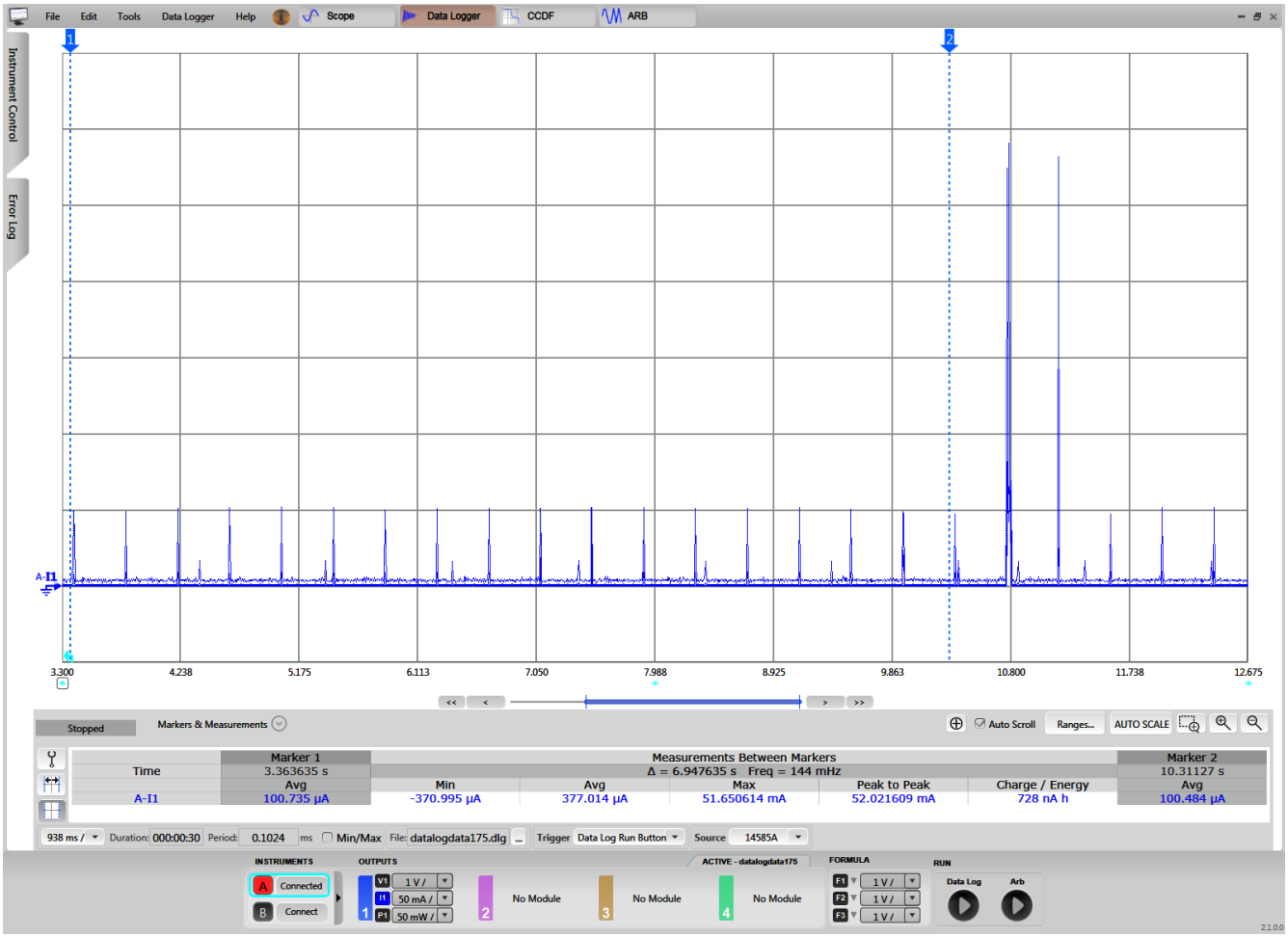


Figure 3-15. CC3220S Current Consumption Versus Time While in Idle Connected Mode With 400-ms LSI

When a message is sent to the TIDC-01005 from the cloud, the CC3220S receives an indication that the AP has buffered a packet for it through the DTIM. In response to the traffic indication, the CC3220S wakes up from low-power mode and polls the AP for the message. The impact on the system power consumption is different depending on which message is received. The first type of message considered is a lock or unlock command. When a lock or unlock command is received, the network processor wakes up, receives the message, and sends the command to the host MCU. The host MCU then parses the command and drives the motor in the appropriate direction for two seconds. The average current consumption of the CC3220S during a lock or unlock command was approximately 12.1 mA and Figure 3-16 shows the current consumption.

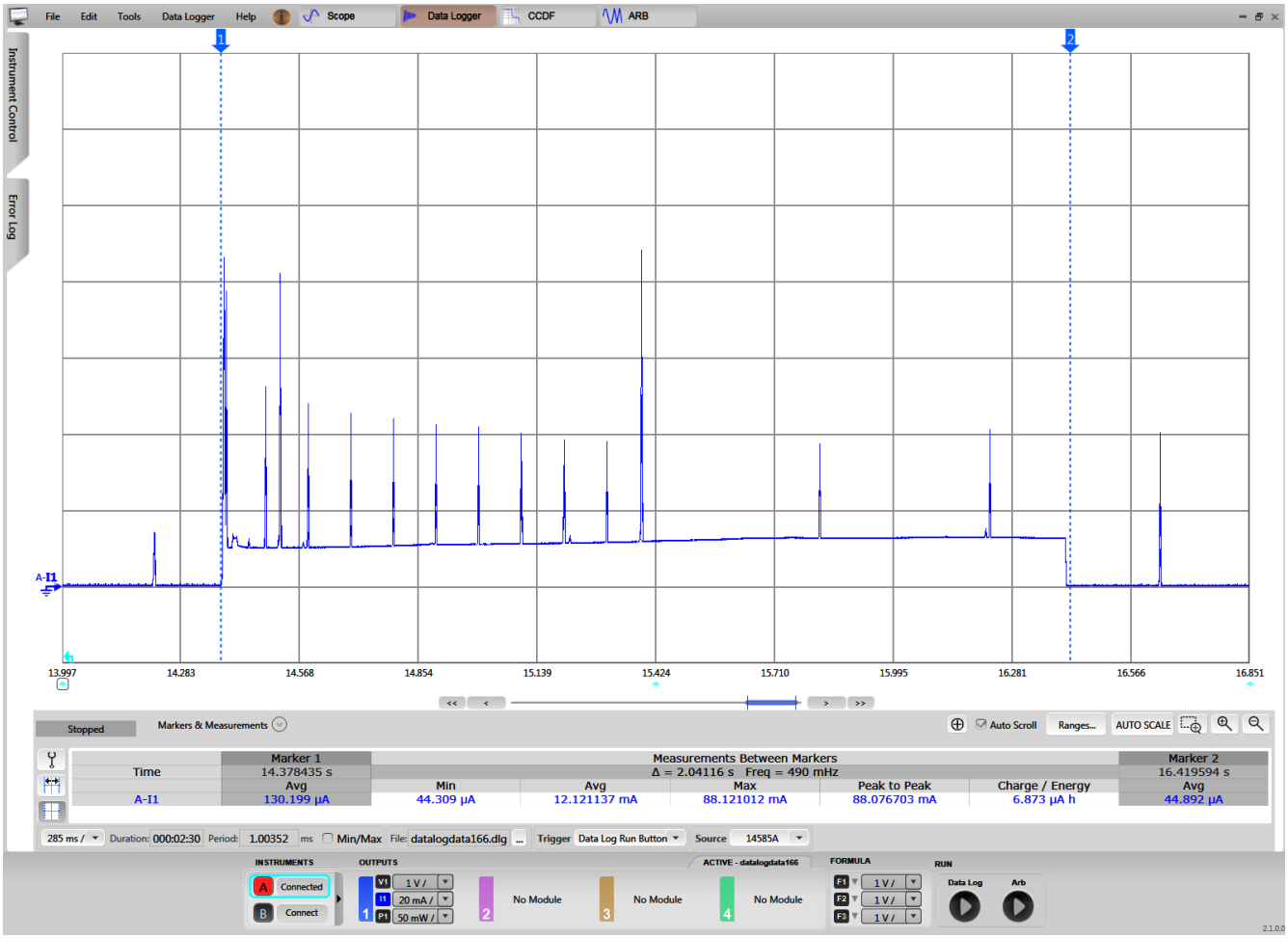


Figure 3-16. CC3220 Current Consumption During Lock and Unlock Cycle

In Figure 3-16, the first spike in current after the cursor represents when the network processor detects a message is ready for it and wakes up to poll the AP. After the message is received, the MCU is waken up by the network processor, and the message is sent to the MCU to be handled. The current consumption remains around the 11 to 12-mA level for the entire two seconds while the MCU is active and driving the motor. The smaller spikes in current that occur while the MCU is active and following the message reception represent AP beacon receptions. These beacon receptions occur because the CC3220S waits to resume LSI until a timeout occurs after a message reception. By not entering LSI immediately, the latency is reduced when multiple messages are sent to the system within a short interval without a large impact on power consumption.

The second type of message that can be received by the TIDC-01005 is a lock status request. When a lock status request is received, the network processor wakes up, receives the message, and sends the request to the host MCU. Then, the MCU parses the request and responds by sending a message indicating the lock state back to the cloud. The average current consumption of the CC3220S, in response to a lock status request, was found to be approximately 11.8 mA. Figure 3-17 shows a graph of the current consumption for the lock status case.

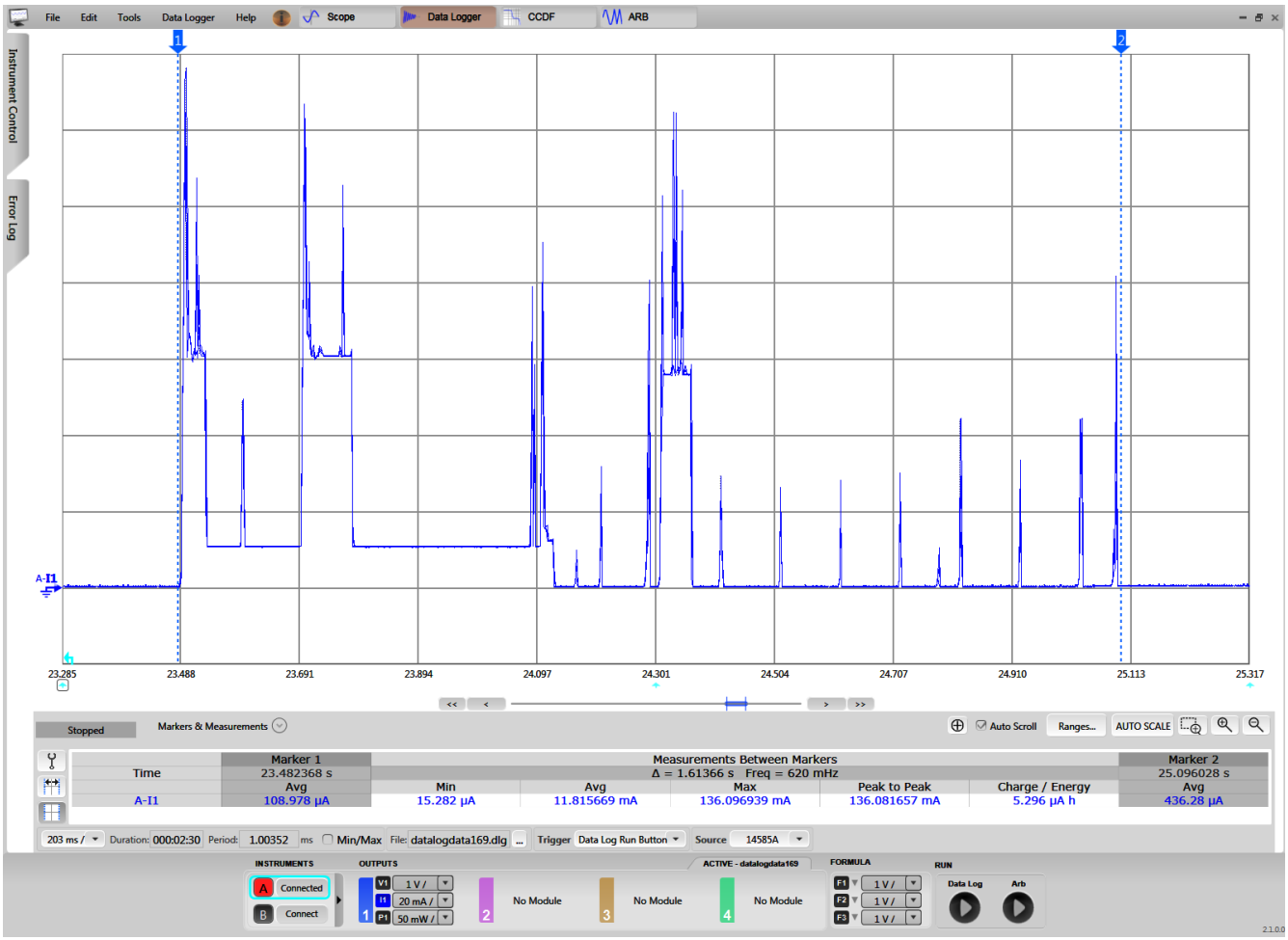


Figure 3-17. CC3220 Current Consumption During Lock Status Request

Similar to the lock or unlock command case, the first spike in current after the cursor represents when the network processor detects a message is ready for it and wakes up to poll the AP. When the message is received, it is passed to the MCU to be parsed. Users can see from Figure 3-17 that the current increases significantly in the middle of the cycle when the lock state is transmitted to the cloud (MQTT broker). After the message is received by the MQTT broker and acknowledged, the system enters the idle connected state. Users can see from Figure 3-17 that the system again waits for a timeout to occur before resuming the configured LSI.

The average current consumed by the CC2640R2F SNP throughout the provisioning process (while advertising, paired, and exchanging AP credentials) was approximately 211.8 µA. While idle (before and after provisioning), the average current consumed by the CC2640R2F was measured to be approximately 40 µA. Figure 3-18 and Figure 3-19 show the current consumption from both measurements.

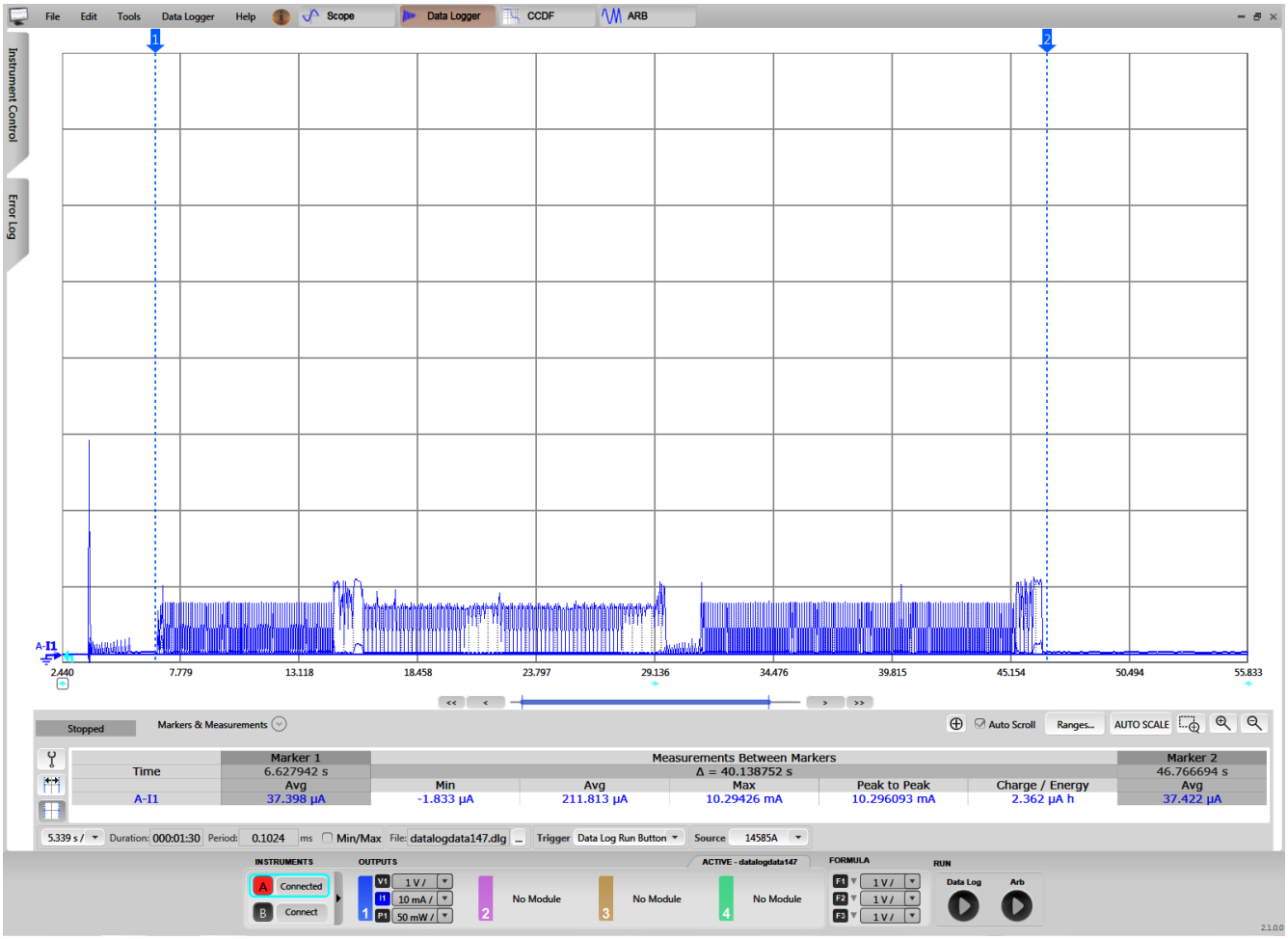


Figure 3-18. CC2640R2F Current Consumption During Provisioning

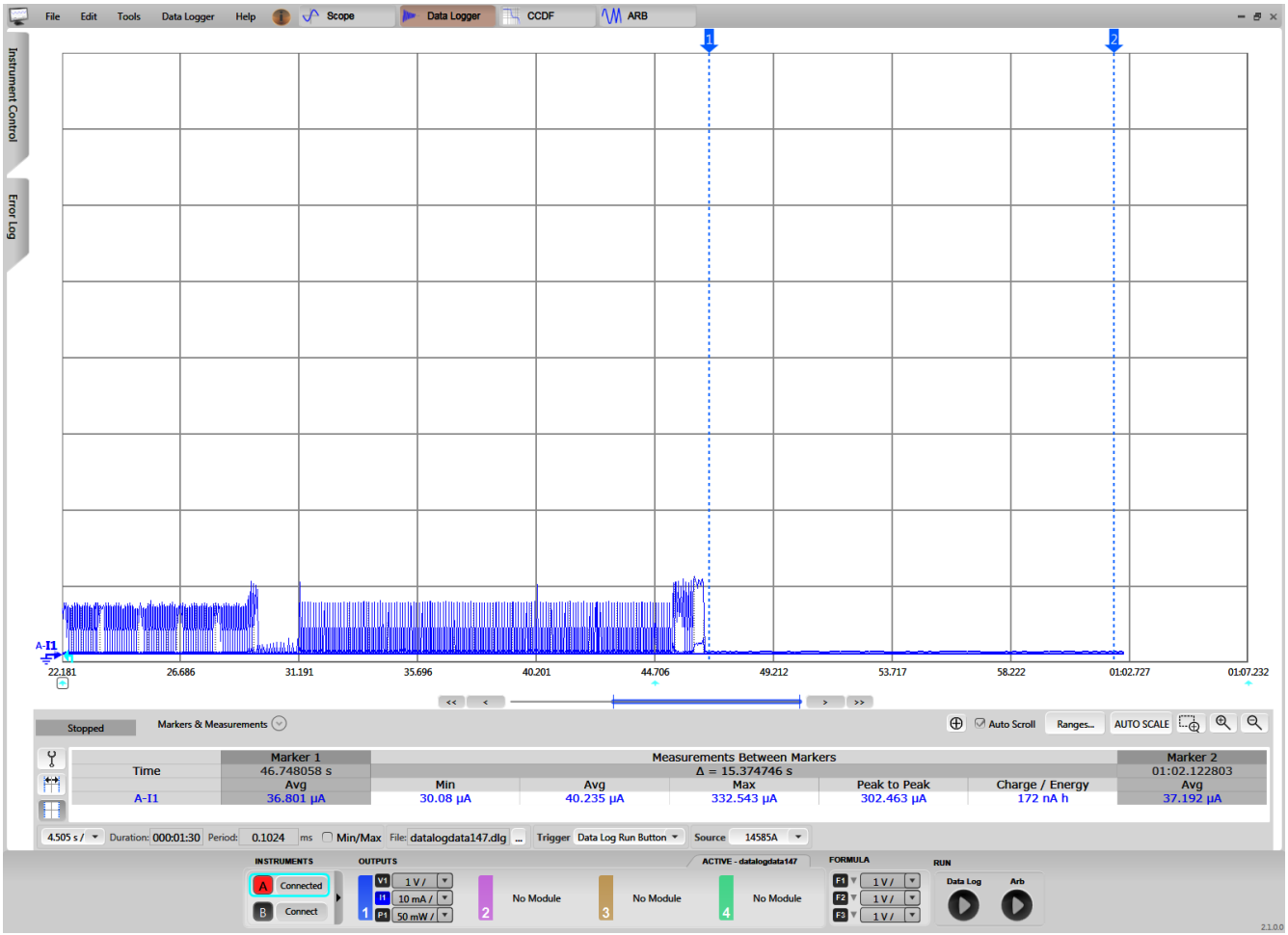


Figure 3-19. CC2640R2F Current Consumption While Idle

The power consumed by the DRV8837 during lock and unlock events was also measured. Specifically, the current consumed while supplying 5 V to the motor power supply of the DRV8337 and driving a load was measured. The load used was a deadbolt installed in a door. The average current consumption while locking and unlocking the deadbolt was approximately 77.6 mA and 73.1 mA, respectively, over the two-second interval that the motor was driven. Figure 3-20 and Figure 3-21 show the current consumption.

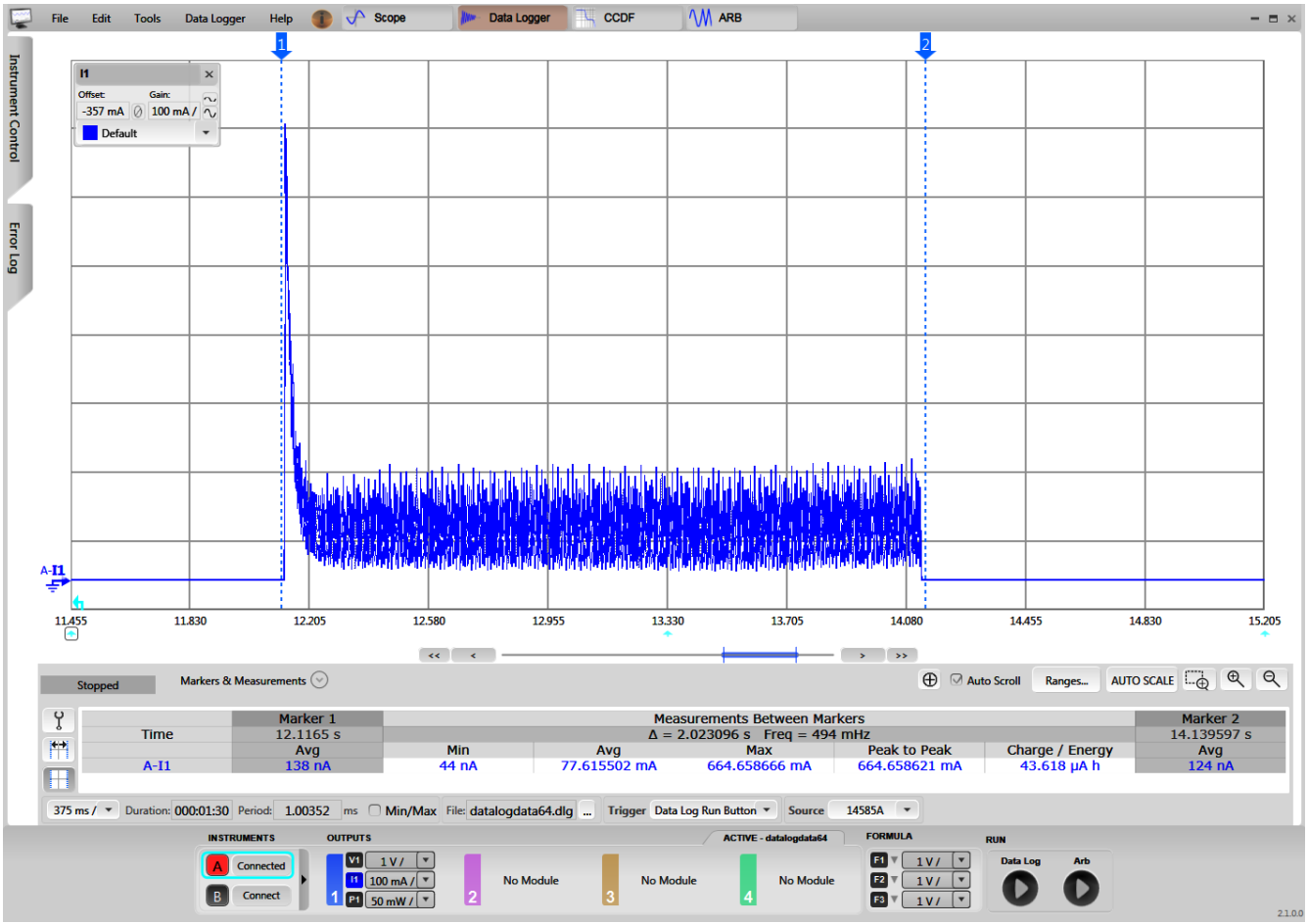


Figure 3-20. DRV8837 Current During Lock Event

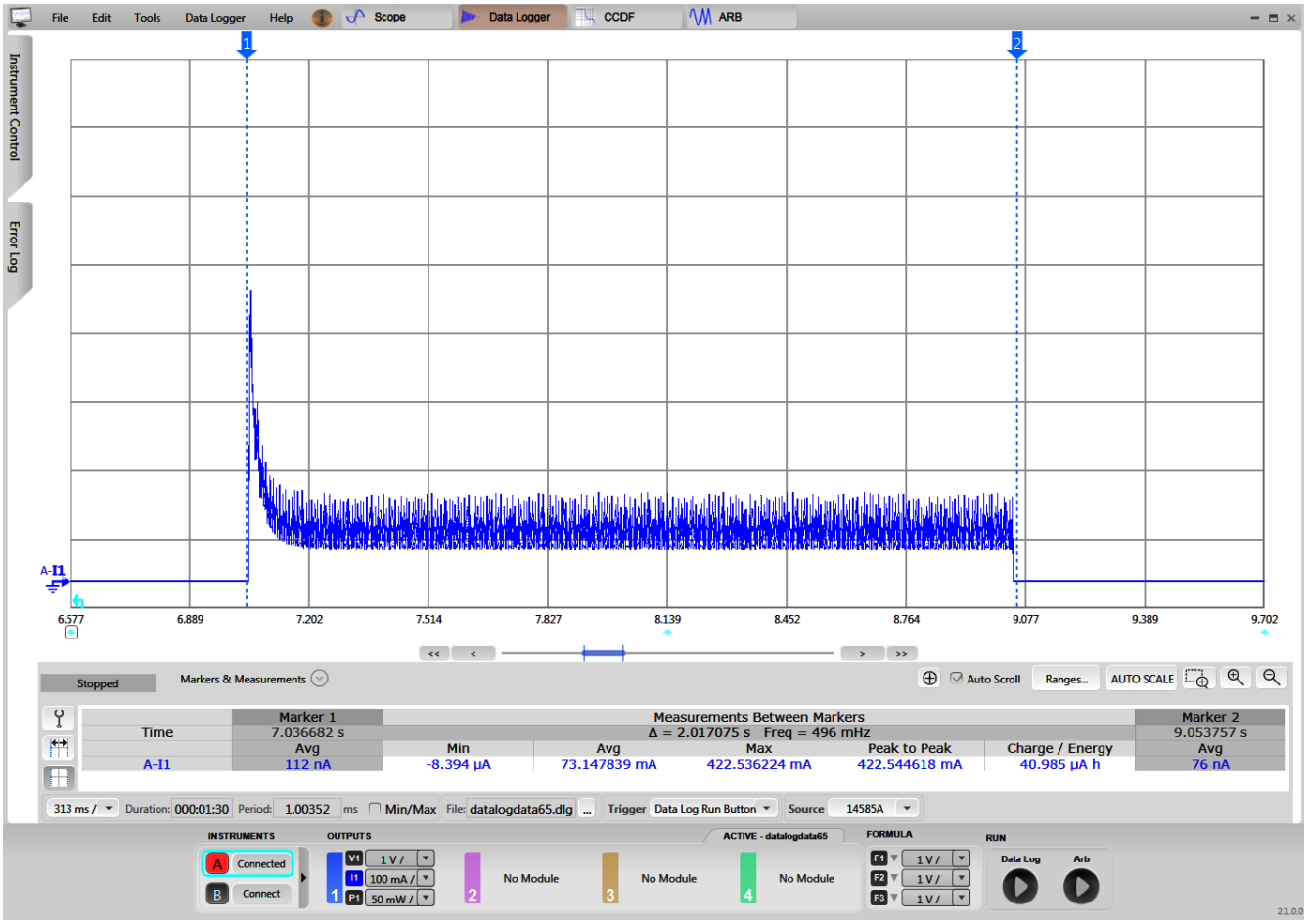


Figure 3-21. DRV8837 Current During Unlock Event

In between lock events, the DRV8837 is held in sleep mode by the CC3220S. The average sleep current of the motor while the DRV8837 was held in sleep was 0.099 µA and Figure 3-22 shows the current measurement.

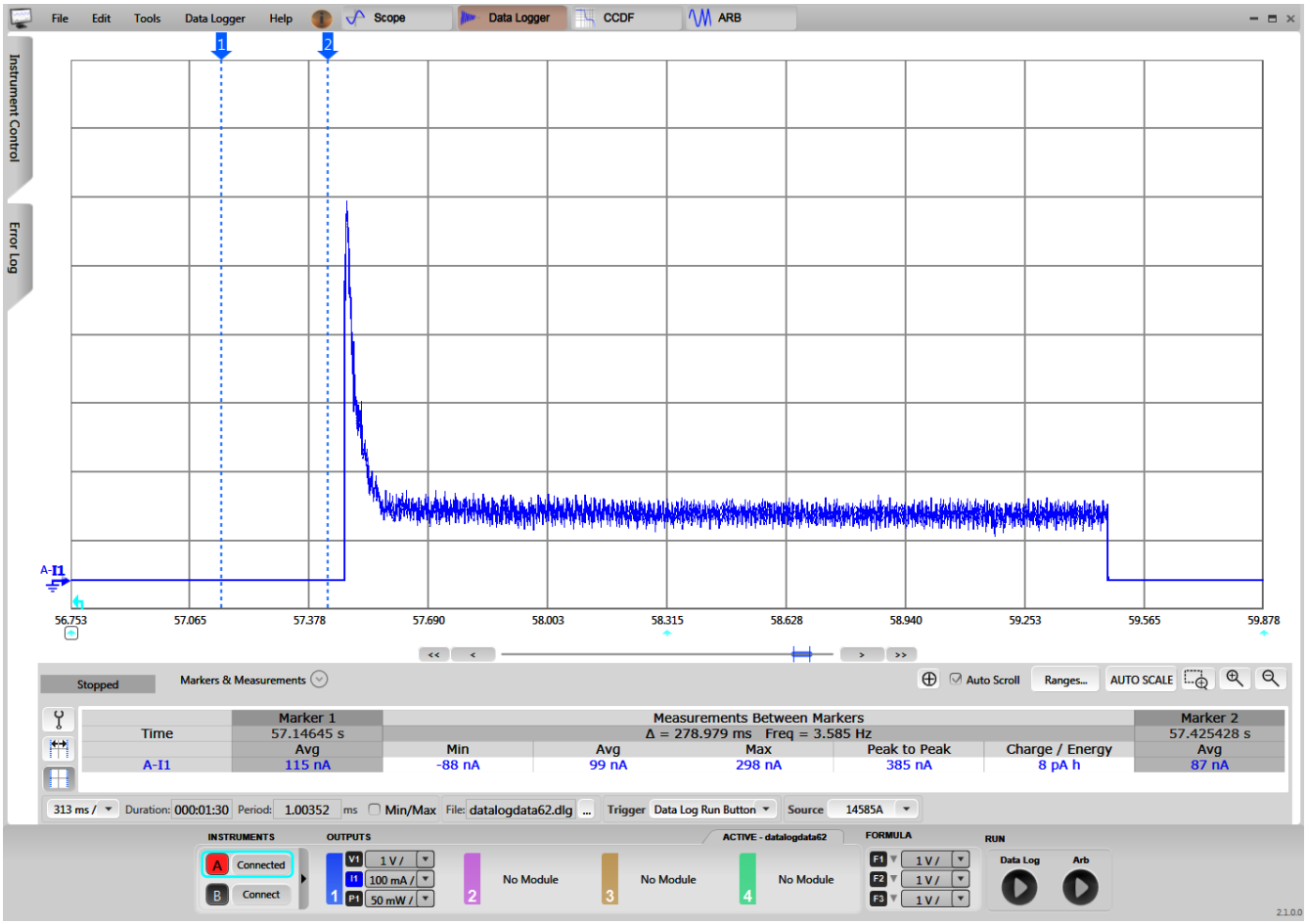


Figure 3-22. DRV8837 Sleep Current

Note

The previously listed results are based on EVMs, not an optimized hardware design. A system with an optimized hardware design may provide improved results.

3.2.5 Battery Life Estimate

The experimental results from Section 3.2.4 can be used to estimate the battery life of a system that is based on this design. Based on the results in Section 3.2.4 and using the assumption that the electronic smart lock is locked or unlocked 24 times per day, users can estimate the power consumed by each subsystem.

The main parameters of the Wi-Fi subsystem that affect the estimated battery life are as follows:

- CC3220 supply voltage: 3.3 V
- Idle connected current: 377 µA
- Average current during a lock or unlock event: 12.12 mA
- Duration of a lock event: 2 seconds
- Average current when reading lock status: 11.82 mA
- Duration of a lock status event: 1.6 seconds

Equation 1 gives the average Wi-Fi idle power.

$$\begin{aligned} P_{\text{wifi\_idle}} &= V_{\text{wifi}} \times I_{\text{avg\_wifi\_idle}} \\ &= 3.3V \times 377\ \mu A \\ &= 1,244\ \mu W \end{aligned} \tag{1}$$

Equation 2 gives the average Wi-Fi power per day, due to locking and unlocking.

$$\begin{aligned} P_{\text{wifi\_lock\_unlock}} &= V_{\text{wifi}} \times I_{\text{avg\_lock\_unlock}} \times \frac{\left(t_{\text{lock\_unlock}} \times num\ total\ events\right)}{86400\ sec} \\ &= 3.3\ V \times 12.12\ mA \times \left(\frac{2 \times 24}{86400}\right) \\ &= 22.22\ \mu W \end{aligned} \tag{2}$$

Equation 3 gives the average Wi-Fi power per day, due to the checking lock status.

$$\begin{aligned} P_{\text{wifi\_lock\_status}} &= V_{\text{wifi}} \times I_{\text{avg\_lock\_status}} \times \left(\frac{t_{\text{lock\_status}} \times num\ total\ events}{86400\ \text{sec}}\right) \\ &= 3.3\ V \times 11.82\ mA \times \left(\frac{1.6 \times 24}{86400}\right) \\ &= 17.32\ \mu W \end{aligned} \tag{3}$$

Equation 4 gives the approximate average Wi-Fi power.

$$\begin{aligned} P_{\text{wifi}} &\cong P_{\text{wifi\_idle}} + P_{\text{wifi\_lock\_unlock}} + P_{\text{wifi\_lock\_status}} \\ &= 1,283.54\ \mu W \end{aligned} \tag{4}$$

The main parameters of the BLE subsystem that affect the estimated battery life are as follows:

- CC2640R2F supply voltage: 3.3 V
- Average current of the CC2640R2F after provisioning: 40.23 µA

Equation 5 gives the approximate average BLE power.

$$\begin{aligned} P_{\text{ble}} &\cong P_{\text{ble\_idle}} \\ &= V_{\text{ble}} \times I_{\text{avg\_ble\_idle}} \\ &= 3.3\,V \times 40.23\ \mu A \\ &= 132.76\,\mu W \end{aligned} \tag{5}$$

The main parameters of the motor driver subsystem that affect the estimated battery life are as follows:

- DRV8837 motor supply voltage: 5 V
- Average DRV8837 sleep current: 0.099 µA
- Average current while locking deadbolt: 77.62 mA
- Average current while unlocking deadbolt: 73.15 mA
- Average time spent driving the motor: 2 seconds

Equation 6 gives the average motor driver idle power.

$$\begin{aligned} P_{\text{md\_idle}} &= V_{md} \times I_{\text{avg\_idle}} \\ &= 5\,V \times 0.099\ \mu A \\ &= 0.49\ \mu W \end{aligned} \tag{6}$$

Equation 7 gives the average motor driver power during a lock event.

$$\begin{aligned} P_{\text{md\_lock}} &= V_{\text{md}} \times I_{\text{avg\_md\_lock}} \times \left(\frac{t_{lock} \times num\,lock\ events}{86400\ \text{sec}} \right) \\ &= 5\,V \times 77.62\ mA \times \left(\frac{2 \times 12}{86400} \right) \\ &= 107.81\ \mu W \end{aligned} \tag{7}$$

Equation 8 gives the average motor driver power during an unlock event.

$$\begin{aligned} P_{\text{md\_unlock}} &= V_{md} \times I_{\text{avg\_md\_unlock}} \times \left(\frac{t_{unlock} \times num\,unlock\ events}{86400\ \text{sec}} \right) \\ &= 5\,V \times 73.15\ mA \times \left(\frac{2 \times 12}{86400} \right) \\ &= 101.60\ \mu W \end{aligned} \tag{8}$$

Equation 9 gives the approximate average motor driver power.

$$\begin{aligned} P_{\text{md}} &\cong P_{\text{md\_idle}} + P_{\text{md\_lock}} + P_{\text{md\_unlock}} \\ &= 0.49\,\mu W + 107.81\,\mu W + 101.60\,\mu W \\ &= 209.90\,\mu W \end{aligned} \tag{9}$$

Using the calculations for the power consumed by the Wi-Fi, BLE, and motor subsystems, users can estimate the total average system power (see Equation 10).

$$
\begin{aligned}
P_{\text{total}} &= P_{\text{wifi}} + P_{\text{ble}} + P_{\text{md}} \\
&= 1,283.54 + 132.76 + 209.90 \\
&= 1,626.2\ \mu W
\end{aligned}
\tag{10}
$$

The estimated total system power can be used to estimate the battery life of the system, using the theoretical energy capacity of the power source. For this estimate, assume an energy capacity of 18,000 mWh and calculate the months of battery life using Equation 11.

$$
\begin{aligned}
months &= \left(\frac{E_{total}\, mWh}{P_{total}\, mW}\right) \times \left(\frac{1\ day}{24\ hrs}\right) \times \left(\frac{1\ month}{30.5\ days}\right) \\
&= \left(\frac{18,000\ mWh}{1.626\ mW}\right) \times \left(\frac{1\ day}{24\ hrs}\right) \times \left(\frac{1\ month}{30.5\ days}\right) \\
&= 15.1\ months\ of\ battery\ life\ \left(1.26\ years\right)
\end{aligned}
\tag{11}
$$

4 Design Files

All the design files for the hardware used in the TIDC-01005 are at the respective EVM product pages.

For the CC3220S LaunchPad design files, see: CC3220S-LAUNCHXL.

For the CC2640R2F LaunchPad design files, see: CC2640R2F-LAUNCHXL.

For the DRV8837 design files, see: DRV8837EVM.

For the Sensor BoosterPack Plug-In Module design files, see: BOOSTXL-SENSORS.

5 Software Files

To download the software files, see the design files on the TIDC-01005 product page.

6 Related Documentation

1. Texas Instruments, *CC3220 SimpleLink Wi-Fi Wireless and Internet-of-Things Solution, a Single-Chip Wireless MCU*, data sheet
2. Texas Instruments, *CC3220 SimpleLink™ Wi-Fi® LaunchPad™ Development Kit Hardware User's Guide*
3. Texas Instruments, *SimpleLink™ CC3120, CC3220 Wi-Fi® Internet-on-a-chip™ Solution Built-In Security Features*, application report
4. Texas Instruments, *CC3120, CC3220 SimpleLink™ Wi-Fi® and Internet of Things Network Processor*, user's guide
5. Texas Instruments, *CC2640R2F SimpleLink™ Bluetooth® low energy Wireless MCU*, data sheet
6. Texas Instruments, *CC2640R2 LaunchPad Quick Start Guide*, user's guide
7. Texas Instruments, *Measuring Bluetooth Low Energy Power Consumption*, application report
8. Texas Instruments, *DRV883x Low-Voltage H-Bridge Driver*, data sheet
9. Texas Instruments, *DRV8837EVM User's Guide*
10. Texas Instruments, *BOOSTXL-SENSORS BoosterPack Plug-in Module Quick Start Guide*
11. Texas Instruments, *BOOSTXL-SENSORS Sensors BoosterPack Plug-in Module*, user's guide

6.1 Trademarks

7 Terminology

Security Enabler	An embedded SoC feature that can help customers to achieve their security objectives. Usually, a hardware feature, such as silicon or software in ROM form.
Certificates	Certificates are standard-formatted files. They typically contain the public key of the subject and a CA signature of the header and public key.
Keys	Keys are used for data encryption, key establishment, and digital signatures. The key lengths and types depend on the algorithm used, their purpose, and the security level.
OTA software update	An over-the-air software update that is transmitted wirelessly to a system.
Simple Network Processor (SNP)	A use-case of the CC2640 device, in which it acts a network processor and abstracts most of the BLE protocol away from an external MCU host.
Simple Application Processor (SAP)	An external MCU host (or library) that is used to control a SNP over a serial communications interface.
Network Processor Interface (NPI)	The protocol used to communicate between the SAP and SNP.

8 About the Author

BENJAMIN MOORE is an applications engineer at Texas Instruments, on the for SimpleLink Wi-Fi Applications team. Benjamin has been working in the Embedded Processing business unit of TI since 2013. He previously worked as part of the C2000™ and MSP430 System Applications teams. Prior to joining TI, Benjamin received his Bachelor of Science in Electrical and Computer Engineering from Ohio State University in Columbus, OH.

9 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision * (February 2018) to Revision A (September 2020)	Page
• Added WPA2 + PMF and WPA3 in Section 2.3.1, *CC3220*	5

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265