Design Guide: TIDA-010003

Simple 6LoWPAN Mesh End-Node Improves Network Performance Reference Design

Description

This reference design implements a simple RF mesh network end node for smart meter Advanced Metering Infrastructure (AMI) networks. The network is an IPv6 over low-power wireless personal area networks (6LoWPAN) solution. The design implements this network in a single CC1310 SimpleLink™ Wireless MCU to optimize performance and minimize system cost. The complete 6LoWPAN mesh network runs on top of the TI-15.4 stack based on IEEE 802.15.4e/g protocols and implements an un-slotted channel hopping (USCH) mode to provide protection against network interference.

Resources

TIDA-010003	Design Folder
CC1310	Product Folder
TPD4E004	Product Folder
TPD6E004	Product Folder
TM4C1294NCPDT	Product Folder
LM4040	Product Folder
TPS796	Product Folder
SN74AVC4T245	Product Folder

TI E2E™ Community

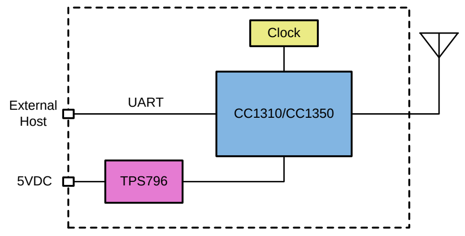
ASK Our E2E™ Experts

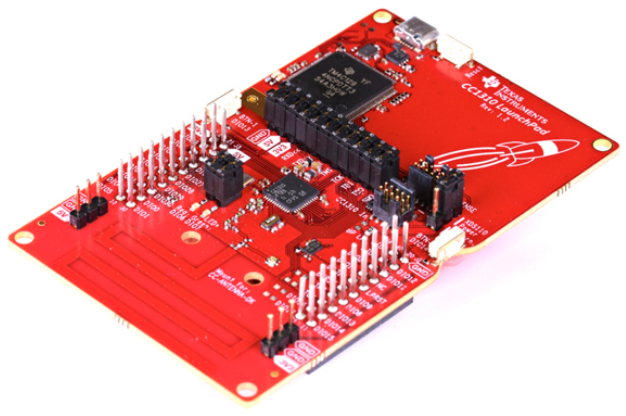
Features

- IPv6 networking over low-power RF in the < 1-GHz ISM bands
- Implements mesh network protocols of 6LoWPAN, RPL, IPv6, ICMPv6 and UDP
- Implements TI-15.4 stack with IEEE 802.15.4e based frequency hopping and MAC data encryption
- Software layer architecture identical to Wi-SUN FAN v1.0
- Fully compatible with the TIDA-01547, TIDA-010024 and TIDA-010032 reference designs to provide a complete network solution
- Supports low-power mode configuration for the battery-powered end nodes

Applications

- Wireless communications
- Electricity meter
- Water meter
- Gas meter





1 System Description

This reference design provides a simple low-power end node that supports 6LoWPAN mesh protocols. A primary design goal is to improve AMI network performance by using frequency hopping (FH) techniques that increase robustness in noisy radio frequency (RF) environments.

FH is a technique of transmitting data by switching one of many channels, where the channel is selected by a pseudo-random sequence known to both sender and receiver. This technique is known as robust versus interference and excellent in coexistence performance. As the communication systems run in sub-1 GHz ISM bands and multiple wireless systems run together, it is important that the system performance is not affected by interference or the coexisting systems. Section 3.2 shows experimental results to verify this by measuring network performance in the existence of out-of-network interference. In addition, Table 1 summarizes key system performance to show network robustness with background noise interference.

Another segment of this design is the 6LoWPAN mesh stacks, which improves network coverage and supports IPv6-based applications. The increased network coverage reduces the total system cost by reducing the number of data collectors that are typically more expensive than smart meters. The smart meters are static in the AMI networks. The mesh networking addresses the connectivity issue through multi-hop transmissions when data collector and smart meters are not reachable with each other.

This design is based upon a SimpleLink MCU of the CC13x0 wireless MCU. Figure 1 shows the overall system architecture. The TI-15.4 Medium Access Control (MAC) supports FH mode. The TI SimpleLink PHY supports IEEE 802.15.4g for 50-kbps, 200-kbps frequency-shift keying (FSK) modes and 5-kbps long range mode. The sub-1 GHz RF on the CC1310 MCU can support three frequency bands: 902 MHz, 863 MHz, and 433 MHz.

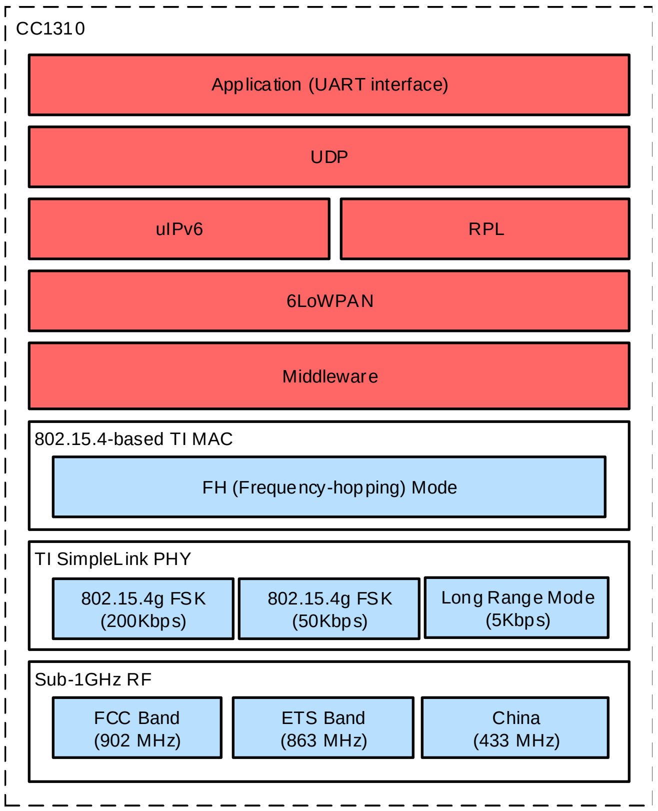


Figure 1. TIDA-010003 System Architecture

1.1 Key System Specifications

Table 1. Key System Specifications

PARAMETER	SPECIFICATIONS	DETAILS
Maximum number of hops	• 64 hops (in software, configurable) • Tested up to 6-hop networks	
Maximum number of route entries	• 10 (in software, configurable) • Tested up to 20-node networks	The overall network size will depend on network topology, not limited by the maximum number of entries.
Maximum number of neighbor nodes	• 10 (in software, configurable) • Tested up to 20-node networks	
Maximum application data size	• 100B (in software, configurable) • Tested up to 100B	
Delivery ratio	• 99.7% (average in 6-hop linear topology with continuous background noise)	Section 3.2.2.2
Round-trip time (RTT)	• 0.19 second (100B over 1-hop node) • 1.17 second (100B over 6-hop node)	Section 3.2.2.3
Goodput	• 8.3 kbps (100B in 1-hop node)	Section 3.2.2.4
MAC data encryption	• IEEE 802.15.4-based encryption supported • MIC-32, MIC-64, MIC-128 • ENC, ENC-MIC-32, ENC-MIC-64, and ENC-MIC-128	Section 3.1.2.2.1.3
Flash, RAM usage	• Flash: 120KB (128KB in total) • RAM: 22KB (20KB + 8KB (cache RAM) in total)	

2 System Overview

2.1 Block Diagram

Figure 2 shows the system block diagram. The CC1310 (or CC1350) MCU is the 6LoWPAN mesh MCU that runs UDP applications, 6LoWPAN mesh network and the TI 15.4-Stack over sub-1 GHz RF.

The external DC/DC converter, as shown in Figure 2, is needed when the external power source supplies the voltage level other than 3.3 V. In this reference design, because the evaluation modules (EVMs) are powered by USB, the TPS796 was chosen to convert 5 V to 3.3 V.

For end-equipment designs, the selection of the power supply depends on the input and output voltage level, and required current consumption. The TI WEBENCH Power Designer provides the detailed design of the power supply with the given input requirements.

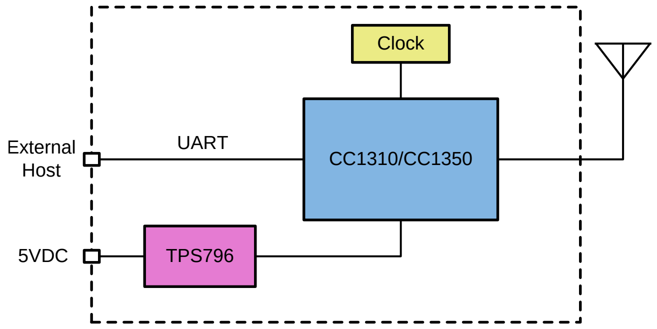


Figure 2. TIDA-010003 Block Diagram

2.2 Design Considerations

For this reference design, these devices perform the following:

- The CC1310 wireless MCU combines an ARM Cortex-M3 MCU with a flexible, ultra-low-power RF transceiver with excellent RX sensitivity to provide a robust link budget and execute the TI 15.4-Stack.
- The TPS796 low-power linear regulator offers high power-supply rejection ratio (PSRR), ultra-low noise, fast start-up, and excellent line and load transient responses.

2.3 Highlighted Products

2.3.1 CC1310

The CC1310 wireless MCU is a member of the Simplelink MCU platform. Its very low current consumption in both active and standby mode provides excellent lifetime when operating from batteries or super capacitors.

The CC1310 combines a flexible, very low-power RF transceiver with a powerful 48-MHz Cortex-M3 MCU in a platform supporting multiple physical layers and RF standards. A dedicated radio controller (Cortex-M0) handles low-level RF protocol commands that are stored in ROM or RAM, thus ensuring ultra-low power and flexibility. Its RF subsystem offers an excellent link budget with receiver sensitivity with –100 dBm at 50 kbps and output power up to +15 dBm. The CC1310 is a highly integrated, true single-chip solution incorporating a complete RF system and an on-chip DC/DC converter. The CC1310 wireless MCU is supported by the SimpleLink Software Development Kit (SDK) that offers 100% application code compatibility across the entire SimpleLink MCU portfolio and includes the integrated TI-RTOS, complete peripheral driver libraries with POSIX-compatible APIs, and encryption-enabled security features.

Figure 3 shows the CC1310 functional block diagram.

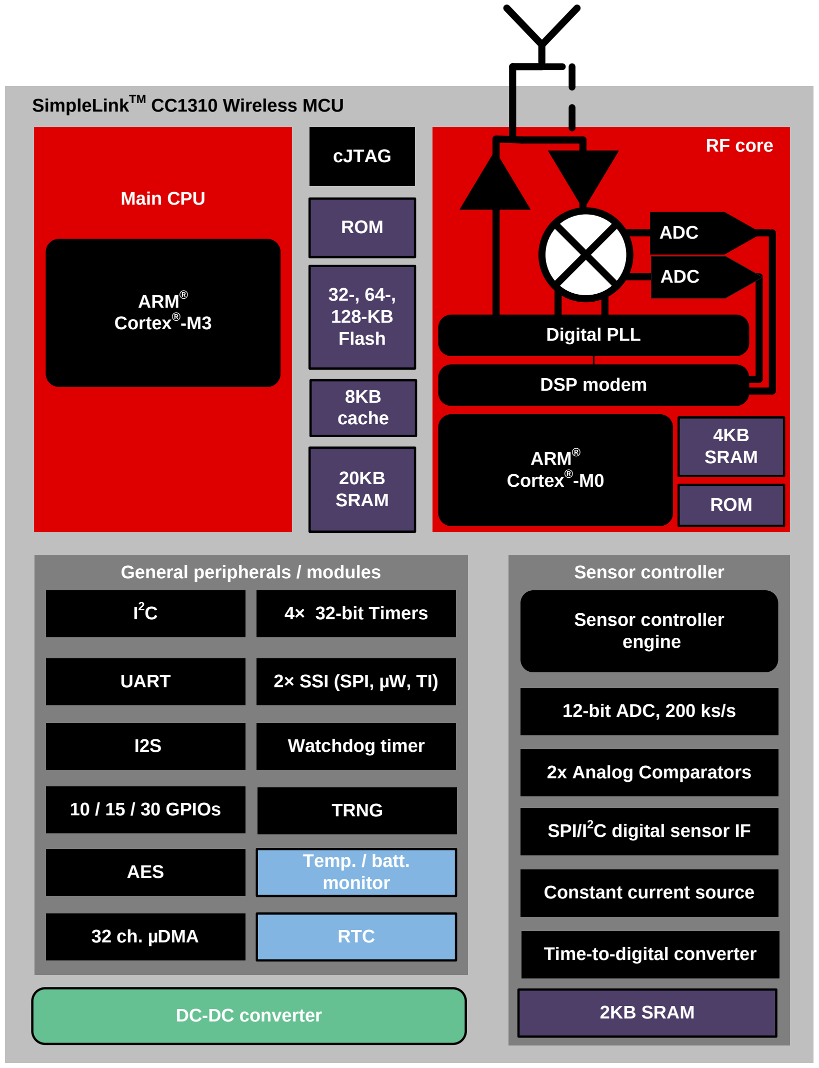


Figure 3. CC1310 Functional Block Diagram

2.3.2 TPS796

The TPS796 family of low-dropout (LDO) low-power linear voltage regulators feature high power-supply rejection ratio (PSRR), ultra-low noise, fast start-up, and excellent line and load transient responses in small outline, 3×3 VSON, SOT223-6, and TO-263 packages. Each device in the family is stable with a small, 1-µF ceramic capacitor on the output. The family uses an advanced, proprietary BiCMOS fabrication process to yield extremely LDO voltages (for example, 250 mV at 1 A).

Each device achieves fast start-up times (approximately 50 µs with a 0.001-µF bypass capacitor) while consuming very low quiescent current (265 µA, typical). Moreover, when the device is placed in standby mode, the supply current is reduced to less than 1 µA. The TPS79630 exhibits approximately 40 μV_{RMS} of output voltage noise at 3.0-V output with a 0.1-µF bypass capacitor. Applications with analog components that are noise sensitive, such as portable RF electronics, benefit from the high PSRR, low-noise features, and fast response time.

Figure 4 and Figure 5 show the TPS796 functional block diagrams.

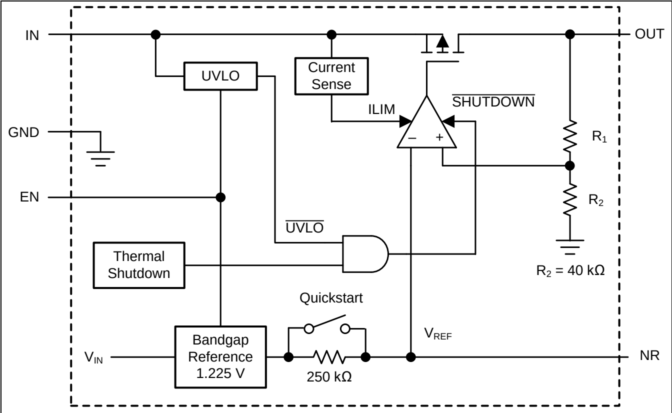


Figure 4. Functional Block Diagram (Fixed Version)

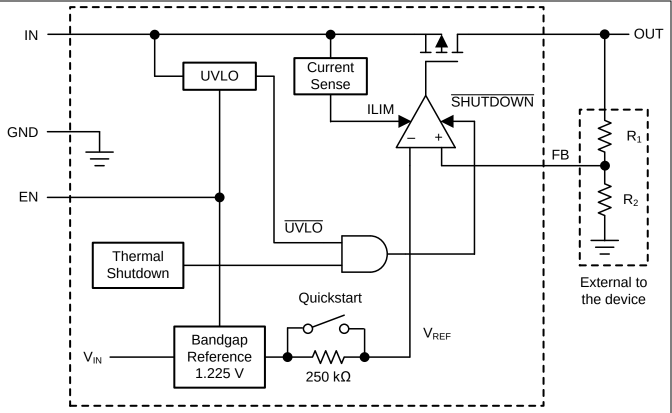


Figure 5. Functional Block Diagram (Adjustable Version)

3 Hardware, Software, Testing Requirements, and Test Results

3.1 Required Hardware and Software

3.1.1 Hardware

This reference design is built with a standard TI EVM of LAUNCHXL-CC1310, as shown in Figure 6. The CC1310 EVM can be replaced with the CC1350 EVM (LAUNCHXL-CC1350) with the same software example.

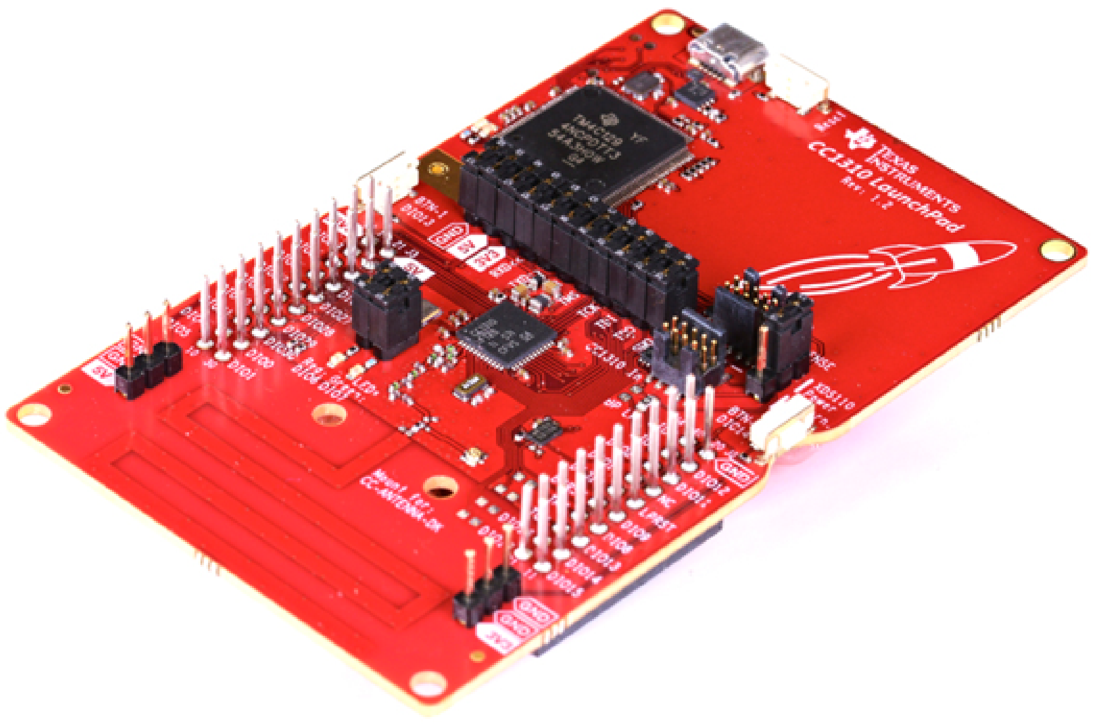


Figure 6. TIDA-010003 Hardware Platform

NOTE: If a software extension with more features is considered, it is strongly recommended to choose the CC13x0 with 7×7 package, which is pin-to-pin compatible to CC13x2 with more memory option.

3.1.2 Software

3.1.2.1 Getting Started

The 6LoWPAN mesh end-node software example was implemented based on the TI-15.4 sensor example from SimpleLink CC13x0 SDK v2.10.00.36. The software example provided with the reference TI design runs on the CC13x0 MCU to support 6LoWPAN, RPL routing, IPv6, ICMPv6, UDP, and simple applications. The pre-requisite tools to build the software example are Code Composer Studio™ v8.0 or above (CCSTUDIO) and SimpleLink CC13x0 SDK v2.40 (SIMPLELINK-CC13X0-SDK).

Figure 7 shows the software example that provides four end-node configuration options. They are UDP poll, UDP push, UDP poll for demo with TIDA-010032 and UDP push for leaf configurations. For the UDP poll example, end nodes send UDP data only when they receive the poll message from the root node. This example is popular in dense networks because this approach can control network traffic effectively regardless of the network size with the cost of polling overheads.

For the UDP push example, end nodes send UDP data whenever they have data to send. Compared to the UDP poll-based approach, this technique does not require the polling overhead while it increases collision probability among transmissions of the end nodes in dense networks.

The debug_poll_demo configuration is an end-node configuration for Internet of Things (IoT) demo with TIDA-010032 data concentrator reference design.

The debug_push_leaf configuration (debug_push_leaf) was designed to provide a low power operation mode for battery-operated devices. To achieve low power consumption, this mode disables routing capability, turns RX off when idle and runs on top of the TI 15.4-Stack sleep mode. An advantage is to extend end-nodes' coverage with the built-in mesh network while keeping power consumption low. A use case will be battery-powered flow meters or in-home display connected to an electricity meter acting as a router node.

Table 2 summarizes end-node configurations supported by this design.

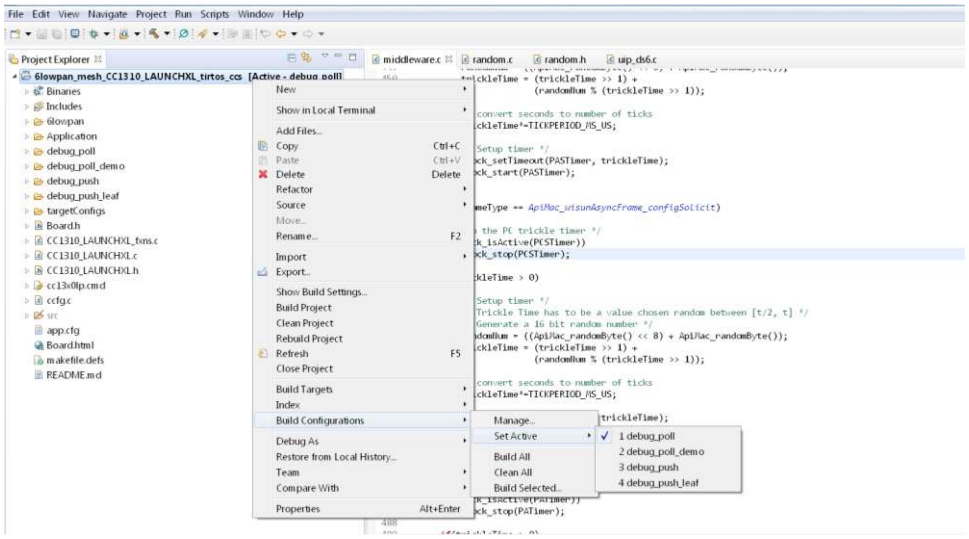
Table 2. End-Node Configuration

FEATURE	INTERMEDIATE	LEAF
Router Capability	Yes	No
RX ON when Idle	Yes	No
TI 15.4-Stack	FH non-sleep mode	FH sleep mode
Build Configuration	• debug_poll • debug_push • debug_poll_demo	• debug_push_leaf

Table 3 shows the summary of software example with CCS build configuration options. The TIDA-01547, TIDA-010024 and TIDA-010032 reference designs are companion TI designs to run as a root node working with the end-node software examples.

Table 3. CCS Build Configurations

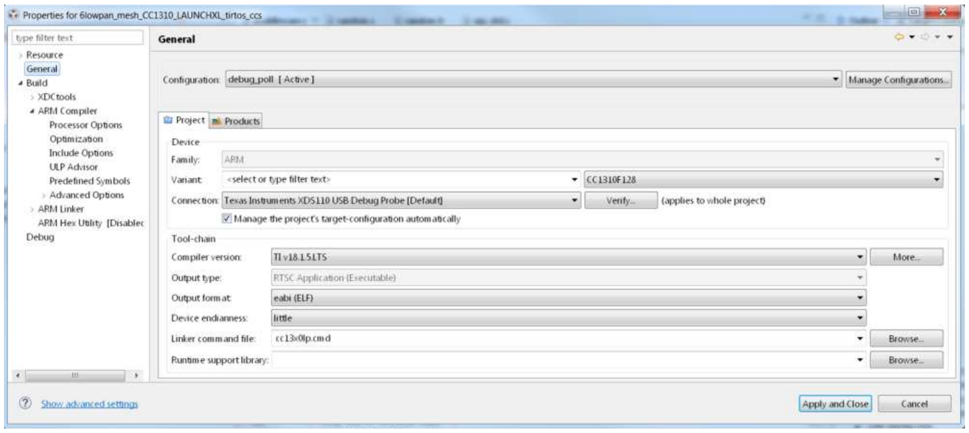
BUILD CONFIGURATION	EXAMPLE	NOTE
debug_poll	UDP Poll	
debug_push	UDP Push	
debug_poll_demo	UDP Poll	To demo with TIDA-010032 data concentrator
debug_push_leaf	UDP Push	Low-power mode operation



```
tickleTime = (trickleTime >> 1) +
             (randomNum % (trickleTime >> 1));
convert seconds to number of ticks
ckleTime*=TICKPERIOD_MS_US;
Setup timer */
ck_setTimeout(PASTimer, trickleTime);
ck_start(PASTimer);
meType == ApiMac_wisunAsyncFrame_configSolicit)
 the PC trickle timer */
k_isActive(PCSTimer))
ck_stop(PCSTimer);
kleTime > 0)
Setup timer */
Trickle Time has to be a value chosen random between [t/2, t] */
Generate a 16 bit random number */
domNum = ((ApiMac_randomByte() << 8) + ApiMac_randomByte());
ckleTime = (trickleTime >> 1) +
           (randomNum % (trickleTime >> 1));
convert seconds to number of ticks
ckleTime*=TICKPERIOD_MS_US;
trickleTime);
k_isActive(PATimer))
ck_stop(PATimer);
```

Figure 7. Code Composer Studio™ Build Configuration

Figure 8 and Figure 9 show Code Composer Studio screen captures to show the ARM compiler (TI v18.1.5.LTS), XDC tools (3.51.1.18_core), and CC13x0 SDK version (v2.40.0.20) for the software example.

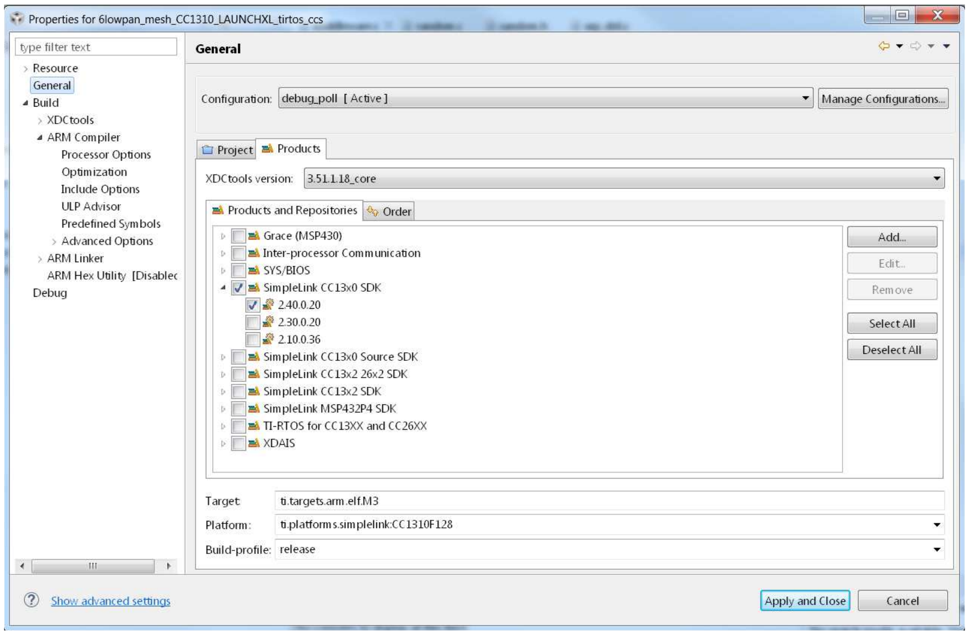


Configuration: debug_poll [Active]

Device
Family: ARM
Variant: <select or type filter text> CC1310F128
Connection: Texas Instruments XDS110 USB Debug Probe [Default] (applies to whole project)
[✓] Manage the project's target-configuration automatically

Tool-chain
Compiler version: TI v18.1.5.LTS
Output type: RTSC Application (Executable)
Output format: eabi (ELF)
Device endianness: little
Linker command file: cc13x0lp.cmd
Runtime support library:

Figure 8. Code Composer Studio™ Compiler Version



Properties for 6lowpan_mesh_CC1310_LAUNCHXL_tirtos_ccs

General

Configuration: debug_poll [Active]

XDCtools version: 3.51.1.18_core

- [ ] Grace (MSP430)
- [ ] Inter-processor Communication
- [ ] SYS/BIOS
- [✓] SimpleLink CC13x0 SDK
  - [✓] 2.40.0.20
  - [ ] 2.30.0.20
  - [ ] 2.10.0.36
- [ ] SimpleLink CC13x0 Source SDK
- [ ] SimpleLink CC13x2 26x2 SDK
- [ ] SimpleLink CC13x2 SDK
- [ ] SimpleLink MSP432P4 SDK
- [ ] TI-RTOS for CC13XX and CC26XX
- [ ] XDAIS

Target: ti.targets.arm.elf.M3

Platform: ti.platforms.simplelink:CC1310F128

Build-profile: release

Figure 9. Code Composer Studio™ RTSC Screen Capture

3.1.2.2 6LoWPAN_TI_15_4_Example

This section starts with software overview, followed by details of the software architecture, and useful tips for debugging and optimizing the software.

NOTE: This reference design provides an open-source based working example that can be a baseline software to develop end-products. The software example is not optimized in RAM or Flash usage and does not guarantee product-level quality.

3.1.2.2.1 Example Overview

This reference design implements a 6LoWPAN mesh network system working with the FH MAC over sub-1 GHz RF. The 6LoWPAN mesh network stacks run on TI-RTOS, which are implemented based on CONTIKI open source.

Figure 10 shows the end-node example software state machine. After power on, the end node starts with *Jdllc_deviceStates_init* state and, after completing the initialization, it goes to the *Jdllc_deviceStates_fh_sync* state where the node starts FH synchronization, which consists of two steps, discussed in Section 3.1.2.2.1.1. Once the node completes the first step of the FH synchronization, the state moves to *Jdllc_deviceStates_PA_done* state. After the end-node completes the FH synchronization, it moves to *Jdllc_deviceStates_joined* state that is a ready state to start RPL route discovery. The green LED (DIO7) on the EVM indicates that the node completes the RPL-level joining process. The state machine transition has been implemented in /Application/middleware.c.

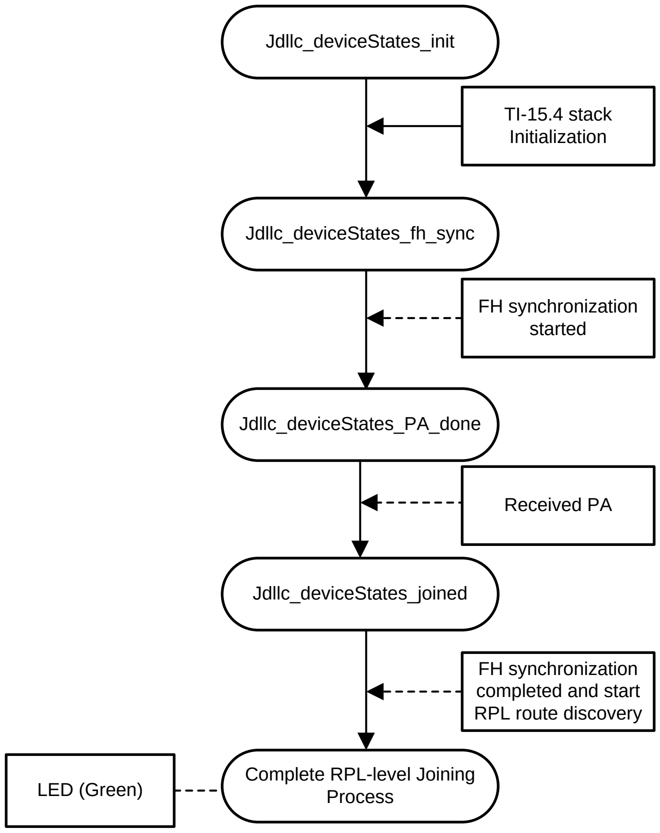

Jdllc_deviceStates_init
TI-15.4 stack Initialization
Jdllc_deviceStates_fh_sync
FH synchronization started
Jdllc_deviceStates_PA_done
Received PA
Jdllc_deviceStates_joined
FH synchronization completed and start RPL route discovery
LED (Green)
Complete RPL-level Joining Process

Figure 10. Software State Machine

3.1.2.2.1.1 FH Synchronization

FH synchronization is required to discover the FH network and to synchronize the FH timing and schedule. The underlying TI 15.4-Stack adopts WI-SUN FAN v1.0-based mechanism using four command frames:

- PAN advertisement (PA)
- PAN advertisement solicit (PAS)
- PAN configuration (PC)
- PAN configuration solicit (PCS)

FH synchronization starts with discovering neighbors, candidates of tracking parents, by performing active scan. The end nodes start with sending PAS commands at the time chosen by the trickle algorithm (RFC 6206). The PAS is sent over all the FH channels from the lowest channel number to the highest in sequence as the nodes do not know the FH timing and schedule at this time. As a response to the PAS, the PA is sent by the nodes that has already joined to the FH network. When the end nodes receive multiple PAs during the scan period (SCAN_TIMEOUT_VALUE), they will choose one of them based on the link-level metric, and then update unicast FH timing, schedule, and the PAN information with the tracking FH parent.

The next step is to send the PCS in the same way as the PAS. Once the nodes receive the PC as a response, they update the broadcast FH timing and schedule and the Group Transient Key (GTK) hash information. Receiving PC as a response of the PCS completes the FH synchronization process, which is ready to receive and send data at the network layers.

The FH timing and schedule correction is done with the received data packets that contain FH unicast and broadcast timing and schedule information elements (IEs).

NOTE: The FH synchronization mechanism implemented in this reference design is TI proprietary and is not WI-SUN FAN standard compliant.

Table 4 summarizes the trickle algorithm parameters used for FH synchronization. Depending on the network size, the parameters may need to be adjusted. These parameters are defined in /Application/middleware.h.

Table 4. Trickle Algorithm Parameters

PARAMETER	VALUE	DESCRIPTION
TRICKLE_TIMEOUT_VALUE	6 seconds	Discovery trickle timer minimum timeout for PAS and PCS
TRICKLE_MAX_BACKOFF	3	Discovery trickle timer backoff exponent for PAS and PCS
SCAN_TIMEOUT_VALUE	20 seconds	Discovery trickle timer timeout for PA and PC

3.1.2.2.1.2 Keep-Alive Mechanism

The goal of the keep-alive mechanism is to detect FH sync loss throughout monitoring sync error conditions such as data transmission and reception failures. The keep-alive mechanism broadcasts keep-alive frames (a 10B TI proprietary message defined in the example) to the link-level neighbors periodically (KEEP_ALIVE_TX_TIMEOUT_VALUE) once end nodes join to the FH network. The keep-alive TX timer is reset when broadcast frames other than keep-alive frames are sent, which reduces the keep-alive traffic overheads to the network.

If end nodes do not receive broadcast frames from their tracking parents in series (KEEP_ALIVE_MAX_ATTEMPTS), FH sync loss occurs and the end nodes move immediately to the Jdllc_deviceStates_fh_sync state to restart the FH synchronization process.

In addition to the keep-alive transmissions, each node traces unicast transmission failures to the target parents. If TX attempts fail in series (MAXIMUM_NUM_DATA_FAILURE), this indicates the FH sync loss, which results in moving to the Jdllc_deviceStates_fh_sync state to re-start the FH synchronization process. Table 5 summarizes the keep-alive parameters. The keep-alive parameters are defined in /Application/middleware.h.

Table 5. Keep-Alive Parameters

PARAMETER	VALUE	DESCRIPTION	COMMENTS
KEEP_ALIVE_TX_TIMEOUT_VALUE	60 seconds	Keep-alive TX interval for parents	UDP polling example only
KEEP_ALIVE_MAX_ATTEMPTS	5 times	The maximum number of failed keep-alive RX in series to indicate FH sync loss	UDP polling example only
MAXIMUM_NUM_DATA_FAILURE	5 packets	The maximum number of TX failure to indicate FH sync loss	UDP polling and UDP push examples

3.1.2.2.1.3 MAC Data Encryption

The MAC data encryption follows IEEE 802.15.4 standard with the security level options of MIC-32, MIC-64, MIC-128, ENC, ENCMIC- 32, ENC-MIC-64, and ENC-MIC-128.

In the software example, the default mode is set to the security level of *ApiMac_secLevel_encMic32* and the key ID mode of *ApiMac_keyIdMode_8*, defined in the /Application/jdllc.c. The TI-15.4 supports the pre-shared key mechanism based on IEEE 802.15.4 standard. The security key must be pre-programmed in the software, and all the nodes in the same network must share the same pre-shared key.

Because the MAC data encryption mechanism requires a node to register its neighbors that use the data encryption, it is required to have a discovery phase throughout unsecured data exchanges. The PAS and PA frames used for the FH synchronization must be sent without data encryption. In addition, the software example uses unsecured transmissions for PC, PCS, and broadcast frames to minimize a chance that some neighbors with a better route are not detected during the discovery phase.

3.1.2.2.1.4 RPL Routing

The 6LoWPAN mesh software example uses the RPL protocol for multi-hop routing. The network formation with the RPL routing is initiated by broadcasting Destination Oriented Directed Acyclic Graph (DODAG) information object (DIO) by the root node. Once child nodes receive the DIOs, they broadcast the DIOs and send back a unicast destination advertisement object (DAO) packet to the parents that provide the best route toward the root. The DIO and DAO transmission times are determined by the trickle algorithm (RFC 6206). For the RPL metric to decide the best route, the expected transmission count (ETX) is used by default. For details on RPL routing, see the RFC standard RFC 6550 or the TI training video on Wireless Network Challenges and Solutions for a Smarter Grid IoT.

3.1.2.2.1.5 6LoWPAN

The goal of the 6LoWPAN protocol is to support the IP services by reducing the gap between IPv6 and lower stacks to serve IPv6 applications on the low-end devices typically restricted in processing power, memory, and energy. The primary tasks of the 6LoWPAN are fragmentation and reassembly, IPv6 and UDP header compression, stateless IPv6 address auto-configuration, and neighbor discovery optimization. For details of the 6LoWPAN protocol, see the RFC standards RFC 4944 and RFC 6282 or the TI training video on Wireless Network Challenges and Solutions for a Smarter Grid IoT.

3.1.2.2.2 Software Architecture

The 6LoWPAN mesh end-node software consists of TI 15.4-Stack, middleware layer interconnecting between TI 15.4-Stack and upper layers, 6LoWPAN mesh stacks, and the application layer. The middleware layer initializes and configures the TI 15.4-Stack and processes incoming data from the TI-15.4 or mesh network stacks. The 6LoWPAN mesh stacks cover 6LoWPAN, RPL, IPv6, ICMPv6 and UDP protocols based on CONTIKI open source. The example software provides two types of UDP applications that can be configured at compile time: UDP poll and UDP push examples.

Figure 11 shows the overall software architecture. The software example was implemented based on the sensor_cc1310p example available in the SimpleLink CC13x0 SDK v2.10.00.36. The middleware layer runs the middleware_process that manages the FH synchronization and keep-alive mechanism, and interfaces between TI-15.4 and mesh network stacks to handle incoming data. The middleware layer follows the same software architecture via iCall service defined in the sensor_cc1310p example. The tcpip_task covers the mesh network layers of 6LoWPAN, uIPv6, RPL and UDP layers. The UDP poll and push examples are parts of the app_task as the application layer.

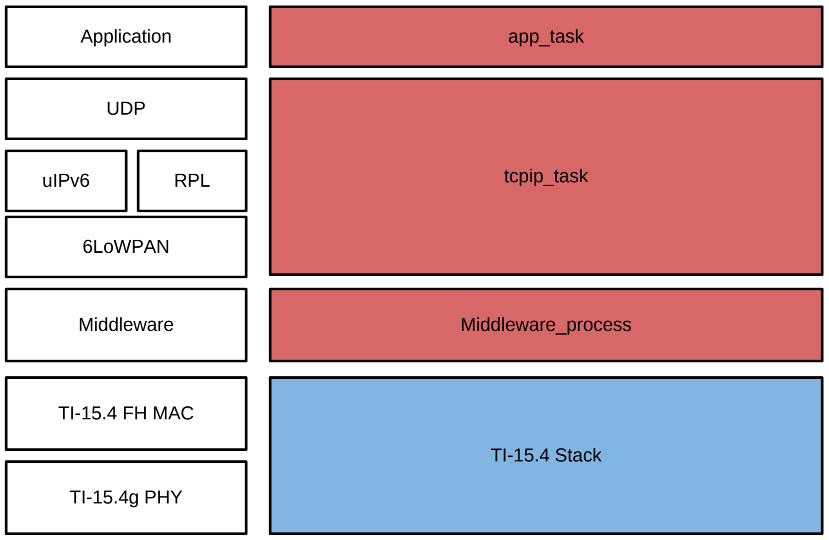


Figure 11. TIDA-010003 Software Architecture

3.1.2.2.2.1 TI-15.4 PHY Configuration

The TI 15.4-Stack supports multiple options for the frequency band and mode to run the FH. The software example configures the PHY mode to APIMAC_STD_US_915_PHY_1. The default configuration can be updated in the /Application/subg/config.h. The following codes list all the options for the PHY ID, which can be found in /Application/api_mac.h.

NOTE: The TI design software example was verified with the FH operation over the PHY mode of APIMAC_STD_US_915_PHY_1.

```
 /*! PHY IDs - 915MHz US Frequency band operating mode # 1 */
#define APIMAC_STD_US_915_PHY_1                 1
/*! 863MHz ETSI Frequency band operating mode #1 */
#define APIMAC_STD_ETSI_863_PHY_3               3
/*! 433MHz China Frequency band operating mode #1 */
#define APIMAC_GENERIC_CHINA_433_PHY_128        128
/*! PHY IDs - 915MHz LRM US Frequency band operating mode # 1 */
#define APIMAC_GENERIC_US_LRM_915_PHY_129       129
/*! 433MHz China LRM Frequency band operating mode #1 */
#define APIMAC_GENERIC_CHINA_LRM_433_PHY_130    130
/*! 863MHz ETSI LRM Frequency band operating mode #1 */
#define APIMAC_GENERIC_ETSI_LRM_863_PHY_131     131
/*! PHY IDs - 915MHz US Frequency band operating mode # 3 */
#define APIMAC_GENERIC_US_915_PHY_132            132
/*! 863MHz ETSI Frequency band operating mode #2 */
#define APIMAC_GENERIC_ETSI_863_PHY_133          133
```

3.1.2.2.2.2 Middleware Layer

The Middleware layer initializes the TI 15.4-Stack, maintains MAC-level keep-alive mechanism, registers security entries to the TI 15.4-Stack, processes packets incoming from the mesh network or the TI 15.4-Stack, handles message timeout and erroneous transmissions, performs FH synchronization, and maintains the state machine which can be one of four states: *Jdllc_deviceStates_init*, *Jdllc_deviceStates_fh_sync*, *Jdllc_deviceStates_PA_done*, and *Jdllc_deviceStates_joined*.

Anend-node starts with *Jdllc_deviceStates_init* state to reset the TI 15.4-Stack, to configure initial MAC PIB and FH PIB values, and to start the PAN as end-node. The state changes to *Jdllc_deviceStates_fh_sync* state when the node initiates the FH synchronization process. As the first step of the FH synchronization, if the node receives PA as a response of the PAS command, it moves to *Jdllc_deviceStates_PA_done* state and proceeds to send the PCS command. After the FH synchronization completes by receiving PC commands, the node moves to *Jdllc_deviceStates_joined* state, which is the ready state to send data at the mesh network layers.

Similar to the sensor_cc1310p example available in the SimpleLink CC13x0 SDK, the communications between TI 15.4-Stack and the middleware is done via iCall messages. Conversely, the message exchanges between the mesh network stacks and the middleware layer are done via mailbox posting or direct API calls. For the details on the iCall framework, refer to ti-15.4-stack-users-guide in docs/ti154stack under the CC13x0 SimpleLink SDK installation directory.

3.1.2.2.2.3 Network Layers (6LoWPAN, IPv6, RPL, and UDP)

The network stacks are implemented based on the CONTIKI open source. The tcpip_task (in 6lowpan/uip_rpl_task.c) processes messages incoming from the application and lower layers through mailbox. For details of the implementations, refer to the CONTIKI open source website.

3.1.2.2.2.4 Application Layer

The example includes two types of UDP applications: UDP poll and UDP push. The app_task opens the UDP socket (UDP client for the end-node) with known port numbers and starts UDP data transmissions. Depending on the UDP examples, the initiator of the UDP data is different. For the UDP poll example, the root node initiates the poll message to each node to read data. For the UDP push example, each node initiates data transmissions whenever there is application data to send.

The Device Language Message Specification (DLMS) and Companion Specification for Energy Metering (COSEM) for smart metering applications use the poll-based mechanism, and the Constraint Application Protocol (CoAP) uses the mix of UDP poll (GET command) and push (periodic OBSERVE command) mechanisms. The target end-product applications can be easily integrated on top of the given UDP application.

3.1.2.2.2.5 LED Configuration

Table 6 summarizes the LED configuration in the software example. The board_led_type_LED1 toggles when transmission or reception events happen. The board_led_type_LED2 turns on when end nodes join to the RPL network.

Table 6. LED Configuration (CC1310)

NAME (PIN NUMBER)	EVENT	ACTION
board_led_type_LED1 (DIO6)	Data TX/RX	TOGGLE (Red)
board_led_type_LED2 (DIO7)	Complete RPL-level Joining	ON (Green)

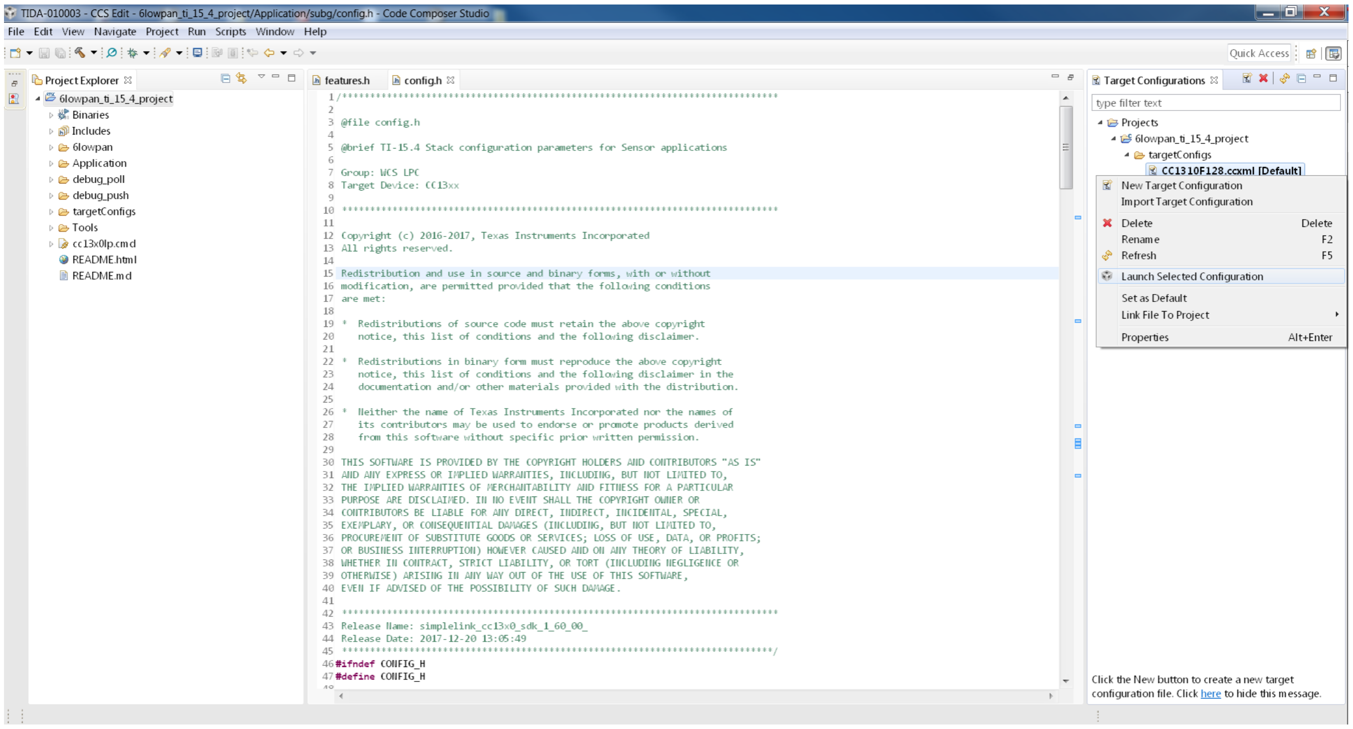
3.1.2.2.3 Tips for Debugging and Optimization

This section provides useful tips with the Code Composer Studio tool to debug and optimize the software example in the end-product development phase.

3.1.2.2.3.1 Running in Debug Mode

To debug software, run the software in debug mode. The Code Composer Studio tool provides the debug mode operation. To run in debug mode with the Code Composer Studio tool:

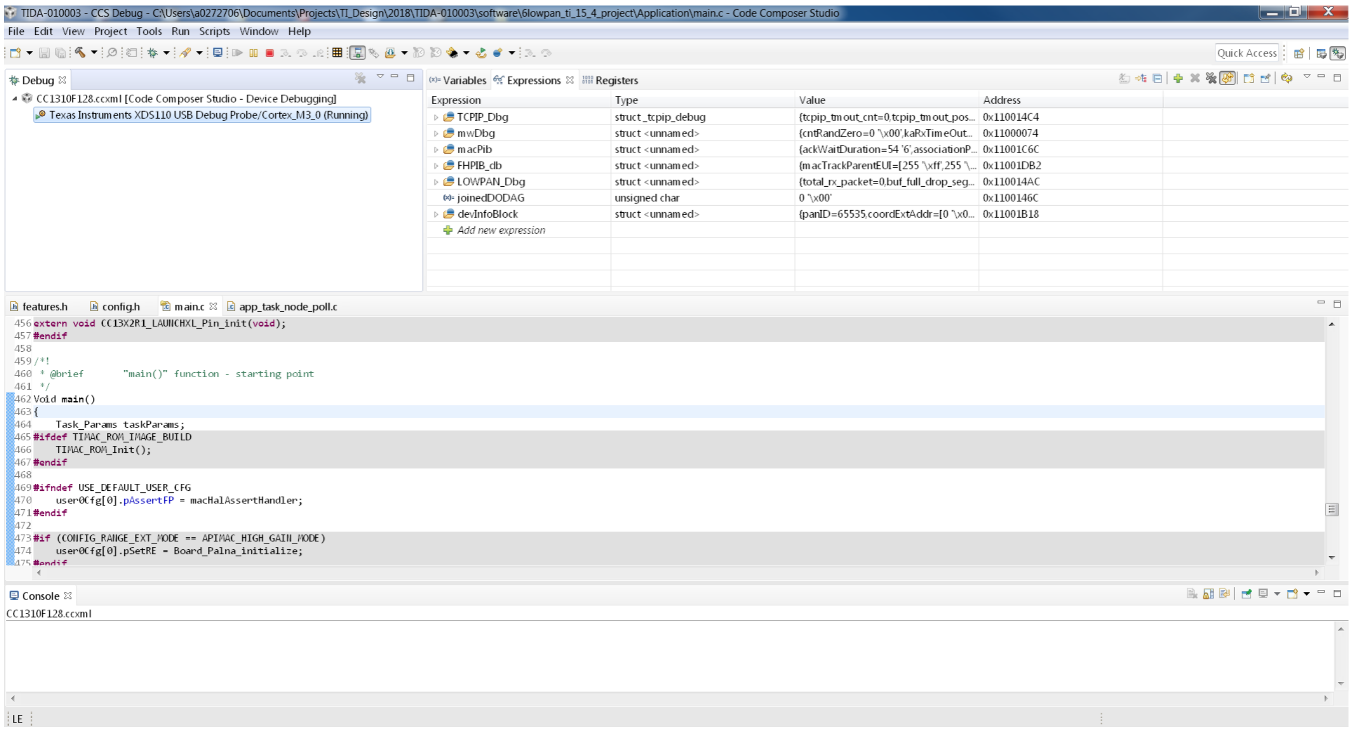
1. Launch the target configuration for the target device (see Figure 12).
2. Connect the target EVM, load the program, and then run (see Figure 13).
3. Add global variables to debug in the *Expressions* tab (see the right top in Figure 13) to trace the variables in the debug mode.



```
/******************************************************************************

 @file config.h

 @brief TI-15.4 Stack configuration parameters for Sensor applications

 Group: WCS LPC
 Target Device: CC13xx

 ******************************************************************************

 Copyright (c) 2016-2017, Texas Instruments Incorporated
 All rights reserved.

 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions
 are met:

 *  Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

 *  Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

 *  Neither the name of Texas Instruments Incorporated nor the names of
    its contributors may be used to endorse or promote products derived
    from this software without specific prior written permission.

 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

 ******************************************************************************
 Release Name: simplelink_cc13x0_sdk_1_60_00_
 Release Date: 2017-12-20 13:05:49
 *****************************************************************************/
#ifndef CONFIG_H
#define CONFIG_H
```

Click the New button to create a new target configuration file. Click here to hide this message.

Figure 12. Code Composer Studio™ Debug: Launching Debug Window


TIDA-010003 - CCS Debug - C:\Users\a0272706\Documents\Projects\TI_Design\2018\TIDA-010003\software\6lowpan_ti_15_4_project\Application\main.c - Code Composer Studio
File Edit View Project Tools Run Scripts Window Help
Quick Access
Debug
CC1310F128.ccxml [Code Composer Studio - Device Debugging]
Texas Instruments XDS110 USB Debug Probe/Cortex_M3_0 (Running)
Variables
Expressions
Registers
features.h
config.h
main.c
app_task_node_poll.c
Console
LE

Expression	Type	Value	Address
TCPIP_Dbg	struct _tcpip_debug	{tcpip_tmout_cnt=0,tcpip_tmout_pos...	0x110014C4
mwDbg	struct <unnamed>	{cntRandZero=0 '\x00',kaRxTimeOut...	0x11000074
macPib	struct <unnamed>	{ackWaitDuration=54 '6',associationP...	0x11001C6C
FHPIB_db	struct <unnamed>	{macTrackParentEUI=[255 '\xff',255 '\...	0x11001DB2
LOWPAN_Dbg	struct <unnamed>	{total_rx_packet=0,buf_full_drop_seg...	0x110014AC
joinedDODAG	unsigned char	0 '\x00'	0x1100146C
devInfoBlock	struct <unnamed>	{panID=65535,coordExtAddr=[0 '\x0...	0x11001B18
Add new expression			

```
extern void CC13X2R1_LAUNCHXL_Pin_init(void);
#endif

/*!
 * @brief       "main()" function - starting point
 */
Void main()
{
    Task_Params taskParams;
#ifdef TIMAC_ROM_IMAGE_BUILD
    TIMAC_ROM_Init();
#endif

#ifndef USE_DEFAULT_USER_CFG
    user0Cfg[0].pAssertFP = macHalAssertHandler;
#endif

#if (CONFIG_RANGE_EXT_MODE == APIMAC_HIGH_GAIN_MODE)
    user0Cfg[0].pSetRE = Board_Palna_initialize;
#endif
```

```
CC1310F128.ccxml
```

Figure 13. Code Composer Studio™ Debug: Running Software in Debug Mode

Table 7 summarizes some global variables to debug the software example.

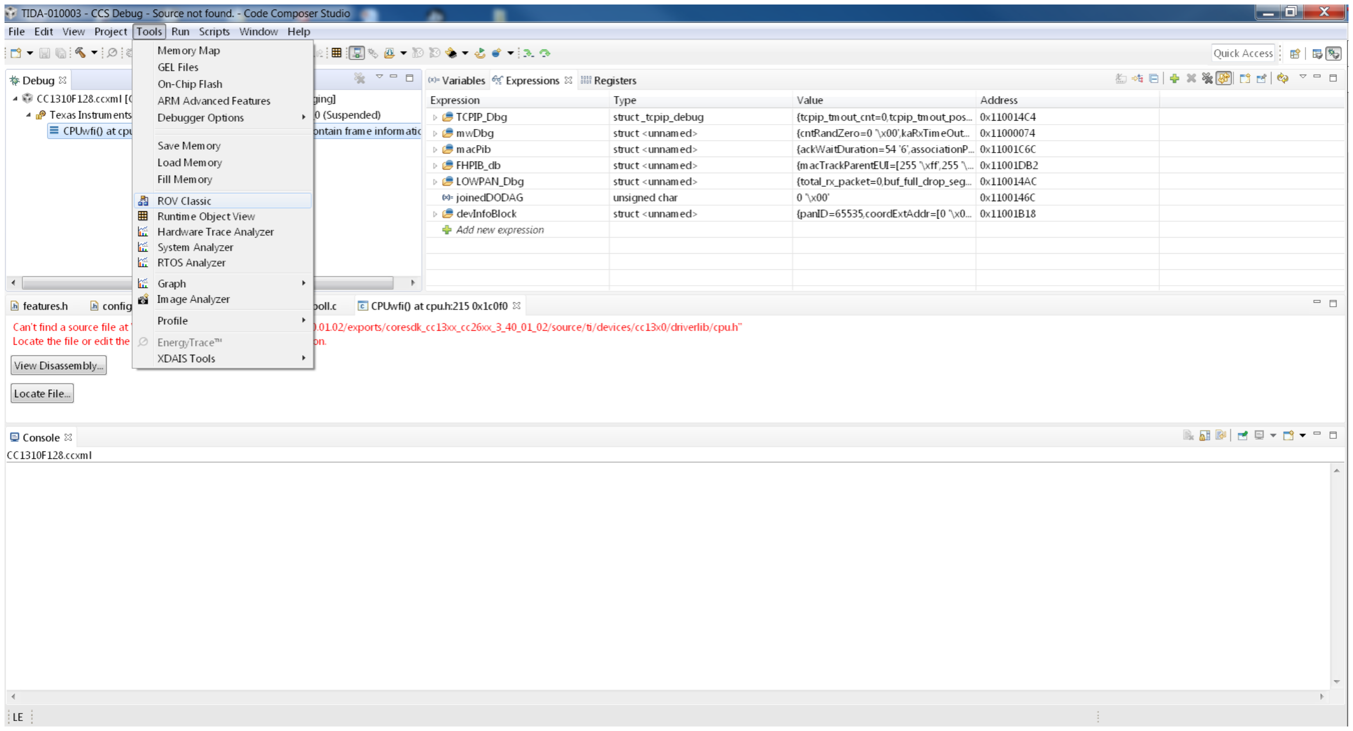
Table 7. Global Variables for Debugging

VARIABLES	DESCRIPTION
TCPIP_Dbg	Debug counts for UDP, IPv6, ICMPv6, RPL layers
LOWPAN_Dbg	Debug counts for 6LoWPAN layer
devInfoBlock	Device state machine and PAN information
mwDbg	Debug counts for middleware layer
macPib	TI-15.4 MAC PIB
FHPIB_db	TI-15.4 FH MAC PIB
joinedDODAG	Flag to indicate RPL DODAG join state
ApiMac_extAddr	8B extended MAC address

3.1.2.2.3.2 *ROV Analysis*

CCS provides a useful tool to debug the software, RTOS Object View (ROV). The ROV tool helps to address various software crash issues or to optimize software examples by analyzing peak memory usage. To debug software with the ROV tool:

1. Suspend debug mode and open ROV as shown in Figure 14. Figure 15 shows the screen capture of ROV (on the right-bottom).
2. As an example, go to the *Detailed* and *CallStacks* taps in the *Task* menu to optimize the stack size, to trace the call stacks, or to address the stack overflow issue.

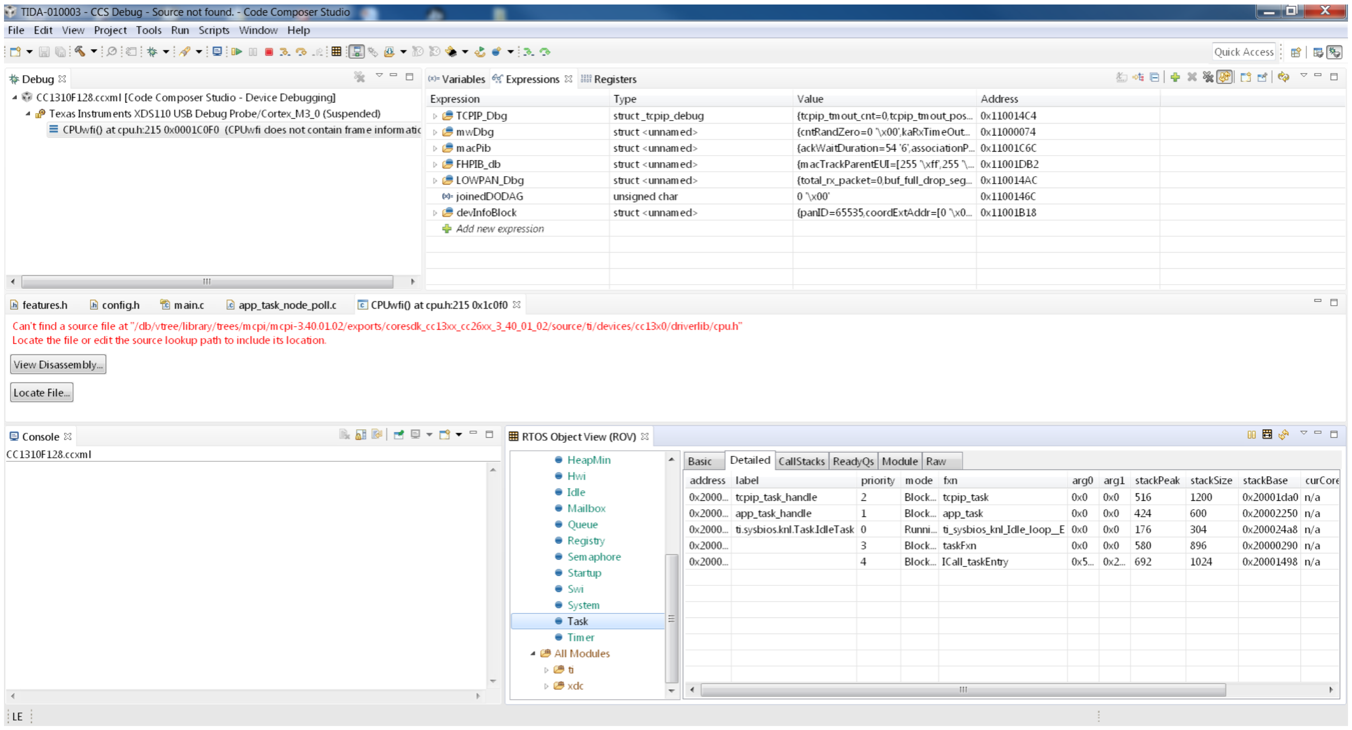


Expression	Type	Value	Address
TCPIP_Dbg	struct _tcpip_debug	{tcpip_tmout_cnt=0,tcpip_tmout_pos...	0x110014C4
mwDbg	struct <unnamed>	{cntRandZero=0 '\x00',kaRxTimeOut...	0x11000074
macPib	struct <unnamed>	{ackWaitDuration=54 '6',associationP...	0x11001C6C
FHPIB_db	struct <unnamed>	{macTrackParentEUI=[255 '\xff',255 '\...	0x11001DB2
LOWPAN_Dbg	struct <unnamed>	{total_rx_packet=0,buf_full_drop_seg...	0x110014AC
joinedDODAG	unsigned char	0 '\x00'	0x1100146C
devInfoBlock	struct <unnamed>	{panID=65535,coordExtAddr=[0 '\x0...	0x11001B18
Add new expression			

```
Can't find a source file at "
.01.02/exports/coresdk_cc13xx_cc26xx_3_40_01_02/source/ti/devices/cc13x0/driverlib/cpu.h"
Locate the file or edit the
on.
```

```
CC1310F128.ccxml
```

Figure 14. Code Composer Studio™ Debug: Launch ROV


TIDA-010003 - CCS Debug - Source not found. - Code Composer Studio
File Edit View Project Tools Run Scripts Window Help
Quick Access
Debug
CC1310F128.ccxml [Code Composer Studio - Device Debugging]
Texas Instruments XDS110 USB Debug Probe/Cortex_M3_0 (Suspended)
CPUwfi() at cpu.h:215 0x0001C0F0 (CPUwfi does not contain frame informatic
Variables
Expressions
Registers
features.h
config.h
main.c
app_task_node_poll.c
CPUwfi() at cpu.h:215 0x1c0f0
View Disassembly...
Locate File...
Console
CC1310F128.ccxml
RTOS Object View (ROV)
HeapMin
Hwi
Idle
Mailbox
Queue
Registry
Semaphore
Startup
Swi
System
Task
Timer
All Modules
ti
xdc
Basic
Detailed
CallStacks
ReadyQs
Module
Raw
LE

Expression	Type	Value	Address
TCPIP_Dbg	struct _tcpip_debug	{tcpip_tmout_cnt=0,tcpip_tmout_pos...	0x110014C4
mwDbg	struct <unnamed>	{cntRandZero=0 '\x00',kaRxTimeOut...	0x11000074
macPib	struct <unnamed>	{ackWaitDuration=54 '6',associationP...	0x11001C6C
FHPIB_db	struct <unnamed>	{macTrackParentEUI=[255 '\xff',255 '\...	0x11001DB2
LOWPAN_Dbg	struct <unnamed>	{total_rx_packet=0,buf_full_drop_seg...	0x110014AC
joinedDODAG	unsigned char	0 '\x00'	0x1100146C
devInfoBlock	struct <unnamed>	{panID=65535,coordExtAddr=[0 '\x0...	0x11001B18
Add new expression			

Can't find a source file at "/db/vtree/library/trees/mcpi/mcpi-3.40.01.02/exports/coresdk_cc13xx_cc26xx_3_40_01_02/source/ti/devices/cc13x0/driverlib/cpu.h"
Locate the file or edit the source lookup path to include its location.

address	label	priority	mode	fxn	arg0	arg1	stackPeak	stackSize	stackBase	curCore
0x2000...	tcpip_task_handle	2	Block...	tcpip_task	0x0	0x0	516	1200	0x20001da0	n/a
0x2000...	app_task_handle	1	Block...	app_task	0x0	0x0	424	600	0x20002250	n/a
0x2000...	ti.sysbios.knl.Task.IdleTask	0	Runni...	ti_sysbios_knl_Idle_loop__E	0x0	0x0	176	304	0x200024a8	n/a
0x2000...		3	Block...	taskFxn	0x0	0x0	580	896	0x20000290	n/a
0x2000...		4	Block...	ICall_taskEntry	0x5...	0x2...	692	1024	0x20001498	n/a

Figure 15. Code Composer Studio™ Debug: Debug Software with ROV Tool

3.2 Testing and Results

3.2.1 Test Setup

Table 8 summarizes the firmware to be flashed to set up the end-nodes and data collector. For these experiments, the end nodes use the CC1310 EVM as Section 3.1.1 shows. The TIDA-01547 implements the 6LoWPAN mesh data collector working with the end nodes. The details of flashing the data collector software are described in the *Flashing the Software Example* section in the TIDA-01547 Design Guide.

Table 8. Firmware for Test Setup

DEVICE	CC1310	MSP432	BUILD CONFIGURATION
END NODE	6lowpan_mesh_CC1310_LAUNCHXL_tirtos_ccs.out	N/A	debug_poll
DATA COLLECTOR	coprocessor_cc13x0_lp.hex	6lowpan_mesh_example_tirtos_ccs.out	TIDA-01547 (debug_root_poll)

3.2.1.1 *Creating Multi-hop Linear Topology*

It is challenging to create multi-hop networks in a small LAB area. For the experiments, the multi-hop topology was created with address filtering in the software. Each node has a list of entries where the node can accept the packet reception. To create a linear topology, each node maintains two entries: one is the target parent and one is the target child.

```
// {0xC6, 0x7F, 0xA4, 0x13, 0x00, 0x4B, 0x12, 0x00} (hop-1)
uint8_t whitelist[NUM_ENTRIES][8]={
{0xF5, 0x2A, 0xF4, 0x14, 0x00, 0x4B, 0x12, 0x00},
{0xC2, 0x2F, 0xF2, 0x13, 0x00, 0x4B, 0x12, 0x00}                              };
```

In the software, this feature can be enabled with the "CONFIG_MULTIHOP_TESTING" macro defined in /Application/subg/config.h.

NOTE: For testing setup, modify the list of entries defined in /Application/middleware.c based on the extended address of the test node. For each node, the extended address set at the initial stage is stored in the global variable of "ApiMac_extAddr".

Figure 16 shows the 6-hop linear topology setup.

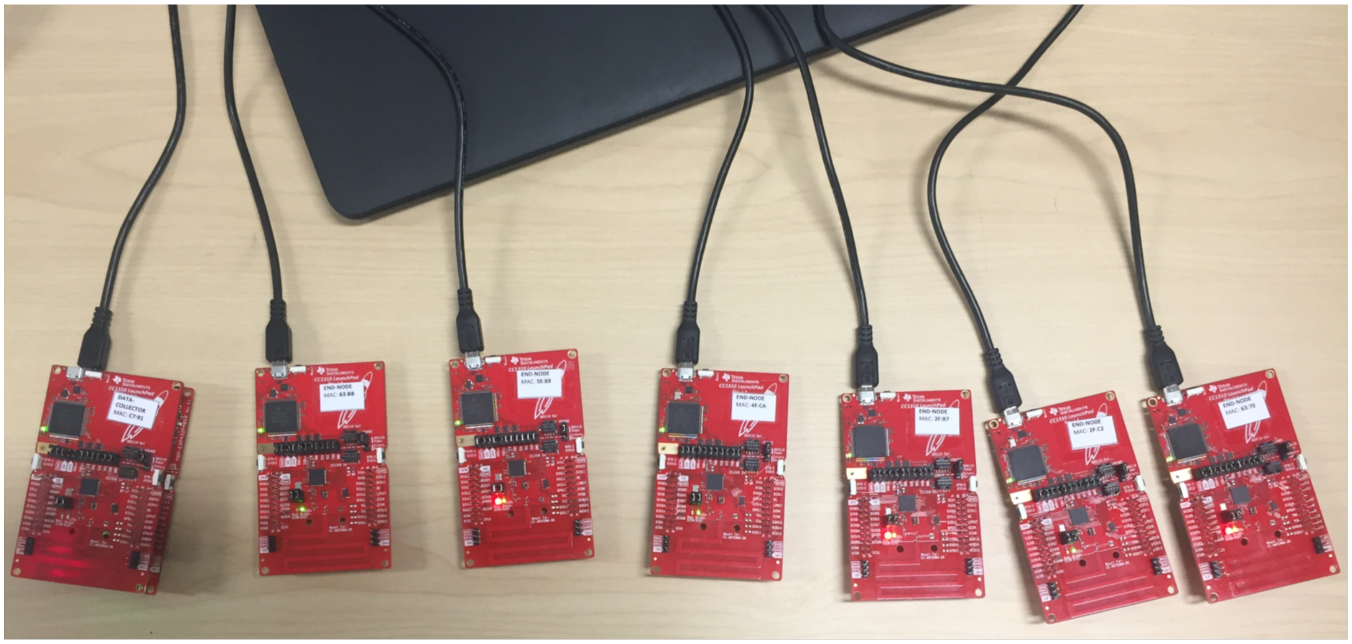


Figure 16. 6-Hop Linear Topology Setup

The multi-hop network with the address filtering technique has pros and cons in terms of network performance because a node can communicate with a dedicated set of nodes only while all the nodes can be seen with each other. A disadvantage is that a node limits the spatial reuse over multi-hop topology while an advantage is that this set-up can avoid hidden node problems.

3.2.2 Impact of Out-of-Network Interference on System Performance

The goal of these experiments is to validate the impact of external noise on the network performance. The external noise can be background channel noise and signals emitted from co-existing sub-1 GHz communication systems.

For these experiments, the UDP poll example is used over a 6-hop linear topology as shown in Figure 17. The data collector sends poll messages every 5 seconds and, as a response, each node sends back the same size data packets to the data collector. To verify the impact of noise, continuous background noise is generated to the network. For details of the noise setup, see Section 3.2.2.1.

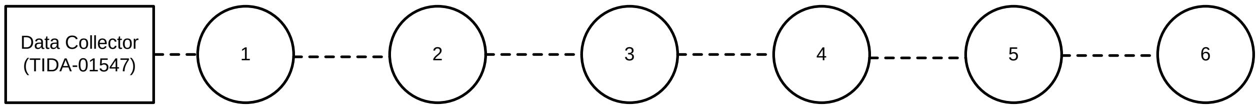
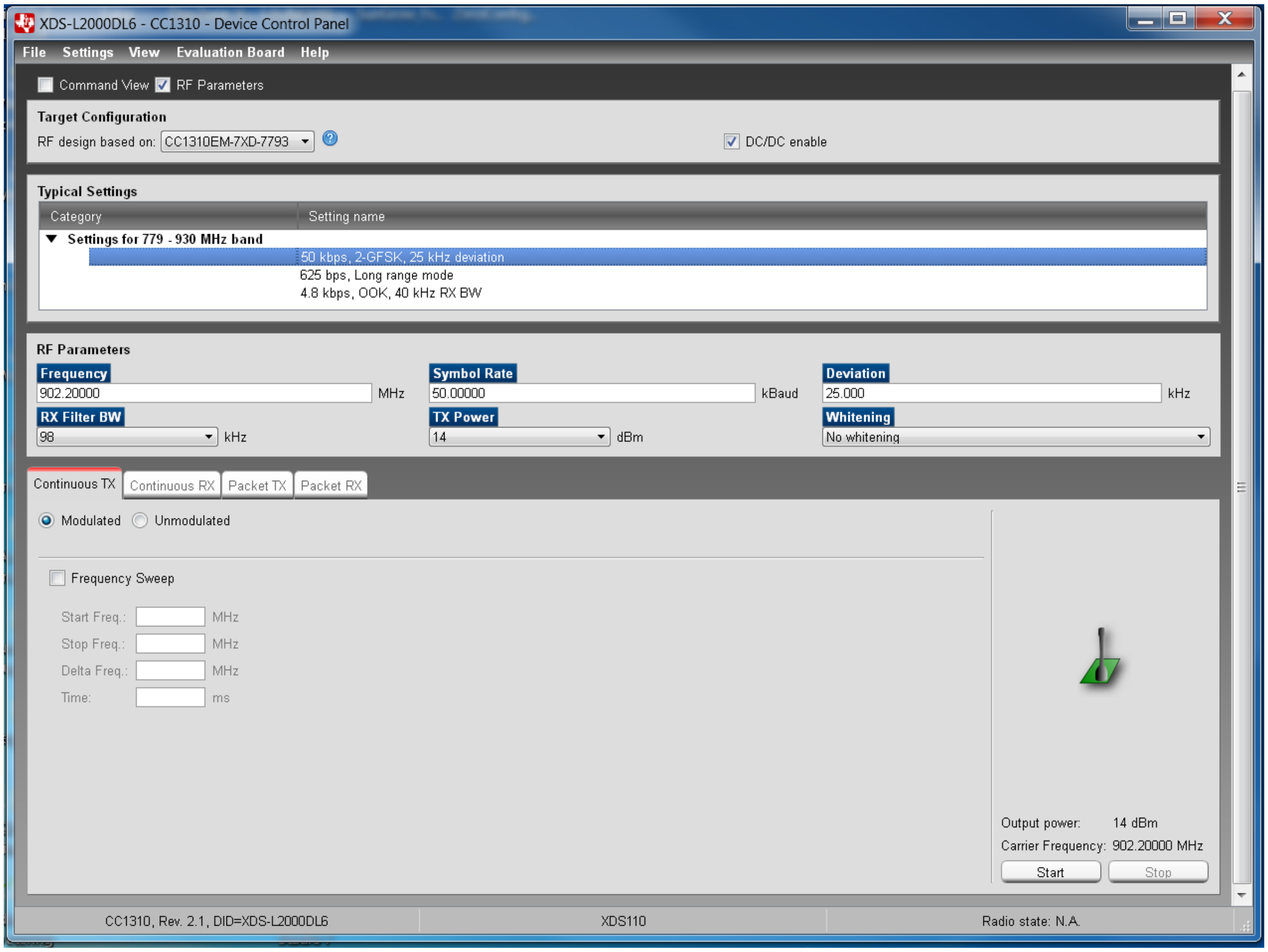


Figure 17. 6-hop Linear Topology

3.2.2.1 Noise Generator

The noise generator is used to verify the noise immunity feature of the FH technique in multi-hop networks. This feature is important especially when the system is running in the unlicensed bands. According to N. Baccour et al., several radio technologies such as wireless sensor networks, telemetry networks, cordless telephones, and mobile phones over the European Global System for Mobile Communications (GSM) band can cause significant in-band or out-of-band noises on the 868-MHz and 915-MHz frequency bands.

Figure 18 shows the SmartRF™ Studio setup to generate continuous noise with the maximum TX power of 14 dBm at a 902.2-MHz frequency [channel #0 in the FCC band (channel #0 to #128)]. Due to the maximum TX power level of the noise, the actual noise effect is not limited to the channel #0. To verify the impact of the continuous noise on the channels, Figure 19 shows the noise capture in the frequency domain. As the channel space is 200 kHz, the number of channels affected by the noise generator is up to six channels (with a threshold of 0 dBm) or eight channels (with a threshold of –10 dBm) as summarized in Table 9.



Target Configuration
RF design based on: CC1310EM-7XD-7793
DC/DC enable [✓]

Typical Settings

Category	Setting name
Settings for 779 - 930 MHz band	
	50 kbps, 2-GFSK, 25 kHz deviation
	625 bps, Long range mode
	4.8 kbps, OOK, 40 kHz RX BW

RF Parameters

Frequency: 902.20000 MHz	Symbol Rate: 50.00000 kBaud	Deviation: 25.000 kHz
RX Filter BW: 98 kHz	TX Power: 14 dBm	Whitening: No whitening

[✓ Modulated] Unmodulated

Frequency Sweep
Start Freq.: MHz
Stop Freq.: MHz
Delta Freq.: MHz
Time: ms

Output power: 14 dBm
Carrier Frequency: 902.20000 MHz

CC1310, Rev. 2.1, DID=XDS-L2000DL6 | XDS110 | Radio state: N.A.

Figure 18. Continuous Noise at 902.2-MHz Frequency (Channel #0 in FCC Band)

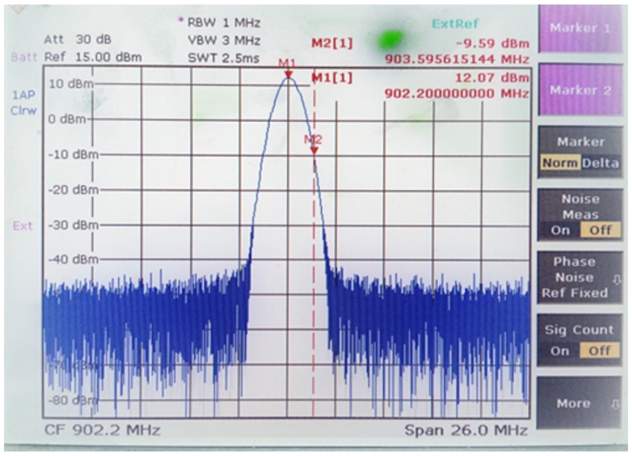


Figure 19. Spectrum Analyzer Capture for Continuous Noise With TX Power of 14 dBm

Table 9. Number of Channels Affected by the Noise Generator

THRESHOLD	FREQUENCY	NUMBER OF CHANNELS ABOVE THE THRESHOLD
0 dBm	903.23 MHz	6
–10 dBm	903.6 MHz	8

3.2.2.2 Delivery Ratio

The delivery ratio is measured to verify the communication reliability of the FH system in noisy channels over multi-hop networks. Typically, the delivery ratio is measured in one direction based on the ratio of the number of TXs at the transmitter and the number of RXs at the receiver. In this testing setup, the delivery ratio is measured based on bidirectional communication based on the ratio of the number of TXs and the number of RXs (echo back from end nodes) at the data collector. Compared to the delivery ratio measured in one direction, the delivery ratio performance for these experiments is more conservative as the successful transmissions in both directions can increase the number of RXs.

Figure 20 shows the delivery ratio performance with and without continuous background noise as a function of the number of hops. The X-axis shows the number of hops and the Y-axis is delivery ratio in percentage. The result shows that the continuous background noise has no impact on the delivery ratio performance. The average delivery ratio over 6-hop networks achieves 99.7% and the worst-case performance shows 99.4% at six-hop distance.

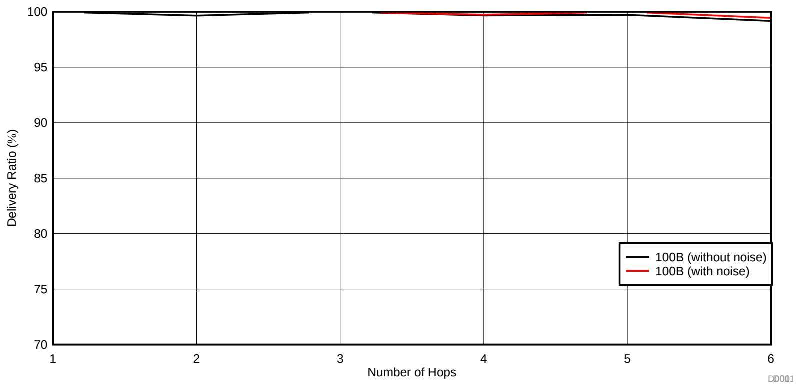


Figure 20. Delivery Ratio as a Function of Number of Hops

3.2.2.3 Round-Trip Time (RTT)

The RTT is measured at the application layer by calculating the time gap between the TX time of poll message sent by the root node and the RX time of data at the root node sent by an end node.

Figure 21 shows the RTT performance with and without the continuous background noise as a function of the number of hops. The X-axis shows the number of hops and the Y-axis shows the RTT in the unit of seconds. The experimental result shows that the RTT performance has a small gap with the continuous noise and the RTT performance gap increases as the number of hops increases. This result is explained by how the TI 15.4-Stack achieves reliable delivery ratio performance even with the continuous background noise. The TI 15.4-Stack implemented a deferred transmission mechanism to minimize the impact of noise on the delivery ratio, which increases the RTT. As the FH system uses 129 channels uniformly in the FCC band, the impact of noise on the noisy channels is negligible for the one-hop node (once every 129 transmissions). The RTT simply adds the delay for each link, and the probability to dwell on the noisy frequency channels increases as the number of hops increases. This is why the impact of noise on the RTT performance increases as the number of hops increases.

The result shows that, for one-hop distance, the average RTT performance achieves 0.19 seconds regardless of noise and, for six-hop distance, the RTT performance shows 1.17 and 1.38 seconds for the FH system without and with noise, respectively.

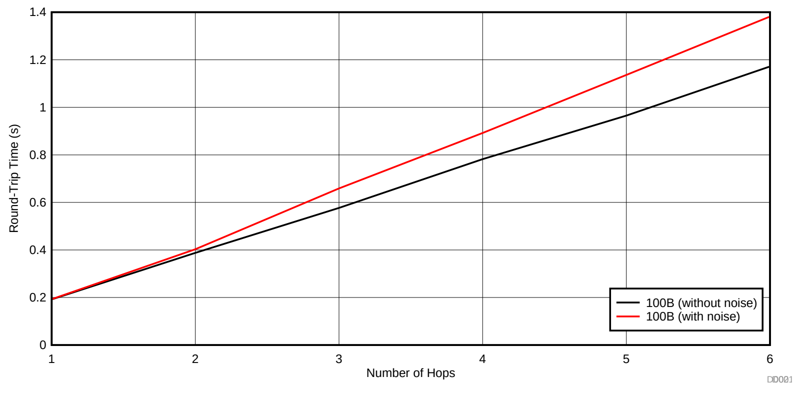


Figure 21. Round-Trip Time as a Function of Number of Hops

3.2.2.4 Goodput

The goodput is defined as the application-level throughput considering all the overheads in the underlying layers. The goodput performance is calculated based on the measured RTT as shown in Figure 21. Because the RTT can be measured on successful transmissions only, the goodput performance does not count on the packet losses caused by channel variation or collisions. Therefore, this can be a reference on the expected goodput when data transmissions are successful.

Figure 22 shows the goodput performance as a function of the number of hops. The X-axis denotes the number of hops and the Y-axis shows the goodput in kbps. As expected, the goodput performance decreases as the number of hop increases due to more transmission times through multiple hops to reach at the destination. The result shows that the goodput performance varies from 8.3 kbps (100B at one-hop distance) to 1.2 kbps (100B at six-hop distance).

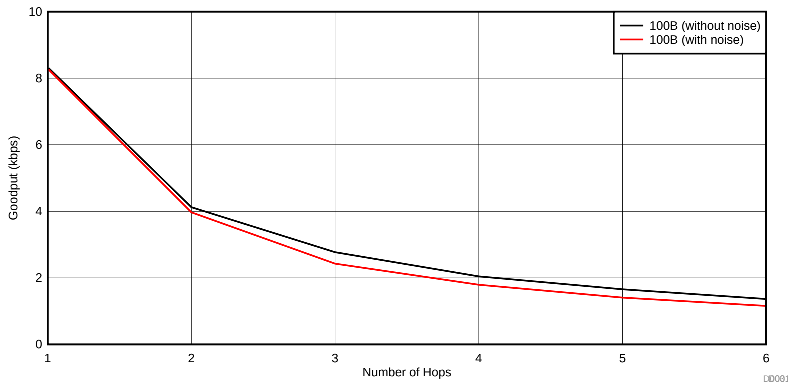


Figure 22. Goodput Performance as a Function of Number of Hops

3.2.3 20-Node Network Testing

The goal of this experiment is to measure delivery ratio performance as well as to verify the software reliability in a real scenario without the filtering technique to create multi-hop discussed in Section 3.2.1.1.

Figure 23 show the delivery ratio performance as a function of the MAC address of each node in the 20-node setup. This experiment uses the UDP polling example with 100B data, and the experiment runs for two days. The X-axis shows the MAC address of each end node and the Y-axis shows the delivery ratio in percentage. The experimental result shows that all the nodes achieve close to 100% delivery ratio performance. The average delivery ratio with 20 nodes is 99.83% and the average RTT is 0.201 second.

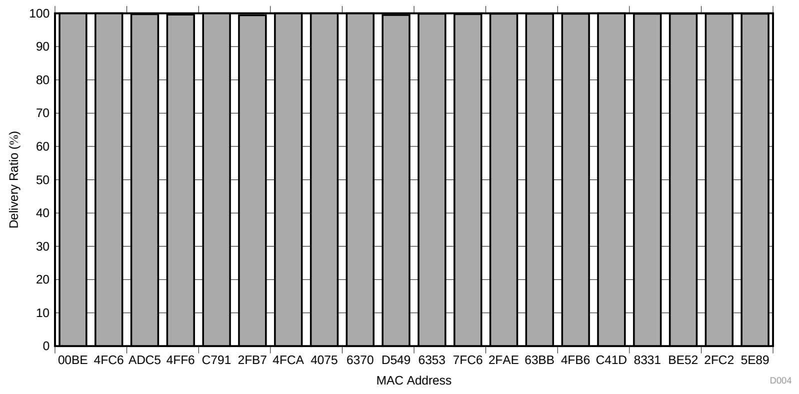


Figure 23. Delivery Ratio Performance With 100B UDP Polling Example

Table 10 summarizes test cases to verify the software reliability and self-healing feature with the 20-node network setup. The experiments use the UDP polling example with 100B data. The test results show that the 6LoWPAN mesh software works reliability without losing connections in the 20-node setup and the keep-alive mechanism implemented in the software is functional by detecting and recovering the connection losses.

Table 10. Software Reliability Test Summary

TEST CASE	DESCRIPTION	PASS or FAIL
Long-run data testing (> one week)	Run 100B UDP polling example in the 20-node setup	Pass (All 20 nodes stayed connection to the data collector)
Self-healing testing (data collector failure)	Power-cycle the data collector to see if all the 20 nodes are reconnected to the network.	Pass
Self-healing testing (end-node failure)	Power-cycle some end-nodes to see if all the 20 nodes are reconnected to the network.	Pass

4 Design Files

4.1 Schematics

To download the schematics, see the design files at TIDA-010003.

4.2 Bill of Materials

To download the bill of materials (BOM), see the design files at TIDA-010003.

4.3 PCB Layout Recommendations

4.3.1 Layout Prints

To download the layer plots, see the design files at TIDA-010003.

4.4 Altium Project

To download the Altium Designer® project files, see the design files at TIDA-010003.

4.5 Gerber Files

To download the Gerber files, see the design files at TIDA-010003 .

4.6 Assembly Drawings

To download the assembly drawings, see the design files at TIDA-010003.

5 Software Files

To download the software files, see the design files at TIDA-010003.

6 About the Author

WONSOO KIM is a system engineer at Texas Instruments, where he is responsible for driving grid communication system solutions, defining future requirements in TI product roadmap, and providing system-level support and training focusing on communication systems for Smart Grid customers. He received a Ph.D. degree in electrical and computer engineering from the University of Texas, Austin, Texas.

Revision History

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

Changes from Original (June 2018) to A Revision **Page**

- Added TIDA-010024 and TIDA-010032 to Features as reference designs that are fully compatible with the TIDA-01547 1
- Added information regarding support for low-power mode configuration for battery-powered end nodes 1
- Added 200-kbps frequency-shift keying as a mode that is supported by TI SimpleLinkPHY for IEEE 802.15.4g 2
- Changed TIDA-010003 System Architecture image 2
- Changed information in Key System Specifications table 3
- Changed information throughout Software section 8
- Changed Code Composer Studio™ Build Configuration image 9
- Changed Code Composer Studio™ Compiler Version image 9
- Changed Code Composer Studio™ RTSC Screen Capture image... 10
- Changed information in Firmware for Test Setup table... 19
- Changed code in Creating Multi-hop Linear Topology section 19