TI Designs: TIDA-01547

Simple 6LoWPAN mesh data collector improves network performance reference design

Description

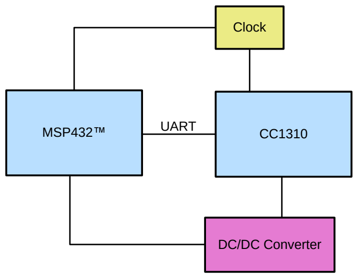
This reference design implements a simple mesh network data collector for smart meter advanced metering infrastructure (AMI) networks. The network is an IPv6 over low-power wireless personal area networks (6LoWPAN) solution. The data collector implements a complete software stack split across a 32-bit host MCU and a separate wireless MCU to optimize performance and flexibility. Network performance is improved by the TI-15.4 stack which implements IEEE 802.15.4e/g protocols. This stack implements an un-slotted channel hopping (USCH) mode that provides protection against in-band interference.

Resources

TIDA-01547	Design Folder
MSP432P401R	Product Folder
CC1310	Product Folder
TPD4E004	Product Folder
TPD6E004	Product Folder
TM4C1294NCPDT	Product Folder
LM4040	Product Folder
TPS796	Product Folder
SN74AVC4T245	Product Folder
TPS735	Product Folder
TPS2012A	Product Folder
MSP430G2452	Product Folder

TI E2E™ Community

ASK Our E2E™ Experts

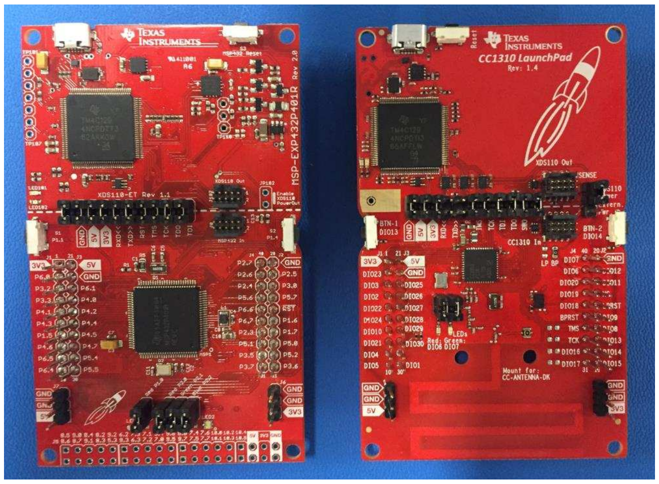


Features

- IPv6 networking over low-power RF in <1-GHz ISM bands
- Data collector and edge router supports up to 200 end-nodes
- Implements mesh network protocols of 6LoWPAN, RPL, IPv6/ICMPv6, and UDP
- Implements TI-15.4 stack with frequency hopping and MAC data encryption
- Adopts layering architecture identical to Wi-SUN FAN v1.0
- Provides open source-based working example with compile-time option to run as data collector (root node) or end-node

Applications

- Wireless communications
- Data collector
- Data concentrator
- Electricity meter



1 System Description

This reference design provides a simple low-power data collector 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 or in-network interference.

Another segment of this design is the 6LoWPAN mesh stacks, which improve 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 two SimpleLink™ microcontrollers (MCUs); the MSP432™ host MCU and the CC1310 wireless MCU. Figure 1 shows the software architecture. The TI-MAC (Medium Access Control) can be configured to operate in one of three modes: non-beacon, beacon, and FH. The TI SimpleLink PHY supports IEEE 802.15.4g for 50-kbps frequency-shift keying (FSK) and long range mode for 5 kbps. The sub-1 GHz RF on the CC1310 MCU can support three frequency bands: 902 MHz, 863 MHz, and 433 MHz.

This reference design configures the TI 15.4-stack with the FH mode on top of the IEEE 802.15.4g-based 50-kbps FSK over the FCC band (902 MHz). The MSP432 MCU runs as a network MCU that provides simple applications, user datagram protocol (UDP), IPv6, ICMPv6, RPL (a routing protocol for low-power and lossy networks), 6LoWPAN, and host interface layer.

The data collector typically supports local network as well as backbone connectivity; in this case, it is called edge (or border) router. The backbone connectivity feature is out of scope for this design. The MSP432 MCU provides enough peripherals to integrate with the external communication modules such as Wi-Fi® or cellular, which allows this design to extend to the edge router.

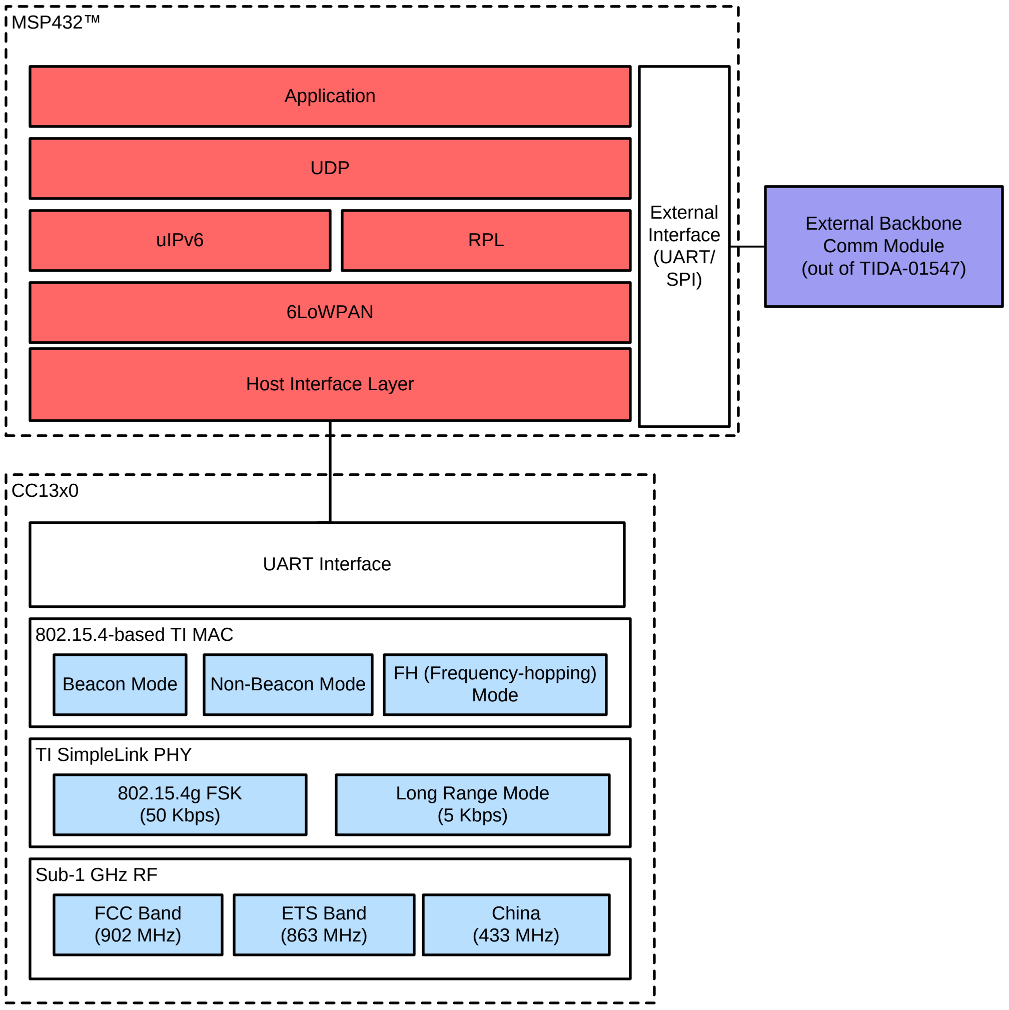


Figure 1. TIDA-01547 System Architecture

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 six-hop networks	
Maximum number of nodes	• 200 (in software, configurable) • 100 for end nodes • Tested up to 20 nodes	Section 3.2.5
Maximum number of neighbors	• 50 (in software, configurable) • Tested up to 20 neighbors	Maximum supported by the TI 15.4-stack (in CC1310)
Maximum application data size	• 200B (in software, configurable)	
Delivery ratio	• 97.9% (in six-hop node with 100% duty-cycle channel noise) • versus 0% delivery ratio with non-FH system	Section 3.2.2.2
Round-trip time (RTT)	• 0.21 seconds (100B over one-hop) • 1.39 seconds (100B over six-hop)	Section 3.2.2.3
Goodput	• 10.5 kbps (500B in one-hop distance)	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

2 System Overview

2.1 *Block Diagram*

Figure 2 shows the system block diagram. The MSP432 MCU is the 6LoWPAN mesh MCU that runs UDP applications, 6LoWPAN mesh network stacks, and host interface protocol connected to the CC1310 through the UART interface. The CC1310 is the connectivity MCU running 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 EVMs are powered by USB, the TPS796 and TPS375 are chosen to convert 5 V to 3.3 V.

For end-equipment designs, the selection of the DC/DC converters depends on the input voltage level and required current level. The TI WEBENCH Power Designer provides the details of the power supply design with the given V_{IN} and V_{OUT}.

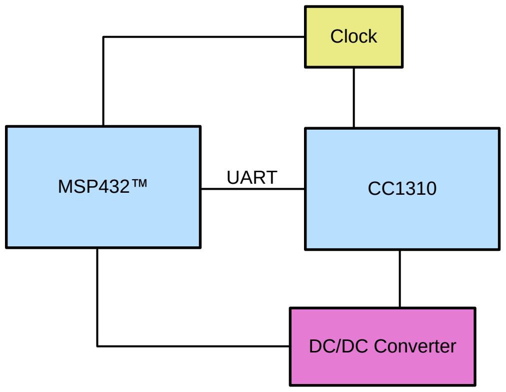


Figure 2. TIDA-01547 System Block Diagram

2.2 *Design Considerations*

For this reference design, these devices perform the following:

- The MSP432 host MCU platform is based upon the Arm® Cortex®-M4F CPU and offers a wide variety of flash and RAM configurations plus the interfaces to support backbone network connectivity.
- 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 and TPS375 low-power linear regulators offer high power-supply rejection ratio (PSRR), low noise, fast start-up, and excellent line and load transient responses.

2.3 *Highlighted Products*

2.3.1 MSP432P401R

The MSP432P401x MCU family is TI's latest addition to its portfolio of efficient ultra-low-power mixed signal MCUs. The MSP432P401x MCUs feature the Arm Cortex-M4 processor in a wide configuration of device options, which include a rich set of analog, timing, and communication peripherals. These options cater to a large number of application scenarios where both efficient data processing and enhanced low-power operation are paramount.

Overall, the MSP432P401x is an ideal combination of the TI MSP430™ low-power DNA, advance mixed signal features, and the processing capabilities of the Arm 32-bit Cortex-M4 RISC engine. The devices ship with bundled peripheral driver libraries and are compatible with standard components of the Arm ecosystem.

Figure 3 shows the MSP432 MCU functional block diagram.

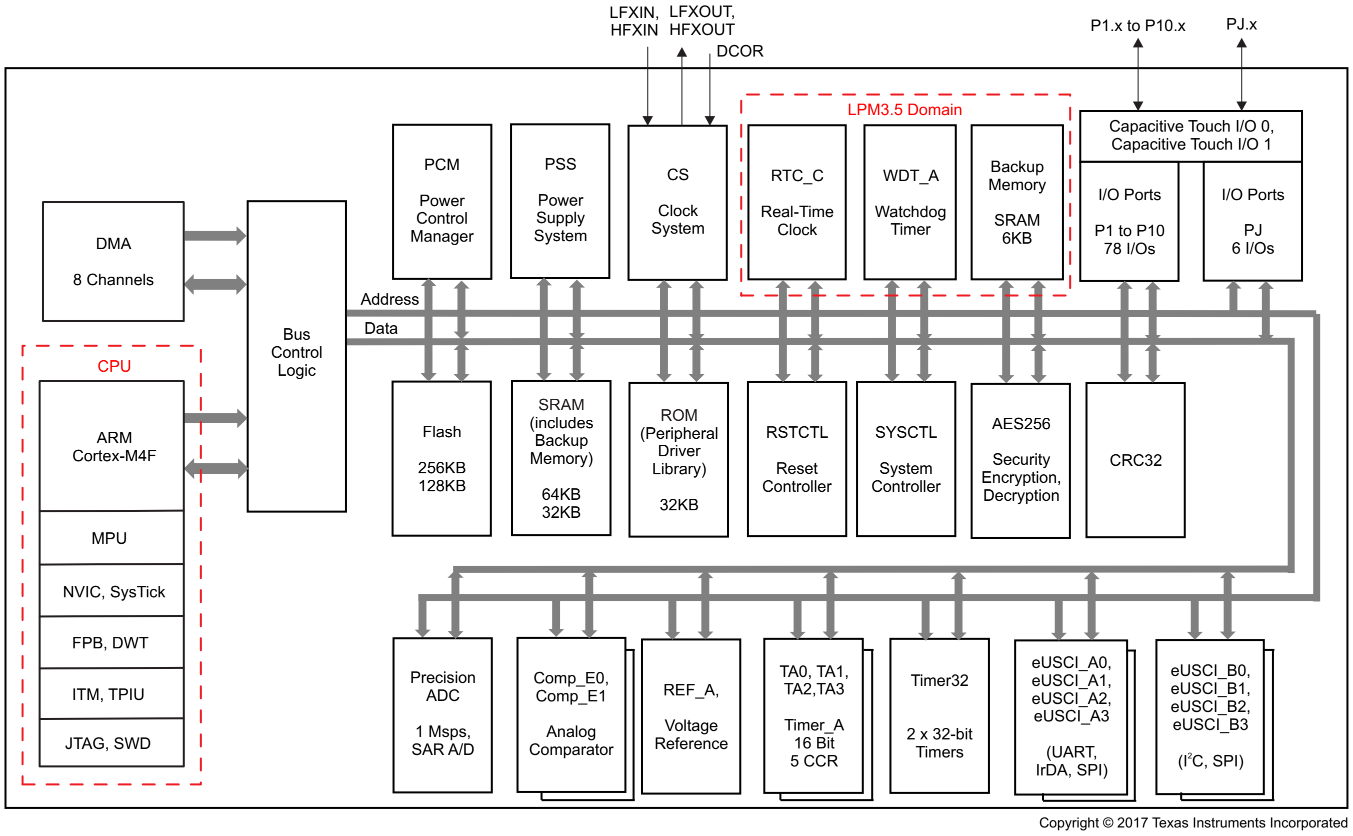

LFXIN, HFXIN
LFXOUT, HFXOUT
DCOR
P1.x to P10.x
PJ.x
LPM3.5 Domain
PCM Power Control Manager
PSS Power Supply System
CS Clock System
RTC_C Real-Time Clock
WDT_A Watchdog Timer
Backup Memory SRAM 6KB
Capacitive Touch I/O 0, Capacitive Touch I/O 1
I/O Ports P1 to P10 78 I/Os
I/O Ports PJ 6 I/Os
DMA 8 Channels
Bus Control Logic
Address
Data
CPU
ARM Cortex-M4F
MPU
NVIC, SysTick
FPB, DWT
ITM, TPIU
JTAG, SWD
Flash 256KB 128KB
SRAM (includes Backup Memory) 64KB 32KB
ROM (Peripheral Driver Library) 32KB
RSTCTL Reset Controller
SYSCTL System Controller
AES256 Security Encryption, Decryption
CRC32
Precision ADC 1 Msps, SAR A/D
Comp_E0, Comp_E1 Analog Comparator
REF_A, Voltage Reference
TA0, TA1, TA2,TA3 Timer_A 16 Bit 5 CCR
Timer32 2 x 32-bit Timers
eUSCI_A0, eUSCI_A1, eUSCI_A2, eUSCI_A3 (UART, IrDA, SPI)
eUSCI_B0, eUSCI_B1, eUSCI_B2, eUSCI_B3 (I²C, SPI)
Copyright © 2017 Texas Instruments Incorporated

Figure 3. MSP432P401R Block Diagram

2.3.2 CC1310

The CC1310 is a member of the CC26xx and CC13xx family of cost-effective, ultra-low-power, 2.4-GHz and sub-1 GHz RF devices. Its very low active RF, MCU current, and low-power mode current consumption provide excellent battery lifetime and allow operation on small coin-cell batteries and in energy-harvesting applications. The device is the first part in a sub-1 GHz family of cost-effective, ultra-low-power wireless MCUs.

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. The low-power consumption of the CC1310 does not come at the expense of RF performance; the CC1310 has excellent sensitivity and robustness (selectivity and blocking) performance. The CC1310 is a highly integrated, true single-chip solution incorporating a complete RF system and an on-chip DC/DC converter.

Sensors can be handled in a very low-power manner by a dedicated autonomous ultra-low-power MCU that can be configured to handle analog and digital sensors; thus, the main MCU (Cortex-M3) is able to maximize sleep time. The CC1310 power and clock management and radio systems require specific configuration and handling by software to operate correctly. These requirements are implemented in TI RTOS, and it is therefore recommended that this software framework is used for all application development on the device. The complete TI-RTOS and device drivers are offered in source code free of charge.

Figure 4 shows the CC1310 functional block diagram.

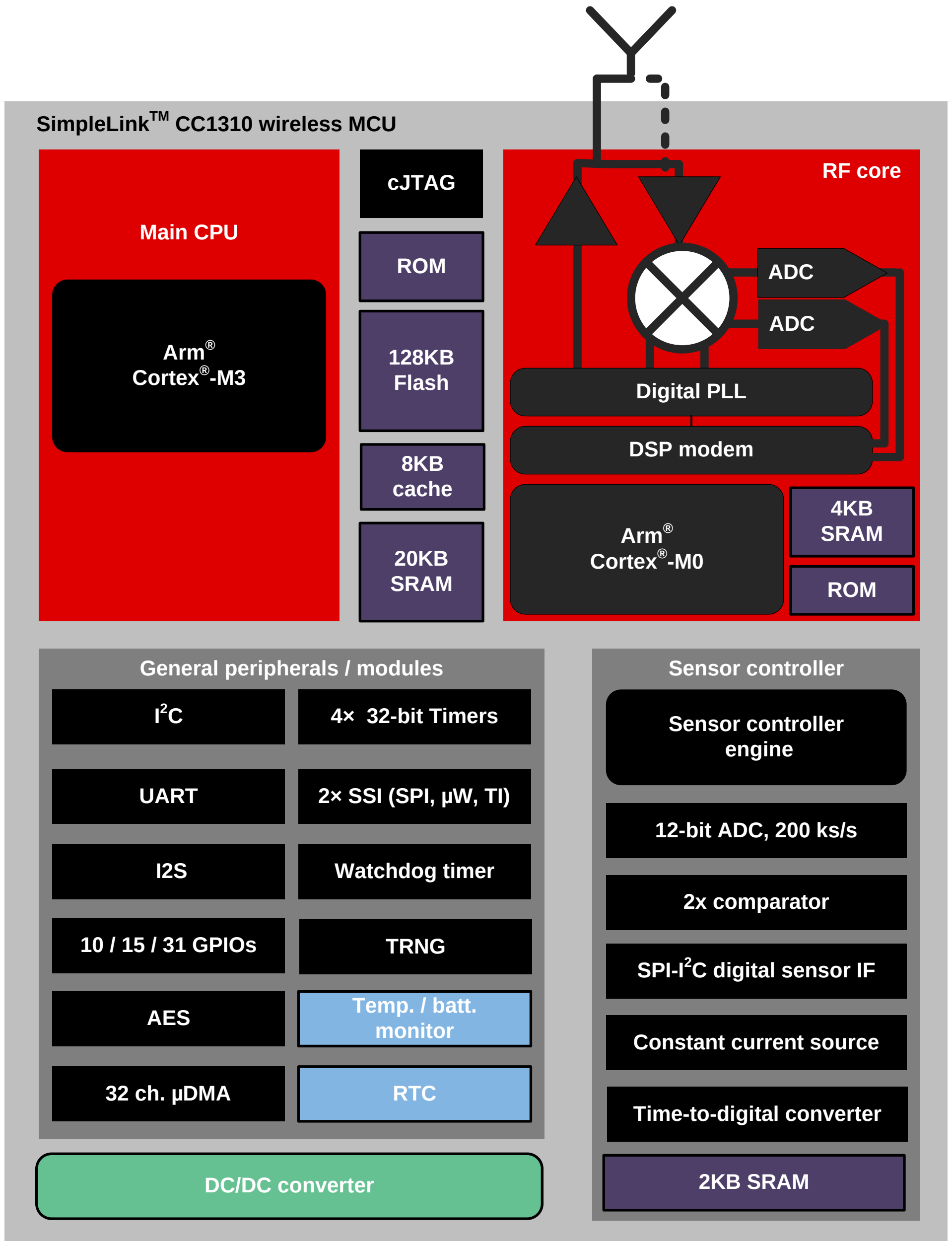


Figure 4. CC1310 Functional Block Diagram

2.3.3 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 5 and Figure 6 show the TPS796 functional block diagrams.

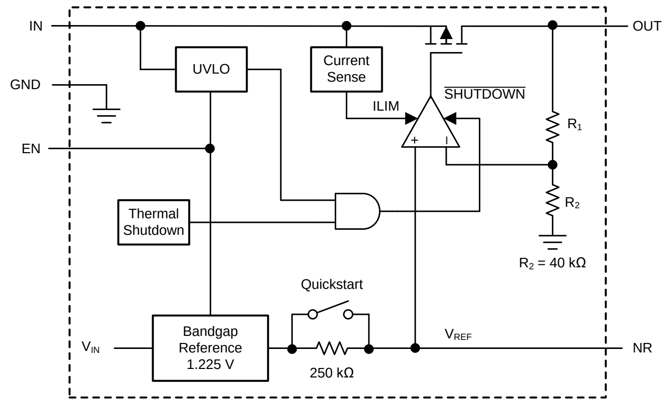


Figure 5. Functional Block Diagram (Fixed Version)

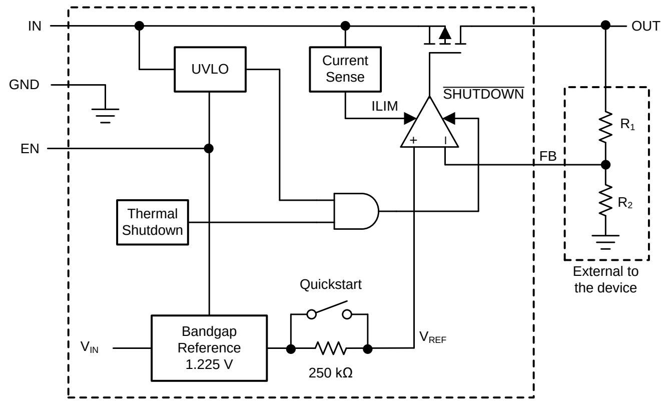


Figure 6. Functional Block Diagram (Adjustable Version)

2.3.4 TPS735

The TPS735 family of LDO, low-power linear regulators offers excellent AC performance with very low ground current. This family provides high PSRR, low noise, fast start-up, and excellent line and load transient responses while consuming a very low 46-µA (typical) ground current.

The TPS735 family of devices is stable with ceramic capacitors and uses an advanced BiCMOS fabrication process to yield a typical dropout voltage of 280 mV at 500-mA output. The TPS735 family of devices uses a precision voltage reference and feedback loop to achieve overall accuracy of 2% (V_{OUT} > 2.2 V) over all load, line, process, and temperature variations. This family of devices is fully specified from T_A = –40°C to +125°C and is offered in a low-profile, 3-mm × 3-mm SON-8 package and a 2-mm × 2-mm SON-6 package.

Figure 7 and Figure 8 show the TPS735 functional block diagrams.

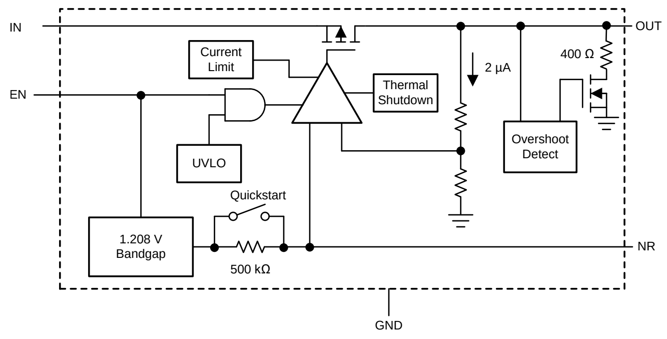


Figure 7. Functional Block Diagram (Fixed Voltage Version)

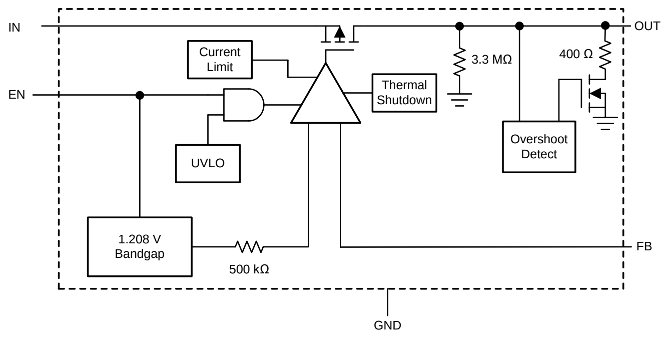


Figure 8. Functional Block Diagram (Adjustable Voltage Version)

3 Hardware, Software, Testing Requirements, and Test Results

3.1 Required Hardware and Software

3.1.1 Hardware

This reference design is built with two standard TI EVMs: LAUNCHXL-CC1310 and MSP-EXP432P401R, as shown in Figure 9. The CC1310 can be replaced with the CC1350 (LAUNCHXL-CC1350) without modifying the software example provided with the design.

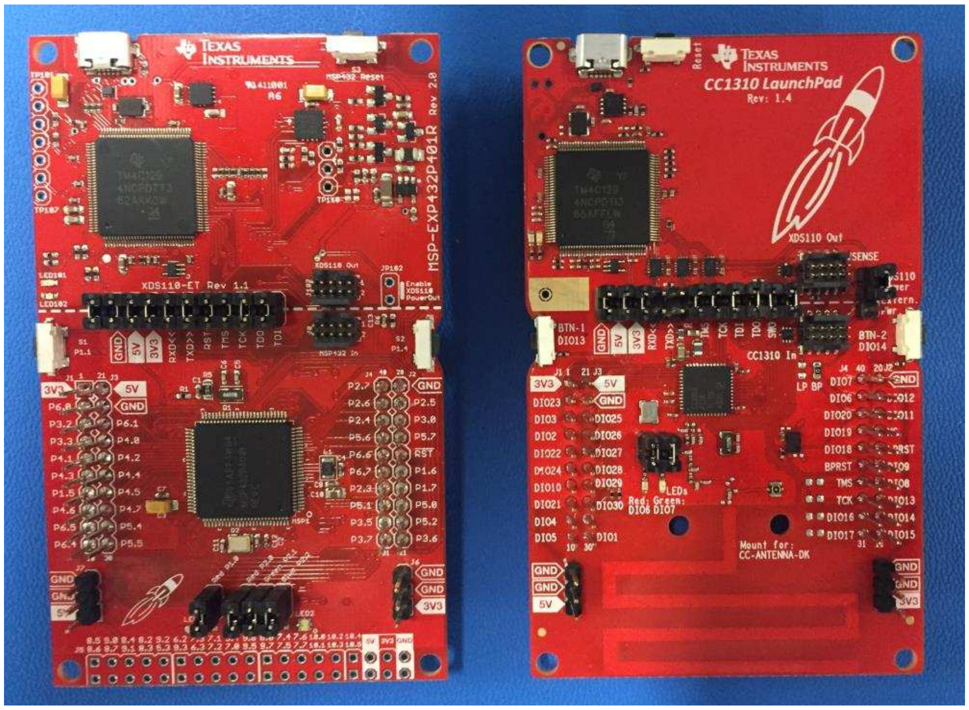


Figure 9. TIDA-01547 Hardware Platform

3.1.1.1 Hardware Configuration

To configure the hardware:

1. Remove two jumpers of RXD and TXD on P4 on the LAUNCHXL-CC1310 (see Figure 9).
2. Plug-in the LAUNCHXL-CC1310 on top of the MSP-EXP432P401R (see Figure 10).
3. Power the EVMs by connecting the USB cable to the MSP-EXP432P401R (or LAUNCHXL-CC1310) EVM on one side and to the power source (for example, PC) on the other side.

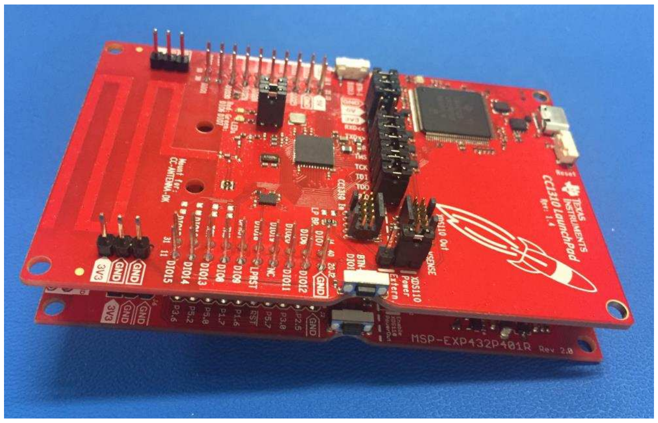


Figure 10. Hardware Configuration (Side)

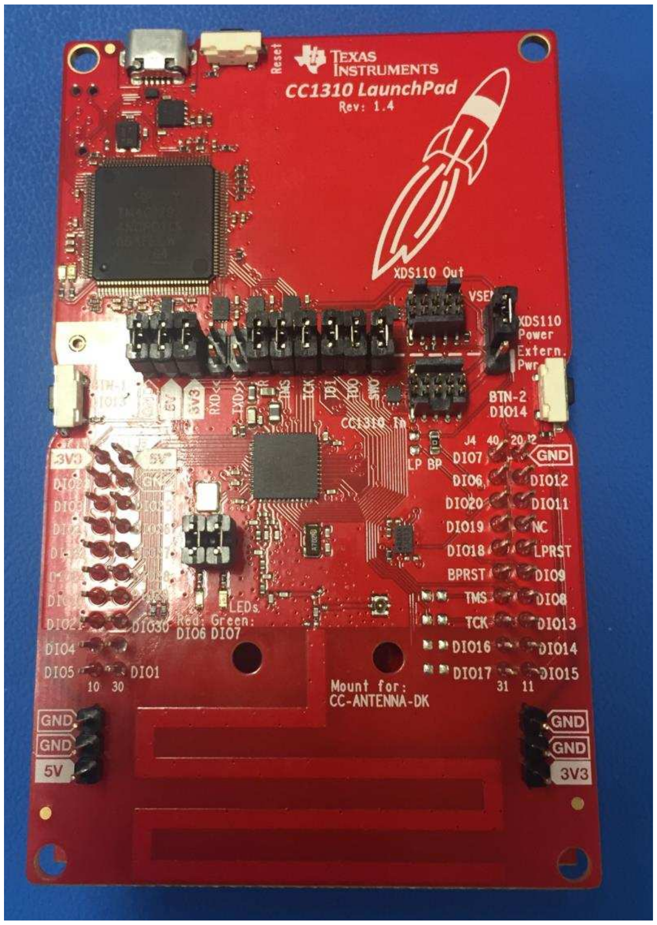


Figure 11. Hardware Configuration (Top)

For software development and short RF range testing, it is convenient to plug the two boards on top of each other, as shown in Figure 10. However, in this hardware configuration, the MSP-EXP432P401R (on the bottom) might cause some interference on the antenna area on the LAUNCHXL-CC1310 (on the top), which greatly reduces the RF range of the board.

An alternative hardware configuration, as shown in Figure 12, avoids this issue and should be used if long RF range is desired. Table 2 summarizes pin mapping between two EVMs.

To achieve long range communications, the antenna performance is very important. For guidelines on the antenna selection, see the following application notes: *Antenna Selection Guide* and *Antenna Selection Quick Guide* .

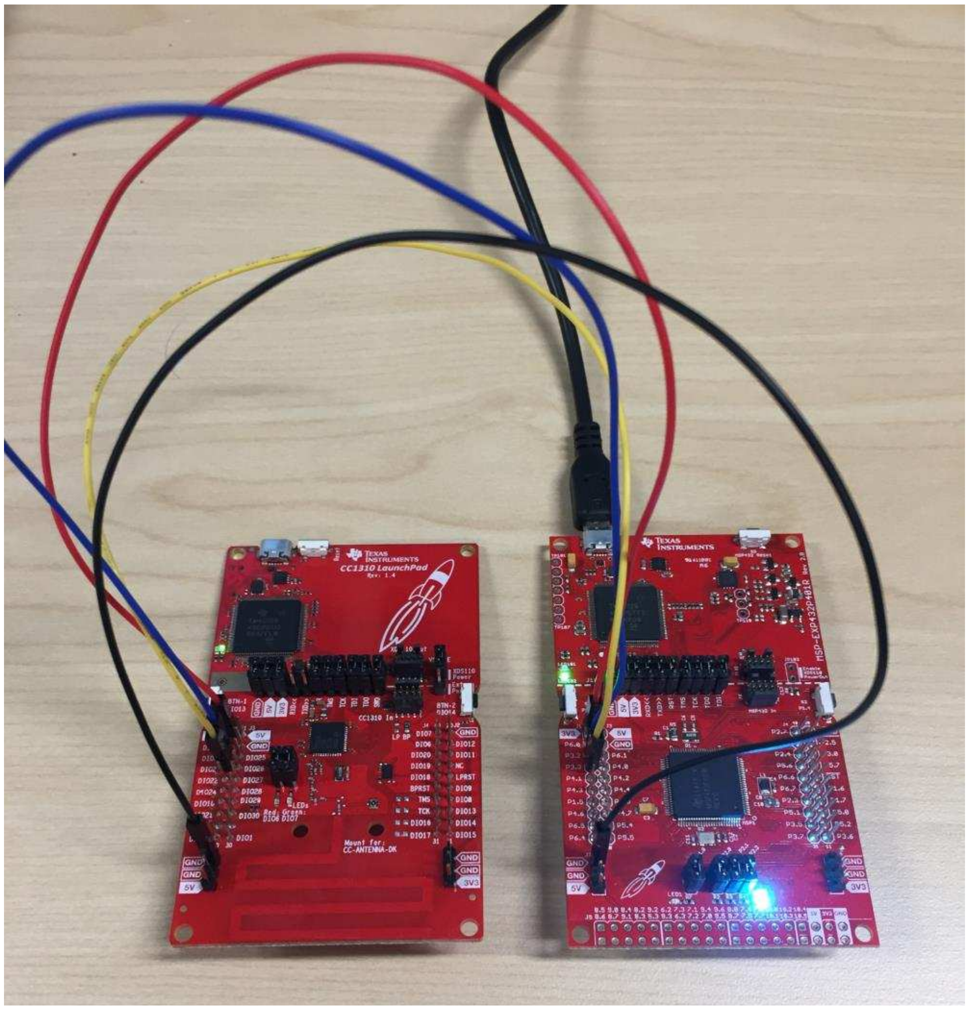


Figure 12. Wiring-Based Hardware Configuration

Table 2. Pin Mapping Between Two EVMs

EVM	INPUT VOLTAGE	GROUND	UART	
LAUNCHXL-CC1310	3V3	GND	DIO2 (RX)	DIO3 (TX)
MSP-EXP432P401R	3V3	GND	P3.3 (TX)	P3.2 (RX)

To verify the communication range between the plug-in based and the wiring-based hardware configuration, Figure 13 shows an experiment with two nodes performed in an indoor environment. The experiment ran two nodes, one data collector and one end-node, with a 100B data transfer in each direction. In Figure 13:

- The black dot denotes the end-node location.
- The red dot shows the location of the data collector with the plug-in based hardware configuration.
- The blue dot shows the location of the data collector with the wiring-based hardware configuration.

For each experiment for the range measurement, the hardware configuration of the end node is changed to match to the hardware configuration of the data collector. The experimental results show that the hardware configuration has an impact on the maximum range and the wiring-based hardware configuration achieved better range compared to the plug-in based hardware configuration. In the experiment, the plug-in based hardware configuration achieved 230 m while the wiring-based hardware configuration achieved 300 m.

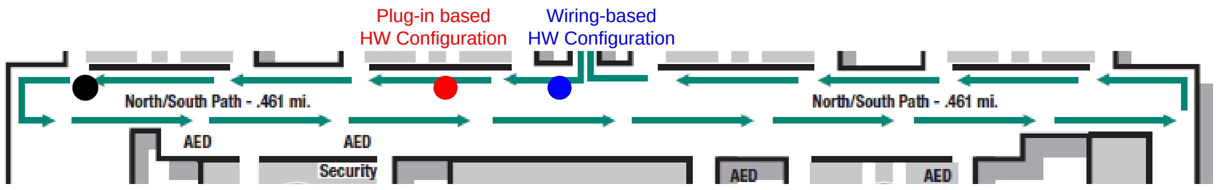


Figure 13. Communication Range Measurement With Two-Node Setup

3.1.2 Software

3.1.2.1 Getting Started

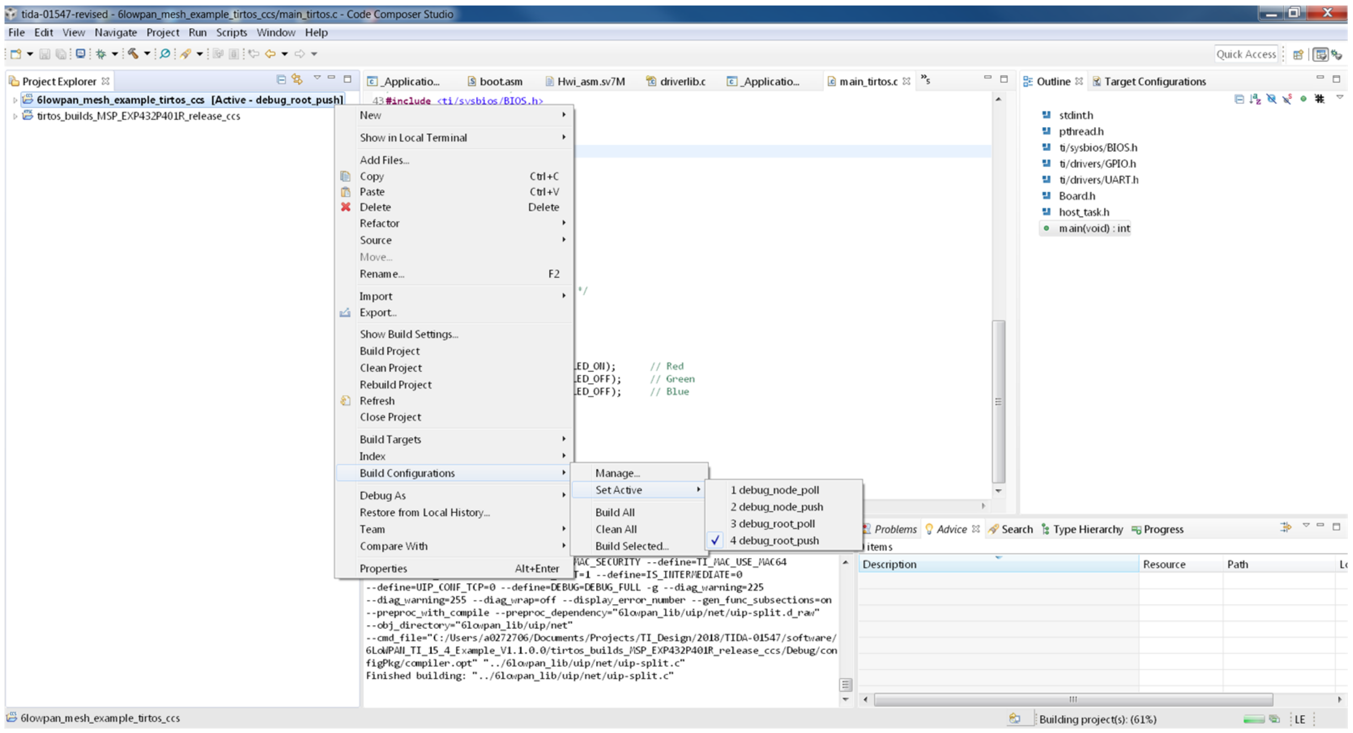
There are two software binaries to build the reference design: one for the CC1310 and one for the MSP432. The TI-15.4 software loaded on the CC1310 can be obtained from the SimpleLink CC13x0 SDK v2.30.0.20. This design uses the TI-15.4 coprocessor firmware (coprocessor_cc1310_lp.hex) without modification. Different from the sensor and collector examples provided with the SimpleLink CC13x0 SDK, the coprocessor example provides the UART host interface along with the TI 15.4-stack, which allows the TI 15.4-stack to communicate with an external device using UART.

The pre-built binary file (coprocessor_cc1310_lp.hex) can be found in C:\TI\simplelink_cc13x0_sdk_2_30_00_20\examples\rtos\CC1310_LAUNCHXL\ti154stack\hexfiles\.

The software example provided with the reference design runs on the MSP432 MCU to support 6LoWPAN, RPL routing, IPv6/ICMPv6, UDP, and applications. The prerequisite tools to build the software example is Code Composer Studio™ (CCS) v8 (or above) with the latest software update (by selecting Help → Check for Updates in the CCS menu) and SimpleLink MSP432P4 SDK v2.40.0.10.

Figure 14 shows the software example that provides four build configuration options to support two types of UDP applications: UDP poll example (end-node and root) and UDP push example (end node and root).

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 but it increases collision probability among end nodes in dense networks.



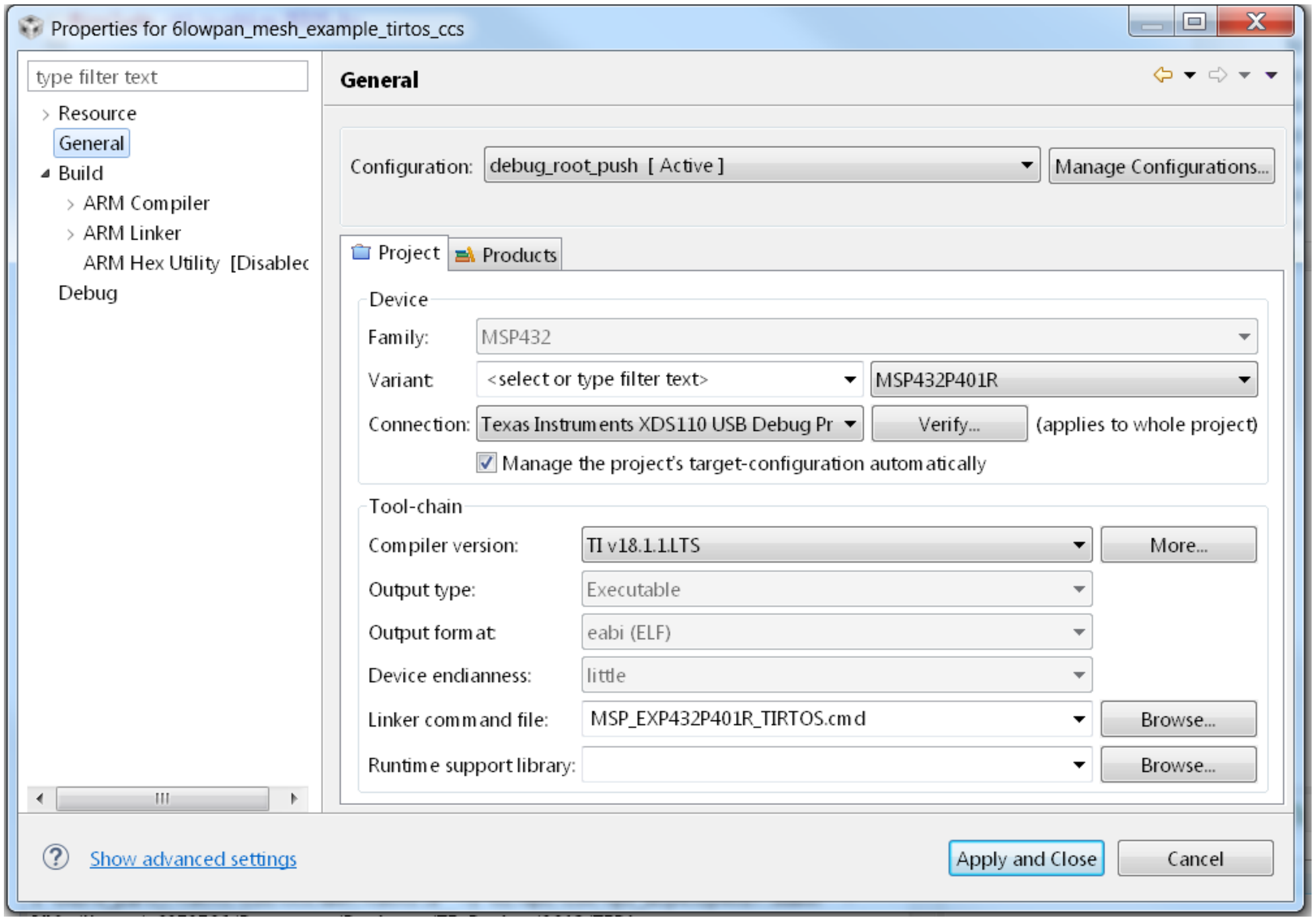
```
43 #include <ti/sysbios/BIOS.h>
```

```
LED_ON);      // Red
LED_OFF);     // Green
LED_OFF);     // Blue
```

```
MAC_SECURITY --define=TI_MAC_USE_MAC64
T=1 --define=IS_INTERMEDIATE=0
--define=UIP_CONF_TCP=0 --define=DEBUG=DEBUG_FULL -g --diag_warning=225
--diag_warning=255 --diag_wrap=off --display_error_number --gen_func_subsections=on
--preproc_with_compile --preproc_dependency="6lowpan_lib/uip/net/uip-split.d_raw"
--obj_directory="6lowpan_lib/uip/net"
--cmd_file="C:/Users/a0272706/Documents/Projects/TI_Design/2018/TIDA-01547/software/
6LoWPAN_TI_15_4_Example_V1.1.0.0/tirtos_builds_MSP_EXP432P401R_release_ccs/Debug/con
figPkg/compiler.opt" "../6lowpan_lib/uip/net/uip-split.c"
Finished building: "../6lowpan_lib/uip/net/uip-split.c"
```

Figure 14. 6LoWPAN_TI_15_4_Example Build Configurations

Figure 15 and Figure 16 show CCS captures to show the CCS compiler and MSP432 SDK versions used to build the software example.


Properties for 6lowpan_mesh_example_tirtos_ccs

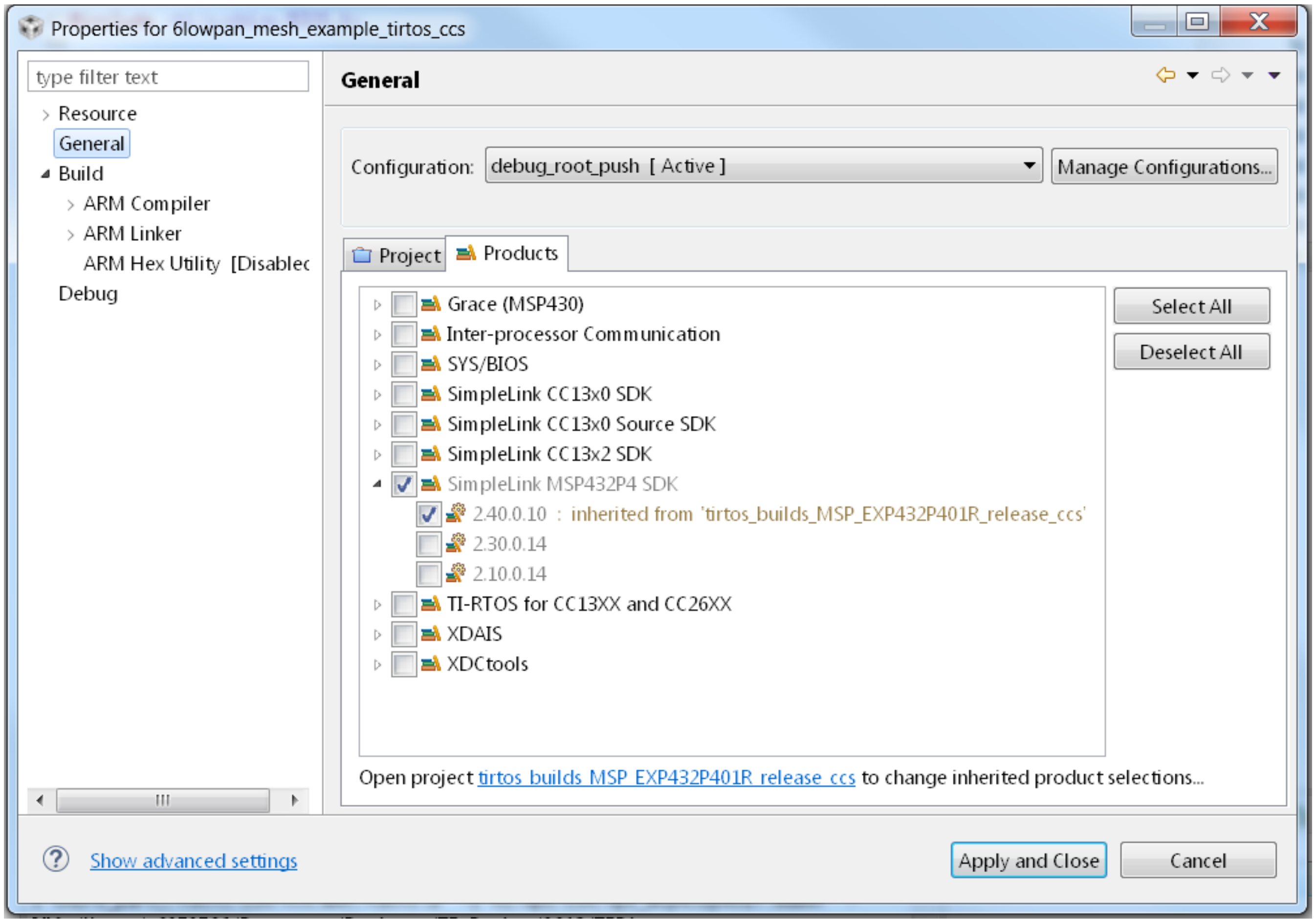
General

Configuration: debug_root_push [Active]

Device
Family: MSP432
Variant: <select or type filter text> MSP432P401R
Connection: Texas Instruments XDS110 USB Debug Pr (applies to whole project)
☑ Manage the project's target-configuration automatically

Tool-chain
Compiler version: TI v18.1.1.LTS
Output type: Executable
Output format: eabi (ELF)
Device endianness: little
Linker command file: MSP_EXP432P401R_TIRTOS.cmd
Runtime support library:

Figure 15. CCS™ Compiler Version



Configuration: debug_root_push [Active]

- [ ] Grace (MSP430)
- [ ] Inter-processor Communication
- [ ] SYS/BIOS
- [ ] SimpleLink CC13x0 SDK
- [ ] SimpleLink CC13x0 Source SDK
- [ ] SimpleLink CC13x2 SDK
- [✓] SimpleLink MSP432P4 SDK
  - [✓] 2.40.0.10 : inherited from 'tirtos_builds_MSP_EXP432P401R_release_ccs'
  - [ ] 2.30.0.14
  - [ ] 2.10.0.14
- [ ] TI-RTOS for CC13XX and CC26XX
- [ ] XDAIS
- [ ] XDCtools

Open project tirtos_builds_MSP_EXP432P401R_release_ccs to change inherited product selections...

Figure 16. CCS™ Products Property

3.1.2.2 *6LoWPAN_TI_15_4_Example*

This section starts with software overview, followed by details of the software architecture, useful tips to debugging and optimizing the software, and flashing the example projects.

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 are implemented based on CONTIKI open source.

Figure 17 and Figure 18 show the example software state machine for the data collector and end node, respectively. The data collector starts with Host_deviceStates_init to initialize the TI 15.4-stack on the CC1310 and then move to Host_deviceStates_joined that is ready state to support end-nodes in the network. The end node has two intermediate states of Host_deviceStates_fh_sync and Host_deviceStates_PA_done where the node starts FH synchronization. Once the node find a tracking FH parent, it moves to Host_deviceStates_joined that is ready state to start RPL route discovery. The LED changes accordingly to indicate the state change without the CCS debugger.

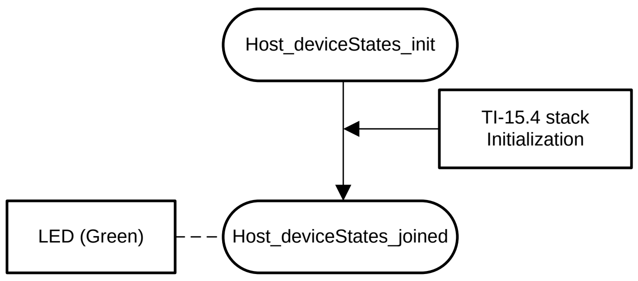


Figure 17. Software State Machine (Data Collector)

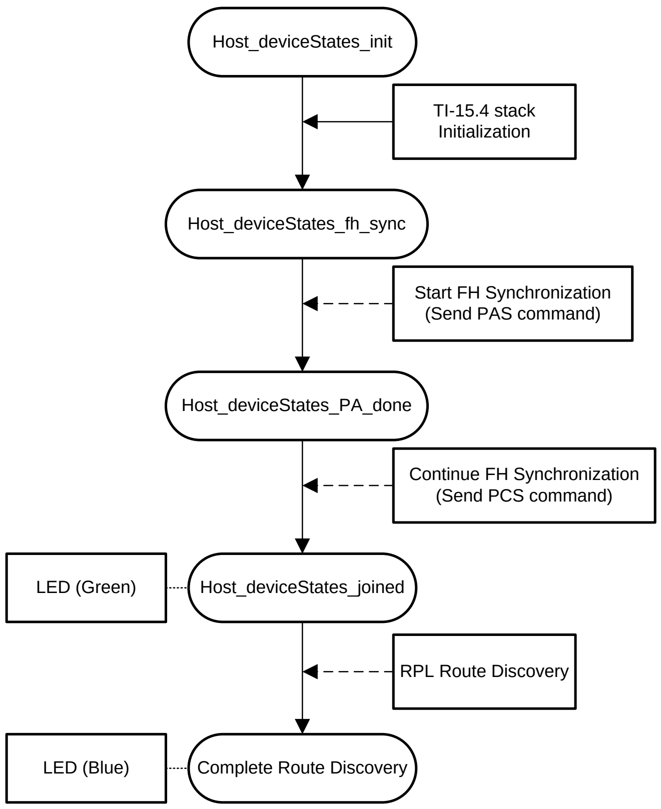

Host_deviceStates_init
TI-15.4 stack Initialization
Host_deviceStates_fh_sync
Start FH Synchronization (Send PAS command)
Host_deviceStates_PA_done
Continue FH Synchronization (Send PCS command)
LED (Green)
Host_deviceStates_joined
RPL Route Discovery
LED (Blue)
Complete Route Discovery

Figure 18. Software State Machine (End-Node)

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 that can be candidates of tracking parents by performing active scan. Once powered up, 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 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 synchronized with the FH network. When the end nodes receive multiple PAs during the scan period (DISC_SCAN_INTERVAL), they choose one of them based on the route cost and then update unicast FH timing and schedule, the tracking FH parent, and the PAN information.

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 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 by data packets that contains 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 3 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 /6lowpan_lib/host-api/host_config.h.

Table 3. Trickle Algorithm Parameters

PARAMETER	VALUE	DESCRIPTION
DISC_IMIN	6 seconds	Discovery trickle timer I_{min} parameter for PAS and PCS
DISC_MAX_COUNT	3	Discovery trickle timer exponent for PAS and PCS
DISC_SCAN_INTERVAL	20 seconds	Discovery trickle timer 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 failure conditions such as data transmission and reception failures. The only difference of the keep-alive mechanism between the UDP polling and push examples is whether the keep-alive frame is used or not. Because the UDP polling application does not allow end nodes to initiate data transmissions, the UDP polling example introduces a new keep-alive frame while the UDP push example does not need the frame.

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_INTERVAL) once end nodes join to the FH network. To reduce the keep-alive traffic overheads, the keep-alive TX timer is reset when broadcast frames other than keep-alive frames are sent.

If end nodes does not receive broadcast frames from their tracking parents in series (MAXIMUM_NUM_FAIL_RX), it is supposed that FH sync loss occurs and the end nodes move immediately to the FH_SYNC state to restart the FH synchronization process.

In addition to the keep-alive transmissions, each node monitors unicast transmission failures to the target parents. If it occurs in series (MAX_RETRY_CNT), this indicates the FH sync loss, which results in moving to the FH_SYNC state to re-start the FH synchronization process. Table 4 summarizes the keep-alive parameters.

Table 4. Keep-Alive Parameters

PARAMETER	VALUE	DESCRIPTION	COMMENTS
KEEP_ALIVE_TX_INTERVAL	60 seconds	Keep-alive TX interval for parents	UDP polling example only
MAXIMUM_NUM_FAIL_RX	5 packets	The maximum number of failed keep-alive RX in series to indicate FH sync loss	UDP polling example only
MAX_RETRY_CNT	5 packets	The maximum number of TX failure to indicate FH sync loss	UDP polling example or UDP push example

3.1.2.2.1.3 *MAC Data Encryption*

The MAC-level data encryption is enabled with "FEATURE_MAC_SECURITY" macro defined in the Predefined Symbols menu in the property of the CCS project. Per TI 15.4-stack, the MAC data encryption follows IEEE 802.15.4 standard and supports the security level of MIC-32, MIC-64, MIC-128, ENC, ENC-MIC-32, ENC-MIC-64, and ENC-MIC-128.

In the 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 6lowpan_lib/host-api/host_api.c. The TI-15.4 supports pre-shared key mechanisms. The security key must be pre-programmed in the software, and all the nodes in the same network must share the same 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.

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 DODAG information object (DIO) by the root node. Once child nodes receive the DIOs, they broadcast the DIOs and send back unicast destination advertisement object (DAO) packet to the parents that provides the best route to 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 6LoWPAN protocol reduces the technical 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/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_TI_15_4_Example software consists of three major stacks: the host interface layer, 6LoWPAN mesh stacks, and the application layer. The host interface layer initializes and configures the TI15.4-stack and handles incoming and outgoing data from and to the TI15.4-stack through UART. 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 19 shows the software architecture. The host interface layer consists of two tasks: host_tx_task and host_rx_task. These tasks are the UART interface to the TI 15.4-stack in the CC1310 MCU. The tcpip_task based on CONTIKI open source covers the network layers from 6LoWPAN to UDP layer. The UDP poll and push examples are parts of the app_task as the application layer.

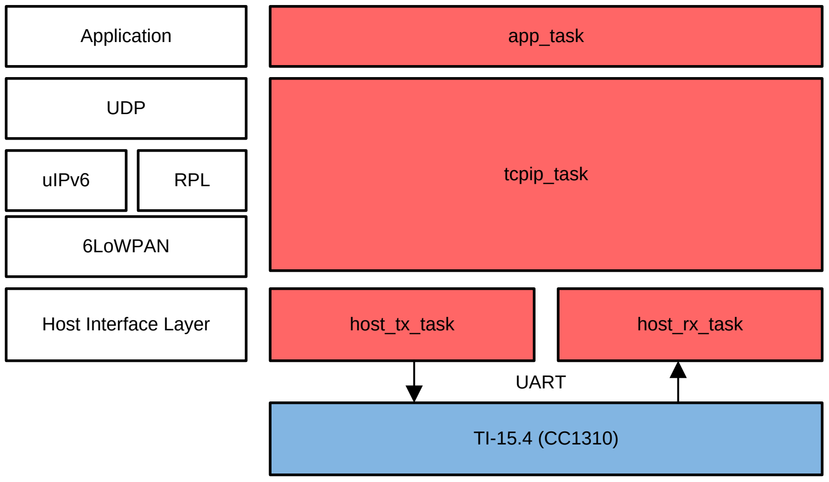


Figure 19. Software Architecture of TIDA-01547

3.1.2.2.2.1 TI 15.4-Stack PHY Configuration

The TI15.4-stack supports multiple options for the frequency band and mode to run the FH. The software example configure the PHY mode to APIMAC_STD_US_915_PHY_1, which is the 50kbps FSK over a 902-MHz frequency band with 129 channels. The default configuration can be updated in host_config.h. The ti-15.4-stack-users-guide (in C:\TI\simplelink_cc13x0_sdk_2_30_00_20\docs\ti154stack), parts of the SimpleLink CC13x0 SDK, provides the details of the PHY operation modes. The following codes list all the options for the PHY ID, which can be found in the 6lowpan_lib/host-api/api_mac.h.

NOTE: The 6LoWPAN_TI_15_4_Example was verified only 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 Host Interface Layer

The host interface layer is the UART interface with the TI 15.4-stack in the CC1310 MCU. The software example configured the Board_UART1 (P3.2 for UART RX and P3.3 for UART TX) with the baud rate of 115,200 bps.

3.1.2.2.2.2.1 host_tx_task

The host_tx_task takes responsible of initializing the TI 15.4-stack, maintaining MAC-level keep-alive mechanism, registering security entries to the TI 15.4-stack, sending packets generated by the upper layers to the TI 15.4-stack, handling message timeout and erroneous transmissions, maintaining the host state machine which can be one of Host_deviceStates_init,Host_deviceStates_fh_sync, Host_deviceStates_PA_done and Host_deviceStates_joined, and performing FH synchronization.

Once powered up, a device starts with Host_deviceStates_init state to reset the TI-15.4 device, to configure MAC PIB and FH PIBs, and to start the PAN as root or end-node. The host state changes to Host_deviceStates_fh_sync when the device starts the FH synchronization process. After completing PAS and PA exchanges, the first step of FH synchronization, the device jumps to Host_deviceStates_PA_done state. After the FH synchronization is completed, the device moves to Host_deviceStates_joined state. At this stage, the task notifies the upper layer to indicate ready-to-send data.

The root node does not require the FH synchronization process. Once the root node completes initialization, the root node jumps to Host_deviceStates_joined state.

The task communicates with the other internal tasks through mailbox (host_mailbox). The mailbox is configured to hold maximum 10 messages. The host messages sent to the TI15.4-stack is formatted based on the ti-15.4-stack-cop-interface-guide document found in C:\TI\simplelink_cc13x0_sdk_2_30_00_20\docs\ti154stack.

3.1.2.2.2.2.2 host_rx_task

The host_rx_task processes incoming UART messages from the TI 15.4-stack. The task first reads 4-byte, which is the minimum of the required host message header including 1B sync, 1B Length, 1B cmd0, and 1B cmd1. Once the task detects the header, it extracts data portion based on the Length information in the header. Depending on the subsystem code (cmd0), the task routes the received stream to one of the three modules: mt_mac (MAC interface), mt_util (UTIL interface), and mt_sys (SYS interface).

The details on the message formatting can be found in the ti-15.4-stack-cop-interface-guide document in C:\TI\simplelink_cc13x0_sdk_2_30_00_20\docs\ti154stack.

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

The network stacks are based on the CONTIKI open source. The tcpip_task (in uip_rpl_task.c) processes messages incoming from the application and host interface layers through mailbox. For details of the implementations, see 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 server for the root or 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 from the node. For the UDP push example, each node initiates data transmissions whenever there is data available in its queue.

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 UDP applications can be easily integrated with the given software examples.

3.1.2.2.2.5 LED Configuration

Table 5 summarizes the LED configuration in the software example. The Board_LED2 turns on when the host state changes to Host_deviceStates_joined. The Board_LED3 turns on when end nodes join to the RPL network. For the root node, the Board_LED3 is not used.

Table 5. LED Configuration (MSP432P401R)

NAME (PIN NUMBER)	EVENT	ACTION
Board_LED0 (P1.0)	UART TX/RX	Toggle (Red)
Board_LED2 (P2.1)	Completed FH SYNC	ON (Green)
Board_LED3 (P2.2)	Joined the RPL Network (only for end-nodes)	ON (Blue)

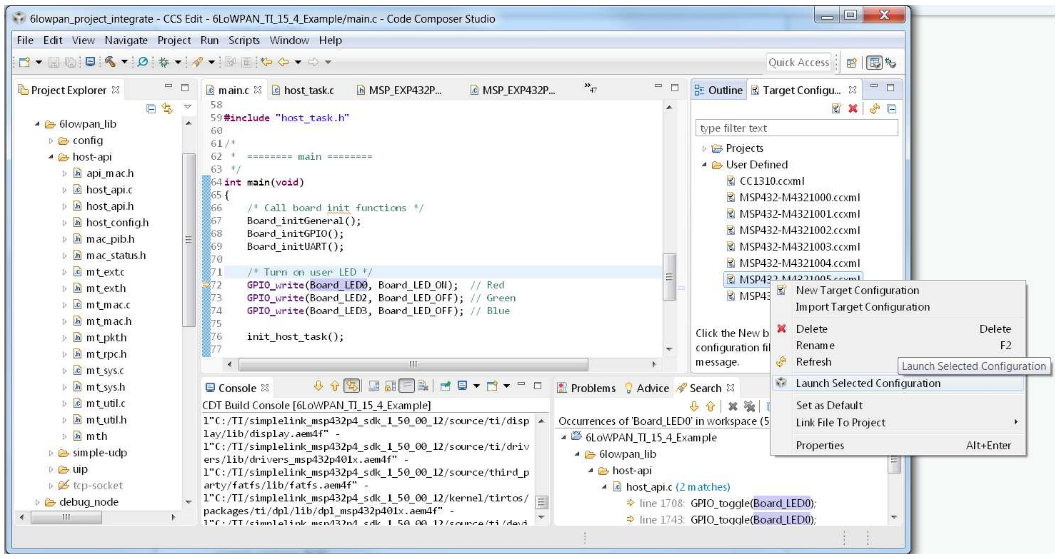
3.1.2.2.3 Tips for Debugging and Optimization

This section provides useful tips with the CCS 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 CCS tool provides the debug mode operation. To run in debug mode with the CCS tool:

1. Launch the target configuration for the target EVM or device (see Figure 20).
2. Connect the target EVM, load the program or symbol, and then run.
3. Add global variables to debug in the Expressions tab to trace the variables in the debug mode.


6lowpan_project_integrate - CCS Edit - 6LoWPAN_TI_15_4_Example/main.c - Code Composer Studio
File Edit View Navigate Project Run Scripts Window Help
Project Explorer
main.c
host_task.c
Outline
Target Configu...
New Target Configuration
Import Target Configuration
Delete
Rename
Refresh
Launch Selected Configuration
Set as Default
Link File To Project
Properties
Console
Problems
Advice
Search

```
#include "host_task.h"

/*
 * ======== main ========
 */
int main(void)
{
    /* Call board init functions */
    Board_initGeneral();
    Board_initGPIO();
    Board_initUART();

    /* Turn on user LED */
    GPIO_write(Board_LED0, Board_LED_ON);  // Red
    GPIO_write(Board_LED2, Board_LED_OFF); // Green
    GPIO_write(Board_LED3, Board_LED_OFF); // Blue

    init_host_task();
```

```
CDT Build Console [6LoWPAN_TI_15_4_Example]
l"C:/TI/simplelink_msp432p4_sdk_1_50_00_12/source/ti/display/lib/display.aem4f" -
l"C:/TI/simplelink_msp432p4_sdk_1_50_00_12/source/ti/drivers/lib/drivers_msp432p401x.aem4f" -
l"C:/TI/simplelink_msp432p4_sdk_1_50_00_12/source/third_party/fatfs/lib/fatfs.aem4f" -
l"C:/TI/simplelink_msp432p4_sdk_1_50_00_12/kernel/tirtos/packages/ti/dpl/lib/dpl_msp432p401x.aem4f" -
```

```
Occurrences of 'Board_LED0' in workspace
line 1708: GPIO_toggle(Board_LED0);
line 1743: GPIO_toggle(Board_LED0);
```

Figure 20. CCS™ Debug: Launching Debug Window

Table 6 provides some global variables useful to debug network and host interface layers.

Table 6. Global Variables for Debugging

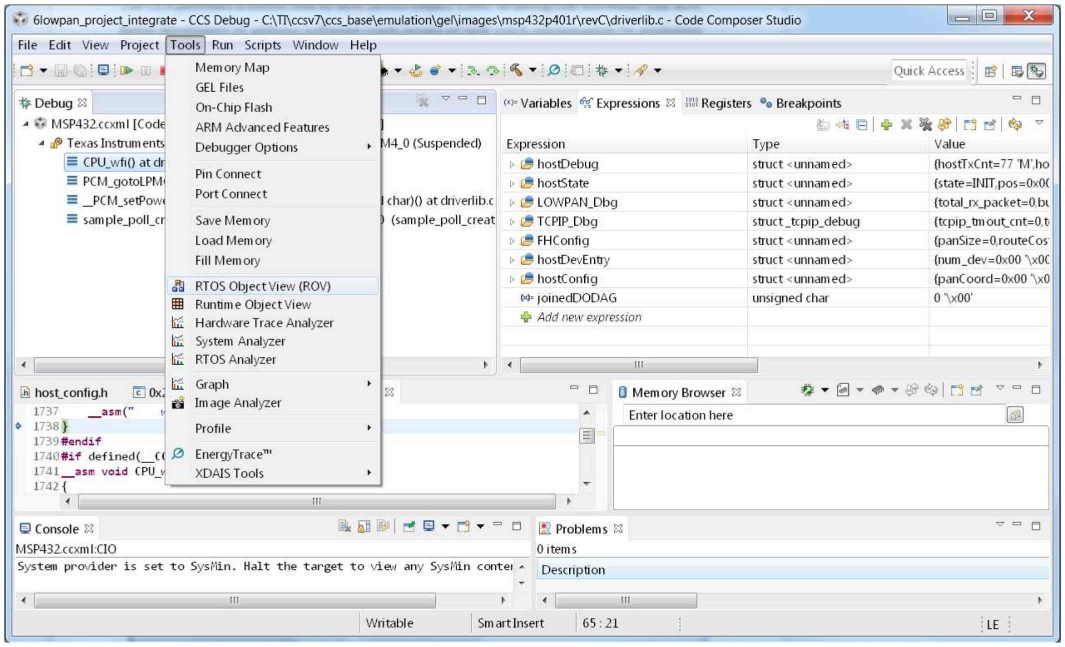
GLOBAL VARIABLES	DESCRIPTION
hostDebug	Debug counts for host interface layer
TCPIP_Dbg	Debug counts for UDP/IPv6/ICMPv6/RPL layers
LOWPAN_Dbg	Debug counts for 6LoWPAN
hostDevInfoBlock	Host configuration parameters
joinedDODAG	Flag to indicate DODAG join state

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 developers address software crash issues or optimize task stack by analyzing stackPeak. To debug software with the ROV tool:

1. Suspend debug mode and open ROV as shown in Figure 21.

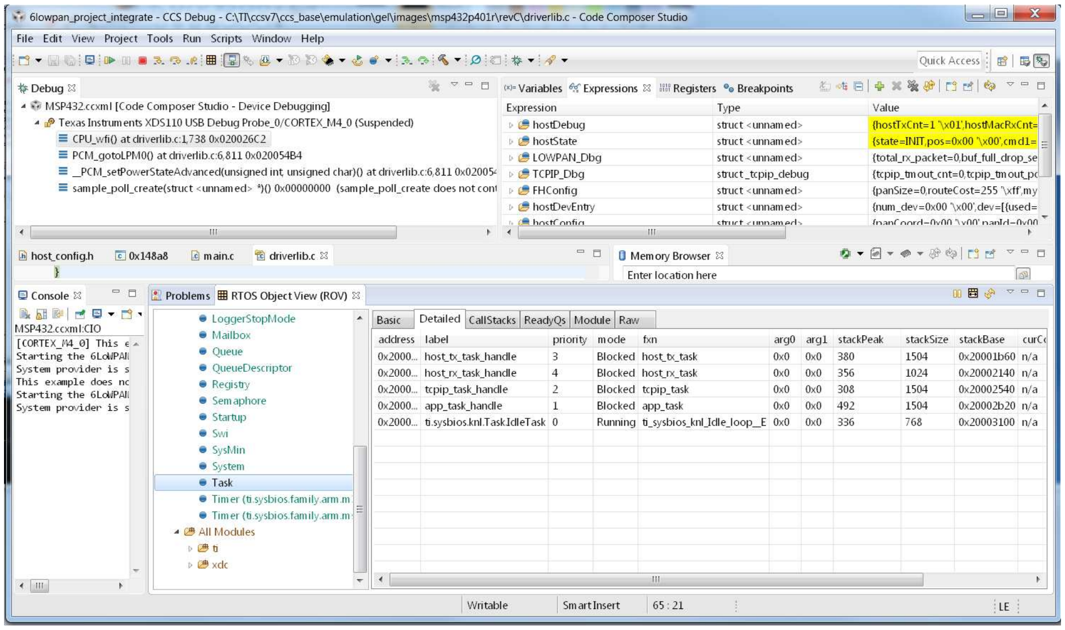
   Figure 22 shows the screen capture of ROV (on the right-bottom).
2. As an example, use the "Detailed" and "CallStacks" taps in the "Task" menu to optimize the stack size, to trace the call stacks, or to address the stack overflow.



Expression	Type	Value
hostDebug	struct <unnamed>	{hostTxCnt=77 'M',ho
hostState	struct <unnamed>	{state=INIT,pos=0x0(
LOWPAN_Dbg	struct <unnamed>	{total_rx_packet=0,bu
TCPIP_Dbg	struct _tcpip_debug	{tcpip_tmout_cnt=0,t
FHConfig	struct <unnamed>	{panSize=0,routeCos
hostDevEntry	struct <unnamed>	{num_dev=0x00 '\x0(
hostConfig	struct <unnamed>	{panCoord=0x00 '\x0
joinedDODAG	unsigned char	0 '\x00'
Add new expression		

```
MSP432.ccxml:CIO
System provider is set to SysMin. Halt the target to view any SysMin conte
```

Figure 21. CCS™ Debug: Launch ROV


6lowpan_project_integrate - CCS Debug - C:\TI\ccsv7\ccs_base\emulation\gel\images\msp432p401r\revC\driverlib.c - Code Composer Studio
File Edit View Project Tools Run Scripts Window Help
Quick Access
Debug
Variables Expressions Registers Breakpoints
host_config.h 0x148a8 main.c driverlib.c
Memory Browser
Enter location here
Console
Problems RTOS Object View (ROV)
LoggerStopMode
Mailbox
Queue
QueueDescriptor
Registry
Semaphore
Startup
Swi
SysMin
System
Task
Timer (ti.sysbios.family.arm.m
Timer (ti.sysbios.family.arm.m
All Modules
ti
xdc
Basic Detailed CallStacks ReadyQs Module Raw
Writable
Smart Insert
65 : 21
LE

```
MSP432.ccxml [Code Composer Studio - Device Debugging]
  Texas Instruments XDS110 USB Debug Probe_0/CORTEX_M4_0 (Suspended)
    CPU_wfi() at driverlib.c:1,738 0x020026C2
    PCM_gotoLPM0() at driverlib.c:6,811 0x020054B4
    __PCM_setPowerStateAdvanced(unsigned int, unsigned char)() at driverlib.c:6,811 0x02005
    sample_poll_create(struct <unnamed> *)() 0x00000000 (sample_poll_create does not cont
```

Expression	Type	Value
hostDebug	struct <unnamed>	{hostTxCnt=1 '\x01',hostMacRxCnt=
hostState	struct <unnamed>	{state=INIT,pos=0x00 '\x00',cmd1=
LOWPAN_Dbg	struct <unnamed>	{total_rx_packet=0,buf_full_drop_se
TCPIP_Dbg	struct _tcpip_debug	{tcpip_tmout_cnt=0,tcpip_tmout_pc
FHConfig	struct <unnamed>	{panSize=0,routeCost=255 '\xff',my
hostDevEntry	struct <unnamed>	{num_dev=0x00 '\x00',dev=[{used=
hostConfig	struct <unnamed>	{panCoord=0x00 '\x00',panId=0x00

```
MSP432.ccxml:CIO
[CORTEX_M4_0] This e
Starting the 6LoWPA
System provider is s
This example does nc
Starting the 6LoWPA
System provider is s
```

address	label	priority	mode	fxn	arg0	arg1	stackPeak	stackSize	stackBase	curC
0x2000...	host_tx_task_handle	3	Blocked	host_tx_task	0x0	0x0	380	1504	0x20001b60	n/a
0x2000...	host_rx_task_handle	4	Blocked	host_rx_task	0x0	0x0	356	1024	0x20002140	n/a
0x2000...	tcpip_task_handle	2	Blocked	tcpip_task	0x0	0x0	308	1504	0x20002540	n/a
0x2000...	app_task_handle	1	Blocked	app_task	0x0	0x0	492	1504	0x20002b20	n/a
0x2000...	ti.sysbios.knl.Task.IdleTask	0	Running	ti_sysbios_knl_Idle_loop__E	0x0	0x0	336	768	0x20003100	n/a

Figure 22. CCS Debug: Debug Software With ROV

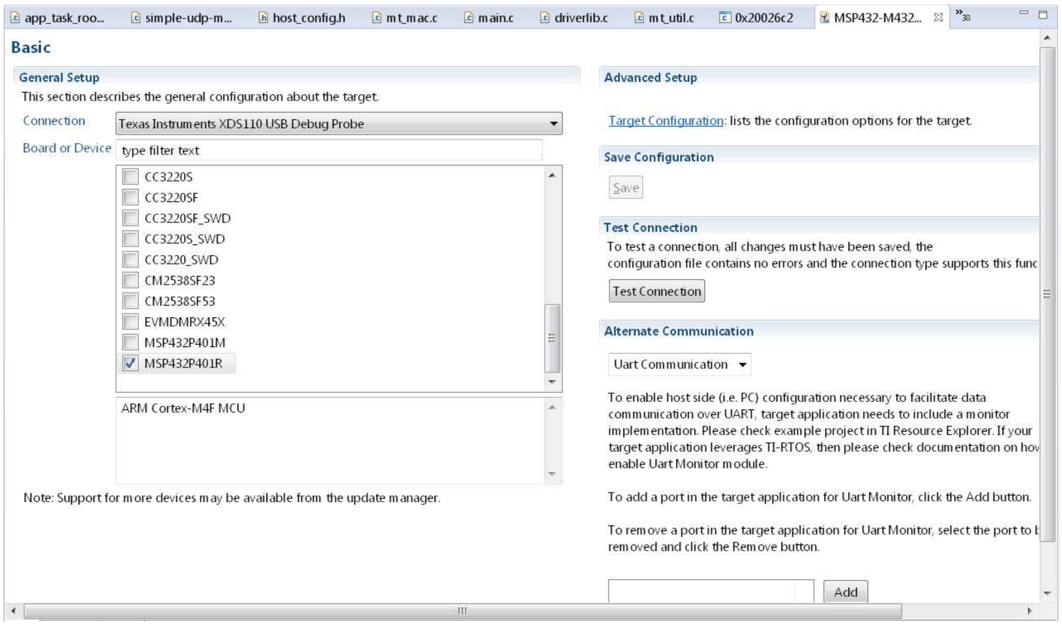
3.1.2.2.4 Flashing the Software Example

To run this reference design, there are two firmware to be flashed: one for the CC1310 MCU and one for the MSP432 MCU. This section provides step-by-step instructions to flash the software.

3.1.2.2.4.1 Flash the 6LoWPAN_TI_15_4_Example on MSP432™ Using CCS™

The output binary for the 6LoWPAN_TI_15_4_Example project is in form of an .out file. To flash the .out file on the MSP432 MCU with the CCS tool:

1. Build the 6LoWPAN_TI_15_4_Example project.
2. Create the CCS Target Configuration for the MSP432 platform (see Figure 23).
3. Launch the target configuration (see Figure 24).
4. Connect the target device and flash the .out binary.



Basic

General Setup

This section describes the general configuration about the target.

Connection: Texas Instruments XDS110 USB Debug Probe

Board or Device: type filter text

- [ ] CC3220S
- [ ] CC3220SF
- [ ] CC3220SF_SWD
- [ ] CC3220S_SWD
- [ ] CC3220_SWD
- [ ] CM2538SF23
- [ ] CM2538SF53
- [ ] EVMDMRX45X
- [ ] MSP432P401M
- [✓] MSP432P401R

ARM Cortex-M4F MCU

Note: Support for more devices may be available from the update manager.

Advanced Setup

Target Configuration: lists the configuration options for the target.

Save Configuration

Test Connection

To test a connection, all changes must have been saved, the configuration file contains no errors and the connection type supports this func

Alternate Communication

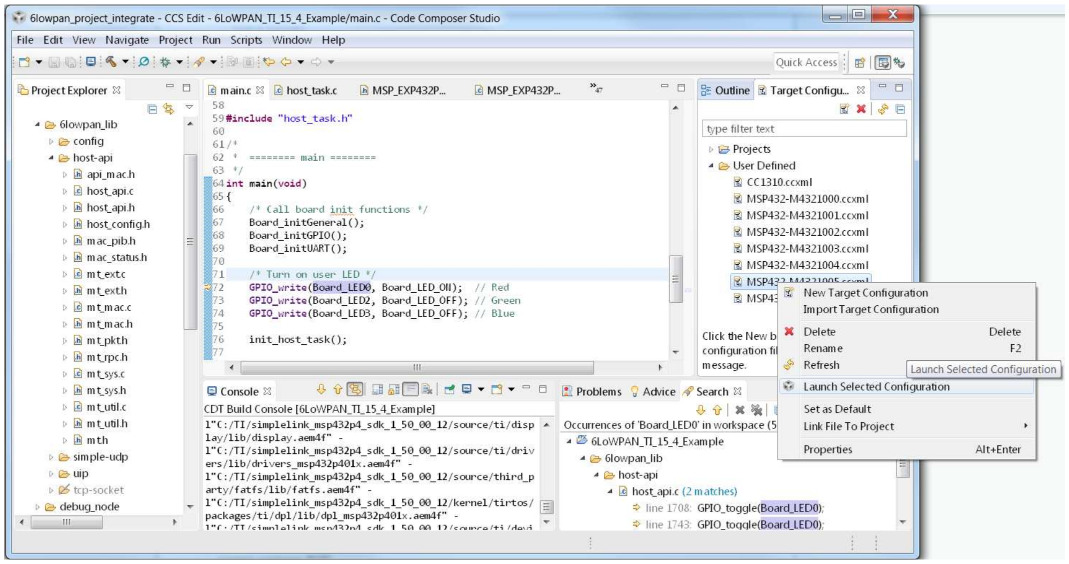
Uart Communication

To enable host side (i.e. PC) configuration necessary to facilitate data communication over UART, target application needs to include a monitor implementation. Please check example project in TI Resource Explorer. If your target application leverages TI-RTOS, then please check documentation on hov enable Uart Monitor module.

To add a port in the target application for Uart Monitor, click the Add button.

To remove a port in the target application for Uart Monitor, select the port to b removed and click the Remove button.

Figure 23. CCS™ Target Configuration for MSP432™



6lowpan_project_integrate - CCS Edit - 6LoWPAN_TI_15_4_Example/main.c - Code Composer Studio

```
#include "host_task.h"

/*
 * ======== main ========
 */
int main(void)
{
    /* Call board init functions */
    Board_initGeneral();
    Board_initGPIO();
    Board_initUART();

    /* Turn on user LED */
    GPIO_write(Board_LED0, Board_LED_ON);  // Red
    GPIO_write(Board_LED2, Board_LED_OFF); // Green
    GPIO_write(Board_LED3, Board_LED_OFF); // Blue

    init_host_task();
```

CDT Build Console [6LoWPAN_TI_15_4_Example]

```
1"C:/TI/simplelink_msp432p4_sdk_1_50_00_12/source/ti/disp
lay/lib/display.aem4f" -
1"C:/TI/simplelink_msp432p4_sdk_1_50_00_12/source/ti/driv
ers/lib/drivers_msp432p401x.aem4f" -
1"C:/TI/simplelink_msp432p4_sdk_1_50_00_12/source/third_p
arty/fatfs/lib/fatfs.aem4f" -
1"C:/TI/simplelink_msp432p4_sdk_1_50_00_12/kernel/tirtos/
packages/ti/dpl/lib/dpl_msp432p401x.aem4f" -
```

Occurrences of 'Board_LED0' in workspace (5

- 6LoWPAN_TI_15_4_Example
  - 6lowpan_lib
    - host-api
      - host_api.c (2 matches)
        - line 1708: GPIO_toggle(Board_LED0);
        - line 1743: GPIO_toggle(Board_LED0);

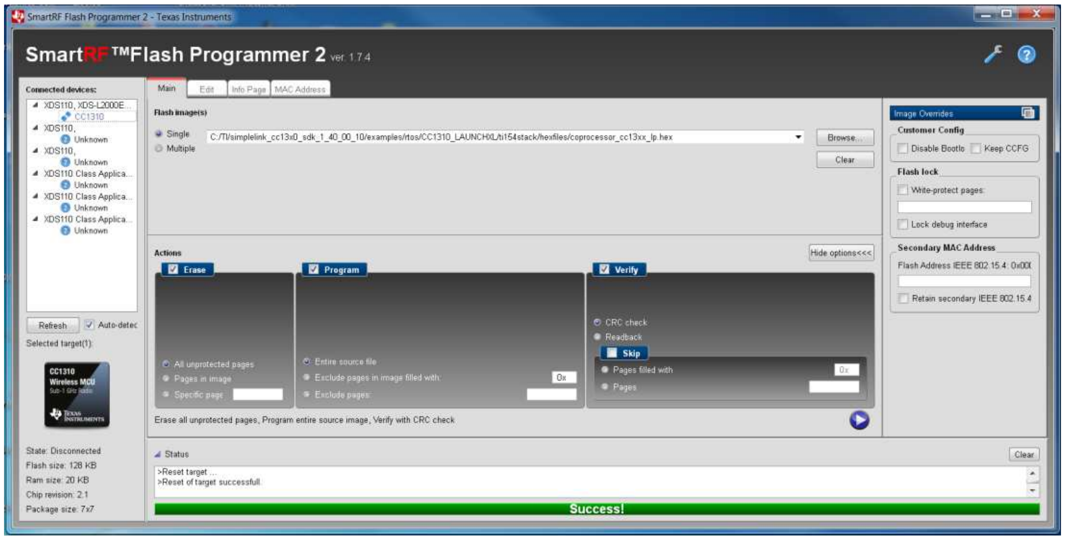
Click the New b configuration fil message.

Figure 24. Launch CCS™ Target Configuration

3.1.2.2.4.2 Flash TI-15.4 Coprocessor Binary on CC1310

The pre-built binary provided with the SimpleLink CC13x0 SDK is in form of hex and the SmartRF™ Flash Programmer can be used to flash the hex file on the CC1310 MCU. Figure 25 shows the screen capture of flashing the firmware with the SmartRF Flash Programmer.

1. Open SmartRF Flash Programmer and select the target device.
2. Load the target TI-15.4 coprocessor hex (coprocessor_cc13x0_lp.hex) and flash the binary.
3. Make sure that when compile all the check boxes (Erase, Program, and Verify) are selected.


SmartRF Flash Programmer 2 - Texas Instruments
SmartRF™Flash Programmer 2 ver. 1.7.4
Main
Edit
Info Page
MAC Address
Browse...
Clear
Hide options<<<
Image Overrides
Refresh

Connected devices: XDS110, XDS-L2000E... CC1310; XDS110, Unknown; XDS110, Unknown; XDS110 Class Applica... Unknown; XDS110 Class Applica... Unknown; XDS110 Class Applica... Unknown

Auto-detec ✓

Selected target(1): CC1310 Wireless MCU

State: Disconnected
Flash size: 128 KB
Ram size: 20 KB
Chip revision: 2.1
Package size: 7x7

Flash image(s): [✓ Single] Multiple

C:/TI/simplelink_cc13x0_sdk_1_40_00_10/examples/rtos/CC1310_LAUNCHXL/ti154stack/hexfiles/coprocessor_cc13xx_lp.hex

Actions:

- Erase ✓: [✓ All unprotected pages] Pages in image; Specific page
- Program ✓: [✓ Entire source file] Exclude pages in image filled with: 0x; Exclude pages:
- Verify ✓: [✓ CRC check] Readback; Skip: Pages filled with 0x; Pages

Erase all unprotected pages, Program entire source image, Verify with CRC check

Customer Config: Disable Bootlo; Keep CCFG
Flash lock: Write-protect pages:; Lock debug interface
Secondary MAC Address: Flash Address IEEE 802.15.4: 0x00[illegible]; Retain secondary IEEE 802.15.4

Status

```
>Reset target ...
>Reset of target successfull
```

Success!

Figure 25. SmartRF™ Flash Programmer (for TI-15.4 Firmware on CC1310)

3.2 Testing and Results

3.2.1 Test Setup

Table 7 summarizes the CCS build configuration to set up test nodes. For the network testing, both data collector and end node use the same hardware platform as shown in Section 3.1.1.1.

Table 7. Firmware for Test Setup

DEVICE	DATA COLLECTOR	END NODE	COMMENTS
CC1310	coprocessor_cc1310_lp.hex	coprocessor_cc1310_lp.hex	SimpleLink™ CC13x0 SDK v2.30.0.20
MSP432	debug_root_poll	debug_node_poll	UDP poll example (Section 3.2.2, Section 3.2.4)
	debug_root_push	debug_node_push	UDP push example (Section 3.2.3)

The experiments compare the RF mesh system performance with two different underlying MAC protocols: the FH MAC protocol using full 129 channels in the FCC band (FH-129) and non-FH MAC (non-FH).

For the FH-129, the TI 15.4-stack was configured to the FH mode activating all 129 channels in the FCC band. For the non-FH, the TI 15.4-stack is configured to the non-beacon mode over the channel #0 (902.2 MHz) in the FCC band.

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 is created with address filtering in the software. Each node has a list of entries where the node can accept the packet reception. There are two entries for each end node: one is the target parent and one is the target child.

```
//Address filtering for multi-hop linear topology testing
#define NUM_ENTRIES     2
//entry #0: target parent, entry #1: target child
uint8_t whitelist[NUM_ENTRIES][8]={
{0x1D, 0xC4, 0xA4, 0x13, 0x00, 0x4B, 0x12, 0x00},
{0xFD, 0xBB, 0xA4, 0x13, 0x00, 0x4B, 0x12, 0x00}                                 };
```

In the software, this feature can be enabled with "MULTIHOP_TESTING" macro defined in 6lopwan_lib/host-api/host_config.h.

NOTE: For testing setup, modify the list of entries defined in the 6lopwan_lib/host-api/mt_mac.c based on the extended address of the test node. For each node, the extended address set at the initial stage is stored in "hostDevInfoBlock.extAddr".

Figure 26 shows the six-hop network setup with a noise generator.

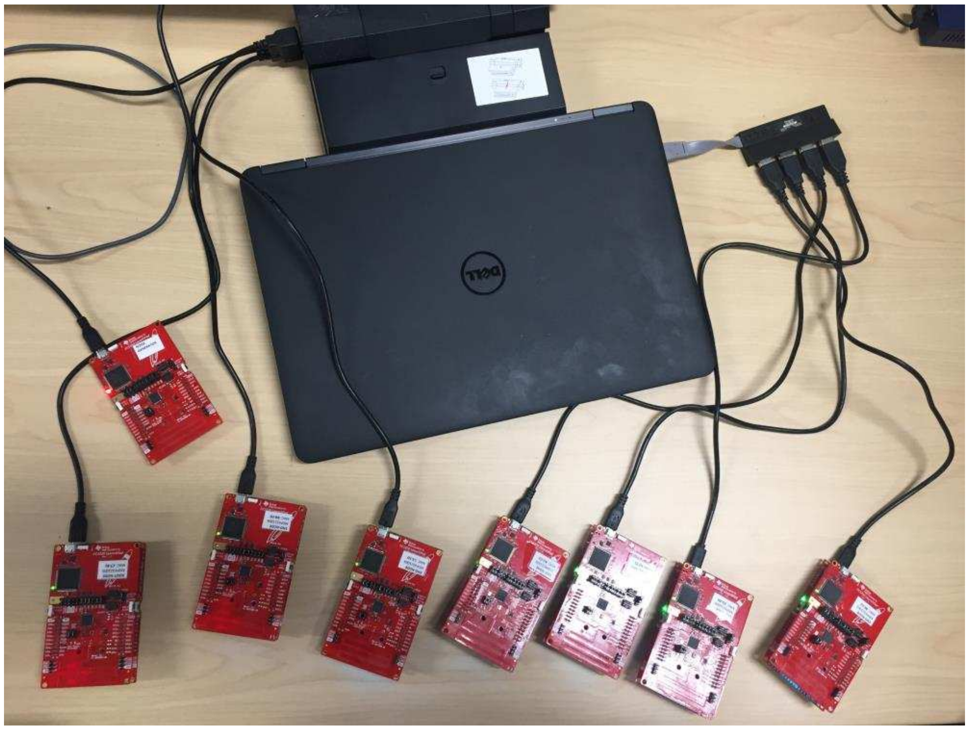


Figure 26. Multi-Hop Network Setup With Noise Generator

The multi-hop network with the address filtering technique has a disadvantage in using network resource. This disadvantage is because all nodes are within transmission range, which is different from real multi-hop networks. This range limits spatial reuse of the network resource. However, the multi-hop network in this setup has an advantage on the hidden node problem over real multi-hop networks because all nodes can be seen with each other.

NOTE: For some experiments to validate network performance as a function of data size, the software example was modified to support up to 1200B.

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 of the FH-129 and the non-FH systems. The external noise sources can be channel noise and signals emitted from adjacent sub-1 GHz communication systems.

For these experiments, the UDP poll example is used over the six-hop linear topology as shown in Figure 27. The data collector (or root node) sends poll messages every 10 seconds, and as a response, each node sends back the same size data packets to the data collector. In addition, to verify the impact of noise, background noise is generated to the network. For details of the noise setup, see Section 3.2.2.1.

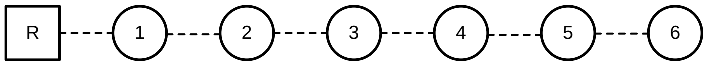
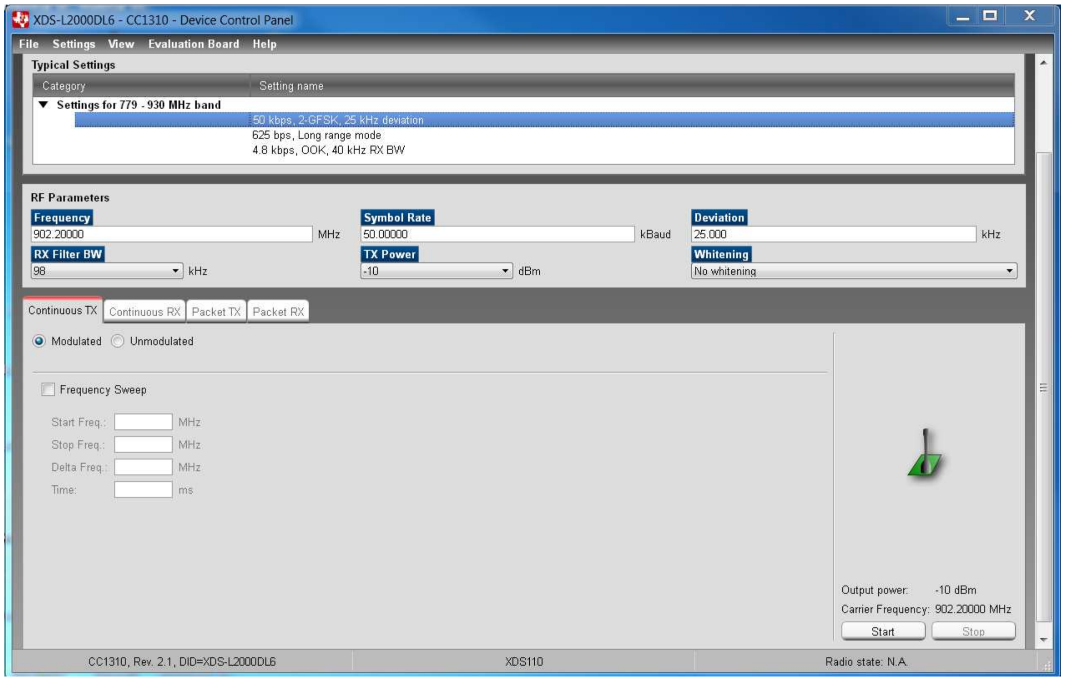


Figure 27. Six-Hop Linear Topology

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 28 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 29 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 8.


XDS-L2000DL6 - CC1310 - Device Control Panel

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: -10 dBm
Whitening: No whitening

Continuous TX: Modulated

Output power: -10 dBm
Carrier Frequency: 902.20000 MHz

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

Figure 28. Continuous Noise at 902.2-MHz Frequency (Channel 0 in FCC Band)

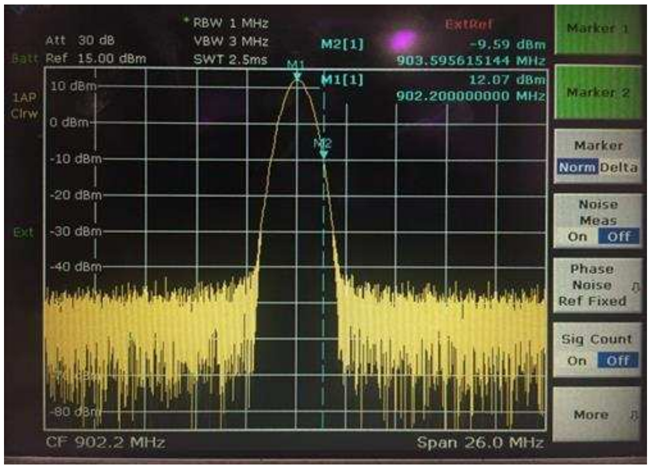
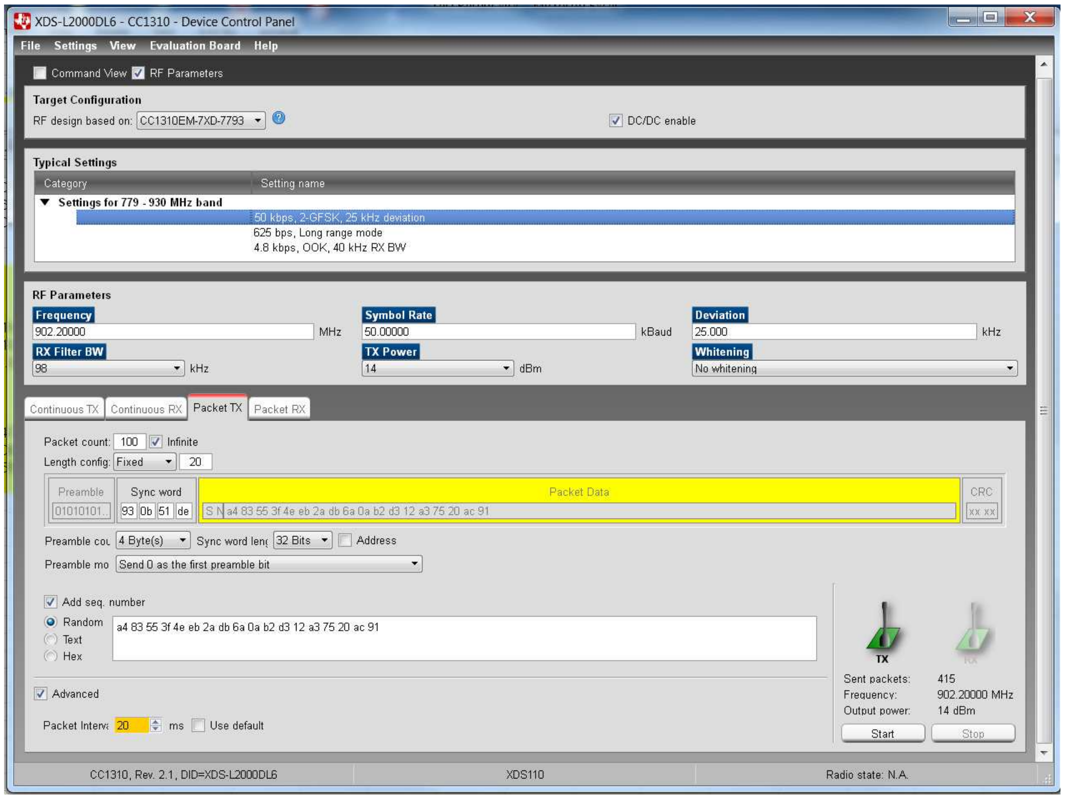


Figure 29. Spectrum Analyzer Capture for Continuous Noise With TX Power of 14 dBm

Table 8. 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

Figure 30 shows the SmartRF Studio capture to generate less than 100% duty-cycle noise. The packet size is fixed at 20B, and the duty cycle is adjusted with the packet interval in the SmartRF Studio setup. The packet intervals used for the experiments are summarized in Table 9.


XDS-L2000DL6 - CC1310 - Device Control Panel
File Settings View Evaluation Board Help
Continuous TX
Continuous RX
Packet TX
Packet RX
TX
RX
Start
Stop

[ ] Command View [✓] RF Parameters

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

Packet count: 100 [✓] Infinite
Length config: Fixed 20

Preamble	Sync word	Packet Data	CRC
01010101..	93 0b 51 de	S N a4 83 55 3f 4e eb 2a db 6a 0a b2 d3 12 a3 75 20 ac 91	xx xx

Preamble cou 4 Byte(s) Sync word len(32 Bits [ ] Address
Preamble mo Send 0 as the first preamble bit

[✓] Add seq. number
(●) Random ( ) Text ( ) Hex
a4 83 55 3f 4e eb 2a db 6a 0a b2 d3 12 a3 75 20 ac 91

[✓] Advanced
Packet Interv: 20 ms [ ] Use default

Sent packets: 415
Frequency: 902.20000 MHz
Output power: 14 dBm

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

Figure 30. Non-Continuous Noise at 902.2-MHz Frequency (Channel 0 in FCC Band)

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 root node. 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 31 shows the delivery ratio as a function of application data size. The x-axis shows the data size in bytes and the y-axis is the average delivery ratio in percentage. For testing, the FH-129 (FH with 129 channels) is used with and without the continuous noise. The continuous noise is generated based on Figure 28.

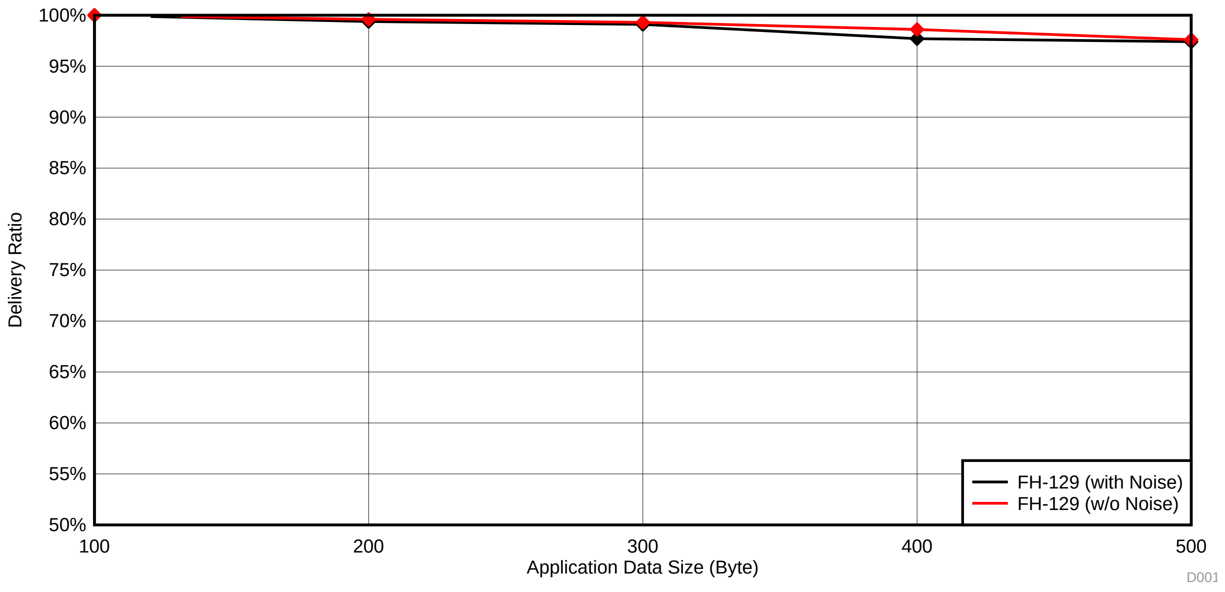


Figure 31. Delivery Ratio (in %) as a Function of Data Size

The results show that there is no performance degradation on the delivery ratio with the continuous noise, which proves that the noise immunity of the FH technique improves the delivery ratio performance significantly.

NOTE: In comparison, the non-FH system shows 0% delivery ratio performance in the existence of the continuous noise.

In addition, the test results show that the delivery ratio degrades slightly as the data size increases. In the given set of testing, the worst performance is 97.4% for 500B data with the continuous noise. Because the UDP poll mechanism controls the network traffic without contentions among end nodes, it is expected that there is no degradation with the data size increment. However, the small degradation comes from the collisions between the periodic keep-alive traffic and the application data. Increasing the keep-alive interval improves the worst-case delivery ratio performance, but it delays the sync loss detection when it happens.

Figure 32 shows the delivery ratio performance 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 results shows that the continuous noise has no impact on the delivery ratio performance even in six-hop distance. The delivery ratio performance achieves greater than 97.9% over all the test cases.

In real fields, the noise level varies over time and sometimes the jamming noise might not be very realistic. The goal of the following experiments is to show the impact of noise level on the delivery ratio for the FH-129 compared to the non-FH. For the experiments, the noise level is changed with the duty cycle of the noise.

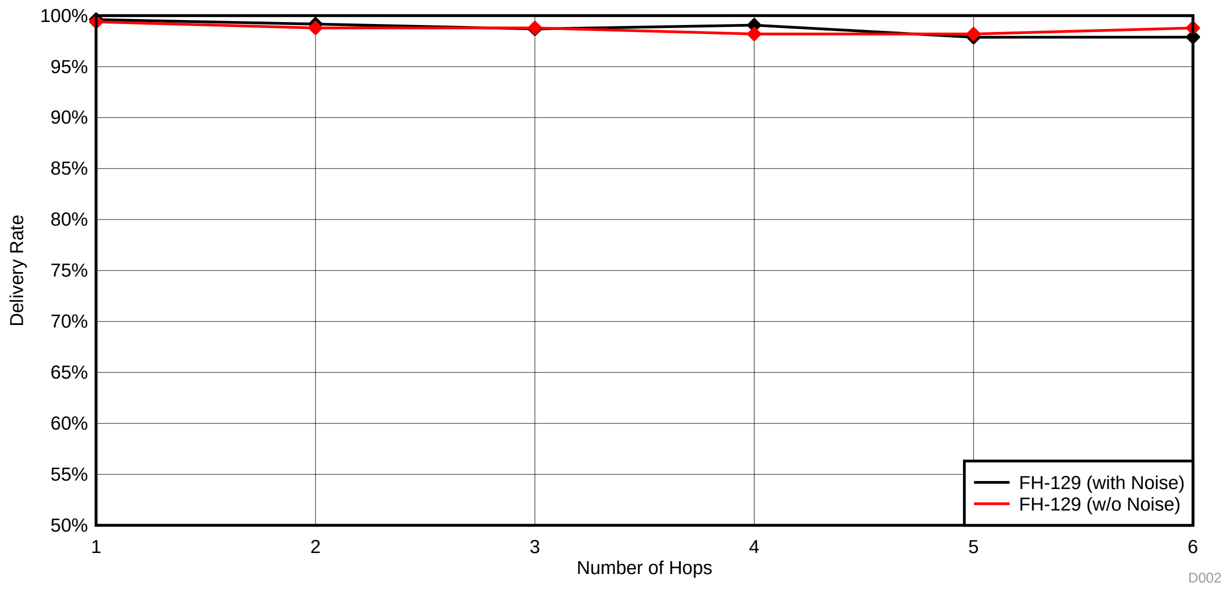


Figure 32. Delivery Ratio (in %) as a Function of Number of Hops

Table 9 summarizes the noise configuration with SmartRF Studio. For all the test cases, SmartRF Studio configured the carrier frequency to 902.2 MHz and the TX power to the maximum of 14 dBm, as shown in Figure 30. The noise duty cycle in percentage is driven as follows:

$$\text{Duty Cycle }(\%) = \frac{\text{Packet Size in Byte} \times 8}{\text{PHY Rate} \times \text{Interval}} \times 100 \quad (1)$$

Table 9. Noise Configuration With SmartRF™ Studio

DUTY CYCLE	PACKET SIZE	INTERVAL	PHY RATE
4%	20B	0.08 s	50 kbps
8%	20B	0.04 s	50 kbps
16%	20B	0.02 s	50 kbps

Figure 33 shows the delivery ratio performance as a function of the noise duty cycle. The x-axis shows the noise duty cycle in percentage and the y-axis denotes the delivery ratio in percentage. The experiments use FH-129 and non-FH systems with three different packet sizes: 100B, 300B, and 500B.

The results show that, for all the sizes, the FH-129 shows no degradation with the noise duty cycle, while the non-FH shows significant degradation as the noise duty cycle increases. The performance degradation of the non-FH becomes significant as the data size increases. This result is because a bigger size means a longer transmission time, which has more chances to get errors in the noisy channels. For the noise duty cycle of 16%, the performance gap between the FH-129 and non-FH shows 33% for 100B and increases to 78% for 500B.

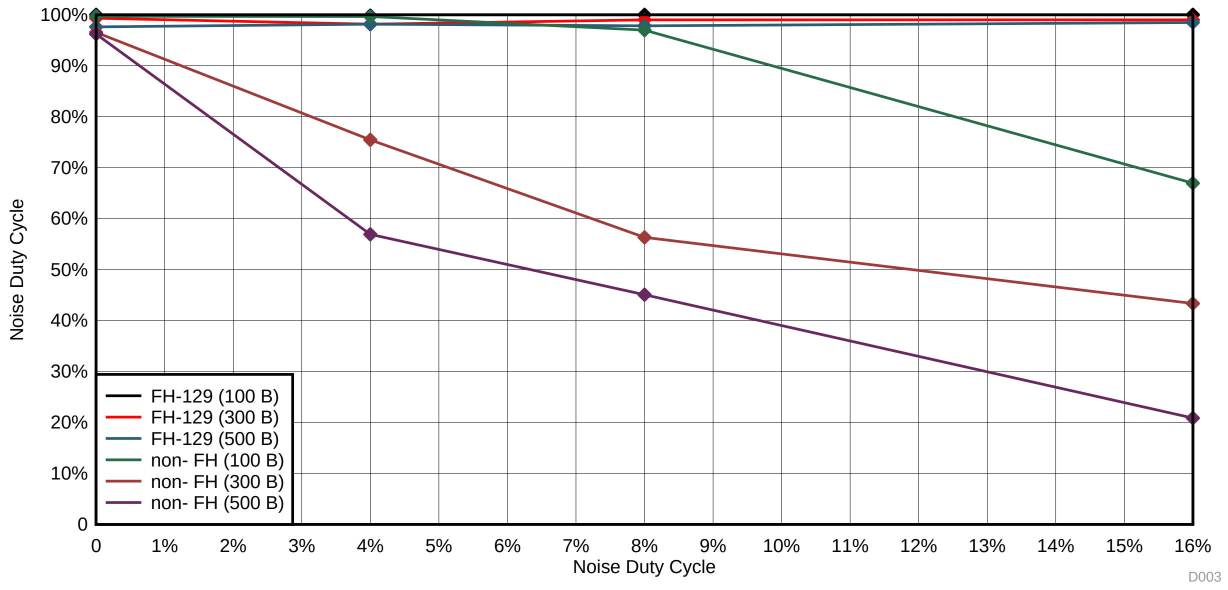


Figure 33. Delivery Ratio (in %) as a Function of Noise Duty Cycle

Figure 34 shows the delivery ratio performance as a function of number of hops for the noise duty cycles of 0%, 8%, and 100%. The goal of this experiment is to show the impact of noise level on the delivery ratio as a function of the number of hops for the FH-129 and the non-FH systems. The x-axis is the number of hops and the y-axis is the delivery ratio in percentage. As expected, for the non-FH, the impact of the noise duty cycle becomes significant as the number of hops increases. In this experiment, for all the hops, the FH-129 achieves greater than 97.44% regardless of the noise duty cycles.

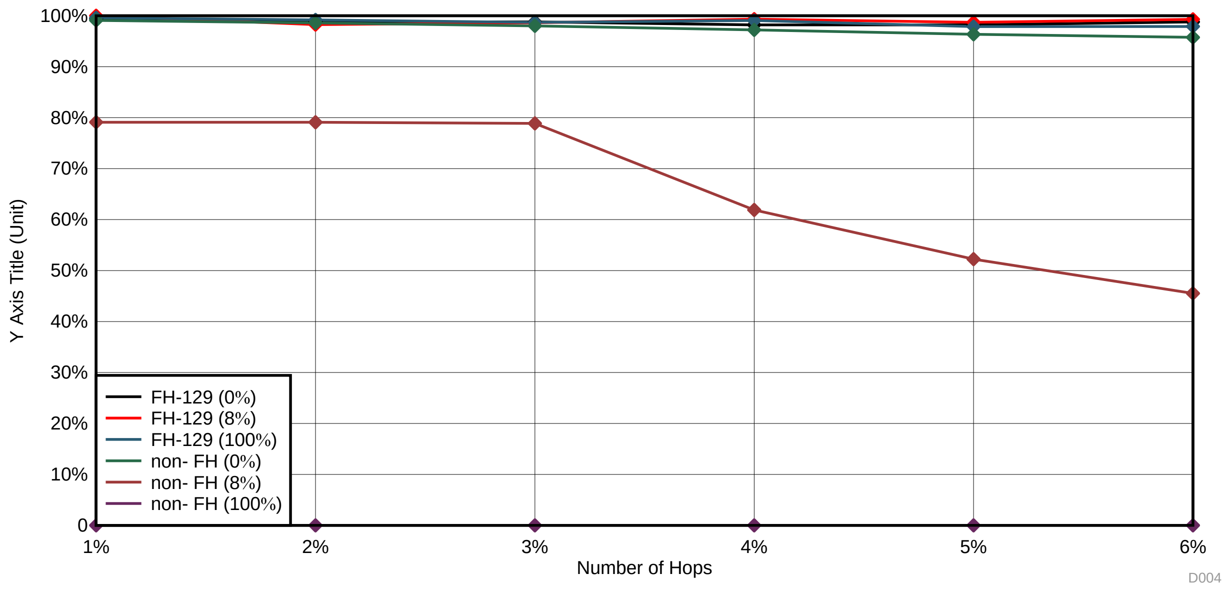


Figure 34. Delivery Ratio (in %) With Noise Duty Cycle as a Function of Number of Hops

3.2.2.3 Round-Trip Time (RTT)

The RTT is measured at the application layer by calculating the time gap between the transmit time and the time echo-back from end nodes at the root node.

Figure 35 shows the RTT performance as a function of number of hops, data size, and with and without the continuous noise. The x-axis shows number of hops and the y-axis shows the RTT in the unit of seconds. For the experiment, the FH-129 is used.

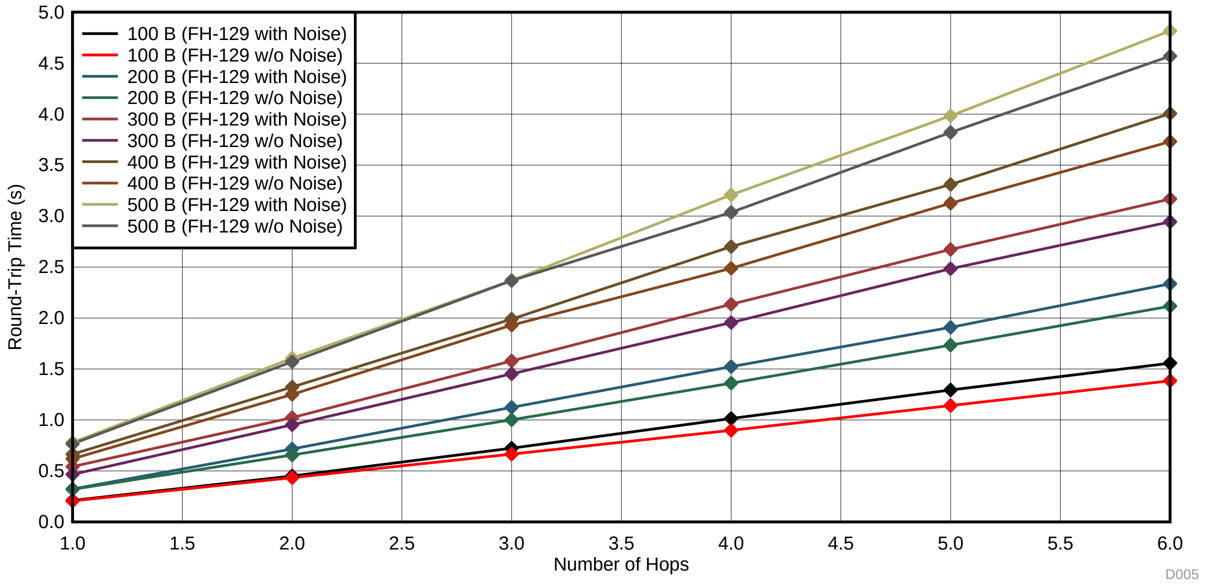


Figure 35. Round-Trip Time (in Seconds) Over Six-Hop Network

The results show 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 the same delivery ratio performance even with the continuous 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-129 uses 129 channels uniformly, the impact of noise on the noisy channels is negligible for one-hop nodes (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 results show that, for one-hop, the RTT performance achieves 0.21 seconds for 100B and 0.78 seconds for 500B and for six-hop, the RTT performance varies from 1.39 seconds for 100B to 4.82 seconds for 500B.

Figure 36 shows a closer look at the RTT performance as a function of the number of hops by averaging the results in Figure 35 over all the data sizes. The x-axis shows the number of hops and the y-axis shows the RTT performance in seconds. The goal of this experiment is to show the impact of the number of hops on the RTT performance in the noisy channel condition.

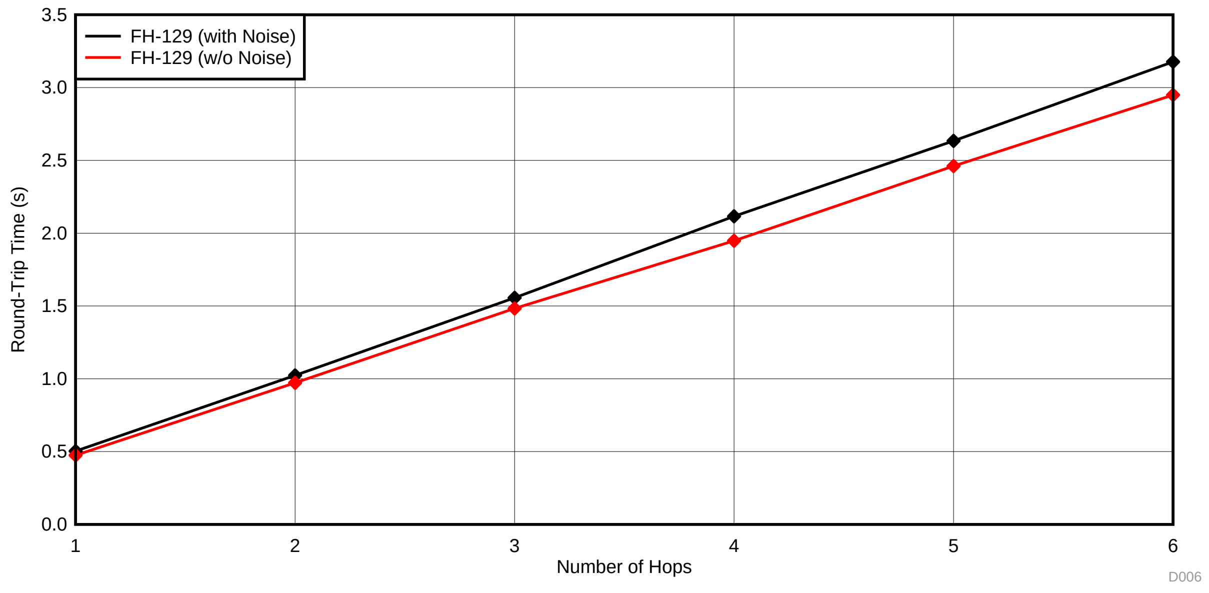


Figure 36. RTT Performance (in Seconds) as a Function of Number of Hop

The results show that, for the one-hop, the RTT performance gap between the FH-129 with and without noise shows 28.4 ms and for six-hop, the performance gap increases to 227.6 ms.

Figure 37 shows a closer look at the RTT performance as a function of the data size by averaging the results in Figure 35 over all the hops. The x-axis shows the application data size in byte and the y-axis shows the average RTT in the unit of seconds.

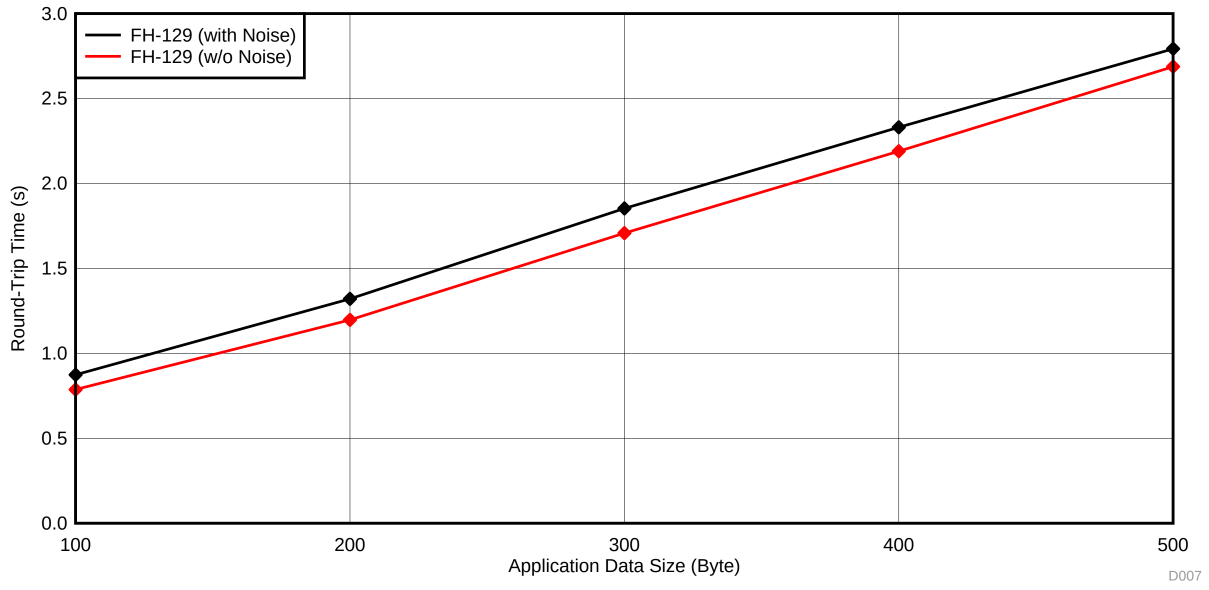


Figure 37. RTT Performance (in Seconds) as a Function of Data Size

The results show that the RTT performance gap between the FH-129 with and without noise slightly increases as the data size increases but less than that for the number of hops. This is because the test results are averaged over the number of hops that dominates the RTT performance. For 500B, the RTT performs 110 ms and for 100B the RTT shows 90 ms.

3.2.2.4 Goodput

The goodput is defined as the application-level throughput considering the overheads of underlying layers. The goodput performance is calculated based on the measured RTT as shown in Figure 35. Because the RTT can be measured on successful transmissions only, the goodput performance does not count on the performance degradation due to the packet losses caused by channel variation or collisions. Therefore, this data can be a reference on the expected goodput when data transmissions are successful.

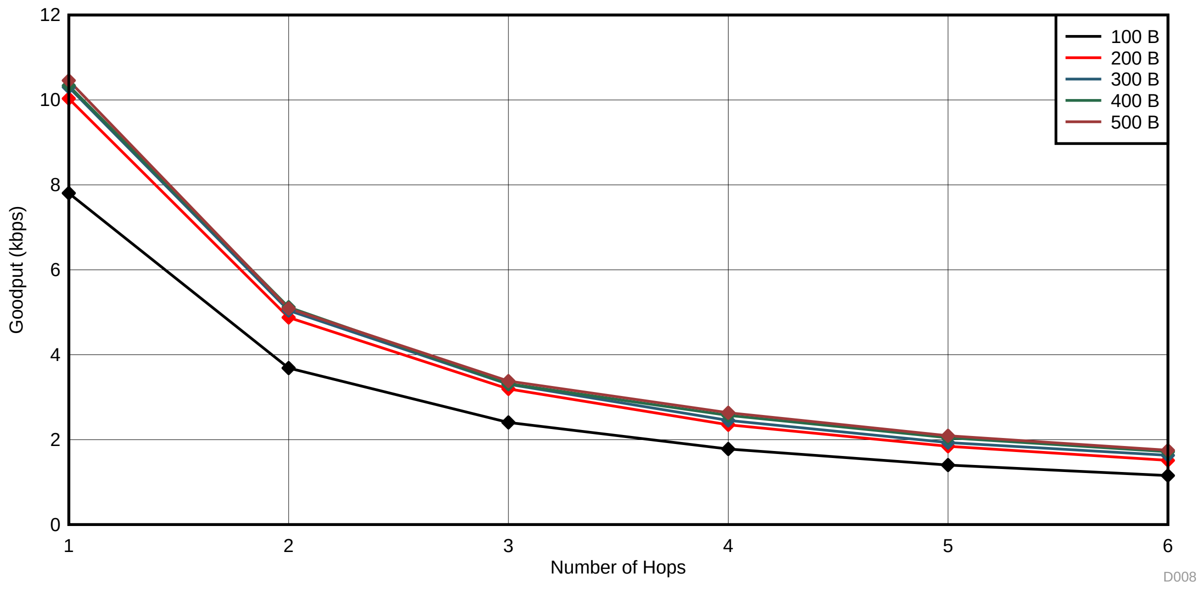


Figure 38. Goodput Performance (in kbps) as a Function of Number of Hops

Figure 38 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. In overall, the goodput performance decreases as the number of hop increases due to more transmissions through the number of hops to reach at the destination. In addition, the goodput performance decreases as the data size decreases. This is because the impact of overhead (for example, TX/RX processing time, packet headers, and propagation delay) becomes significant as the data size decreases. The results show that the goodput performance varies from 1.51 kbps (with 100B over six-hop) to 10.5 kbps (with 500B over one-hop).

3.2.3 Impact of In-Network Interference on System Performance

The goal of this experiment is to verify how reliably the FH-129 performs with in-network interference, mostly packet collisions among nodes, compared to the non-FH system. To verify this, the multi-hop network with two three-hop branches is created as shown in Figure 39.

This experiment uses the UDP push example because the experimental goal is to see the impact of in-network interference on the system performance. The UDP push example is more likely to have packet collisions due to the distributed packet transmissions among nodes. The application data size is fixed at 500B.

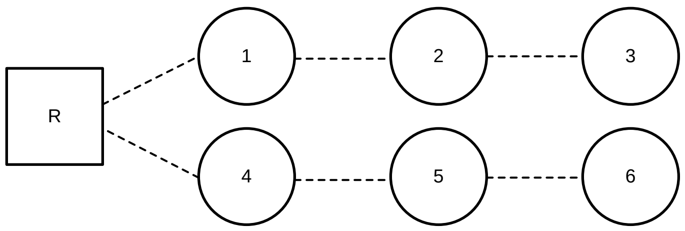


Figure 39. Multi-Hop Topology With Two Three-Hop Network Branches

3.2.3.1 Delivery Ratio

Figure 40 shows the delivery ratio performance as a function of the packet interval. The experiment compares the performance of the FH-129 with the non-FH system. The x-axis shows the packet interval in seconds and the y-axis shows the delivery ratio in percentage.

The experimental results show that compared to the FH-129, the non-FH performance is more sensitive to the network traffic as the performance degradation becomes significant as the packet interval decreases. The performance gap increases to 29% at the packet interval of 5 seconds. The performance gap comes from that the FH-129 reduces probability of collisions among nodes in two branches due to the frequency hopping. For the non-FH, packet losses occur when there are simultaneous transmissions in more than one links and the impact becomes significant with higher traffic.

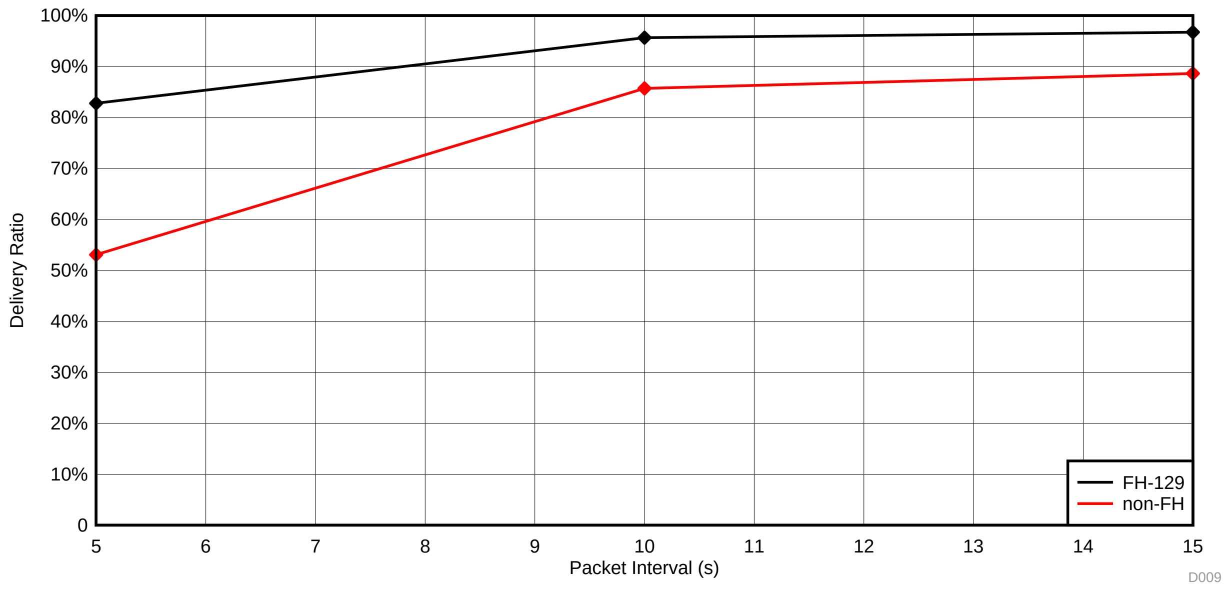


Figure 40. Delivery Ratio Performance as a Function of Packet Interval

3.2.4 Coexistence Performance

The coexistence in the unlicensed bands is important. Compared to the 2.4-GHz ISM band, the coexistence is more critical in the sub-1 GHz RF because the transmission range is wider due to its longer RF propagation property.

To validate how the FH system can perform in terms of coexistence performance, this experiment runs two three-hop networks as shown in Figure 41. For the experiment, the UDP poll example is used with the packet size of 500B and the poll interval of 5 seconds.

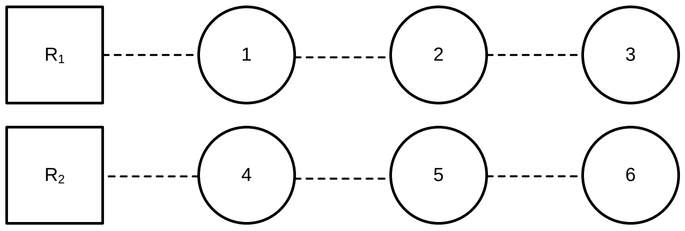


Figure 41. Two Three-Hop Networks for Coexistence Scenario

3.2.4.1 Delivery Ratio

Figure 42 shows the delivery ratio performance as a function of various coexistence scenario. The x-axis shows the scenario and the y-axis shows the delivery ratio in percentage. The delivery ratio is averaged over all three hops. This experiment evaluates three scenario: sole network with the non-FH for reference, coexisting networks with non-FH and non-FH, and coexisting networks with non-FH and FH-129.

The test results show that for the sole network of the non-FH, the delivery ratio is 99.66% while in the co-existence scenario with two non-FH systems, the delivery ratio shows significant drop to 70.23% because adjacent non-FH network interference. However, in the coexistence scenario with non-FH and FH-129 systems, the FH-129 system has no impact on the delivery ratio of the coexisting non-FH system, which proves that the FH-129 shows outstanding coexistence performance.

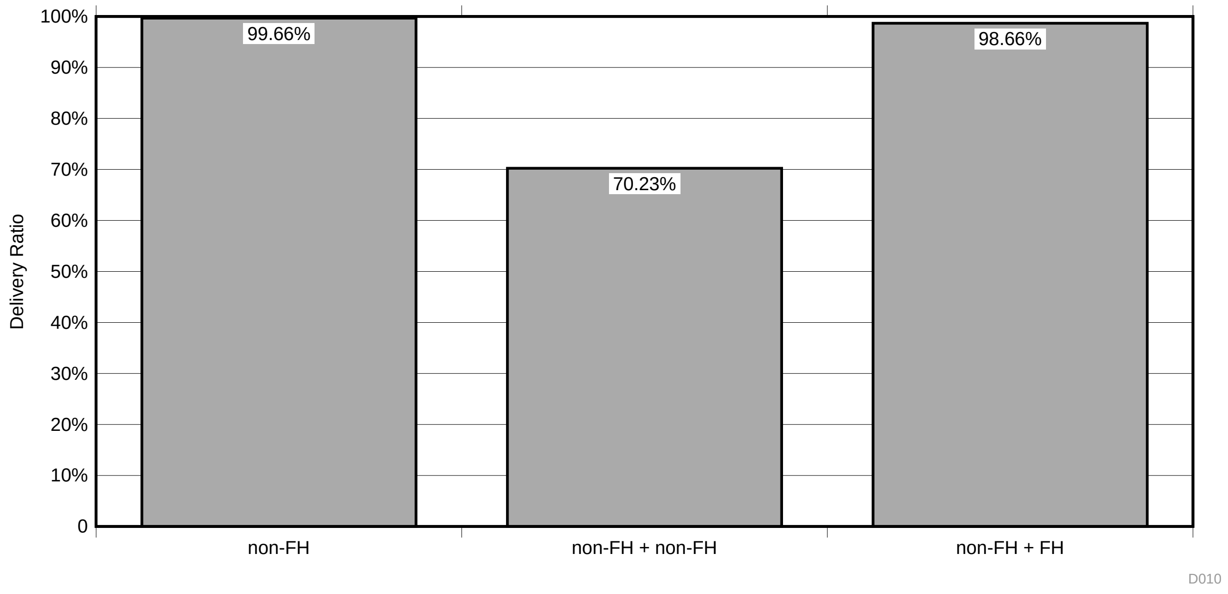


Figure 42. Delivery Ratio Performance of Non-FH System in Various Coexistence Scenario

Figure 43 and Figure 44 show a closer look at the delivery ratio performance in the two coexistence scenario to see how the system performance is affected with each other. Figure 43 is for the coexistence scenario of two non-FH systems, and Figure 44 is for the non-FH with the FH system.

The experimental results show that running two non-FH systems together degrades the delivery ratio performance significantly by interfering with each other. The delivery ratio performance is 70.23% and 67.55% in this scenario. However, in the coexistence scenario of non-FH with FH-129, the delivery ratio performance achieves 99.33% for the FH-129 and 98.66% for the non-FH system. These results prove that the FH-129 system minimizes the impact of interference on the non-FH system performance by switching channels instead of using the dedicated channel shared with the coexisting non-FH.

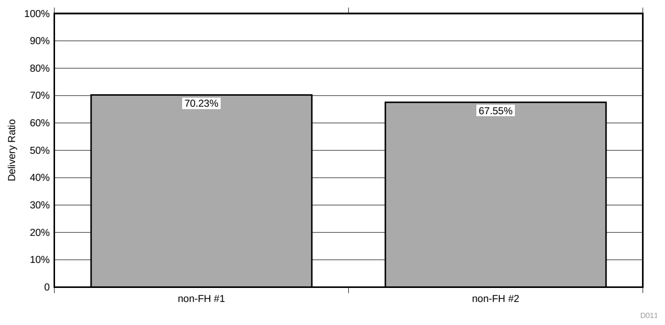


Figure 43. Delivery Ratio in Non-FH With Non-FH Scenario

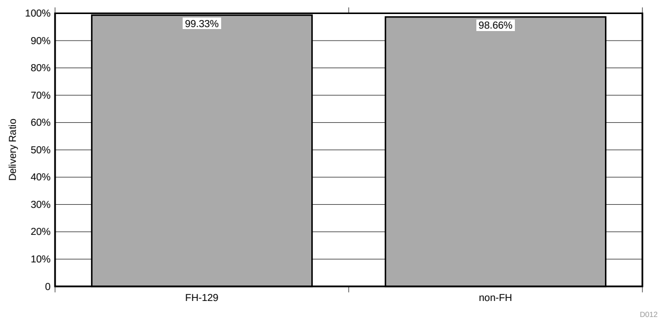


Figure 44. Delivery Ratio in Non-FH With FH-129 Scenario

3.2.5 20-Node Network Testing

The goal of these experiments is to measure delivery ratio performance as well as to verify the software reliability in a real scenario where 20 end-nodes and 1 data collector are distributed in the lab. Figure 45 shows the details of the 20-node setup with locations and the last two bytes of MAC address information. These experiments do not use the software filtering to create multi-hop networks discussed in Section 3.2.1.1.

NOTE: By default, the TI 15.4 coprocessor binary in the SimpleLink SDK 2.30.0.20 supports up to 10 device table entries (MAX_DEVICE_TABLE_ENTRIES defined in the CCS pre-defined symbol), which makes network unstable in a large network. To address this, the TI 15.4 co-processor can be rebuilt with an increased device entry size until memory is allowed. The TI 15.4 co-processor will notify with the SYS_RESET_IND if memory issue happens. Another option without rebuilding the TI 15.4 co-processor is to use the TI 15.4 coprocessor with CC1312R that supports more device entries by default.

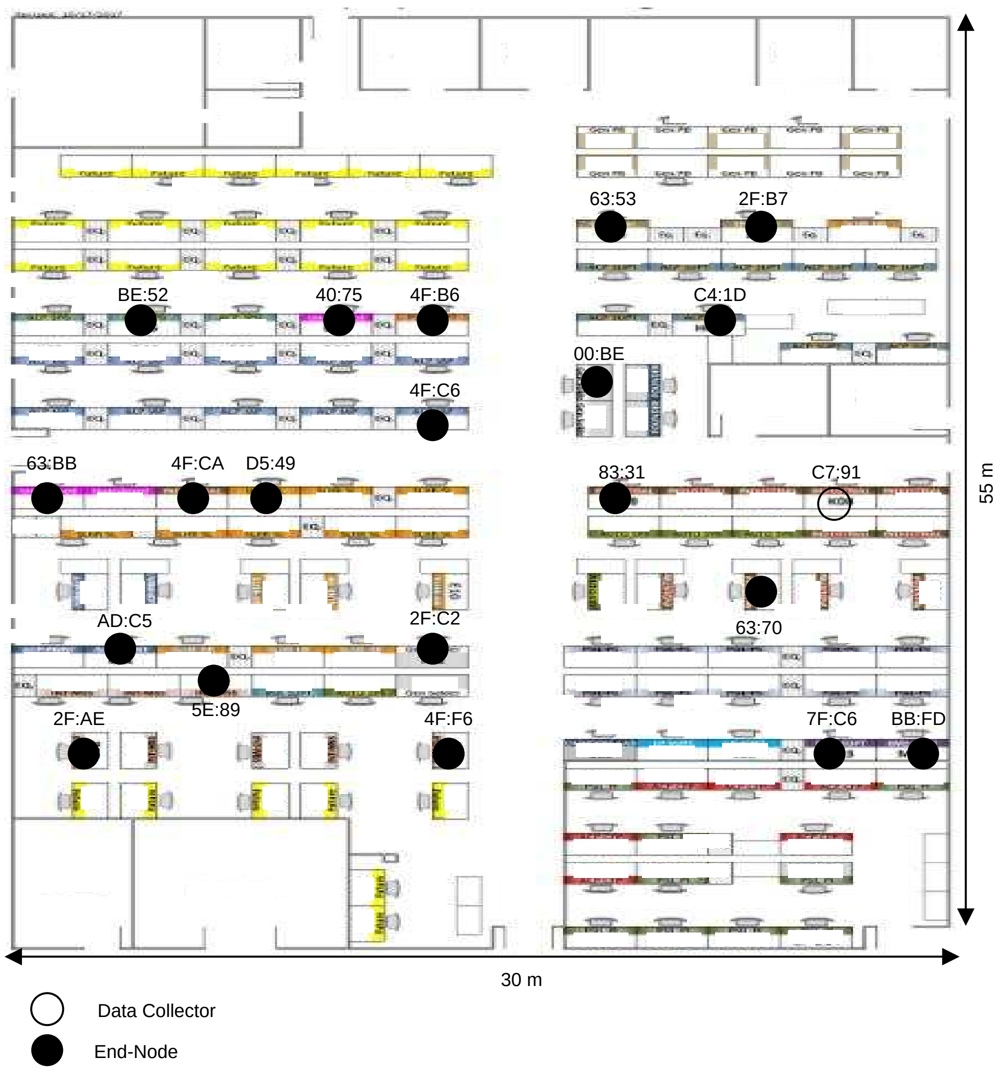


Figure 45. 20-Node Network Setup

Figure 46 and Figure 47 show the delivery ratio performance as a function of the MAC address of each end node in the 20-node setup. This experiment uses the UDP polling example with 100B, 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 results show that all of nodes achieve close to 100% delivery ratio performance for all the nodes. The average delivery ratio with 20 nodes is 99.97% and the average RTT is 0.331 seconds.

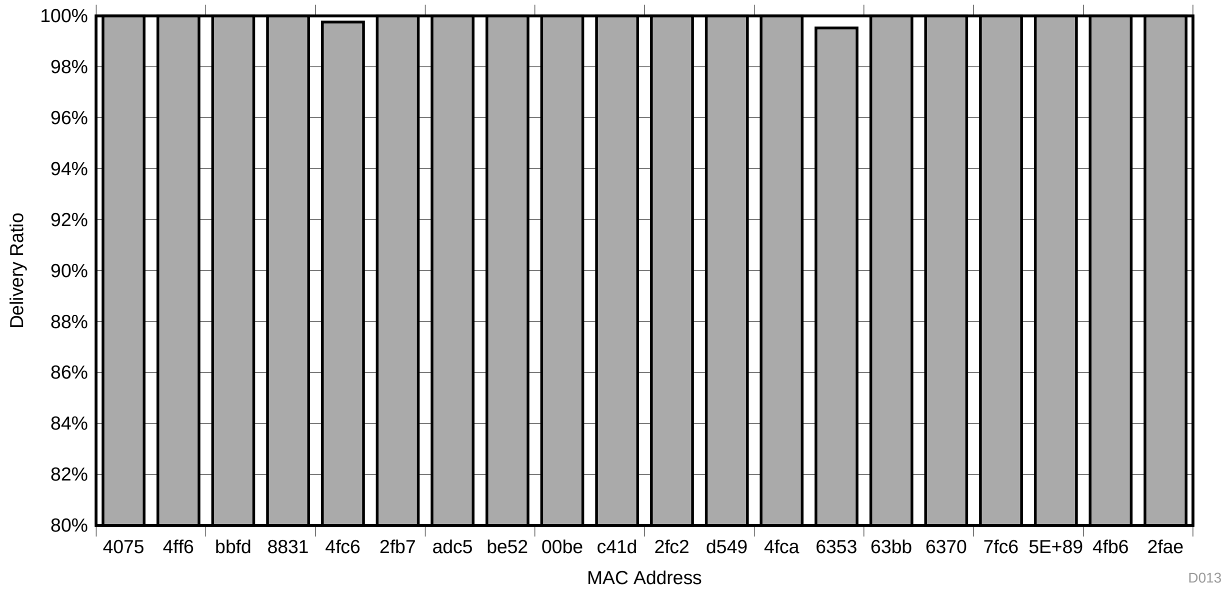


Figure 46. Delivery Ratio Performance With 100B UDP Polling Example

Figure 47 shows the delivery ratio performance as a function of the MAC address of each end node in the 20-node setup. This experiment uses the UDP polling example with 100B 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.

Compared to 100B experiment, the average delivery ratio performance is degraded from 99.97% to 92.23% because 500B application data generates more than one fragment at lower layers. This result increases the total number of transmissions over RF, which increases the chance to be collided between keep-alive and data packets. The measured average RTT is 1.59 seconds.

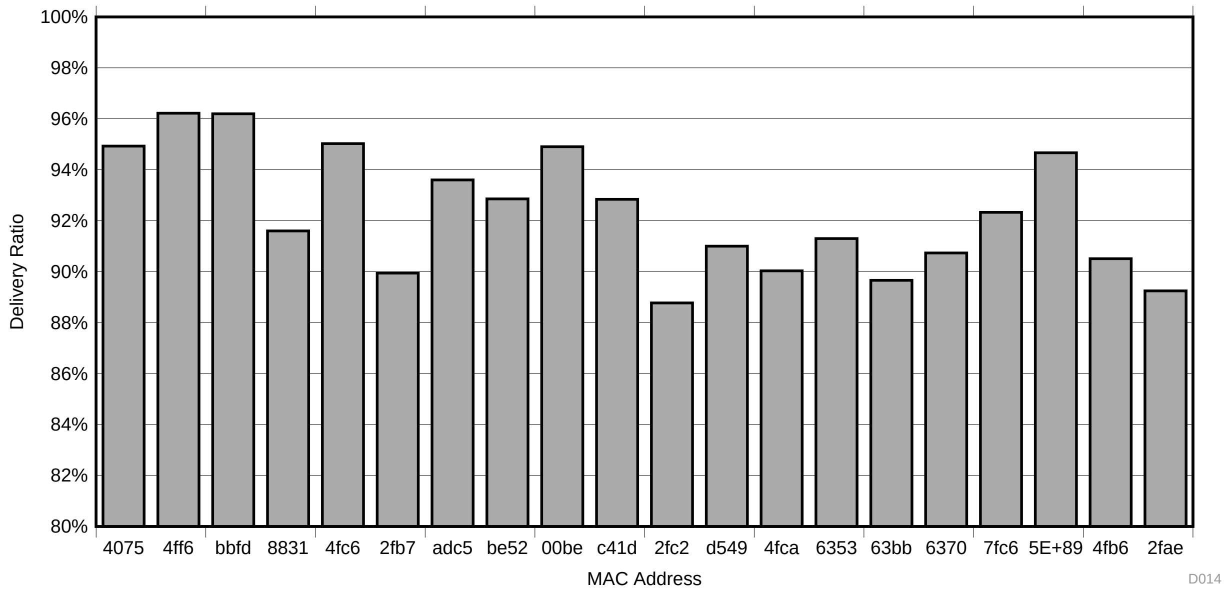


Figure 47. Delivery Ratio Performance With 500B UDP Polling Example

Table 10 summarizes test cases to verify the software reliability in the 20-node network setup. The experiments use the UDP polling example with 1200B 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 automatically.

Table 10. Software Reliability Test Summary

TEST CASE	DESCRIPTION	PASS/FAIL
Long-run data testing (> one week)	Run 1200B UDP polling example in the 20-node setup	Pass (All 20 nodes stayed connection to the DC)
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 all the one-hop 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-01547.

4.2 Bill of Materials

To download the bill of materials (BOM), see the design files at TIDA-01547.

4.3 PCB Layout Recommendations

4.3.1 Layout Prints

To download the layer plots, see the design files at TIDA-01547.

4.4 Altium Project

To download the Altium project files, see the design files at TIDA-01547.

4.5 Gerber Files

To download the Gerber files, see the design files at TIDA-01547.

4.6 Assembly Drawings

To download the assembly drawings, see the design files at TIDA-01547.

5 Software Files

To download the software files, see the design files at TIDA-01547.

6 Trademarks

E2E, SimpleLink, MSP432, MSP430, Code Composer Studio, SmartRF are trademarks of Texas Instruments.
Arm, Cortex are registered trademarks of Arm Limited (or its subsidiaries).
Wi-Fi is a registered trademark of Wi-Fi Alliance.
All other trademarks are the property of their respective owners.

7 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 (March 2018) to A Revision	Page
• Changed information in Table 1	4
• Changed coprocessor_cc13x0_lp.hex to coprocessor_cc1310_lp.hex	12
• Changed SimpleLink CC13x0 SDK software version from v1.50.0.8 to v2.30.0.20	12
• Changed Code Composer Studio software version from v7.1.0 to v8	12
• Changed SimpleLink MSP432P4 SDK software version from v1.50.0.12 to v2.40.0.10	12
• Changed Figure 14	13
• Deleted XDC tools	13
• Changed Figure 15	14
• Changed Figure 16	15
• Changed Figure 16 caption from CCS™ RTSC Screen Capture to CCS™ Products Property	15
• Changed information in Section 3.1.2.2	15
• Changed Figure 17	15
• Changed Figure 18	15
• Changed information in Section 3.2	25