UG104: Testing and Debugging Applications for the Silicon Labs EM35x and EFR32MG Platforms

This manual provides an overview of testing and debugging strategies for applications developed for Silicon Labs EM35x and EFR32MG platforms, and explores in more detail using Simplicity Studio for debugging. This overview is supplemented by the following documents

- *AN1019: Using the NodeTest Application*
- *UG409: RAILtest User's Guide*

KEY FEATURES

- Strategies for testing and debugging Zigbee applications
- Analyzing Network Analyzer session files

1. About This Document

1.1 Purpose

This manual provides an overview of testing and debugging strategies for applications developed for Silicon Labs wireless platforms, and explores in more detail using Simplicity Studio for debugging. It is designed for EmberZNet PRO users who are considering test strategies for their products. This overview is supplemented by documents *AN1019: Using the NodeTest Application* and *UG409: RAILtest User's Guide*. Documents that address issues specific to manufacturing testing include:

- *AN718: Manufacturing Test Overview*
- *AN700.0: Manufacturing Test Guidelines for the EM35x Family*
- *AN700.0: Manufacturing Test Guidelines for the EFR32 Family*
- *Using the Manufacturing Test Library*

1.2 Audience

This document is intended for project managers, embedded software engineers, and test engineers who are responsible for building a successful embedded mesh networking solution using Silicon Labs radio, network stacks, and tools.

1.3 Getting Help

Development kit customers are eligible for training and technical support. You can use the Silicon Labs website to obtain information about all Silicon Labs products and services, and to sign up for product support. You can also contact Customer Support at www.silabs.com/support.

2. Strategies for Testing and Debugging Mesh Networking Applications

2.1 Introduction

Development and testing of embedded applications has always relied on specific testing and debug strategies, such as the use of emulators or JTAG, to isolate problems. This has worked well when a problem exists on a single device, or between two devices communicating with each other. Commercial development and deployment of larger distributed embedded network applications has required another level of testing strategies. Without planning of the testing and debug process, development projects can stall in a cycle of bug fixing, field testing, bug fixing, and field testing until hopefully all problems have been resolved. Silicon Labs has participated or assisted in the development and deployment of a number of commercially available wireless products, and recommends a series of specific testing strategies throughout the development process. The development strategy and testing processes required are similar even if the products being developed are aimed at different marketplaces.

This chapter reviews the typical successful testing methodology, and the hardware and software tools required to qualify products. It provides a testing outline and methodology that can be used for software qualification. Consideration of this testing methodology is critical from the beginning of the development efforts to ensure suitable means for qualification are built into the software, including appropriate means for simulating testing and recording test results. Real world examples and test setups are used to illustrate the proposed testing strategies. The key areas of testing to be covered are:

- Hardware and application considerations for testing and debug
- Initial development and lab testing
- Beta criteria and field trials
- Release testing process and criteria for release

2.2 Hardware and Application Considerations for Testing and Debug

2.2.1 Initial Software Application Development using Development Kit Hardware

Initial development of customer applications typically starts using development kit hardware (see the following figure) that is available from the chip and software supplier. These kits universally have some level of serial or Ethernet debug capabilities, to allow viewing what is occurring at the application as well as at the software stack. It is important for a developer to start with these tools to evaluate what information is available and how it can be used in development, debug and testing.

Some typical choices to be made early include:

- Will the software provide debug information out a debug port or serial port for later use in debug and testing? Such information can be critical to evaluating problems later and can significantly shorten the debug process. However, use of a serial port may slow the application and impact normal operation. A serial port may also not be available during normal system operation. Even if not normally available, a serial or debug port can be populated on prototype hardware for early stages of testing. For example, Silicon Labs hardware includes a dedicated debug port that can be used by the application and also includes stack trace and packet trace information.
- How will software be monitored and upgraded during internal testing? Software must be regularly updated during development and initial testing. For these systems it is recommended that a means of rapidly upgrading software on all devices be included even if it is not included on final field hardware. For example, on Silicon Labs Wireless Gecko development kits the dedicated debug and programming port can be accessed over an Ethernet or USB backchannel. During initial development and testing, use of this wired backchannel is recommended for both software upgrading and system monitoring.
- How will software be monitored and upgraded during field testing? Initial field testing will also result in periodic updating of software. However, mechanisms used during internal testing may be impractical during actual field testing. An over-the-air (OTA) bootloader is recommended for software upgrading during field testing. A debug channel is impractical for all devices during field testing. Instead, it is recommended that selected nodes such as gateways be monitored with a debug channel and that sniffers be used to monitor the network.
- What mechanisms will be provided for system testing and qualification? It is important to be able to trigger system-level events and normal operations during qualification testing. This can be done either manually by operating the system or it can be done using backchannel or serial port mechanisms, but it must be considered early in the design.

Initial application development and testing efforts on development kit hardware should consider each of these questions. Depending on the particular hardware being designed, it may or not be practical to include all of the recommendation above but they should be considered.

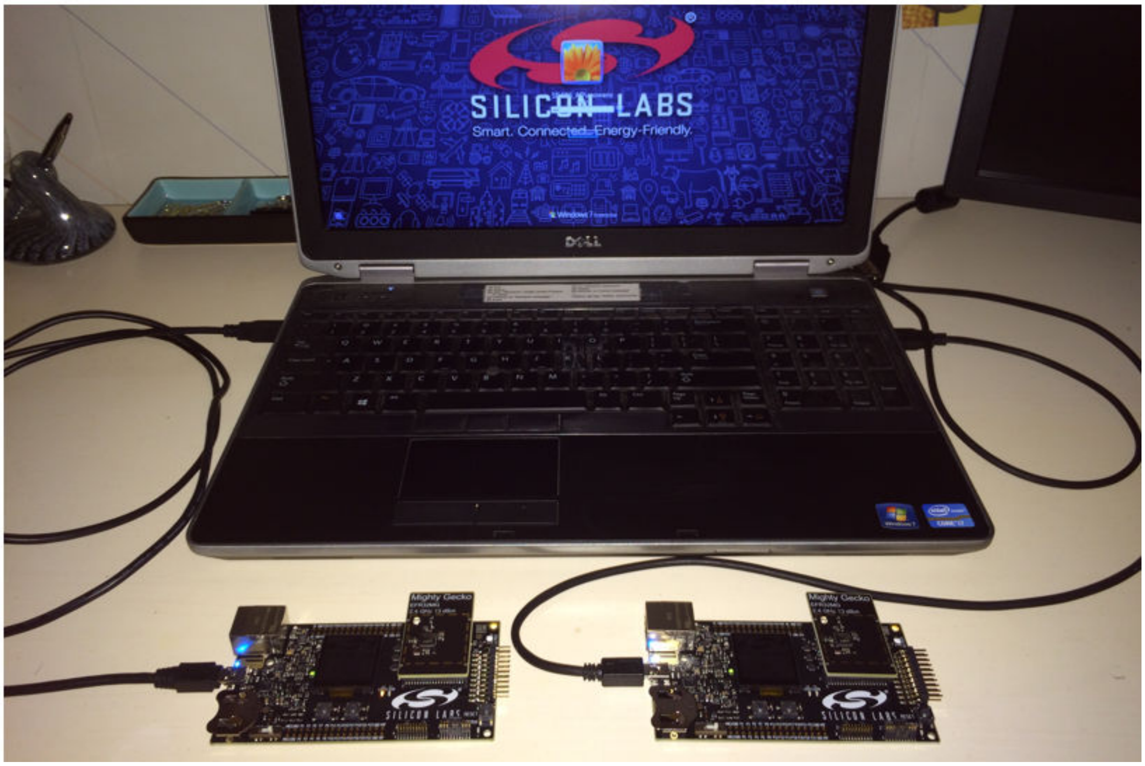


Figure 2.1. Typical EFR32/WSTK Development Kit

2.2.2 Transition to Custom Hardware

Most customers move to the hardware specifically designed for their application as soon as practical. However, the limitations and constraints of the custom hardware design may make it impractical to include monitoring and debug ports. At a minimum, tests points should be included to allow access when required.

If monitoring and debug ports are not practical on the custom hardware, it is recommended that development continue on both custom hardware and the development kit hardware so full debugging capabilities are available on at least some of the hardware. For devices based on Silicon Labs wireless SoCs, the debug port is often the programming port. This should be available on initial hardware designs since devices typically will be programmed many times during development and debug.

Many networks have a centralized base station, gateway or controller node. This plays a critical role in the network and therefore plays a critical role in monitoring and debugging the network. If other devices cannot have a debug and monitoring port, this device should include one.

Questions about debug access and programming must be considered on the customer hardware during the design phase to avoid a hardware respin later to add these capabilities. Software engineers should review schematics to ensure the capabilities provided in hardware match those required for software development and debugging.

2.3 Initial Development Environment and System Testing

Initial development and testing is typically done on a desktop or benchtop environment to ensure a basic application is operational before expanding to a larger network.

2.3.1 Debug Library

During prototyping, developers usually will want to link debugging functionality in their project. This is typically provided through an optional plugin selected from within the Simplicity Studio IDE. For example, the Zigbee application framework provides Debug Basic Library and Debug Extended Library plugins for supplementing the stack with debugging capabilities. Available APIs for debugging are described in an ember-debug.h file, typically included in a stack/include subdirectory within the stack installation. A full description of these interfaces can be found in the online stack API reference.

Note: Once the application has met all design goals, the debugging functionality can be turned off in the final build by removing the related plugins. Debug access should be secured by the method applicable to the platform. See *UG103.5: IoT Endpoint Security Fundamentals* for more information on this and other security best practices.

2.3.2 Single Device Testing and Debug

One of the early steps in starting a new application is to isolate the network to several devices and focus on the message flow and application logic. This may be as simple as light and a switch or a controller and a single device. This testing validates the simple messaging protocols.

For devices with very specific and time-critical operations, debugging on a single device is necessary to ensure proper operation. This is common on devices doing lighting control or dimming, where precise timing is needed to maintain lighting levels. In these cases it is important to develop and debug the application using specific debug tools connected to the individual device. Because timing may be affected by operation of the network stack, this testing must also be done during network-level testing, to verify proper operation of the individual device.

2.3.3 System Test Scenarios

Once single device testing and debugging is completed, more directed system-level testing is required. This directed testing should reflect the expected network operating environment and conditions, in order to detect problems that otherwise would occur in the field.

For any system-level testing, it is important to define the expected operations under specific network and application scenarios. These include the following:

1. System Start Up – The expected normal system start up and commissioning should be validated and tested.
2. Typical System Operation – The expected normal system operation and data flow through the network should be tested to ensure smooth and consistent operation.
3. Security – Many applications run extensive lab and field testing before turning on security. The use of security can make troubleshooting and debug more difficult, depending on the sniffer tools used. It is important to have a debug system that decrypts the packets prior to display. Use of security impacts the payload size available for the application, and increases system level latency due to more node processing time. The impact of security on system level performance needs to be identified early in the system-level testing.
4. Stressed System Operation – If the system has particular devices that generate messages based on user interaction, these items should be tested well beyond normal operation to understand the behavior when the system is more stressed. The acceptable behavior under these conditions must be defined and then tested. For example, in some systems it may be acceptable to discard a message under high traffic conditions and send another message later. In other systems the application should retry sending the message.
5. Power Failure and Restart – Systems may lose power either partially or totally due to a power outage or building maintenance. The system has to react and recover from power failure and resume operations. Battery-operated sleeping devices must conserve power during this loss of network connectivity and then rejoin the network once it restarts. The expected time of the restart varies based on the application design and use of sleeping devices. The behavior of the application on this restart should be defined and tested.
6. Performance Testing – Many applications have specific requirements for end-to-end latency, delivery reliability or other system criteria. These specific metrics should be defined and a means for clearly measuring the metrics during testing should be determined.
7. Typical Application Failure Cases – Failure mechanisms may vary for different systems, depending on network topology. However, typical failure cases should be identified and added to the test plan for each stage of development. Typical scenarios include:
   - low battery on end device
   - loss of end device
   - loss of parent (child fail over to new parent)
   - loss of routers in network
   - loss of gateway, border router, or other access point

2.3.4 System Level Test Networks

In addition to the test scenarios above, dedicated test setups are necessary to establish the conditions expected in the field. In Silicon Labs' experience, ad hoc testing does not uncover all the expected use cases and therefore directed testing to force these network conditions is required.

In each of these test setups, it is not necessary to run directed or specific tests. However, the full range of typical expected operating conditions should be exercised.

Multihop Test Network

Actual field conditions are not always known, and actual radio range under the expected installed environment are not known. A multi-hop test environment should be established. Sometimes testing is done on an ad-hoc basis with desktop or office networks but more than one hop is not tested. One method we have used to ensure testing a multihop network is to build a wired test network with splitters and attenuators in the RF path. This provides a repeatable test environment for regression testing. The following figure shows such a set up. It includes RF cabling, connectors, attenuators and splitters. Each splitter can represent three nodes at a particular network depth. The signal is then attenuated to the next splitter that forms the next hop in the network. A network can be built of as many hops as are expected. This test should be built to replicate as many hops as expected in the field installations, or at least five hops.

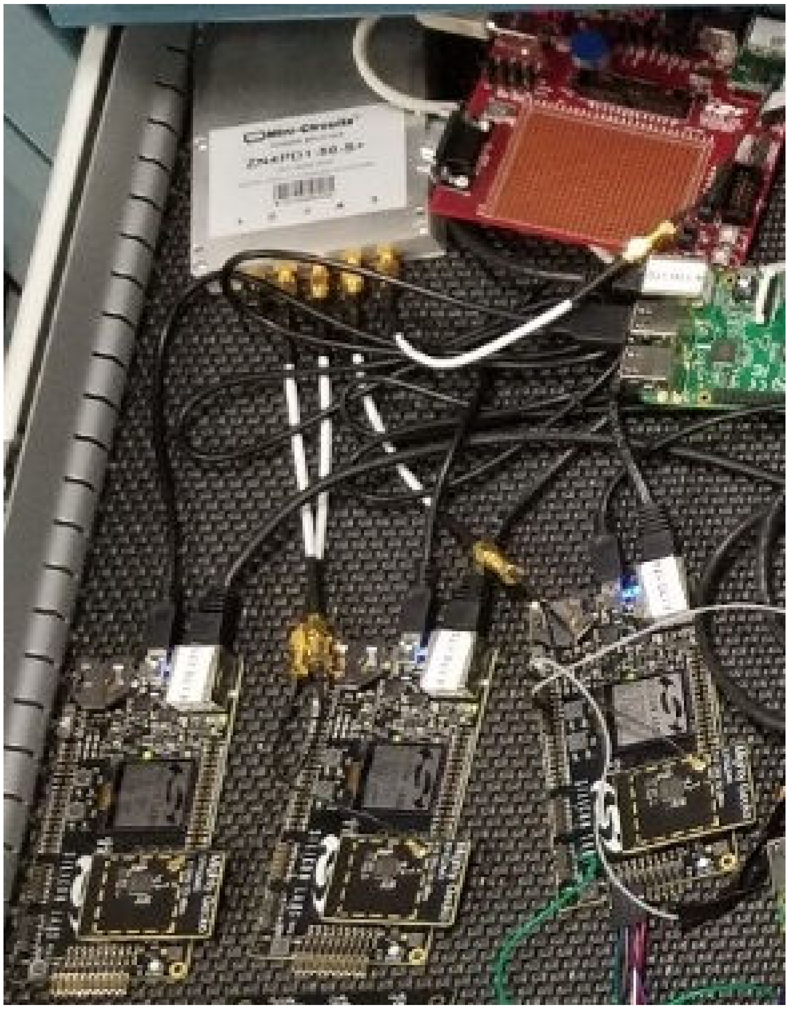


Figure 2.2. Multihop Test Network

Dense Network

Dense networks often can present a challenge, as the network has to deal with increase in table sizes and message flow. The application also experiences more delays in message flow since all radios within range must share time on the air. It is recommended that at least 18 nodes be included in this testing. A compact layout for EM3x dense network testing is shown in the following figure.

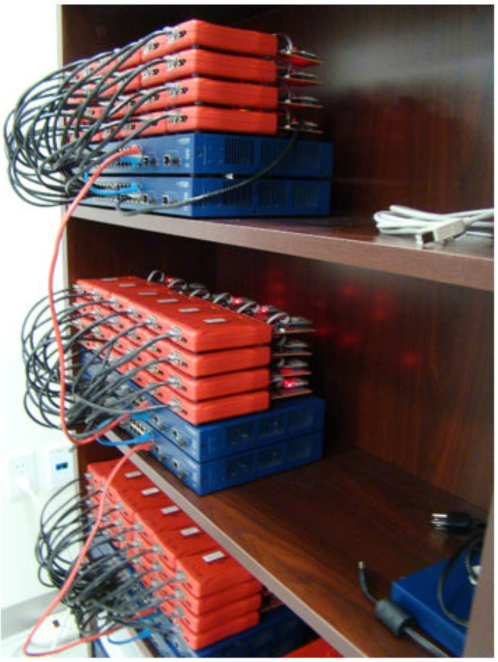


Figure 2.3. Compact EM35x Node Density Test Setup

Often other devices operating in or near the test environment can interfere with this and other testing. In that case, isolating the test environment in Ramsey boxes is recommended, as shown in the following figure.

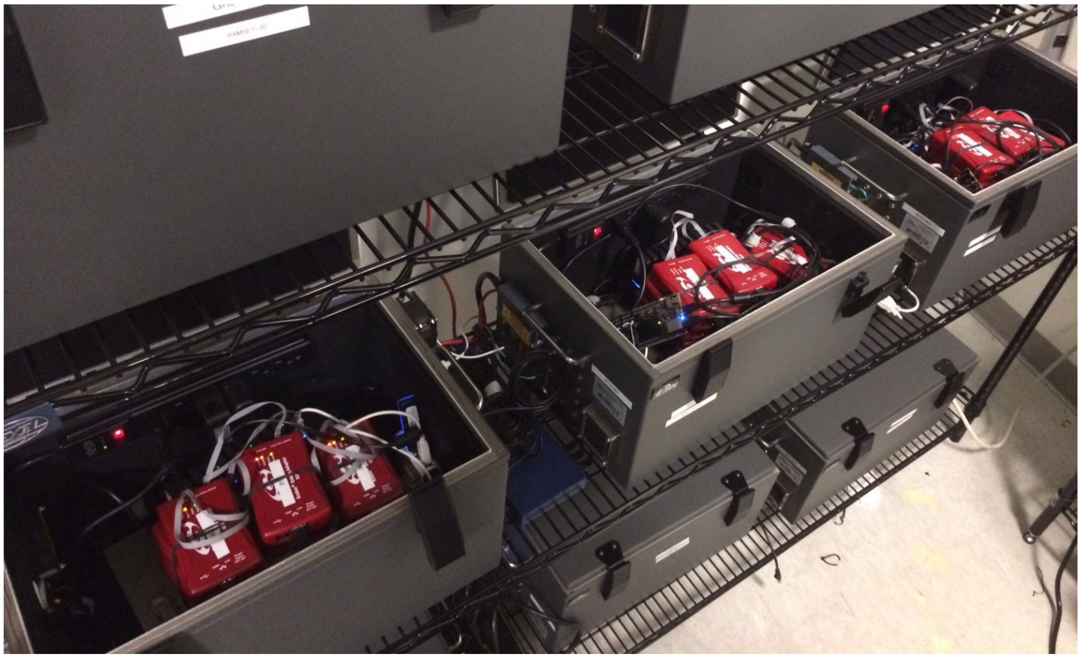


Figure 2.4. Test Network Isolated in Ramsey Boxes

Mobile Device Testing

Many system designs have one or more mobile devices. These may be remote controls, handheld devices or operator indication devices. The mobility of such devices and their ability to reconnect to the network should be tested before field testing. Problems with mobile devices can be difficult to track in the field because the device is moving around the network, and therefore is harder to troubleshoot. Simple mobile node testing was added to the existing wired test installations with a switchable attenuator under control of a java application. This allows moving the device from one side of the network to another and back again. Silicon Labs Software Quality Assurance uses the RCDAT-6000-90 programmable attenuator.

Large Network Testing

After the above dedicated tests, software can then be tested in a large scale test network, as shown in the following figure. The test clusters are controlled through a private Ethernet backchannel to allow:

- Firmware updates
- Command line interface
- Scripting
- Timing analysis
- Packet capture

A central test server manages and controls devices.

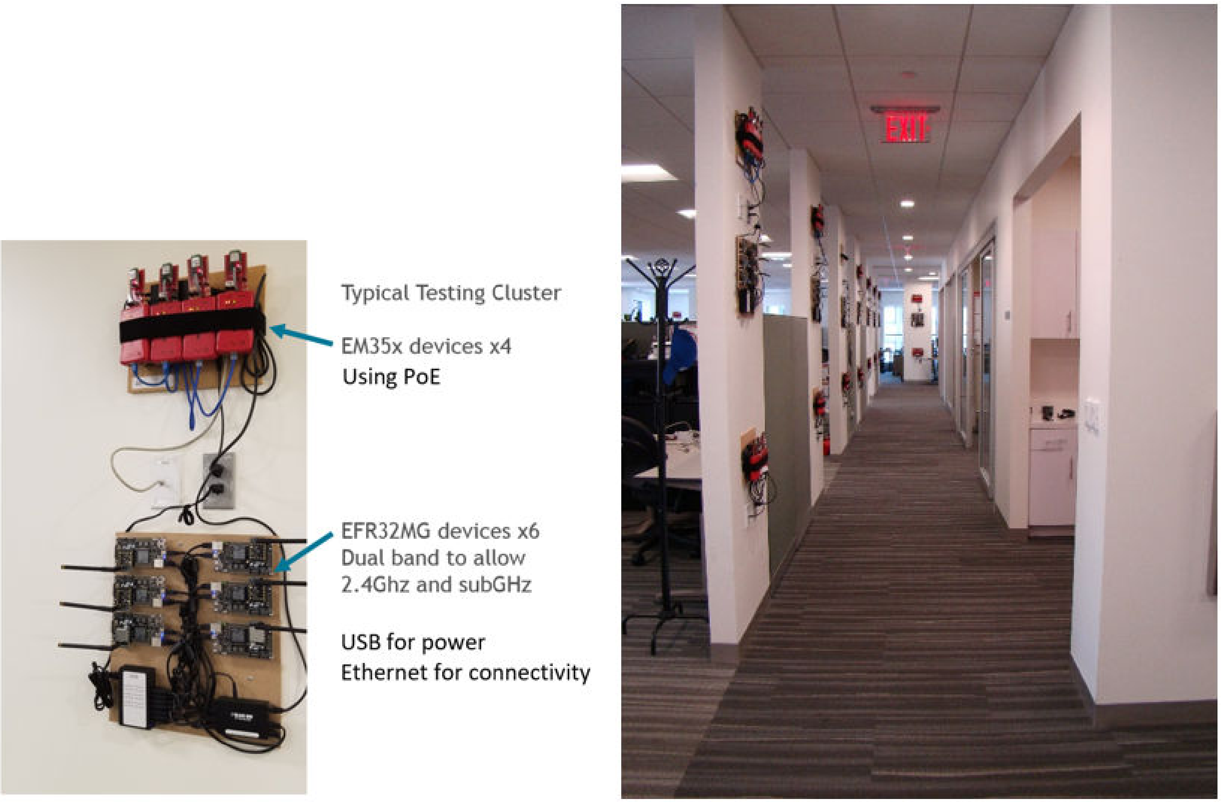


Figure 2.5. Large Network Test Setup

Coexistence and Interference Testing

Testing should also consider other wireless systems that may be operating in the expected environment. Almost any type of typical installation is likely to have to share the 2.4 GHz band with Wifi or 802.11 type traffic, Bluetooth, cordless phones, baby monitors and the many other devices that share the 2.4 GHz band. Depending on the expected field conditions, consider including any of these devices during system-level testing.

2.4 Moving to Beta and Field Trials

2.4.1 Hardware and Test System for Larger System Testing

Beyond the more dedicated test systems discussed above, it is necessary to move to a wider system level test environment within development and then within initial field testing.

While a stack provider like Silicon Labs can have dedicated setups for various conditions, as well as a larger test network with complete debug connectivity, this is not always practical for the typical application developer. Instead, the developer must focus on realistic tests that can be set up and operated within the budget and space constraints of their project.

First and most importantly, it is usually not practical to have full backchannel analysis capability for a wider scale system on custom hardware. This does not mean the developer should give up on the use and analysis of debug data, just that it has to be acquired in different means. Some techniques are:

1. Full Sniffer-Based Environment - If the custom hardware does not have any access for debug data, dedicated sniffers can be used. To be useful they should be spread throughout the system to capture and send as much data as possible back to one central log file.
2. Partial Sniffer and Debug Data - If some of the custom hardware has accessible debug ports, a hybrid system of some debug ports and additional sniffers to fill out the network picture can be used. This is especially valuable if some central control devices can be monitored using debug while more remote areas of the network are monitored using sniffers.
3. Full Debug Environment - If problems are seen with either of the above setups, ask the stack provider to operate the application in their more dedicated test network with full debug access. While this involves porting the application to development hardware, it does provide full debug capabilities for more difficult problems.

For any of the test environments used for the full system testing, several critical items must be included for proper testing and debugging. These include:

1. Mechanisms to activate typical application-level operations. The full system setup should provide scripting or other means to force application-level actions, so the system can be repeatedly tested. A problem that is seen once but cannot be replicated because of an uncontrolled test environment is difficult to resolve.
2. Centralized logging of all sniffer and debug information. Where possible it is important to have a central computer that is collecting data into a single log file. This is critical for time-stamping events, duplicate detection and removal, and proper analysis of the events leading up to a problem.
3. Means for simple code upgrade in all devices in the network. The best designed system will have bugs and features that require upgrading of software in the test network. A means to upgrade all devices easily offers a faster development and testing process.

2.4.2 Reproducing Common Field Conditions or Problems

Typical field problems occur due to several conditions that are not replicated in testing prior to initial field deployments. Evaluation under expected conditions while still in a controlled test setup saves many hours and later trips to the field installations.

Many of these conditions occur under error conditions in the network that were not tested in initial testing.

The most common problems Silicon Labs has experienced in field settings are as follows:

1. Multi-hop Performance - While it sounds unusual, many customers operate on desktop test networks that are generally one hop, and do not test multi-hop performance until an actual field installation occurs. Adding hops in the network obviously adds routing complexity that the network is intended to handle. However, it also increases packet latency and the response time an application can expect. If the application cannot handle this higher latency then message failures or excessive retries occur in the field.
2. Density or Sparsity of Network - Desktop testing of units, even those units that are on the final field hardware design, often does not replicate expected field conditions. Often devices are installed in walls or ceilings, within metal enclosures, with metal panels or shields over the device, or more widely spaced than any lab testing. In addition, use of particular network features such as broadcasts or multicasts is impacted by the density or sparsity of the network.
3. Required Application Interrupt Timing Under Network Loading - Some applications require tight control over interrupt latency to perform critical functions. However, the networking stack also requires interrupt-level processing on receipt of messages. While the network has the ability to buffer messages, this buffering is limited based on the memory of the device. Under actual network conditions, nodes that run out of buffer space temporarily lose the ability to send and receive messages. Alternatively, servicing network-level interrupts over application level interrupts results in poor application-level performance.
4. Excessive Network Traffic - A common complaint is a network that appeared to be operating well has a decrease in throughput or increase in latency. Upon investigation excessive traffic in the network resulted in congestion, lost messages and degraded performance. The most common reason is ongoing broadcasts or multicasts within the network. Failed messages can then lead to more broadcasted route discoveries, leading to further decreases in performance.

2.4.3 Initial Field Deployments

Initial field deployments are a critical step in the development process. These tests validate decisions and testing done throughout the development process and uncover any issues prior to wider deployment. These deployments are normally restricted to several sites, and are carefully monitored and controlled.

The following items are critical for initial field deployments:

1. Monitoring and Analysis Plan in Place - The initial deployments of a field system require the ability to monitor and analyze the system performance. This can be done at the system level by keeping track of application-level failures, or it can be done at a lower level by including some level of sniffers or debug capability in the field network.
2. Clear Operational Criteria and Success Criteria - Before installing a field system, the operating criteria and success or failure criteria should be established so these items can be monitored throughout the deployment.
3. Ability to Escalate Problems - Many networks have third party components or systems, or critical subsystems such as the networking stack. A clear process for escalation and resolution of bugs is critical to ensure issues are identified and closed in a timely manner.

2.4.4 Release Testing and Criteria for Release

For release of a wireless product, it is important to follow a progressive testing process that starts with desktop testing, and proceeds through laboratory testing, initial field deployments, and release testing. The earlier a problem is identified the simpler it is identify and resolve. Bugs are estimated to take 10 to 50 times longer to resolve in a field deployment than in laboratory or office testing.

The release process is an accumulation of the testing discussed above. However, it is typical to have some final test process and validation for final release. This criterion varies among different companies but often includes the following elements:

1. Normal System Operation - Typically this involves running a normal system in a typical environment for some period of time with no failures.
2. Power Cycling and Restart - This testing is done on a repetitive basis to ensure uncontrolled power loss and restarting does not result in failed devices. Typically Silicon Labs sees this done on a small network with automated power cycling. No failures are allowed.

The following table shows which testing strategies are employed during the different phases of development.

Table 2.1. Testing Strategies During Various Development Phases

Monitor Mechanism	Single Device Development	Desktop Development	Lab Testing	Initial Field Testing	Release Testing
Full debug access	XX	XX			
Partial debug			XX	XX	XX
Sniffer based					XX
Dedicated Test Types					
High density			XX		XX
Multi-hop		XX	XX		XX
Mobile node			XX	XX	XX
Large Network			XX	XX	
Coexistence / Interference			XX	XX	
System Test Cases					
System Start Up		XX		XX	XX
Normal operation		XX		XX	XX
Security			XX	XX	
Stressed Operation			XX		XX
Power failure and restart		XX			XX
Performance		XX		XX	
Typical failure mechanisms		XX			XX

The following figure shows how to set up a wired network as described in this document.

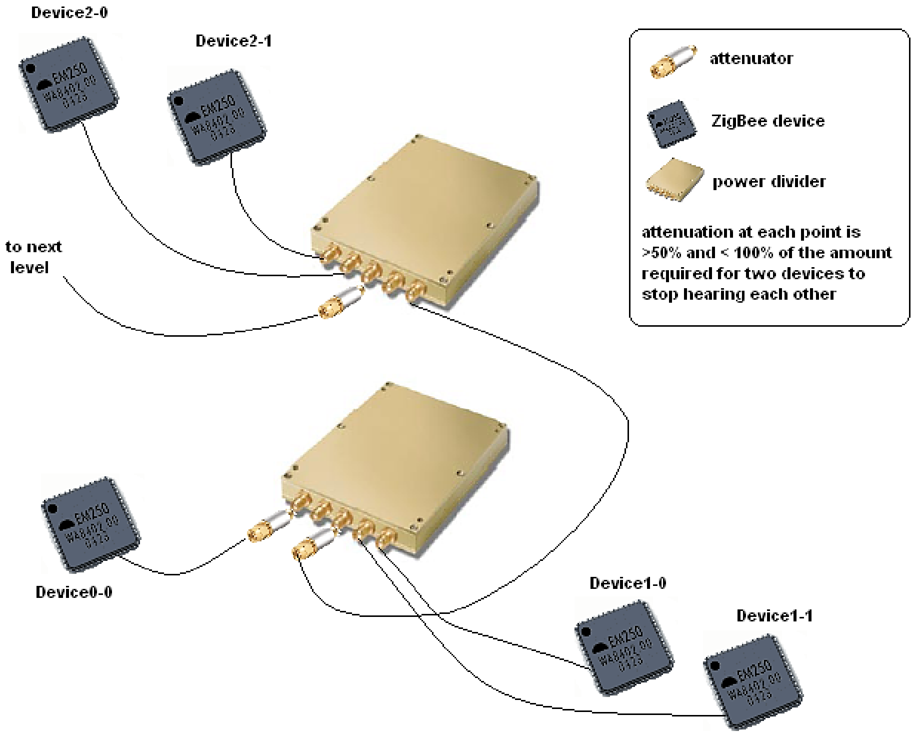

Device2-0
Device2-1
attenuator
ZigBee device
power divider
attenuation at each point is >50% and < 100% of the amount required for two devices to stop hearing each other
to next level
Device0-0
Device1-0
Device1-1

Figure 2.6. Setting up a Wireless Network

The following table gives a list of required parts needed to build the wired networks described in this document.

Table 2.2. Parts Used for Wired Testing

Part	Supplier	Part Number
Mini-Circuits 8-port power divider	Mini-Circuits	ZB8PD-4-S+
Mini-Circuits 4-port power divider	Mini-Circuits	ZN4PD1-50-S+
Mini-Circuits 2-port power divider	Mini-Circuits	ZN2PD2-50-S+
Mini-Circuits 10 dB attenuator	Mini-Circuits	VAT-10
Mini-Circuits 20 dB attenuator	Mini-Circuits	VAT-20
Mini-Circuits 30 dB attenuator	Mini-Circuits	VAT-30
Mini-Circuits terminator Male SMA	Mini-Circuits	ANNE-50L+
Richardson Elec 24" SMA-to-SMA patch cable	Richardson	1-3636-461-5224 (last two digits are the length)

3. Analyzing Network Analyzer Session Files

Simplicity Studio's Network Analyzer debugging tool provides a rich and flexible interface to Silicon Labs embedded networks, which helps you develop and debug new network applications.

This chapter shows how you can use the Simplicity Studio Network Analyzer tool to analyze and debug network behavior.

In this chapter, 'debug adapter' can mean either an ISA3 device for the EM35x or a WSTK board for the EFR32.

3.1 Network Analyzer Environment

The Network Analyzer environment contains the following work areas:

- Devices view - lists all debug adapters and their connected hardware that are accessible to Network Analyzer.
- Capture sessions - contain data captured from a node or set of nodes.
- Editor panes - display the data of a given capture session.

3.1.1 Adapters View

On startup, Simplicity Studio automatically discovers and lists all debug adapters that are connected to the local subnet. You can also configure Simplicity Studio to discover debug adapters on other subnets.

3.1.2 Capture Sessions

Network Analyzer can display one or more capture sessions. Each capture session stores data that it receives from one or more nodes, and displays it in the editor panes.

Network Analyzer supports two types of capture sessions:

- Live sessions display data as it is captured; the display is continuously refreshed as new data arrives. Network Analyzer maintains one live session at a time.
- Saved sessions contain captured data that has been saved to permanent storage in a named .isd file. You can load a saved session and play it back at any time.

3.1.3 Editor Panes

Editor panes provide different views of capture session data. Up to five editor panes can be open at any one time:

- **Map** provides a graphical view of the network, where nodes are displayed with their network identifiers. The map also displays network activity.
- **Transactions** displays high-level node interactions that might comprise multiple events.
- **Events** displays information about all packets transmitted and received during a capture session.
- **Event Detail** displays the decoded contents of the packet that is currently selected in the Events pane.
- **Hex Dump** displays the data of the selected event in raw bytes. Network Analyzer highlights bytes that map to the data currently selected in the Event Detail pane.

3.2 Capturing and Saving Data

You can capture data from the nodes of any connected debug adapters, either from one node at a time, or from multiple nodes simultaneously. You can also capture all network traffic over the current channel by capturing data from a connected sniffer node.

Note: The types of data captured depend on the capabilities of the node's radio chip and debug adapter.

3.2.1 Capturing from a Sniffer Node

A sniffer node has a sniffer application loaded on its wireless SoC, which enables it to capture all over-the-air transmissions between nodes over the designated channel. When you start capturing from a sniffer node, it captures all packets that are exchanged by the nodes on the designated channel.

Note: As of this writing, the EFR32 does not have a sniffer application.

3.2.2 Capturing from Multiple Nodes

You can capture data from multiple nodes simultaneously. This is typically done for nodes that belong to the same network.

Network Analyzer captures all incoming and outgoing packet data through the selected debug adapters, whether or not the host nodes have sniffer applications. The data includes failed transmissions, and debug messages from node applications that are compiled in debug mode.

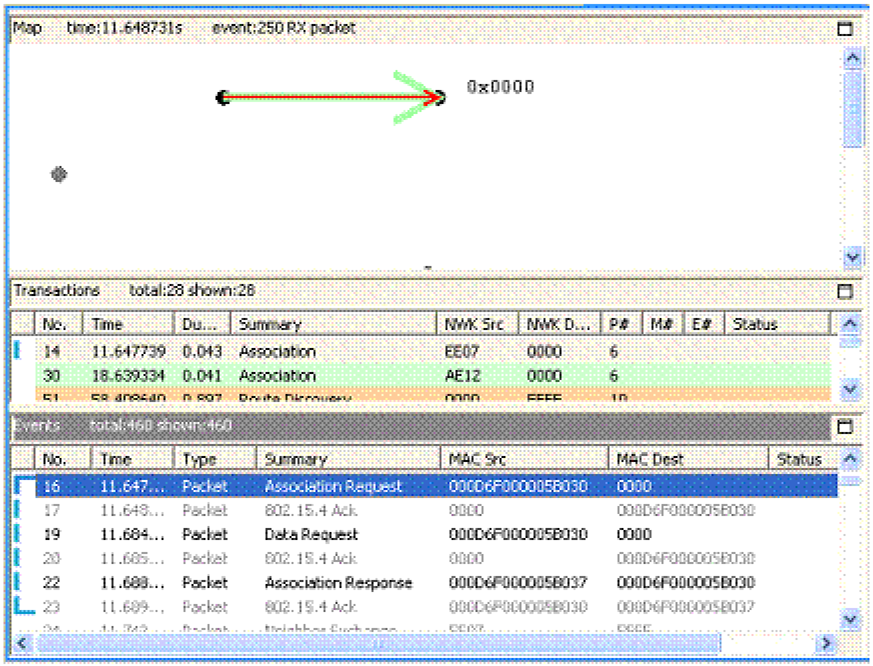
3.2.3 Perfect Trace Session

Capturing from all nodes in a network yields a perfect trace session. The session data compiles all incoming and outgoing data from each node in chronological real time, providing a richly layered view of all activity within a network. This can be especially useful for debugging a network while it undergoes development.

Note: A sniffer capture intercepts all over-the-air transmissions among network nodes; however, the sniffer cannot detect any data transmissions that are known only to their senders such as messages that fail to arrive at their destination, and debug data such as API traces. These are captured in a perfect trace session.

3.3 Reviewing Capture Session Events

The following figure shows part of a capture session, where events 16 through 23 make a single Association transaction (shown in detail in the following table). The transaction begins with a node's request to a coordinator node to join its network. Each message is paired with an 802.15.4 acknowledgement from the message recipient. The transaction ends when the requesting node acknowledges receipt of the coordinator's Association response:



Map time:11.648731s event:250 RX packet

Transactions total:28 shown:28

No.	Time	Du...	Summary	NWK Src	NWK D...	P#	M#	E#	Status
14	11.647739	0.043	Association	EE07	0000	6			
30	18.639334	0.041	Association	AE12	0000	6			
51	58.408640	0.897	Route Discovery	0000	FFFF	10			

Events total:460 shown:460

No.	Time	Type	Summary	MAC Src	MAC Dest	Status
16	11.647...	Packet	Association Request	000D6F000005B030	0000	
17	11.648...	Packet	802.15.4 Ack	0000	000D6F000005B030	
19	11.684...	Packet	Data Request	000D6F000005B030	0000	
20	11.685...	Packet	802.15.4 Ack	0000	000D6F000005B030	
22	11.688...	Packet	Association Response	000D6F000005B037	000D6F000005B030	
23	11.689...	Packet	802.15.4 Ack	000D6F000005B030	000D6F000005B037	
24	11.742...	Packet	Neighbor Exchange	EE07	FFFF	

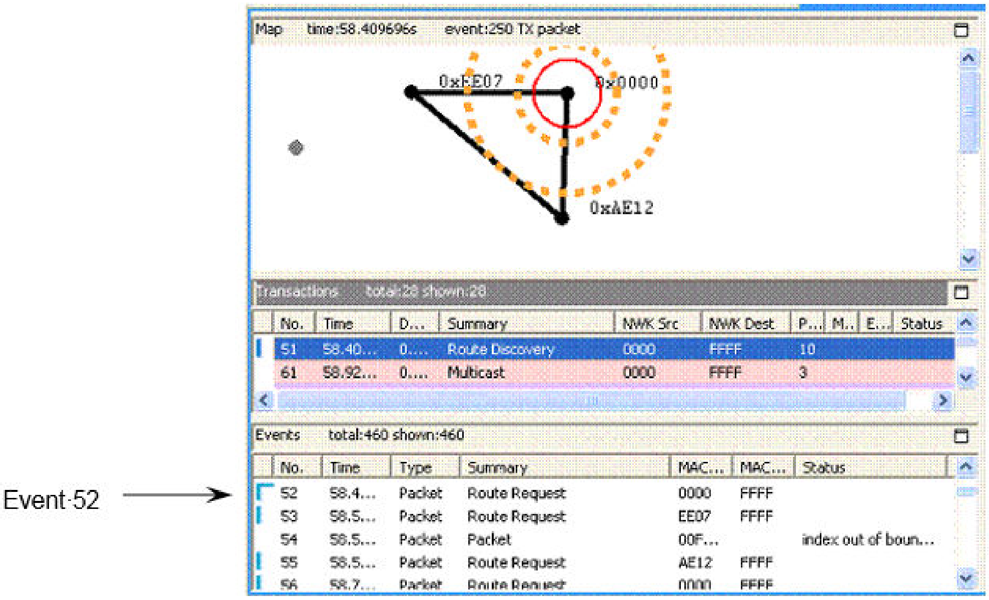
Figure 3.1. Association Transaction

Table 3.1. Association Transaction

Event	Summary	Description
16	Association request	Node requests permission to join the network from the network coordinator.
17	802.15.4 Ack	Coordinator acknowledges message receipt.
19	Data request	Node requests network data.
20	802.15.4 Ack	Coordinator acknowledges message receipt.
22	Association response	Coordinator permits joining.
23	802.15.4 Ack	Node acknowledges message receipt.

3.3.1 Viewing Connectivity

The toolbar's link connectivity button lets you view connections between network nodes. The connectivity diagram shows cumulative connections up to the current event. The following figure shows connections that were established between the network's three-nodes as of Event 52:



Transactions total:28 shown:28

No.	Time	D...	Summary	NWK Src	NWK Dest	P...	M..	E...	Status
51	58.40...	0....	Route Discovery	0000	FFFF	10			
61	58.92...	0....	Multicast	0000	FFFF	3			

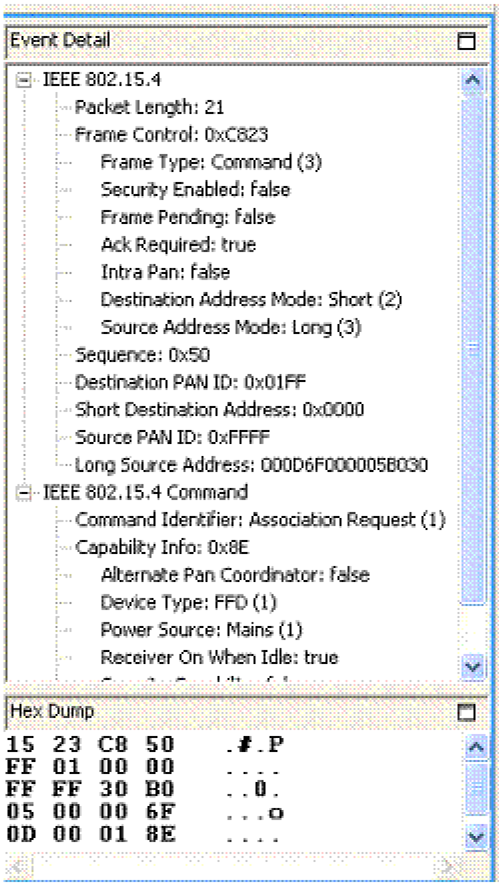
Events total:460 shown:460

No.	Time	Type	Summary	MAC...	MAC...	Status
52	58.4...	Packet	Route Request	0000	FFFF	
53	58.5...	Packet	Route Request	EE07	FFFF	
54	58.5...	Packet	Packet	00F...		index out of boun...
55	58.5...	Packet	Route Request	AE12	FFFF	
56	58.7...	Packet	Route Request	0000	FFFF	

Figure 3.2. Network Connectivity

3.4 Viewing Packet Data

You can view the contents of an individual packet in the Event Detail and Hex Dump editor panes. These display packet data in decoded format and raw bytes, respectively. For example, Network Analyzer shows the Association Request data for Event 16 in the following figure.



Event Detail

```
IEEE 802.15.4
    Packet Length: 21
    Frame Control: 0xC823
        Frame Type: Command (3)
        Security Enabled: false
        Frame Pending: false
        Ack Required: true
        Intra Pan: false
        Destination Address Mode: Short (2)
        Source Address Mode: Long (3)
    Sequence: 0x50
    Destination PAN ID: 0x01FF
    Short Destination Address: 0x0000
    Source PAN ID: 0xFFFF
    Long Source Address: 000D6F000005B030
IEEE 802.15.4 Command
    Command Identifier: Association Request (1)
    Capability Info: 0x8E
        Alternate Pan Coordinator: false
        Device Type: FFD (1)
        Power Source: Mains (1)
        Receiver On When Idle: true
```

Hex Dump

```
15 23 C8 50    .#.P
FF 01 00 00    ....
FF FF 30 B0    ..0.
05 00 00 6F    ...o
0D 00 01 8E    ....
```

Figure 3.3. Decoded and Hexadecimal Content of an Association Request Message Packet

3.4.1 Filtering Captured Data

By default, the Events pane displays all session events. You can build and apply filters that constrain Network Analyzer to show only events that are of interest. By filtering events, you can analyze results more efficiently.

3.4.2 Setting Filters

Each capture session has its own filter settings. When you change a session's filters, Network Analyzer immediately refreshes the display. When you exit Network Analyzer, all session filters are cleared and must be reapplied when you restart.

Network Analyzer provides two ways to edit filters:

- **Filter Manager:** Maintains a set of saved filters that you can review and edit. You can also add new filters. You specify any of the saved filters for display on the Filters menu, where they are accessible for use in one or more sessions.
- **Filter Bar:** An editor that attaches to a given session, where you can enter one or more filter expressions on the fly. Network Analyzer discards filter bar expressions for all sessions when it exits.

3.4.3 Quick Filters

Network Analyzer also provides several preset context-sensitive quick filters that are available from the Transactions and Events editor panes. You access these filters by right-clicking on an event or transaction and choosing a context menu option.

The quick filter options that are available depend on the transaction or event you select. You can specify to hide all events/transactions of the selected type, or to show only that type. Further, if you select an event such as a neighbor exchange that has a source and/or destination address, the context menu also contains options that let you filter events according to that address. This lets you isolate data for a given node.

3.4.4 Filtering Message Types

It is often useful to hide messages that are not relevant to the current analysis. For example, you typically don't need to view neighbor exchanges-stack-level messages that are exchanged among nodes.

By default, the Filters menu has a single saved filter option, **Hide Neighbor Exchange**, which the Filter Manager defines as follows:

```
!isPresent(neighborExchange)
```

By choosing this option, Network Analyzer hides all neighbor exchange messages from the Events pane. You can customize the Filters menu to include other saved filters. You can also set filters manually in the filter bar.

3.4.5 Isolating Node Data

You might want to isolate all messages for a given node, for one of the following reasons:

- Evaluate the amount of traffic between it and other nodes.
- Analyze traffic pattern anomalies.
- Determine whether and when it stopped sending or receiving messages.

To show only traffic for a given node:

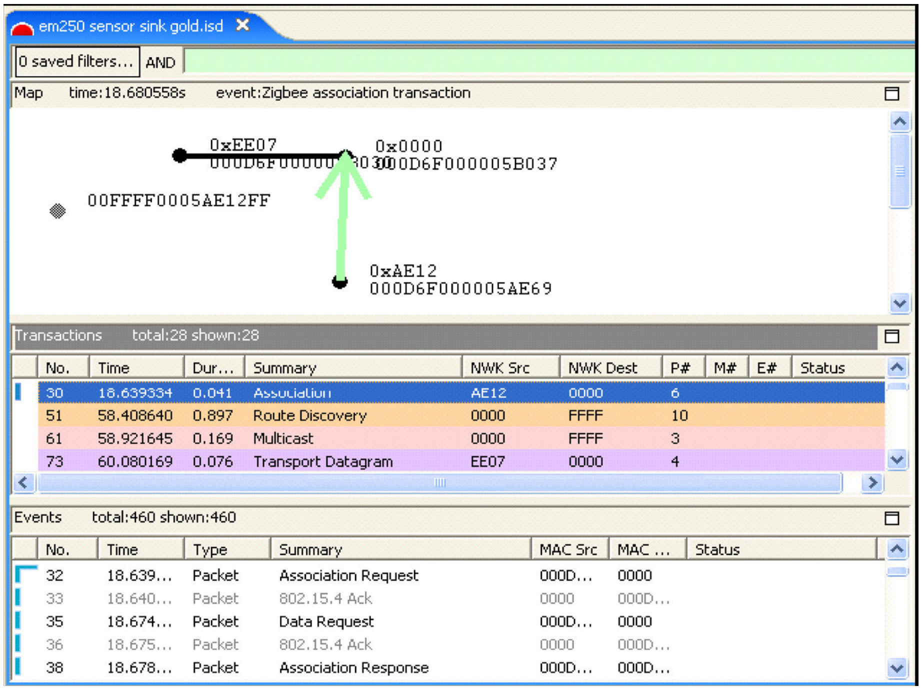
1. Right click on a message that specifies a source and destination.
2. From the context menu, choose one of the following options:
   - Show only destination: node-short-ID
   - Show only source: node-short-ID

For example, to view only events from node AE12, as shown in the following figure:

1. Right-click on transaction 30.
2. From the context menu, choose Show only source AE12.

The filter bar shows a filter expression like this:

fifteenFour.source == 44562



Transactions total:28 shown:28

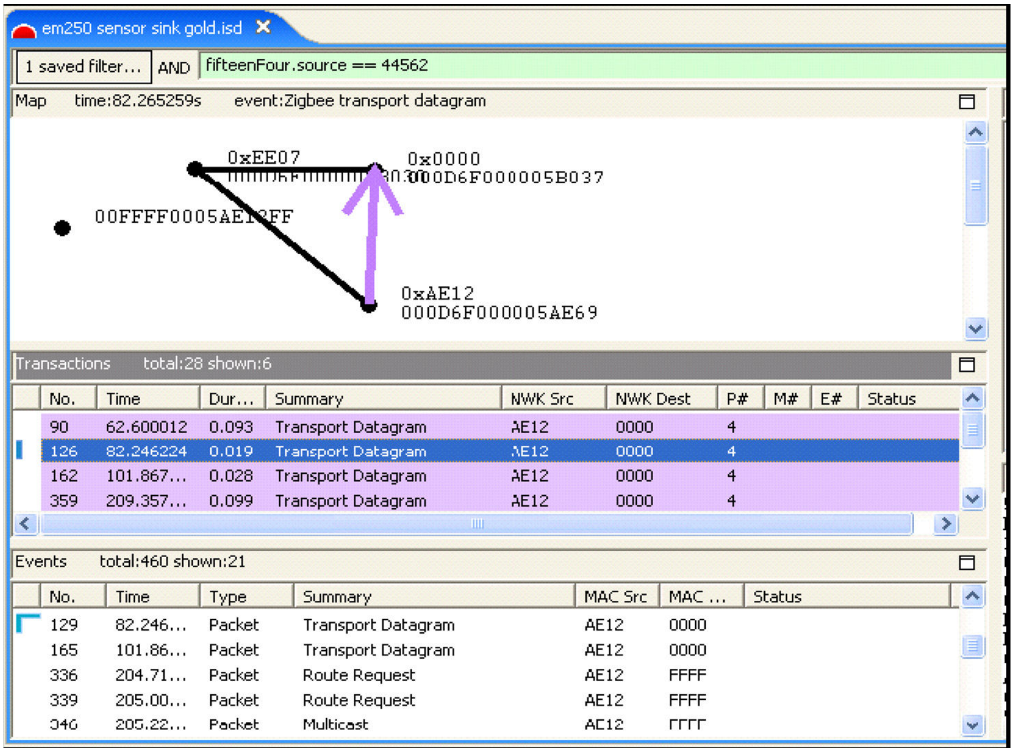
No.	Time	Dur...	Summary	NWK Src	NWK Dest	P#	M#	E#	Status
30	18.639334	0.041	Association	AE12	0000	6			
51	58.408640	0.897	Route Discovery	0000	FFFF	10			
61	58.921645	0.169	Multicast	0000	FFFF	3			
73	60.080169	0.076	Transport Datagram	EE07	0000	4			

Events total:460 shown:460

No.	Time	Type	Summary	MAC Src	MAC ...	Status
32	18.639...	Packet	Association Request	000D...	0000	
33	18.640...	Packet	802.15.4 Ack	0000	000D...	
35	18.674...	Packet	Data Request	000D...	0000	
36	18.675...	Packet	802.15.4 Ack	0000	000D...	
38	18.678...	Packet	Association Response	000D...	000D...	

Figure 3.4. Selecting a Transaction in Order to Filter on Messages from a Source Node

You can further refine the filter by ANDing it with other conditions.-For example, you can specify to hide all neighbor exchange messages from this node by choosing the menu option Filters | Hide Neighbor Exchange, as shown in the following figure.



em250 sensor sink gold.isd

1 saved filter... AND fifteenFour.source == 44562

Map time:82.265259s event:Zigbee transport datagram

Transactions total:28 shown:6

No.	Time	Dur...	Summary	NWK Src	NWK Dest	P#	M#	E#	Status
90	62.600012	0.093	Transport Datagram	AE12	0000	4			
126	82.246224	0.019	Transport Datagram	AE12	0000	4			
162	101.867...	0.028	Transport Datagram	AE12	0000	4			
359	209.357...	0.099	Transport Datagram	AE12	0000	4			

Events total:460 shown:21

No.	Time	Type	Summary	MAC Src	MAC ...	Status
129	82.246...	Packet	Transport Datagram	AE12	0000	
165	101.86...	Packet	Transport Datagram	AE12	0000	
336	204.71...	Packet	Route Request	AE12	FFFF	
339	205.00...	Packet	Route Request	AE12	FFFF	
340	205.22...	Packet	Multicast	AE12	FFFF	

Figure 3.5. Showing Only Messages from a Single Node, Excluding Neighbor Exchanges

By doing this, you can determine node-specific data, such as how many non-neighbor exchange events originate from that node and the intervals between them. After analyzing data for that node, you can clear all current filters and set filters on other nodes, in order to compile their data and activity.

Note: Filters are helpful for isolating network issues-for example, quickly ascertaining when a node stops sending data. However, to analyze an event in the context of network operations, you must clear all filters.

3.5 Enabling Debug Output in Simplicity Studio

Debug output from your application can be added to your compiled image. Once debug output is included in your compiled application image, it can be turned on and off at runtime in your running application. In order to compile debug output into your application in the Simplicity Studio IDE, go to the **Printing and CLI** tab in your application configuration. Check the **Enable Debug Printing** checkbox, which turns all of the debug printing output on or off globally (all areas). The Debug Configuration section lists the functional areas available to produce debug output from the code. Functional areas include General Purpose debug printing, such as security or reporting, individual plugin debug printing, and application-specific debug printing.

Once you have selected the sections of the debug code that you wish to compile into your application, take a look at the generated <application>.h header file. This generated file contains the defines with associated mask values for the debug areas you have chosen to include. Some frameworks provide a means of turning specific debug areas on or off at any time from the command line. For example the EmberZNet ZCL application framework provides a `debugprint` command for this purpose. The `debugprint` command is documented along with all the other CLI commands in the Application Framework API Reference.

For example, if you compile in debug printing for Core, Application, and Attributes you will see a section in the <application>.h header file as shown:

```
// Global switch
#define EMBER_AF_PRINT_ENABLE
// Individual areas
#define EMBER_AF_PRINT_CORE 0x0001
#define EMBER_AF_PRINT_APP 0x0002
#define EMBER_AF_PRINT_ATTRIBUTES 0x0004
#define EMBER_AF_PRINT_BITS { 0x07 }
#define EMBER_AF_PRINT_NAMES { \
  "Core",\
  "Application",\
  "Attributes",\
  NULL\
}
#define EMBER_AF_PRINT_NAME_NUMBER 3
```

If your framework's CLI provides the `debugprint` command, you can then use the value 0x0004 in this example to turn attribute printing on and off on the command line using the following command lines:

```
> debugprint on 0x0004
> debugprint off 0x0004
```

Please note that you may only turn on and off debug printing that has previously been compiled into the application. There is no way to turn debug printing on and off for a specific debug printing area if that area has not been compiled into the application in advance.

3.6 Multinetwork Considerations

If your application uses nodes operating on multiple networks, this information can be reflected when reviewing capture sessions. Currently, however, Network Analyzer cannot auto-detect multinetwork nodes. At the onset, unless all traditional, conceptual nodes constituting the multinetwork node are assigned EUI64s, each conceptual node shows up as a separate node in the Map editor pane. Each such node must be assigned the **same** EUI64 by right-clicking on the node in the Map editor pane and selecting “Assign EUI64…” A dialog pops up that facilitates the assignment, and a checkbox indicating “Multinetwork EUI64” must be checked to inform Network Analyzer that the node is indeed a multinetwork node. This sequence is illustrated in the following two figures.

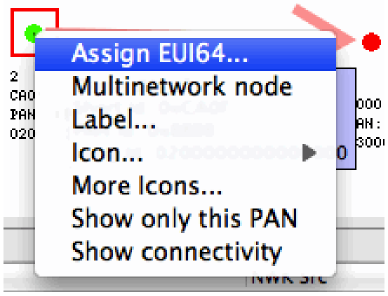
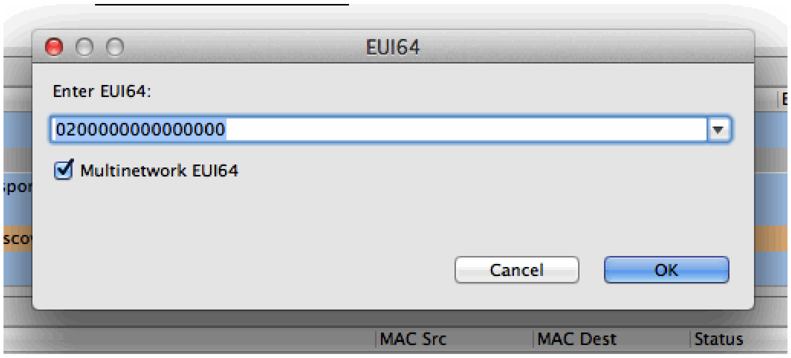


Figure 3.6. Assigning a Multinetwork EUI64



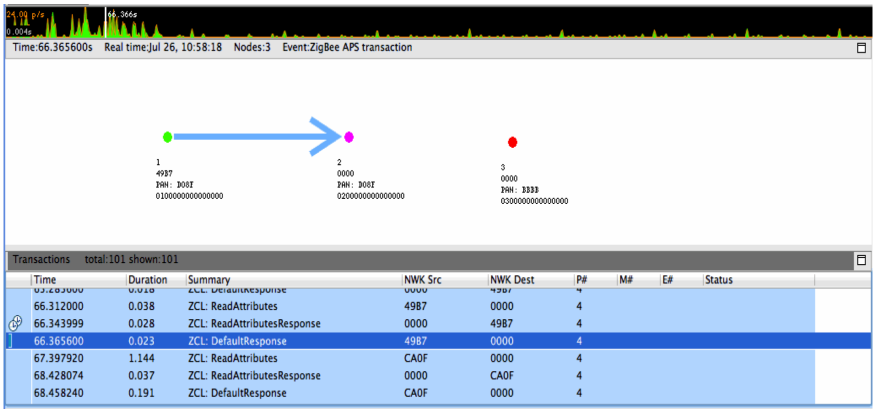
Enter EUI64:

0200000000000000

☑ Multinetwork EUI64

Figure 3.7. EUI64 Dialog

Once all the constituent conceptual nodes have been assigned the same EUI64, Network Analyzer coalesces them into one node in the Map editor pane. Alternatively, if the all the conceptual nodes know the EUI64 or all have been coalesced but Network Analyzer has not been informed that the node is multinetwork, the “Multinetwork node” menu item, seen in Figure 3.6 Assigning a Multinetwork EUI64 on page 22, can be toggled. Once a node is known to Network Analyzer to be multinetwork, it is indicated as such by being colored magenta in the Map editor pane, as seen in the following two figures. These figures also demonstrate the multinetwork capabilities of the Network Analyzer Map editor pane by illustrating events involving the same node on multiple PANs.

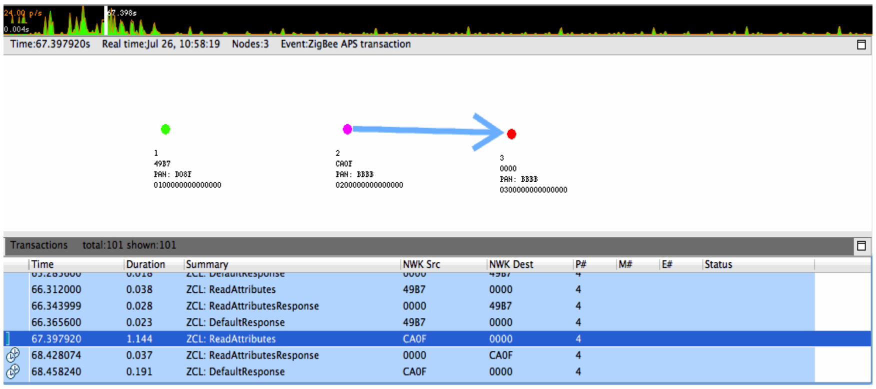


Time:66.365600s Real time:Jul 26, 10:58:18 Nodes:3 Event:ZigBee APS transaction

Transactions total:101 shown:101

Time	Duration	Summary	NWK Src	NWK Dest	P#	M#	E#	Status
65.283000	0.018	ZCL: DefaultResponse	0000	49B7	4			
66.312000	0.038	ZCL: ReadAttributes	49B7	0000	4			
66.343999	0.028	ZCL: ReadAttributesResponse	0000	49B7	4			
66.365600	0.023	ZCL: DefaultResponse	49B7	0000	4			
67.397920	1.144	ZCL: ReadAttributes	CA0F	0000	4			
68.428074	0.037	ZCL: ReadAttributesResponse	0000	CA0F	4			
68.458240	0.191	ZCL: DefaultResponse	CA0F	0000	4			

Figure 3.8. Multinetwork Node in the Map Editor Pane, First PAN

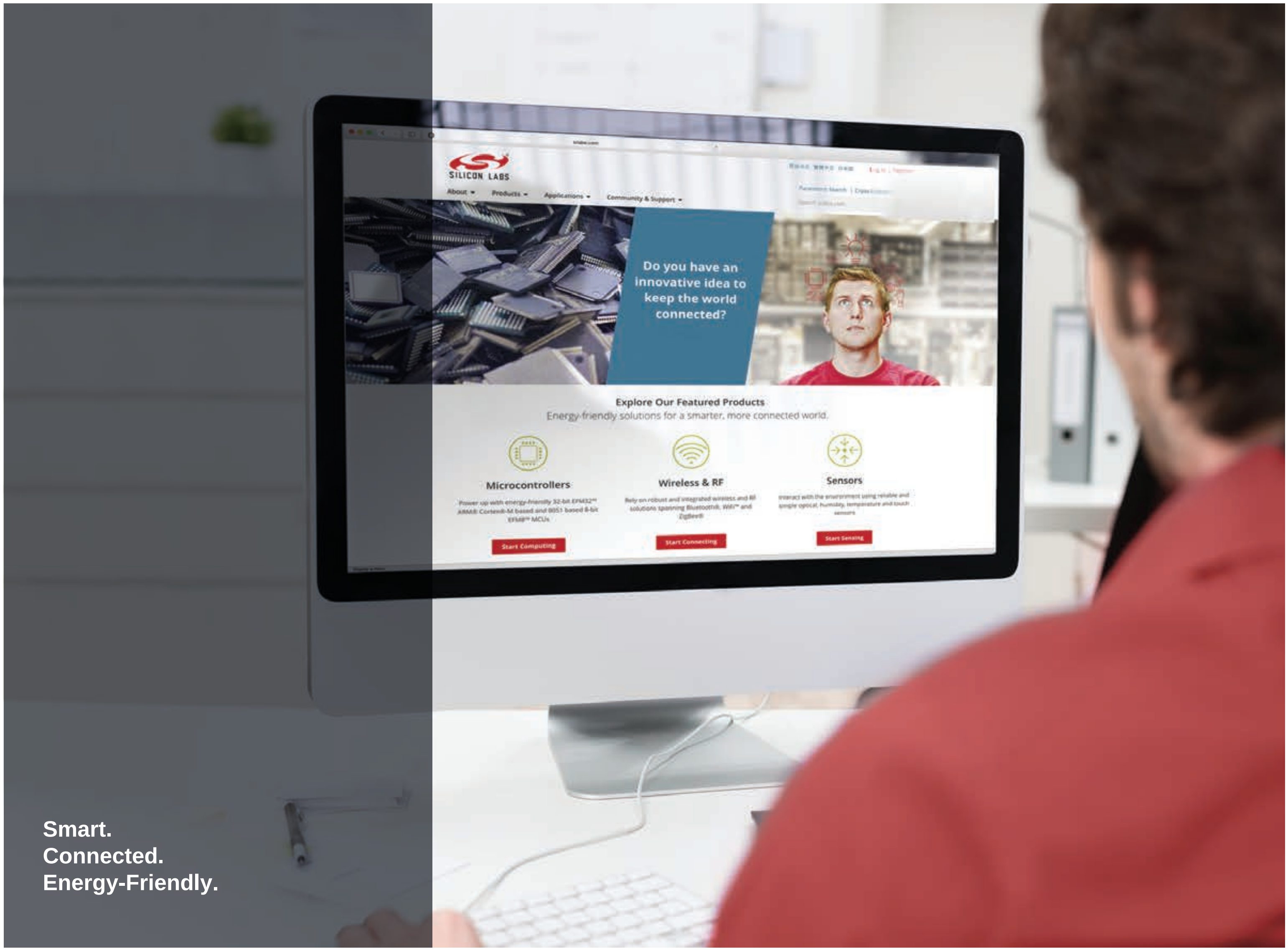


Time:67.397920s Real time:Jul 26, 10:58:19 Nodes:3 Event:ZigBee APS transaction

Transactions total:101 shown:101

Time	Duration	Summary	NWK Src	NWK Dest	P#	M#	E#	Status
65.283000	0.018	ZCL: DefaultResponse	0000	49B7	4			
66.312000	0.038	ZCL: ReadAttributes	49B7	0000	4			
66.343999	0.028	ZCL: ReadAttributesResponse	0000	49B7	4			
66.365600	0.023	ZCL: DefaultResponse	49B7	0000	4			
67.397920	1.144	ZCL: ReadAttributes	CA0F	0000	4			
68.428074	0.037	ZCL: ReadAttributesResponse	0000	CA0F	4			
68.458240	0.191	ZCL: DefaultResponse	CA0F	0000	4			

Figure 3.9. Multinetwork Node in the Map Editor Pane, Second PAN


SILICON LABS
About
Products
Applications
Community & Support
Do you have an innovative idea to keep the world connected?
Explore Our Featured Products
Energy-friendly solutions for a smarter, more connected world.
Microcontrollers
Wireless & RF
Sensors
Start Computing
Start Connecting
Start Sensing

Smart.
Connected.
Energy-Friendly.

Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

Disclaimer

Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice to the product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Without prior notification, Silicon Labs may update product firmware during the manufacturing process for security or reliability reasons. Such changes will not alter the specifications or the performance of the product. Silicon Labs shall have no liability for the consequences of use of the information supplied in this document. This document does not imply or expressly grant any license to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any FDA Class III devices, applications for which FDA premarket approval is required, or Life Support Systems without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons. Silicon Labs disclaims all express and implied warranties and shall not be responsible or liable for any injuries or damages related to use of a Silicon Labs product in such unauthorized applications.

Trademark Information

Silicon Laboratories Inc.®, Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, ClockBuilder®, CMEMS®, DSPLL®, EFM®, EFM32®, EFR, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, Gecko OS, Gecko OS Studio, ISOmodem®, Precision32®, ProSLIC®, Simplicity Studio®, SiPHY®, Telegesis, the Telegesis Logo®, USBXpress® , Zentri, the Zentri logo and Zentri DMS, Z-Wave®, and others are trademarks or registered trademarks of Silicon Labs. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. Wi-Fi is a registered trademark of the Wi-Fi Alliance. All other products or brand names mentioned herein are trademarks of their respective holders.

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com