



Series 2: BG22 Bluetooth® SoCs and Modules

Next Generation IoT Wireless Connectivity

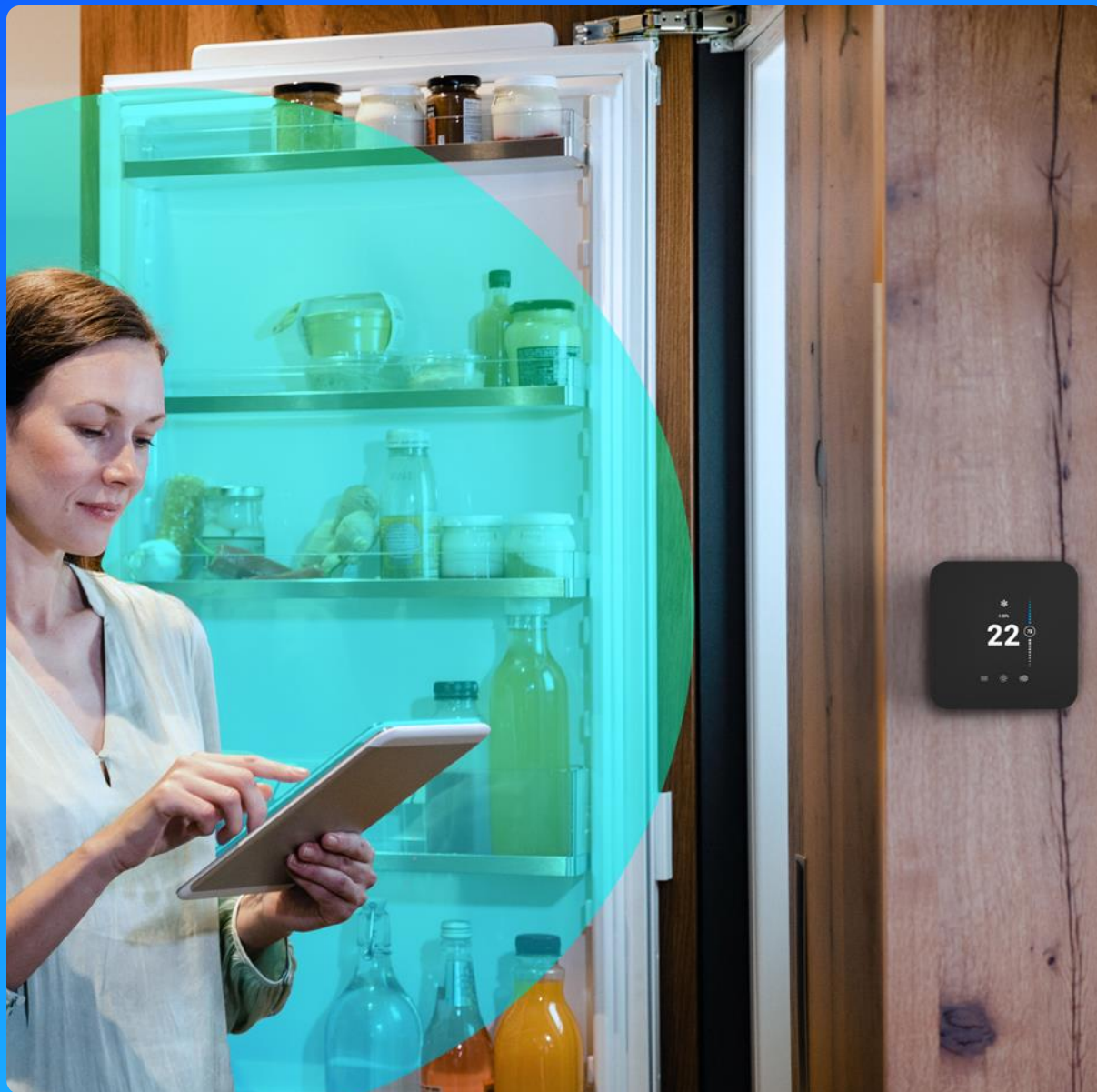


Introduction to Proprietary Wireless

- What is a Proprietary Wireless Application?
- Two EFR32 Application Development Paths: RAIL vs. Connect
- RAIL-based Application Development
 - Simplicity Studio v5
 - SSv5 Radio Configurator
 - API Feature Review
 - Walk-thru of Simple TRX RAIL sample application

High Level Overview

of EFR32 Proprietary Wireless



What is Proprietary Wireless and when is it Appropriate?

- **When the application demands**

- Backwards-compatibility with existing/legacy proprietary protocol(s)
- High degree of protocol optimization
 - For energy consumption
 - For wireless range
- Techtalks - discussing the advantages of having full control over the protocol:
 - <https://www.silabs.com/support/training/long-range-connectivity-using-proprietary-rf-solution>
 - <https://www.silabs.com/support/training/sub-ghz-proprietary-and-connect-software-stack>

- **... at the expense of**

- Requires a higher level of expertise, in-depth knowledge of RF, regulatory specifications, and protocol design.
- More difficult development, longer “time to market”
- Security holes that can remain hidden for a long time due to the difficulty of the analysis
 - But once discovered, exploiting them is usually easy (high obfuscation, not necessarily high security)

Typical Proprietary Wireless Solutions



Smart Meters



Home Automation and Security



Garage Door Openers



Public Infrastructure

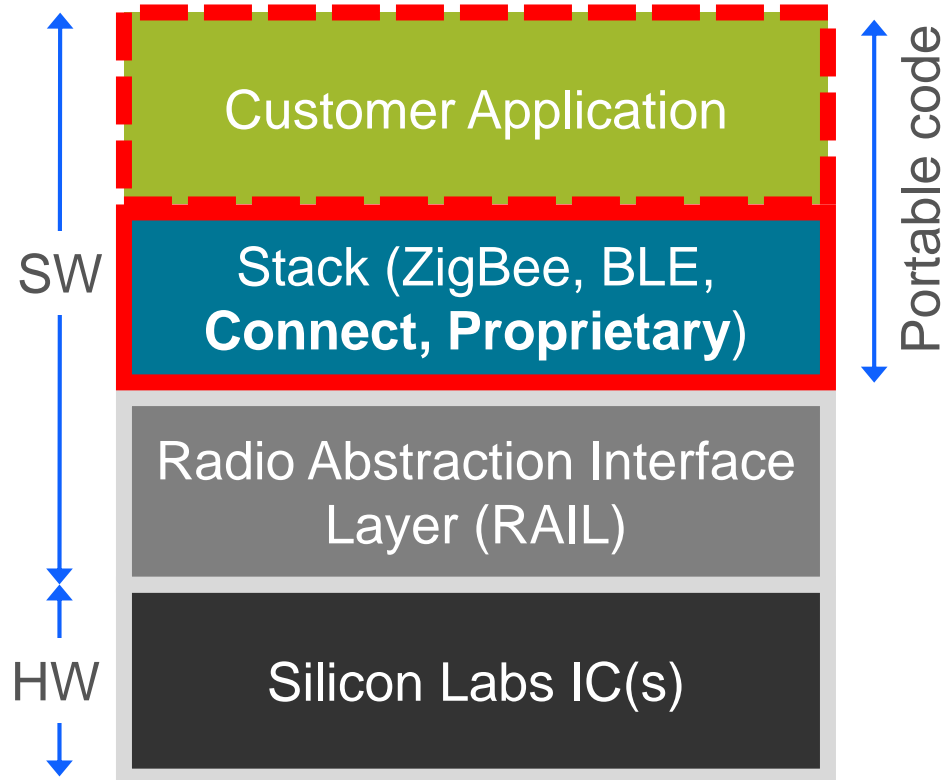


Agriculture



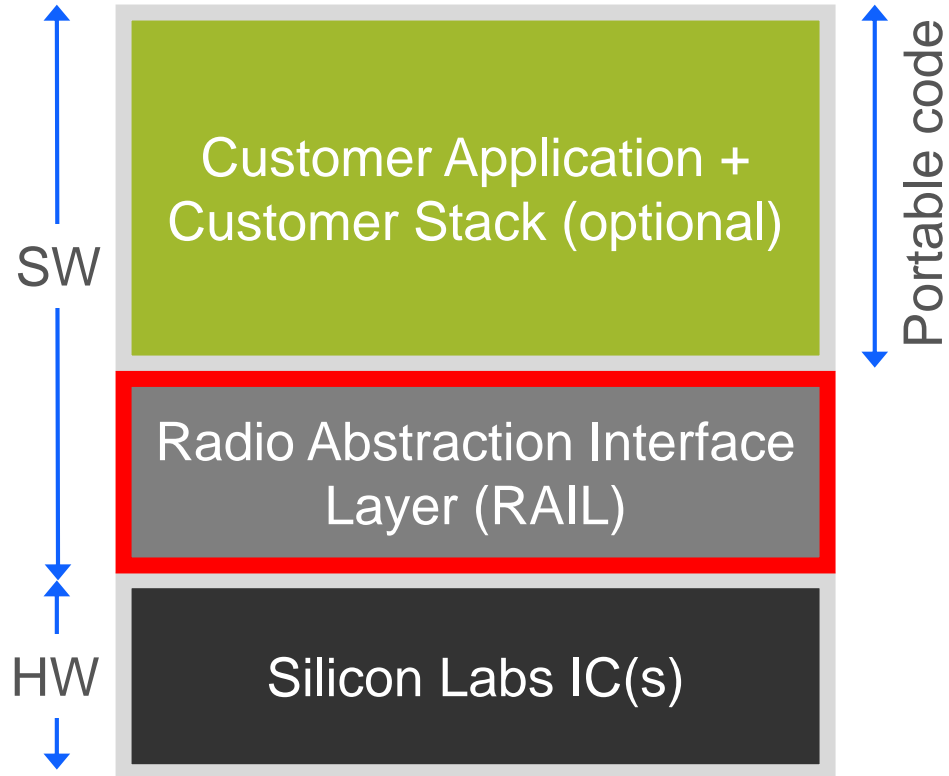
Asset Tracking & ESL

EFR32 Development Path #1: Connect-based Application



- Stack, up to the Network layer
 - Based on RAIL
 - Supports an extended star network topology
 - Configurable PHY (pre-set PHYs available for all ISM regions)
 - 15.4 based MAC (Frame format)
- Also supports MAC mode
 - Pure IEEE 802.15.4 MAC implementation
- Also includes some application layer features
 - Task scheduler
 - OTA bootloader image distribution

EFR32 Development Path #2: RAIL-based Application



- Radio Abstraction Interface Layer
- Library, used to access radio transceiver hardware
- Has some MAC features that can be accelerated by HW
 - Auto ACK
 - Address filtering
 - CSMA/CA or LBT
 - Scheduling and timestamping
- RAIL provides a common API across all supported chips
- All Silicon Labs stacks are implemented on top of RAIL

Best Path Depends on the Application Needs

RAIL

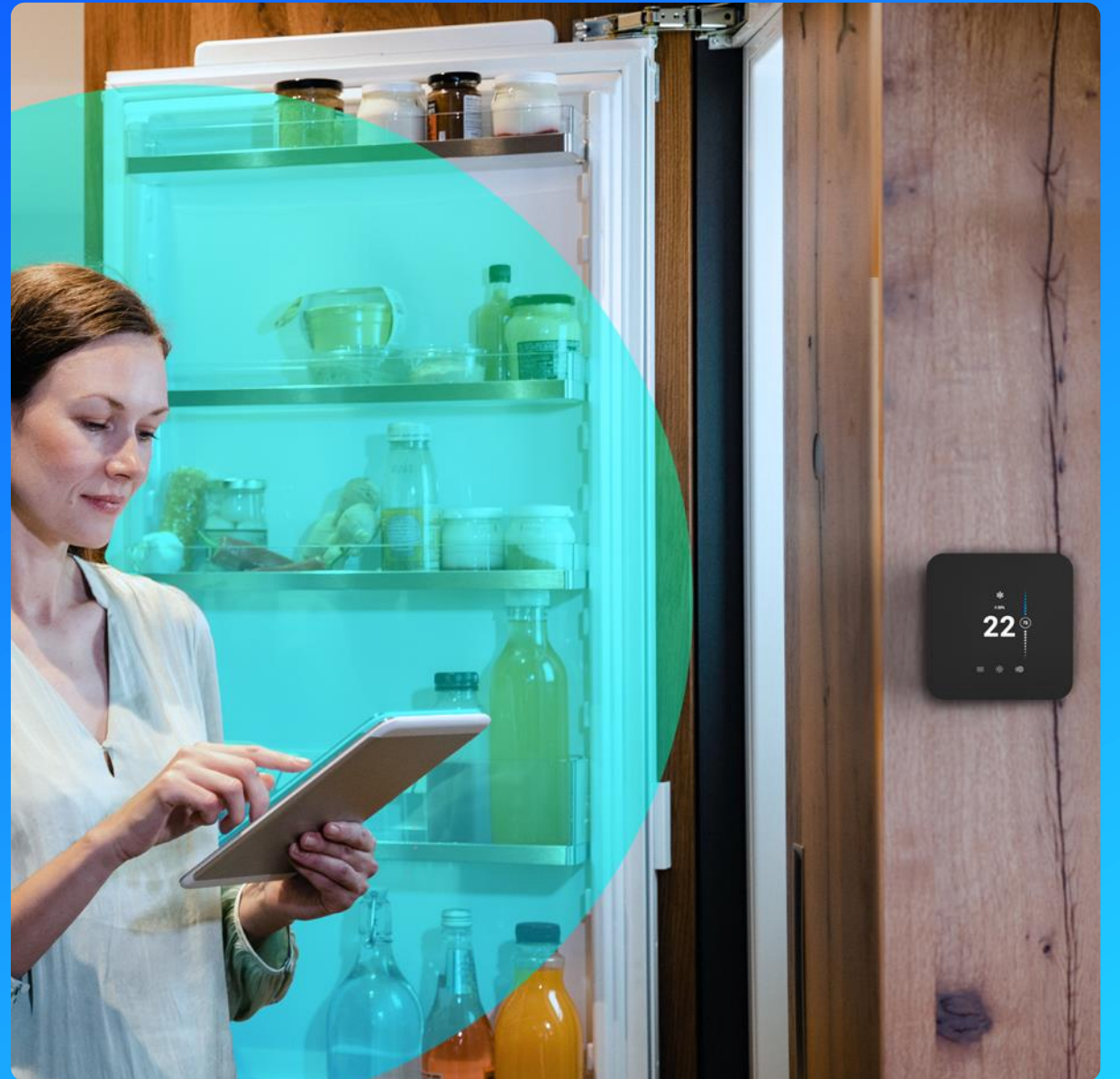
- Pros:
 - Very flexible
 - Support legacy proprietary systems
 - Can be based on custom PHY configuration
 - More efficient resource footprint
- Cons:
 - No network layer, so no multi-hop support
 - No application features like OTA
 - Security must be done in application

Connect

- Pros:
 - Full featured stack, including network layer
 - Task Scheduler
 - OTA bootloader
 - MAC provides 15.4 security
 - Preconfigured Worldwide PHYs
- Cons:
 - Fixed frame format, can't connect to existing networks
 - Not very flexible, e.g. difficult to set up deeper than EM2 sleep
 - RTC oscillator is required for scheduler
 - Larger resource footprint

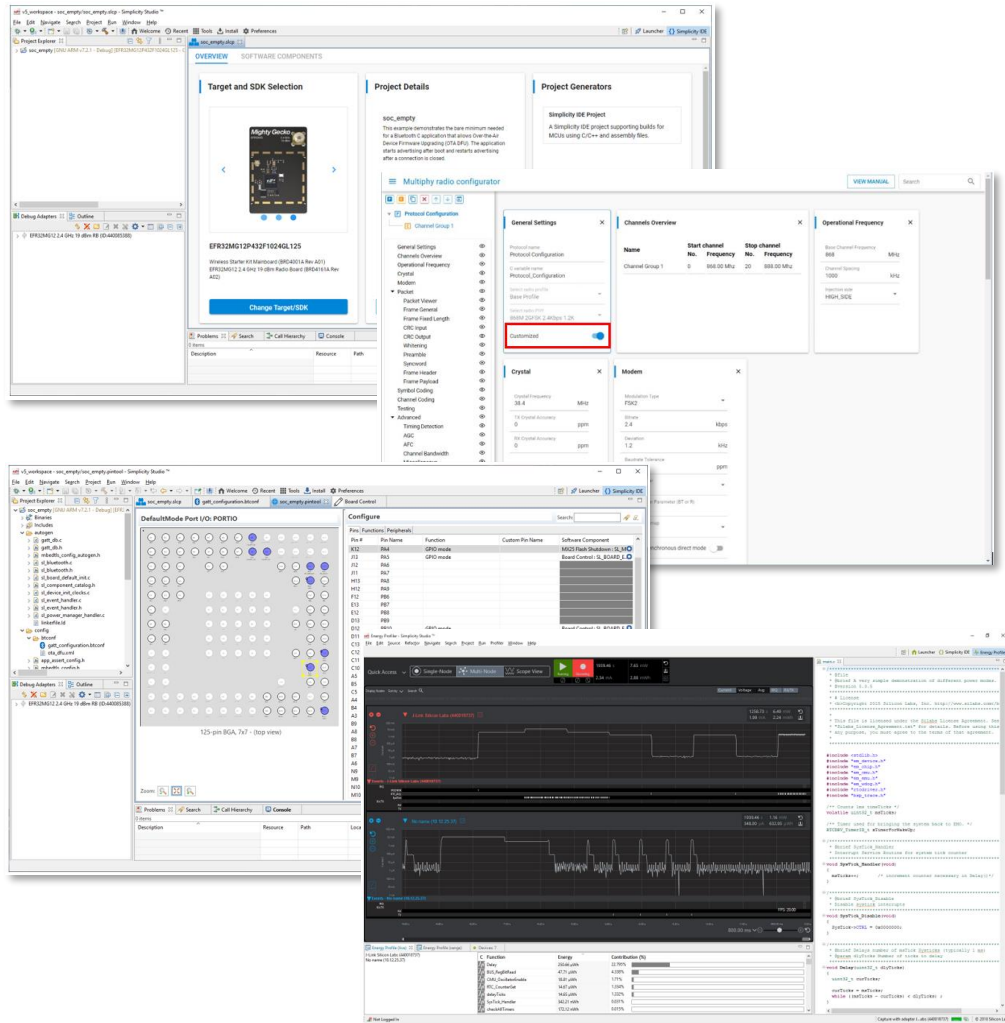
Creating a RAIL Project

In Simplicity Studio v5



Tools and API

SSv5 Tools



RAIL API

- Transmit/Receive
- Automatic State Transitions
 - E.g. automatically go to rx after tx
- Frame Buffering
 - Maintains buffer for both tx and rx
- Timekeeping, Timestamping and Timers
- Scheduled Transmit
- Scheduled Receive
- CCA with Retransmission
 - Supports CSMA/CA and LBT
- Address Filtering
 - With two fixed offset, max 4B address or 802.15.4 addressing
- Auto ACK
 - Preconfigured ACK packet automatically transmitted on every packet that passed all filtering or 802.15.4 ACK

Studio v5 – Starting up with a known product

The screenshot displays the Simplicity Studio v5 interface. On the left, a sidebar titled "My Products" is open, showing a list of boards and parts. The "Boards" section is expanded, listing two boards: "EFR32FG12 2400/868 MHz 10 dBm Dual Band Radio Board (BRD4254A Rev A02)" and "EFR32FG12 2400/915 MHz 19 dBm Dual Band Radio Board (BRD4253A Rev A02)". The "Parts" section is also expanded, listing 15 different part numbers. The main window displays a "Welcome to Simplicity Studio" message, followed by a "Get Started" section with a "Start" button. Below this, there is a "Recent Projects" section with a "Create New Project" button. The "Learn and Support" section contains five links: "Simplicity Studio User's Guide" (OPEN), "Training and Tutorials" (START), "Tips and Tricks" (START), "Silicon Labs Community" (JOIN), and "Technical Support" (START). The status bar at the bottom shows the user's email address, memory usage (326M of 542M), and the copyright notice for Silicon Labs.

v5_workspace - Simplicity Studio™

File Edit Navigate Search Project Run Window Help

Welcome Recent Tools Install Preferences

My Products

EFR32FG12

Boards

- EFR32FG12 2400/868 MHz 10 dBm Dual Band Radio Board (BRD4254A Rev A02)
- EFR32FG12 2400/915 MHz 19 dBm Dual Band Radio Board (BRD4253A Rev A02)

Parts

- EFR32FG12P231F1024GL125
- EFR32FG12P231F1024GM48
- EFR32FG12P231F1024GM68
- EFR32FG12P231F512GM68
- EFR32FG12P232F1024GL125
- EFR32FG12P232F1024GM48
- EFR32FG12P431F1024GL125
- EFR32FG12P431F1024GM48
- EFR32FG12P431F1024GM68
- EFR32FG12P431F1024IM48
- EFR32FG12P431F512GM68
- EFR32FG12P432F1024GL125
- EFR32FG12P432F1024GM48

Welcome to Simplicity Studio

Everything you need to develop, research, and configure devices for IoT applications.

Get Started

Select a connected device or search for a product by name to see available documentation, example projects, and demos.

Connected Devices All Products

Connected Devices

EFR32MG12 2.4 GHz 19 dBm RB, WSTK Mainboard (IP: 10.150.12.31)

Start

Recent Projects

You haven't created a project yet

Create New Project

Learn and Support

Simplicity Studio User's Guide

The official Simplicity Studio 5 User's Guide

OPEN

Training and Tutorials

Our collection of Simplicity Studio training and tutorial videos

START

Tips and Tricks

Useful tips and tricks to help you optimize your tools setup

START

Silicon Labs Community

Where users come together to learn, get help, and grow their skills

JOIN

Technical Support

Get technical support directly from Silicon Labs

START

andras.gnandt@silabs.com

326M of 542M

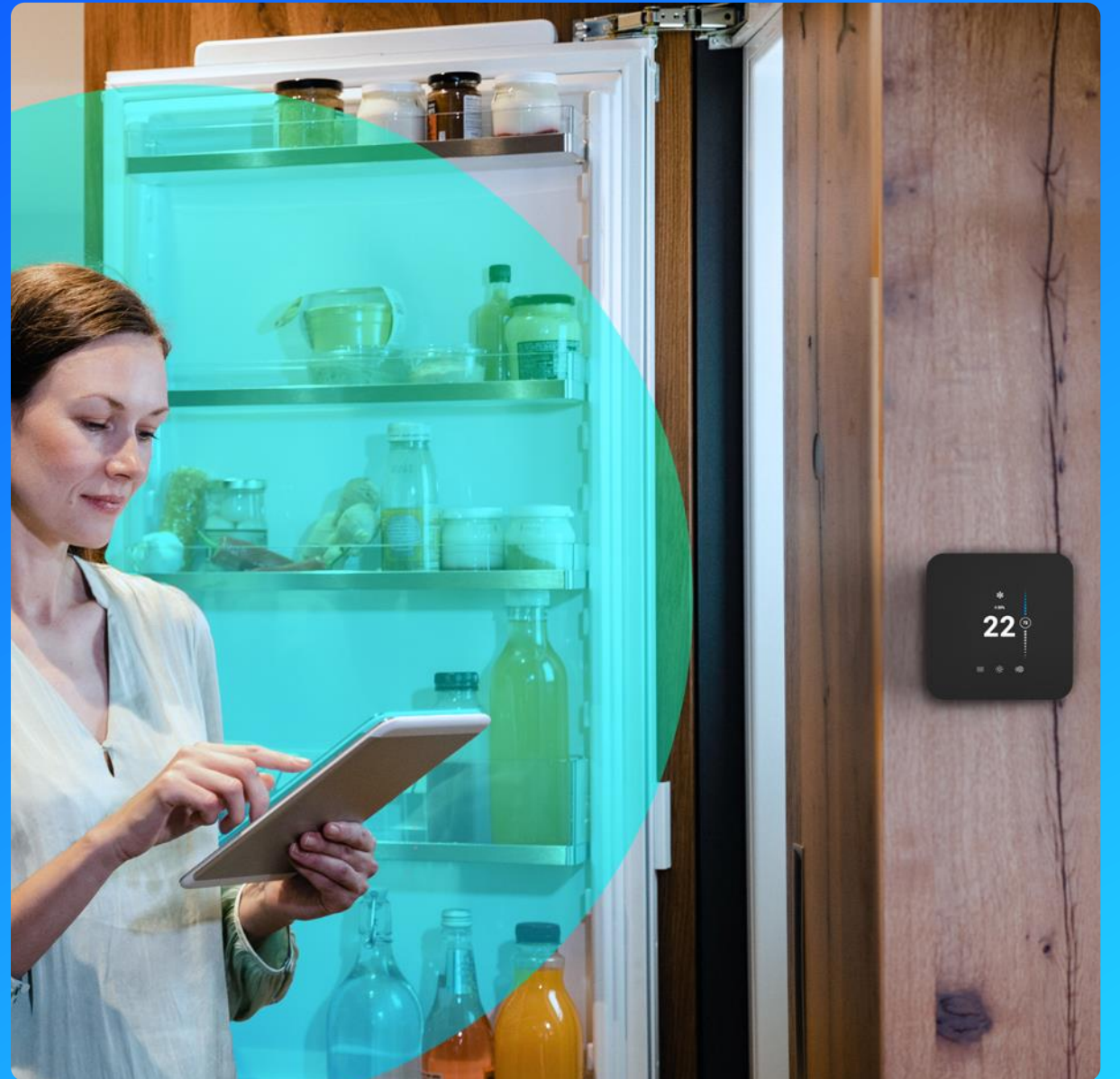
© 2021 Silicon Labs

Studio v5 – Starting up with a connected device

The screenshot displays the Simplicity Studio v5 interface. The top-left pane shows a list of debug adapters, with 'EFR32MG12 2400/868 MHz 10 dBm RB (ID:440208486)' selected. The main workspace area is titled 'EFR32MG12 2400/868 MHz 10 dBm RB, WSTK Mainboard (ID: 000440208486)'. Below the title are tabs for 'OVERVIEW', 'EXAMPLE PROJECTS & DEMOS', 'DOCUMENTATION', and 'COMPATIBLE TOOLS'. A 'Create New Project' button is visible in the top right. The 'General Information' section shows 'Connected Via: J-Link Silicon Labs', 'Debug Mode: Onboard Device (MCU)', and 'Adapter FW: 1v3p4b1035'. The 'Recommended Quick Start Guides' section lists several guides, including 'AN1254: Transitioning from the v2.x to the v3.x Proprietary Flex SDK'. The 'Board' section shows a 'Wireless Starter Kit Mainboard (BRD4001A Rev A01)'. The 'Target Part' section shows 'EFR32MG12 2400/868 MHz 10 dBm Dual Band Radio Board (BRD4163A Rev A02)'. The bottom status bar shows 'andras.gnandt@silabs.com', '581M of 858M', and '© 2021 Silicon Labs'.

RAIL SimpleTRX Walkthrough

Create Project in SSV5



Studio v5 – Example Projects for Proprietary

The screenshot displays the Simplicity Studio v5 interface. The main window is titled "EFR32MG12 2400/868 MHz 10 dBm RB, WSTK Mainboard (ID: 000440208486)". The left sidebar shows a tree view of "Debug Adapters" with the selected adapter highlighted. Below this, there are sections for "Demos" and "Example Projects". The "Example Projects" section is active, showing a list of projects filtered by "Proprietary" technology type. The main content area displays a list of 28 resources found, each with a "CREATE" button. The resources include:

- Flex (Connect) - SoC Direct Mode Device**: This sample app allows direct commissioning of nodes and exchange data between them via CLI commands.
- Flex (Connect) - SoC ECDH Key Exchange**: This sample application illustrates how we could share a network key between two devices in a secure way. The application works via CLI commands which break down the steps to understand and analyze this mechanism.
- Flex (Connect) - SoC Empty**: The Connect Empty project is a barebone Connect app that can be a basis of streamlined proprietary solutions.
- Flex (Connect) - SoC Empty Example DMP**: The Connect Empty DMP example is an RTOS-based project that provides a skeleton for Connect but not functions, beside a BLE Task with a basic CLI interface.
- Flex (Connect) - SoC Light Example DMP**: The purpose of the application is to demonstrate a simple wireless communication between two or more boards, using the connect SDK. In combination with the Connect-Switch sample application, it creates a basic light functionality, where the LEDs can be toggled on the Light node. After power up, the Light (network coordinator) create its own local network, and...
- Flex (Connect) - SoC MAC Mode Device**: A 802.15.4 sample app that provides CLI commands to form a network or join an existing network, send data to another node based on short or long addresses.
- Flex (Connect) - SoC Sensor**: Demonstrates how to properly setup a star network topology in which communication occurs in both directions between the Sink and the Sensor(s) nodes. The PB0 pushbutton can be used to enable or disable sleep, combined with changing the VCOM option 'Enable reception when sleeping'.
- Flex (Connect) - SoC Sink**: The Sink example is the counterpart of the Sensor example. It receives reports of Sensor nodes joining to its network.
- Flex (Connect) - SoC Switch Example**
- Flex (RAIL) - Burst Duty Cycle**

The bottom of the interface shows the user's email address "andras.gnandt@silabs.com" and the memory usage "686M of 858M".

Studio v5 – SimpleTRX Example Project

The screenshot displays the Simplicity Studio v5 interface. On the left, a sidebar shows a tree view of 'Debug Adapters' and 'My Products'. The 'Debug Adapters' list includes various radio boards like BGM13532, EFR32MG12, and EFR32MG13. The 'My Products' section has a search bar and a list of products, with 'My Products 1' selected. The main content area is titled 'EFR32MG12 2400/868 MHz 10 dBm RB, WSTK Mainboard (ID: 000440208486)'. Below the title are tabs for 'OVERVIEW', 'EXAMPLE PROJECTS & DEMOS', 'DOCUMENTATION', and 'COMPATIBLE TOOLS'. A message says 'Run a pre-compiled demo or create a new project based on a software example.' Below this is a filter section with 'Filter on keywords', 'Demos' (disabled), and 'Example Projects' (enabled). A list of 'Technology Type' and 'Provider' filters is shown, with 'Proprietary' selected. The main area displays a grid of project cards, each with a title, description, and a 'CREATE' button. The first card is 'Flex (RAIL) - Simple TRX', which is expanded to show its details: 'This application demonstrates the simplest exchange of transmit and receive operation between two nodes. Both nodes are capable of sending and receiving messages. On the WSTK, any button press (PB0/PB1) will send a message. LED0 will toggle on message send and LED1 will toggle on message receive. CLI can also be used for sending and showing...'.

Create SimpleTRX Sample Project

The screenshot displays the Simplicity Studio IDE interface. The main window is titled "v5_workspace - Simplicity Studio™" and shows a "New Project Wizard" dialog box in the foreground. The wizard is in the "Configuration" step, where the project name is "simple_trx" and the location is "C:\Users\langnandt\SimplicityStudio\v5_workspace\simple_trx". The "Use default location" checkbox is checked. The "With project files" section has "Link sdk and copy project sources" selected. The background shows a list of debug adapters and a sidebar with project templates like "AIL - Simple TRX Multi-PHY" and "AIL - Simple TRX with Auto-ACK".

Project Configuration
Select the project name and location.

Target, SDK Examples Configuration

Project name: simple_trx

Use default location

Location: C:\Users\langnandt\SimplicityStudio\v5_workspace\simple_trx BROWSE

With project files:

- Link to sources
- Link sdk and copy project sources
- Copy contents

CANCEL BACK NEXT FINISH

None Specified (0)

PRODUCTION (28)

AIL - Simple TRX Multi-PHY
This application demonstrates the usage of multiple PHYs by channels. By default there are 2 channels, one at 2.4GHz and one at 4.7GHz. Pressing PB0 will transmit on channel 0 and pressing PB1 will transmit on channel 1. Both channels are capable of sending messages and receiving ACKs. Transmission and reception is reported on serial terminal and LED1 (for channel 0/1, respectively). To send using...

CREATE

AIL - Simple TRX with Auto-ACK
This application demonstrates the simplest exchange of TX (software based) operation between two nodes. Both nodes are capable of sending messages and receiving ACKs. In the STK, any button press (PB0/PB1) will send a message. LED0 will toggle on message send and LED1 will toggle on ACK receive. With one click this baremetal sample...

CREATE

AIL - Switch Standards
The purpose of the application is to demonstrate a simple communication between two or more boards. In addition to the Light sample application it creates a Light functionality, where the light can be toggled in the application. After power up, the node is in SCAN state. It is capable of receiving broadcast messages of the light modules can be...

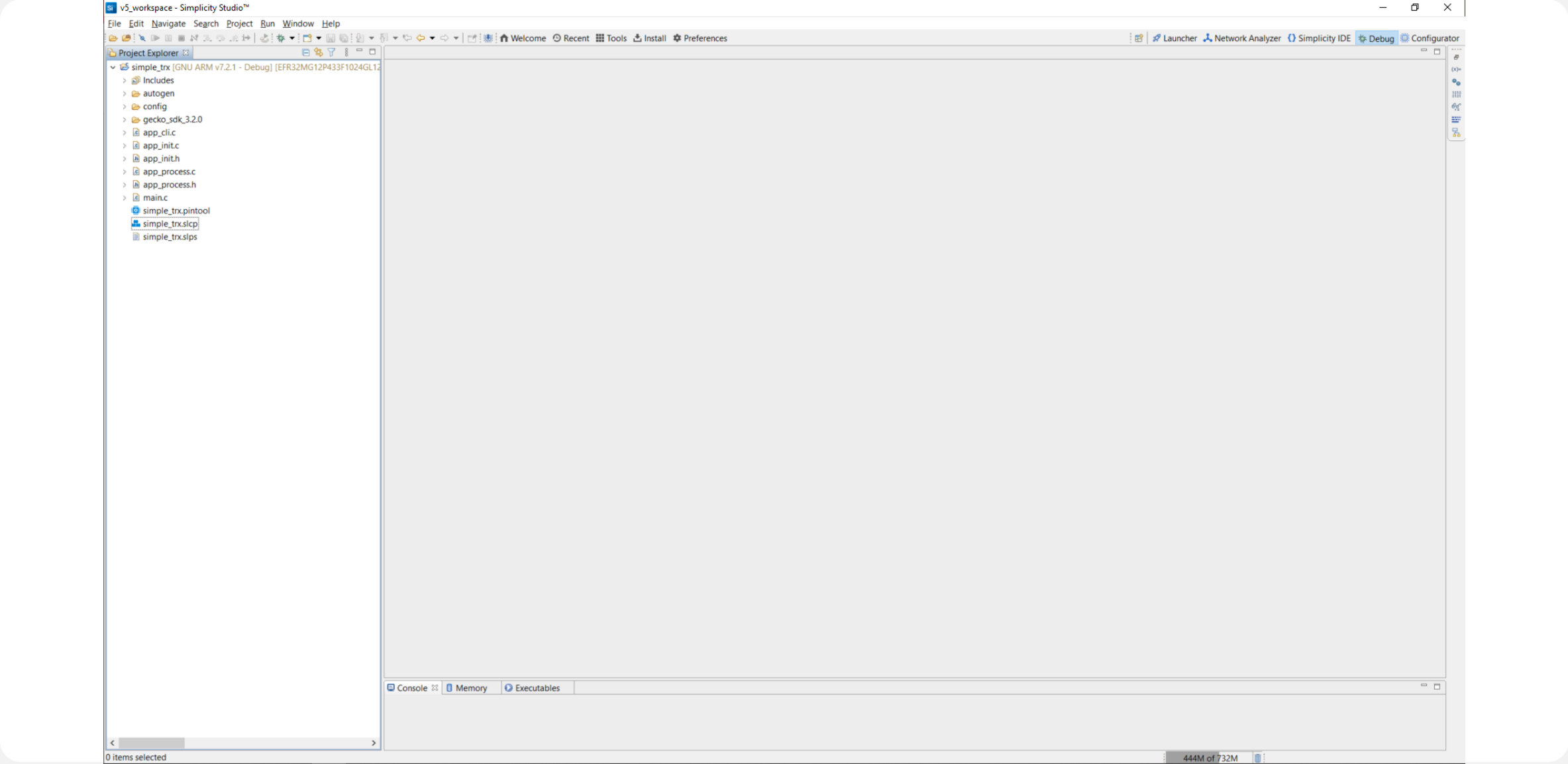
CREATE

AIL - Wireless M-bus Meter
This application demonstrates a Wireless M-Bus collector application. Uses the M-Bus configurator and capable of limited multi-PHY operation. It is capable of limited multi-PHY operation and features, like asymmetric bidirectional modes. For details, see...

CREATE

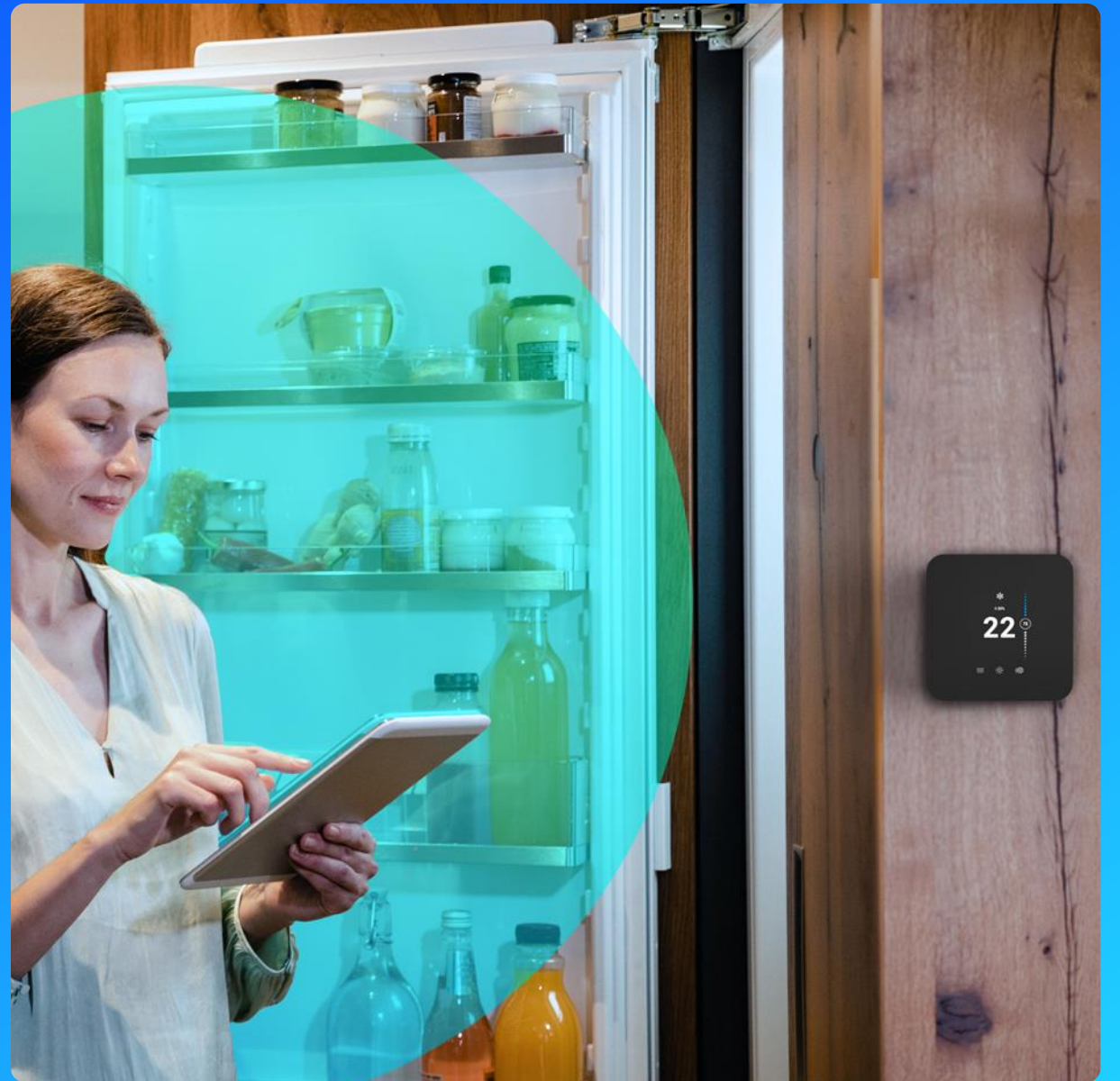
andras.gnandt@silabs.com 709M of 858M © 2021 Silicon Labs

SimpleTRX Project Components



Software Components

Demo



SimpleTRX Project Details

The screenshot displays the Simplicity Studio IDE interface for the SimpleTRX project. The main workspace is divided into three panels:

- Target and SDK Selection:** Shows a 'Mighty Gecko' board image and the target 'EFR32MG12P433F1024GL125'. Below the image, it lists: 'Wireless Starter Kit Mainboard (BRD4001A Rev A01)' and 'EFR32MG12 2400/868 MHz 10 dBm Dual Band Radio Board (BRD4163A Rev A02)'.
- Project Details:** Describes the 'simple_trx' application as a 'Simplicity IDE Project' that demonstrates the simplest exchange of transmit and receive operation between two nodes. It notes that the application can be run on an OS (MicriumOS and FreeRTOS) and provides instructions for terminal commands like 'rx 1' and 'rx 0'. It also includes a 'Category' of 'RAIL Examples' and a 'Preferred SDK' list including Gecko SDK Suite, Amazon, Bluetooth 3.2.0, Bluetooth Mesh 2.1.0, EmberZNet 6.10.0.0, Flex 3.2.0.0, HomeKit 1.0.0.0, and Micrium OS Kernel.
- Project Generators:** Contains a box for 'Simplicity IDE Project' with the text: 'A Simplicity IDE project supporting builds for MCUs using C/C++ and assembly files.'

The Console window at the bottom shows the following output:

```
Adapter Pack Console
C:\SiliconLabs\SimplicityStudio\v5\developer\adapter_packs\usb\usb_monitor.exe C:\SiliconLabs\SimplicityStudio\v5\developer\adapter_packs\usb\usb_monitor.exe -d -s
C:\SiliconLabs\SimplicityStudio\v5\developer\adapter_packs\inspect_emdll\inspect_emdll.exe C:\SiliconLabs\SimplicityStudio\v5\developer\adapter_packs\inspect_emdll\inspect_emdll.exe -slist -sn 000
list: 413c:b06e@012:005[B&F2A022A&0&5] 0bda:402e@013:004[209901010001] 413c:b06f@013:005[C&CA88A5D&0&3] 413c:301a@013:003[C&CA88A5D&0&3] 413c:2107@013:002[C&CA88A5D&0&2] 0bda:8153@014:004[11100000]
# Simplicity Studio device detection

deviceCount=1

device(440208486) {
  serialNumber=440208486
Connecting to device with serial number 440208486 .. ok.
FirmwareString is Silicon Labs J-Link Pro OB compiled Apr  4 2019 10:30:33
EMDLL version: 0.17.18b581
```

Drivers – USART vcom

The screenshot shows the Simplicity Studio interface with the 'SOFTWARE COMPONENTS' tab selected. The 'vcom' component is highlighted in the component list on the left. The right pane displays the 'Function Documentation' for `sl_iostream_usart_init()`.

```
sl_status_t
sl_iostream_usart_init ( sl_iostream_uart_t * iostream_uart,
                        sl_iostream_uart_config_t * uart_config,
                        USART_InitAsync_TypeDef * init,
                        sl_iostream_usart_config_t * usart_config,
                        sl_iostream_usart_context_t *
                        )
```

USART Stream init.

Parameters

Parameter	Description
[in] iostream_uart	IO Stream UART handle.
[in] uart_config	IO Stream UART config.
[in] init	USART initialization modes.
[in] usart_config	USART configuration.
[in] usart_context	USART Instance context.

Buttons at the bottom of the component view include: **Uninstall**, **Add New Instances**, and **View Dependencies & Instances**.

Drivers – Simple button

The screenshot displays the Simplicity Studio IDE interface for configuring a driver. The main window is titled "simple_trx" and shows the "SOFTWARE COMPONENTS" tab. On the left, a tree view shows the project structure, with "Simple Button" selected under the "Driver" category. The central pane displays the "Simple Button Configuration" dialog, which includes a description of the driver and a code snippet for its implementation. The code snippet shows the initialization of a simple button instance and the definition of its context structure.

Simple Button Configuration

Simple buttons use the `sl_button_t` struct and their `sl_simple_button_context_t` struct. These are automatically generated into the following files, as well as instance specific headers with macro definitions in them. The samples below are for a single instance called 'inst0'.

```
1 // sl_simple_button_instances.c
2
3 #include "sl_simple_button.h"
4 #include "sl_simple_button_inst0_config.h"
5
6 sl_simple_button_context_t simple_inst0_context = {
7     .state = 0,
8     .history = 0,
9     .port = SL_SIMPLE_BUTTON_INST0_PORT,
10    .pin = SL_SIMPLE_BUTTON_INST0_PIN,
11    .mode = SL_SIMPLE_BUTTON_INST0_MODE,
12 };
13
14 const sl_button_t sl_button_inst0 = {
15     .context = &simple_inst0_context,
16     .init = sl_simple_button_init,
17     .get_state = sl_simple_button_get_state,
18     .poll = sl_simple_button_poll_step,
19 };
20
21 const sl_button_t *sl_simple_button_array[] = {&sl_button_inst0};
22 const uint8_t simple_button_count = 1;
23
24 void sl_simple_button_init_instances(void)
25 {
```

At the bottom of the configuration pane, there are buttons for "Uninstall", "Add New Instances", and "View Dependencies & Instances".

Drivers - USART

The screenshot displays the SImplicity Studio IDE interface for configuring a project named 'simple_trx'. The 'SOFTWARE COMPONENTS' tab is active, showing a tree view of components under the 'Platform' and 'Driver' categories. The 'UART' category is expanded, and 'UARTDRV USART' is selected. The right-hand pane provides detailed information for this component, including a description, quality level (PRODUCTION), and an 'Install' button. The console at the bottom shows the output of the adapter pack console.

simple_trx OVERVIEW SOFTWARE COMPONENTS CONFIGURATION TOOLS

Filter: Configurable Components Installed Components Components Installed by You

Search keywords, component's name

UARTDRV USART [Install](#)

[Add component to project](#)

Description

The UART USART driver supports the UART capabilities of the USART peripheral. The driver is fully reentrant and supports multiple driver instances. Both blocking and non-blocking transfer functions are available. The non-blocking transfer functions use DMA to transfer data and report transfer completion with callback functions. Note that the blocking functions are not suitable for low energy applications because they use CPU polling.

If the Services->Runtime->System Setup->System Init component is included in a project, the driver instances will be initialized automatically, using the instance configurations, during the `sl_system_init()` call in `main.c`.

Selecting this component will also include the UARTDRV Core component, which is the implementation of the UARTDRV driver itself.

Quality
PRODUCTION

[Open in Browser](#)

UARTDRV - UART Driver

Description

Universal Asynchronous Receiver/Transmitter Driver.

[View Dependencies](#)

Console Memory Executables

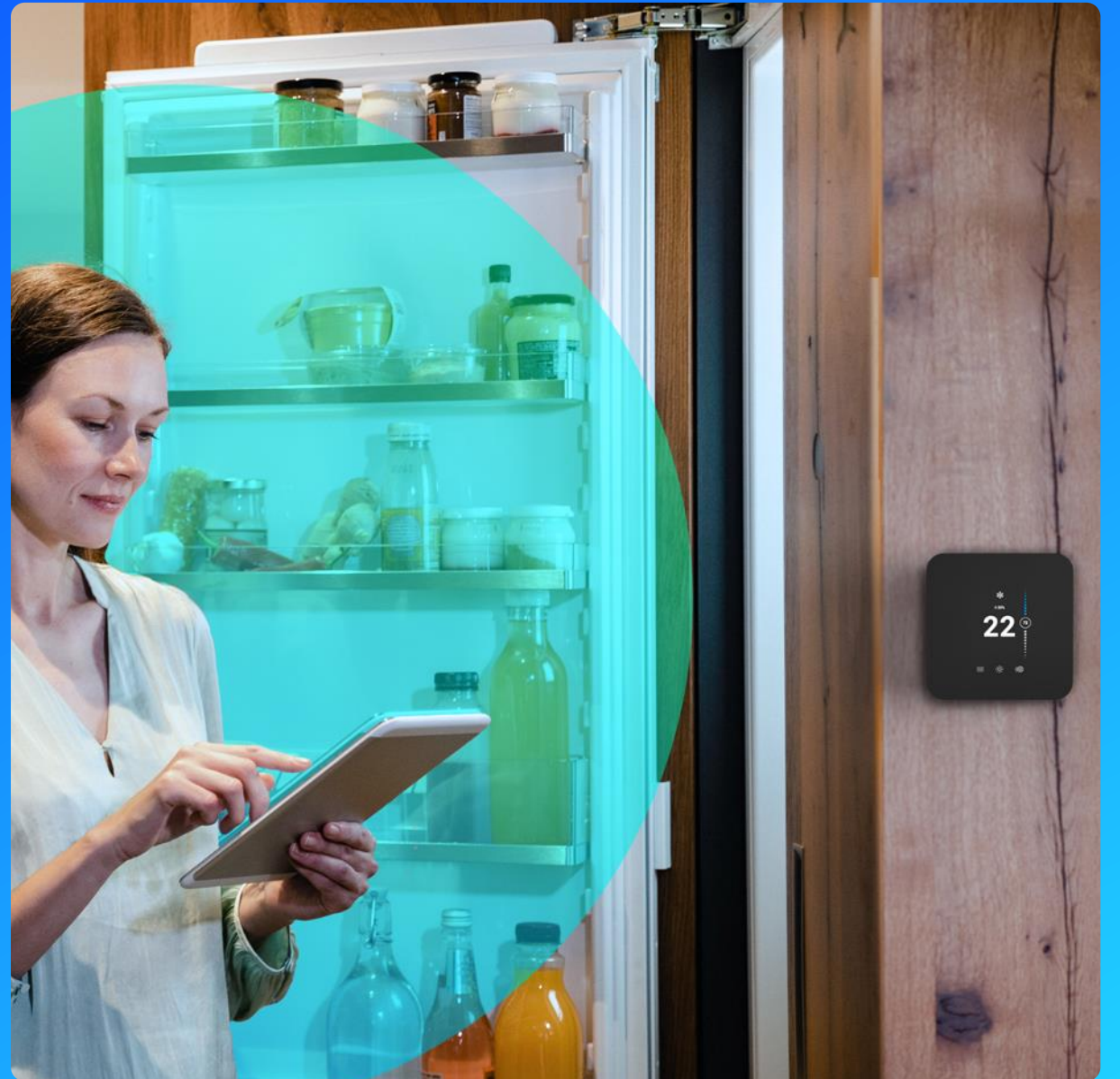
Adapter Pack Console

arrived: e5b7:0811@015:001[20160823]

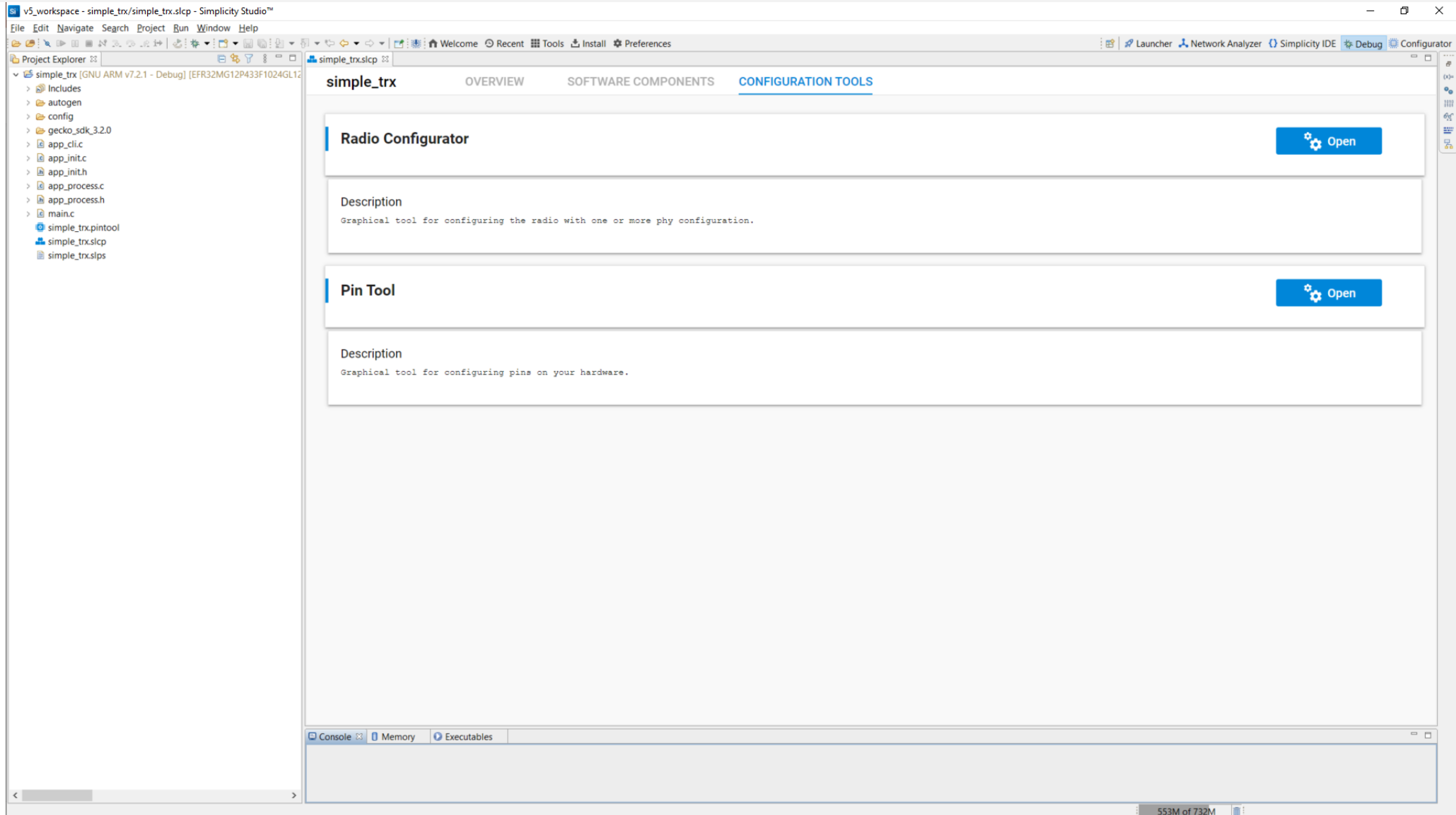
1189M of 2232M

Radio Configurator Flow

Demo



Configuration Tools



Configuration Tools – Pin Tool

The screenshot displays the Simplicity Studio Pin Tool interface. On the left, a project explorer shows the file structure for 'simple_tx'. The main area is divided into two panes: 'DefaultMode Port I/O: PORTIO' and 'Configure'.

The 'DefaultMode Port I/O: PORTIO' pane shows a 7x7 grid of pins for a 125-pin BGA package. Pin H12 is highlighted with a yellow box. The grid includes pins A1 through N13, with various functions like GPIO, USART, and I2C assigned to specific pins.

The 'Configure' pane shows a table of pins and their functions. A dropdown menu is open for pin H12, showing options like 'None', 'GPIO mode', 'ETM_TD3', 'I2C1_SCL', 'I2C1_SDA', and 'LESENSE_ALTEX1'.

Pin #	Pin Name	Function	Custom Pin Name	Software Component
M13	PA0	USART0_TX		IO Stream: USART (vcom) : SL_L
L13	PA1	USART0_RX		IO Stream: USART (vcom) : SL_L
L12	PA2	USART0_CTS		IO Stream: USART (vcom) : SL_L
K13	PA3	USART0_RTS		IO Stream: USART (vcom) : SL_L
K12	PA4	GPIO mode		MX25 Flash Shutdown with usar
J13	PA5	GPIO mode		Board Control : SL_BOARD_ENA
J12	PA6			
J11	PA7			
H13	PA8			
H12	PA9	None		
F12	PB6	GPIO mode		
E13	PB7	ETM_TD3		
E12	PB8	I2C1_SCL		
D13	PB9	I2C1_SDA		
D12	PB10	LESENSE_ALTEX1		Board Control : SL_BOARD_ENA
D11	PB11	PCNT1_S0IN		
C13	PB12	PCNT1_S1IN		RAIL Utility, PTI : SL_RAIL_UTIL_PO
C12	PB13	PCNT2_S0IN		RAIL Utility, PTI : SL_RAIL_UTIL_PO
C11	PB14	PCNT2_S1IN		
C10	PB15	USART2_CLK		
A5	PC0	USART2_CS		
B5	PC1	USART2_CTS		
C5	PC2	USART2_RTS		
A4	PC3	USART2_TX		
B4	PC4	WTIMER0_CC0		
A3	PC5	WTIMER0_CC1		
B9	PC6	WTIMER0_CC2		
A8	PC7	WTIMER0_CDT10		
B8	PC8			
A7	PC9			
B7	PC10			
A6	PC11			
N9	PD8			
M9	PD9			
N10	PD10			
M10	PD11			
N11	PD12			
M11	PD13			
N12	PD14			
N13	PD15	GPIO mode		Board Control : SL_BOARD_ENA
RR	PFN			

Configuration Tool – Radio Configurator

The screenshot displays the Radio Configurator tool within the Simplicity Studio IDE. The main window is titled "Radio Configurator" and is divided into several sections:

- Project Explorer:** Shows the project structure for "simple_tx" (GNU ARM v7.2.1 - Debug) [EFR32MG12P433F1024GL12].
- Protocol Configuration:** The active configuration window for "Channel Group 1". It includes:
 - General Settings:** Protocol name (Protocol Configuration), C variable name (Protocol_Configuration), Select radio profile (Base Profile), and Select radio PHY (868M 2GFSK 500Kbps 125K).
 - Channels Overview:** A table showing channel configurations for "Channel Group 1".

Name	Start channel No.	Start channel Frequency	Stop channel No.	Stop channel Frequency
Channel Group 1	0	868.00 Mhz	20	888.00 Mhz

The "Select radio PHY" dropdown menu is open, showing the following options:

- 169MHz 2GFSK 2.4Kbps 1.2KHz
- 169MHz 2GFSK 2.4Kbps 1.2KHz ETSI
- 169MHz 2GFSK 38.4Kbps 20KHz
- 2450M 2GFSK 1Mbps 500K
- 2450M 2GFSK 250Kbps 125K
- 2450M 2GFSK 2Mbps 1M
- 285MHz 2GFSK 2.4Kbps 1.2KHz
- 285MHz 2GFSK 500Kbps 125KHz
- 315MHz 2GFSK 2.4Kbps 1.2KHz
- 315MHz 2GFSK 38.4Kbps 20KHz

Radio Configurator - Customized

The screenshot displays the Radio Configurator interface within Simplicity Studio. The interface is divided into several sections:

- Project Explorer:** Shows the project structure for 'simple_trx' (GNU ARM v7.2.1 - Debug) [EFR32MG12P433F1024GL12].
- Radio Configurator:** The main configuration area, featuring a search bar and a 'View Manual' button.
- Protocol Configuration:** A tree view showing 'Channel Group 1' with sub-panels for General Settings, Channels Overview, Operational Frequency, Crystal, Modem, Packet, Symbol Coding, Channel Coding, Testing, and Advanced.
- General Settings:** Displays protocol details: Protocol name (Protocol Configuration), C variable name (Protocol_Configuration), radio profile (Base Profile), and radio PHY (868M 2GFSK 500Kbps 125K). A 'Customized' toggle is turned on.
- Channels Overview:** A table showing channel details for 'Channel Group 1':

Name	Start channel No.	Start channel Frequency	Stop channel No.	Stop channel Frequency
Channel Group 1	0	868.00 Mhz	20	888.00 Mhz
- Operational Frequency:** Shows Base Channel Frequency (868 MHz), Channel Spacing (1000 kHz), and Injection side (HIGH_SIDE).
- Crystal:** Shows Crystal Frequency (38.4 MHz), TX Crystal Accuracy (0 ppm), and RX Crystal Accuracy (0 ppm).
- Modem:** Shows Modulation Type (FSK2), Bitrate (500 kbps), Deviation (125 kHz), Baudrate Tolerance (0 ppm), Shaping Filter (Gaussian), and Shaping Filter Parameter (BT or R) (0.5).

The bottom of the interface includes a Console, Memory, and Executables panel, and a status bar showing 498M of 732M.

Radio Configurator – Modem settings

The screenshot displays the Radio Configurator interface within Simplicity Studio. The main window is titled "Radio Configurator" and shows the configuration for a radio channel. The interface is divided into several sections:

- Protocol Configuration:** Shows "Channel Group 1" selected.
- Operational Frequency:** Displays settings for the radio channel:
 - Base Channel Frequency: 868 MHz
 - Channel Spacing: 1000 kHz
 - Injection side: HIGH_SIDE
- Crystal:** Displays settings for the crystal frequency:
 - Crystal Frequency: 38.4 MHz
 - TX Crystal Accuracy: 0 ppm
 - RX Crystal Accuracy: 0 ppm
- Modem:** Displays settings for the modem configuration:
 - Modulation Type: FSK2
 - Bitrate: 500 kbps
 - Deviation: 125 kHz
 - Baudrate Tolerance: 0 ppm
 - Shaping Filter: Gaussian
 - Shaping Filter Parameter (BT or R): 0.5
 - FSK symbol map: MAP0
 - Enable Asynchronous direct mode:
- Packet:** Shows a visual representation of the packet structure with fields for Preamble, Syncword, Payload, and CRC.

The interface also includes a Project Explorer on the left, a Console/Executables panel at the bottom, and a status bar at the very bottom showing "488M of 732M".

Radio Configurator – Packet structure settings

The screenshot displays the Radio Configurator interface within Simplicity Studio. The main window is titled "Radio Configurator" and shows the "Packet" configuration tab. The interface is organized into several sections:

- Protocol Configuration:** A sidebar on the left lists various configuration categories, with "Packet" selected.
- Packet Structure:** A top bar shows the packet structure: Preamble, Syncword, Payload, and CRC. The Payload section is highlighted with a yellow and black striped border.
- Frame General:** Includes settings for Header Enable (disabled), Frame Coding Method (NONE), Frame Length Algorithm (FIXED_LENGTH), and Frame Bit Endian (LSB_FIRST).
- Frame Fixed Length:** Shows a Fixed Payload Size of 16 bytes.
- CRC Input:** Includes CRC Polynomial (CRC_16), CRC Input Bit Endian (LSB_FIRST), CRC Seed (0), and CRC Input Padding (disabled).
- CRC Output:** Includes CRC Byte Endian (LSB_FIRST), CRC Output Bit Endian (MSB_FIRST), and CRC Invert (disabled).
- Whitening:** Includes Whitening Output Bit (0), Whitening Polynomial (NONE), and Whitening Seed (FF FF).
- Preamble:** Includes Preamble Base Pattern (1), Preamble Length Total (40 bits), and Preamble Pattern Length (2 bits).
- Syncword:** Includes Sync Word 0 (F6 8D), Sync Word 1 (0), Sync Word Length (16 bits), and Sync Word TX Skip (disabled).
- Frame Payload:** Includes Payload Whitening Enable (disabled) and Insert/Check CRC after payload (enabled).

The bottom of the interface shows a Console, Memory, and Executables panel. The status bar at the bottom right indicates "522M of 732M".

Radio Configurator – Packet structure settings

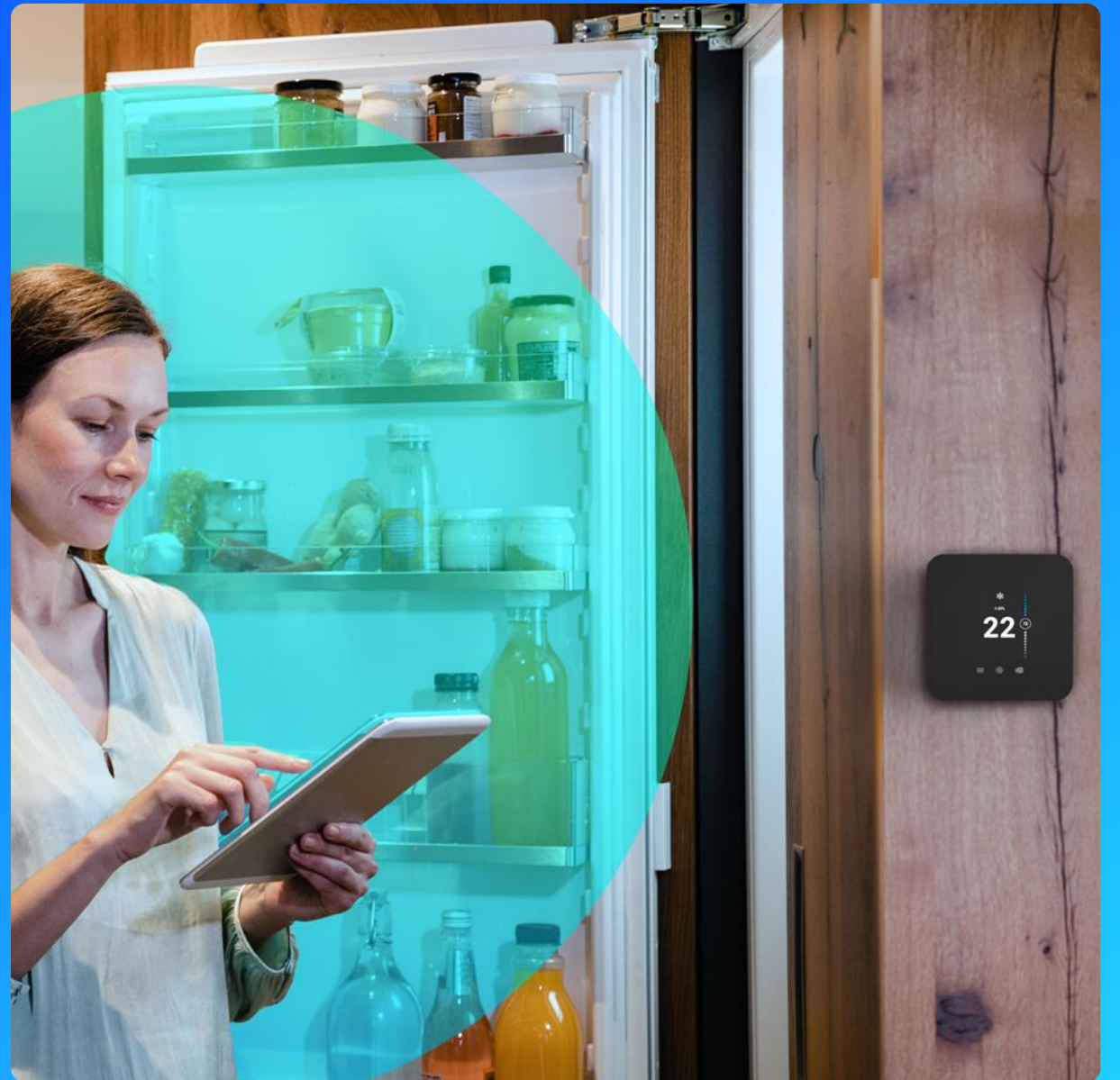
The screenshot displays the Radio Configurator interface within Simplicity Studio. The main window is titled "Radio Configurator" and shows the configuration for "Channel Group 1". The interface is divided into several sections:

- Protocol Configuration:** A sidebar on the left lists various configuration categories: General Settings, Channels Overview, Operational Frequency, Crystal, Modem, Packet, Symbol Coding, Channel Coding, Testing, and Advanced. The "Packet" category is currently selected.
- Symbol Coding:** A panel with settings for DSSS Chipping Code Base (0), DSSS Chipping Code Length (0), DSSS Spreading Factor (0), Differential Encoding Mode (DISABLED), Manchester Code Mapping (Default), and Symbol Encoding (NRZ).
- Channel Coding:** A panel with a dropdown menu for FEC Algorithm, currently set to NONE. Other options include FEC_154G, FEC_154G_K7, FEC_154G_NRNNSC_INTERLEAVING, FEC_154G_RSC_INTERLEAVING, and FEC_154G_RSC_NO_INTERLEAVING.
- Testing:** A panel with a toggle switch for "Reconfigure for BER testing".
- Advanced:** A section containing three sub-panels:
 - Timing Detection:** Includes checkboxes for "Number of Errors Allowed in ..." (0) and "Number of Timing Windows L..." (1).
 - AGC:** Includes checkboxes for "AGC Speed" (FAST) and "AGC Period" (0), along with a field for "AGC Settling Delay".
 - AFC:** Includes checkboxes for "Frequency Compensation ..." (DISABLED) and "Frequency Offset Compensation (...)" (0).

The interface also features a search bar and a "View Manual" button at the top right. The bottom status bar shows "619M of 732M".

RAIL SimpleTRX Walkthrough

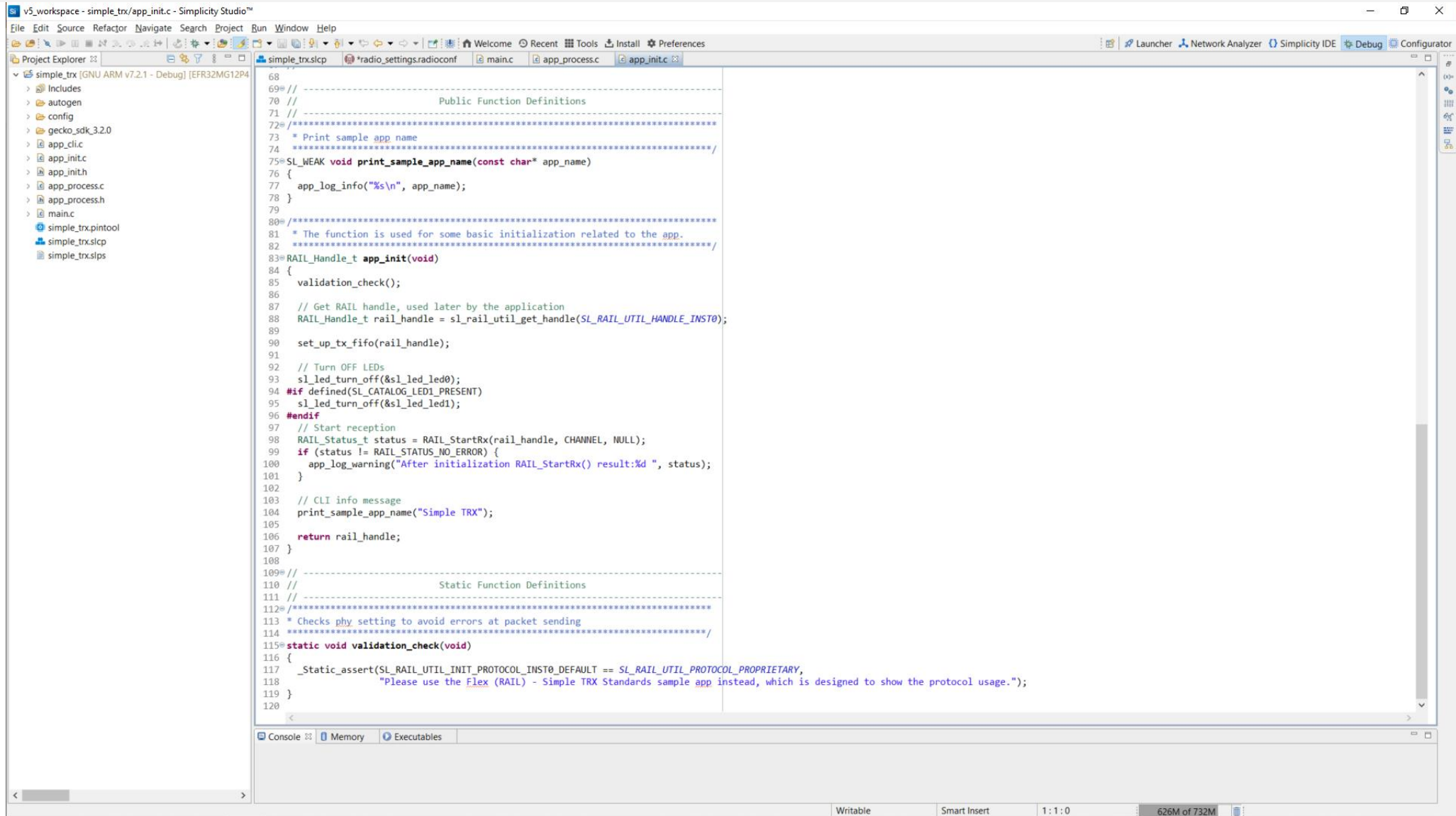
Init



SimpleTRX – main.c

```
v5_workspace - simple_trx/main.c - Simplicity Studio™
File Edit Source Refactor Navigate Search Project Run Window Help
simple_trx.sclp *radio_settings.radioconf main.c
Project Explorer
simple_trx [GNU ARM v7.2.1 - Debug] [EFR32MG12P4]
  Includes
  autogen
  config
  gecko_sdk_3.2.0
  app_cli.c
  app_init.c
  app_init.h
  app_process.c
  app_process.h
  main.c
  simple_trx.pintool
  simple_trx.sclp
  simple_trx.slps
59
60 // -----
61 //                               Static Variables
62 // -----
63 #if defined(SL_CATALOG_KERNEL_PRESENT)
64 // A static handle of a RAIL instance
65 static RAIL_Handle_t rail_handle;
66 #endif
67 // -----
68 //                               Public Function Definitions
69 // -----
70 // =====
71 * Main function
72 =====
73 int main(void)
74 {
75 // Initialize Silicon Labs device, system, service(s) and protocol stack(s).
76 // Note that if the kernel is present, processing task(s) will be created by
77 // this call.
78 sl_system_init();
79
80 // Initialize the application. For example, create periodic timer(s) or
81 // task(s) if the kernel is present.
82 #if defined(SL_CATALOG_KERNEL_PRESENT)
83 app_task_init();
84 #else
85 rail_handle = app_init();
86 #endif
87
88 #if defined(SL_CATALOG_KERNEL_PRESENT)
89 // Start the kernel. Task(s) created in app_init() will start running.
90 sl_system_kernel_start();
91 #else // SL_CATALOG_KERNEL_PRESENT
92 while (1) {
93 // Do not remove this call: Silicon Labs components process action routine
94 // must be called from the super loop.
95 sl_system_process_action();
96
97 // Application process.
98 app_process_action(rail_handle);
99
100 #if defined(SL_CATALOG_POWER_MANAGER_PRESENT)
101 // Let the CPU go to sleep if the system allows it.
102 sl_power_manager_sleep();
103 #endif
104 }
105 #endif // SL_CATALOG_KERNEL_PRESENT
106 }
107
108 // -----
109 //                               Static Function Definitions
110 // -----
111
```


SimpleTRX – RAIL App Init



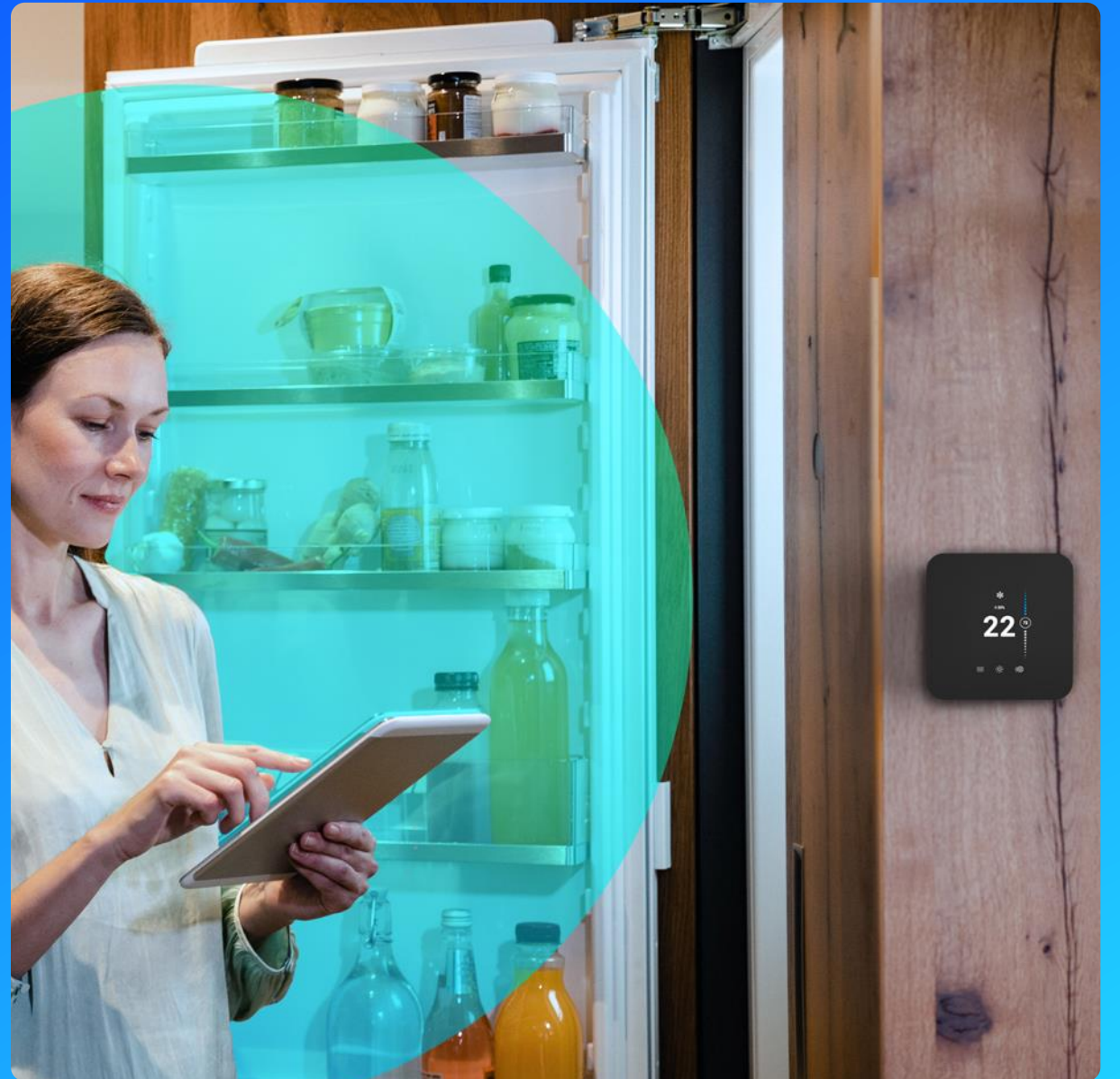
```
68
69 // -----
70 //                                     Public Function Definitions
71 // -----
72 //*****
73 * Print sample app name
74 //*****
75 SL_WEAK void print_sample_app_name(const char* app_name)
76 {
77     app_log_info("%s\n", app_name);
78 }
79
80 //*****
81 * The function is used for some basic initialization related to the app.
82 //*****
83 RAIL_Handle_t app_init(void)
84 {
85     validation_check();
86
87     // Get RAIL handle, used later by the application
88     RAIL_Handle_t rail_handle = sl_rail_util_get_handle(SL_RAIL_UTIL_HANDLE_INST0);
89
90     set_up_tx_fifo(rail_handle);
91
92     // Turn OFF LEDs
93     sl_led_turn_off(&sl_led_led0);
94     #if defined(SL_CATALOG_LED1_PRESENT)
95     sl_led_turn_off(&sl_led_led1);
96     #endif
97     // Start reception
98     RAIL_Status_t status = RAIL_StartRx(rail_handle, CHANNEL, NULL);
99     if (status != RAIL_STATUS_NO_ERROR) {
100         app_log_warning("After initialization RAIL_StartRx() result:%d ", status);
101     }
102
103     // CLI info message
104     print_sample_app_name("Simple TRX");
105
106     return rail_handle;
107 }
108
109 // -----
110 //                                     Static Function Definitions
111 // -----
112 //*****
113 * Checks phy setting to avoid errors at packet sending
114 //*****
115 static void validation_check(void)
116 {
117     _Static_assert(SL_RAIL_UTIL_INIT_PROTOCOL_INST0_DEFAULT == SL_RAIL_UTIL_PROTOCOL_PROPRIETARY,
118         "Please use the Flex (RAIL) - Simple TRX Standards sample app instead, which is designed to show the protocol usage.");
119 }
120
```

RAIL Simple TRX Init - Summary

- Initialized through components:
 - Clocks, DCDC, UART etc
 - PA (required for tx only)
 - PTI (optional)
 - RSSI offset (required for accurate RSSI reading)
 - RAIL -> rail_handle, event handler is ready, sl_rail_util_on_event()
 - Load radio config
 - Automatic state transitions (optional)
 - Some events (recommended to do in code)
- Initialized through app_init:
 - Tx fifo (required for tx only)
 - Start rx (sets starting radio state)

RAIL SimpleTRX Walkthrough

Process



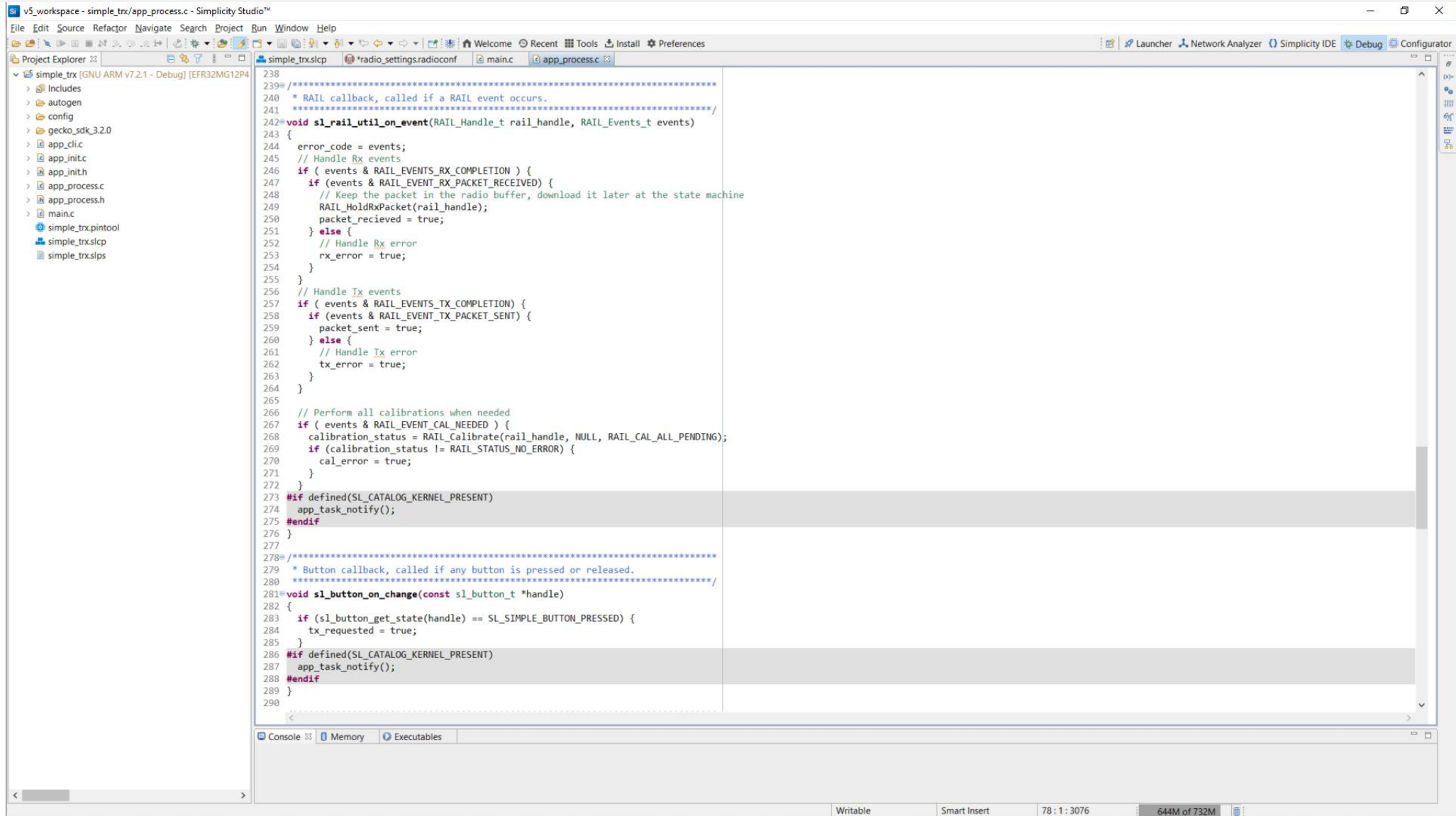
SimpleTRX – Process - Functions

```
49 // -----  
50 //                                     Macros and Typedefs  
51 // -----  
52 // Size of RAIL RX/TX FIFO  
53 #define RAIL_FIFO_SIZE (256u)  
54 // Transmit data length  
55 #define TX_PAYLOAD_LENGTH (16u)  
56  
57 // State machine of simple_trx  
58 typedef enum {  
59     S_PACKET_RECEIVED,  
60     S_PACKET_SENT,  
61     S_RX_PACKET_ERROR,  
62     S_TX_PACKET_ERROR,  
63     S_CALIBRATION_ERROR,  
64     S_IDLE,  
65 } state_t;  
66  
67 // -----  
68 //                                     Static Function Declarations  
69 // -----  
70 //-----  
71 * The function printfs the received rx message.  
72 *  
73 * @param rx_buffer Msg buffer  
74 * @param length How many bytes should be printed out  
75 * @returns None  
76 //-----  
77 static void printf_rx_packet(const uint8_t * const rx_buffer, uint16_t length);  
78 |  
79 //-----  
80 * The API helps to unpack the received packet, point to the payload and returns the length.  
81 *  
82 * @param rx_destination Where should the full packet be unpacked  
83 * @param packet_information Where should all the information of the packet stored  
84 * @param start_of_payload Pointer where the payload starts  
85 * @return The length of the received payload  
86 //-----  
87 static uint16_t unpack_packet(uint8_t *rx_destination, const RAIL_RxPacketInfo_t *packet_information, uint8_t **start_of_payload);  
88  
89 //-----  
90 * The API prepares the packet for sending and load it in the RAIL TX FIFO  
91 *  
92 * @param rail_handle Which rail handlers should be used for the TX FIFO writing  
93 * @param out_data The payload buffer  
94 * @param length The length of the payload  
95 //-----  
96 static void prepare_package(RAIL_Handle_t rail_handle, uint8_t *out_data, uint16_t length);  
97  
98 // -----  
99 //                                     Global Variables  
100 // -----  
101 // Flag, indicating transmit request (button has pressed / CLI transmit request has occurred)
```

SimpleTRX – Process – State Machine

```
172 switch (state) {
173     case S_PACKET_RECEIVED:
174         // Packet received:
175         // - Check whether RAIL_HoldRxPacket() was successful, i.e. packet handle is valid
176         // - Copy it to the application FIFO
177         // - Free up the radio FIFO
178         // - Return to IDLE state i.e. RAIL Rx
179         rx_packet_handle = RAIL_GetRxPacketInfo(rail_handle, RAIL_RX_PACKET_HANDLE_OLDEST_COMPLETE, &packet_info);
180         while (rx_packet_handle != RAIL_RX_PACKET_HANDLE_INVALID) {
181             uint8_t *start_of_packet = 0;
182             uint16_t packet_size = unpack_packet(rx_fifo, &packet_info, &start_of_packet);
183             rail_status = RAIL_ReleaseRxPacket(rail_handle, rx_packet_handle);
184             if (rail_status != RAIL_STATUS_NO_ERROR) {
185                 app_log_warning("RAIL_ReleaseRxPacket() result:%d", rail_status);
186             }
187             if (rx_requested) {
188                 printf_rx_packet(start_of_packet, packet_size);
189             }
190             sl_led_toggle(&sl_led_led0);
191             rx_packet_handle = RAIL_GetRxPacketInfo(rail_handle, RAIL_RX_PACKET_HANDLE_OLDEST_COMPLETE, &packet_info);
192         }
193         state = S_IDLE;
194         break;
195     case S_PACKET_SENT:
196         app_log_info("Packet has been sent\n");
197         #if defined(SL_CATALOG_LEDI_PRESENT)
198             sl_led_toggle(&sl_led_led1);
199         #else
200             sl_led_toggle(&sl_led_led0);
201         #endif
202         state = S_IDLE;
203         break;
204     case S_RX_PACKET_ERROR:
205         // Handle Rx error
206         app_log_error("Radio RX Error occurred\nEvents: %l1X\n", error_code);
207         state = S_IDLE;
208         break;
209     case S_TX_PACKET_ERROR:
210         // Handle Tx error
211         app_log_error("Radio TX Error occurred\nEvents: %l1X\n", error_code);
212         state = S_IDLE;
213         break;
214     case S_IDLE:
215         if (tx_requested) {
216             prepare_package(rail_handle, out_packet, sizeof(out_packet));
217             rail_status = RAIL_StartTx(rail_handle, CHANNEL, RAIL_TX_OPTIONS_DEFAULT, NULL);
218             if (rail_status != RAIL_STATUS_NO_ERROR) {
219                 app_log_warning("RAIL_StartTx() result:%d ", rail_status);
220             }
221             tx_requested = false;
222         }
223         break;
224 }
```

SimpleTRX – Process - RAIL



```
238
239 //*****
240 * RAIL callback, called if a RAIL event occurs.
241 //*****
242 void sl_rail_util_on_event(RAIL_Handle_t rail_handle, RAIL_Events_t events)
243 {
244     error_code = events;
245     // Handle Rx events
246     if ( events & RAIL_EVENTS_RX_COMPLETION ) {
247         if (events & RAIL_EVENT_RX_PACKET_RECEIVED) {
248             // Keep the packet in the radio buffer, download it later at the state machine
249             RAIL_HoldRxPacket(rail_handle);
250             packet_recieved = true;
251         } else {
252             // Handle Rx error
253             rx_error = true;
254         }
255     }
256     // Handle Tx events
257     if ( events & RAIL_EVENTS_TX_COMPLETION ) {
258         if (events & RAIL_EVENT_TX_PACKET_SENT) {
259             packet_sent = true;
260         } else {
261             // Handle Tx error
262             tx_error = true;
263         }
264     }
265
266     // Perform all calibrations when needed
267     if ( events & RAIL_EVENT_CAL_NEEDED ) {
268         calibration_status = RAIL_Calibrate(rail_handle, NULL, RAIL_CAL_ALL_PENDING);
269         if (calibration_status != RAIL_STATUS_NO_ERROR) {
270             cal_error = true;
271         }
272     }
273     #if defined(SL_CATALOG_KERNEL_PRESENT)
274     app_task_notify();
275     #endif
276 }
277
278 //*****
279 * Button callback, called if any button is pressed or released.
280 //*****
281 void sl_button_on_change(const sl_button_t *handle)
282 {
283     if (sl_button_get_state(handle) == SL_SIMPLE_BUTTON_PRESSED) {
284         tx_requested = true;
285     }
286     #if defined(SL_CATALOG_KERNEL_PRESENT)
287     app_task_notify();
288     #endif
289 }
290
```

RAIL Simple TRX Process

Transmit

- Init:
 - PA configuration (RAIL PA component)
 - RAIL_SetTxFifo()
- Main Loop:
 - RAIL_WriteTxFifo()
 - RAIL_StartTx()
- Event handler:
 - Wait for RAIL_EVENT_TX_PACKET_SENT
 - Go back to RX state: auto state transition in RAIL init component

Receive

- Init:
 - RAIL_StartRx()
- Event handler
 - Wait for RAIL_EVENT_RX_PACKET_RECEIVED
 - RAIL_HoldRxPacket()
 - Go back to RX state: auto state transition in RAIL init component
- Main loop:
 - RAIL_GetRxPacketInfo() for RAIL_RX_PACKET_HANDLE_OLDEST_COMPLETE
 - If returns packetHandle:
 - RAIL_CopyRxPacket()
 - RAIL_ReleaseRxPacket()
 - Repeat until returns valid packetHandle

Resources

- Proprietary Flex SDK v3.x Quick Start Guide -- QSG168
- RAIL Fundamentals -- UG103.13
- Connect Fundamentals -- UG103.12
- Multiprotocol Fundamentals -- UG103.16
- Dynamic Multiprotocol User's Guide -- UG305
- [Simplicity Studio® 5 User's Guide](#)
- EFR32 Migration Guide for Proprietary Applications -- AN1244
- About the Connect v3.x User's Guide -- UG435.01
- Building Low Power Networks with the Silicon Labs Connect Stack v3.x -- AN1252
- [Silicon Labs Connect API Reference Guide](#)
- EFR32 Radio Configurator Guide for Simplicity Studio 5 -- AN1253
- RAILtest User's Guide -- UG409
- EFR32 RF Evaluation Guide -- AN972
- [Silicon Labs RAIL API Reference Guide](#)
- <https://www.silabs.com/support/training/rail>
- [RAIL Tutorials](#)



works with
BY SILICON LABS
VIRTUAL CONFERENCE

