works with
BY SILICON LABS
VIRTUAL CONFERENCE

SILICON LABS

Series 2: BG22 Bluetooth® SoCs and Modules

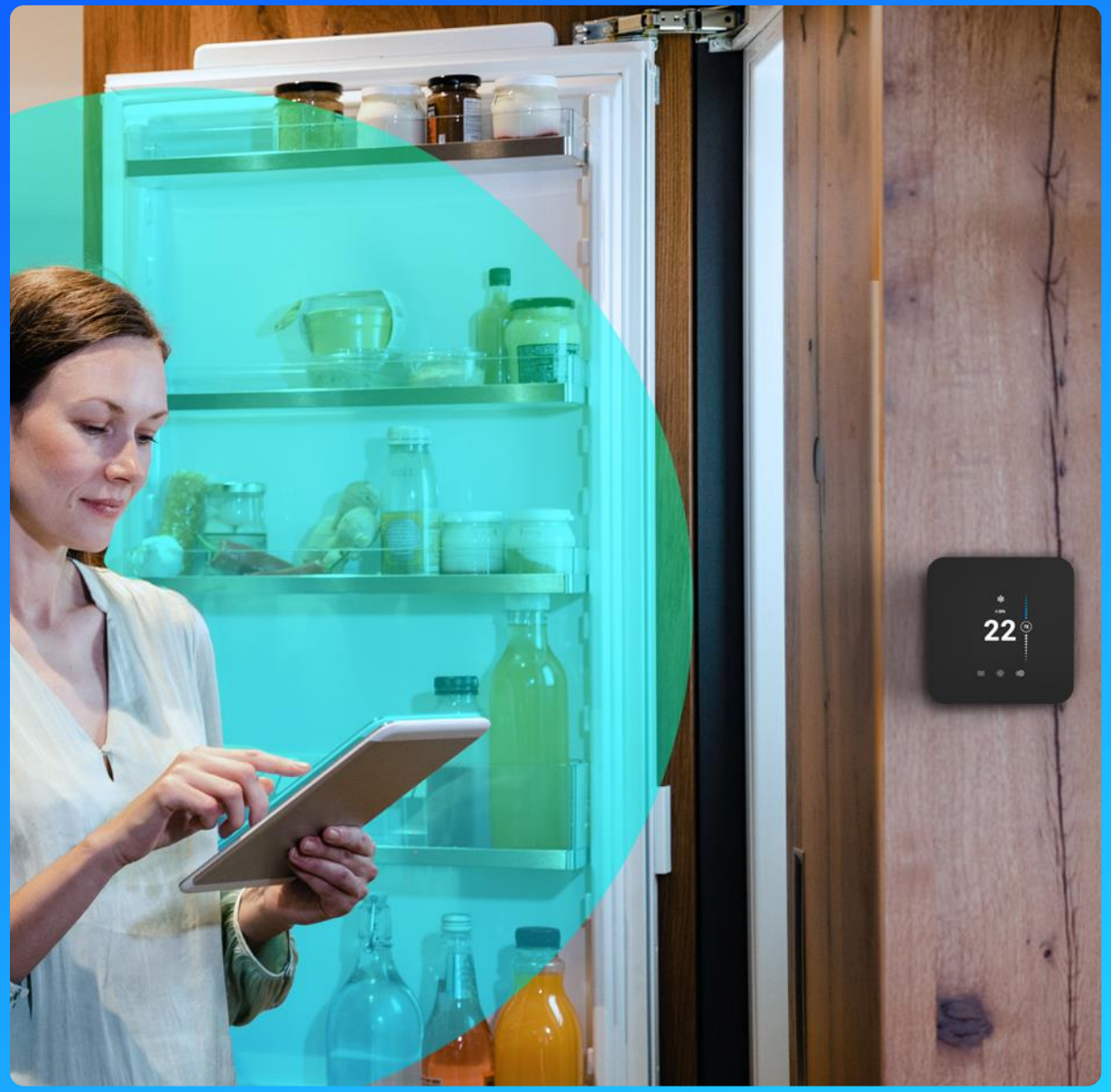
Next Generation IoT Wireless Connectivity

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

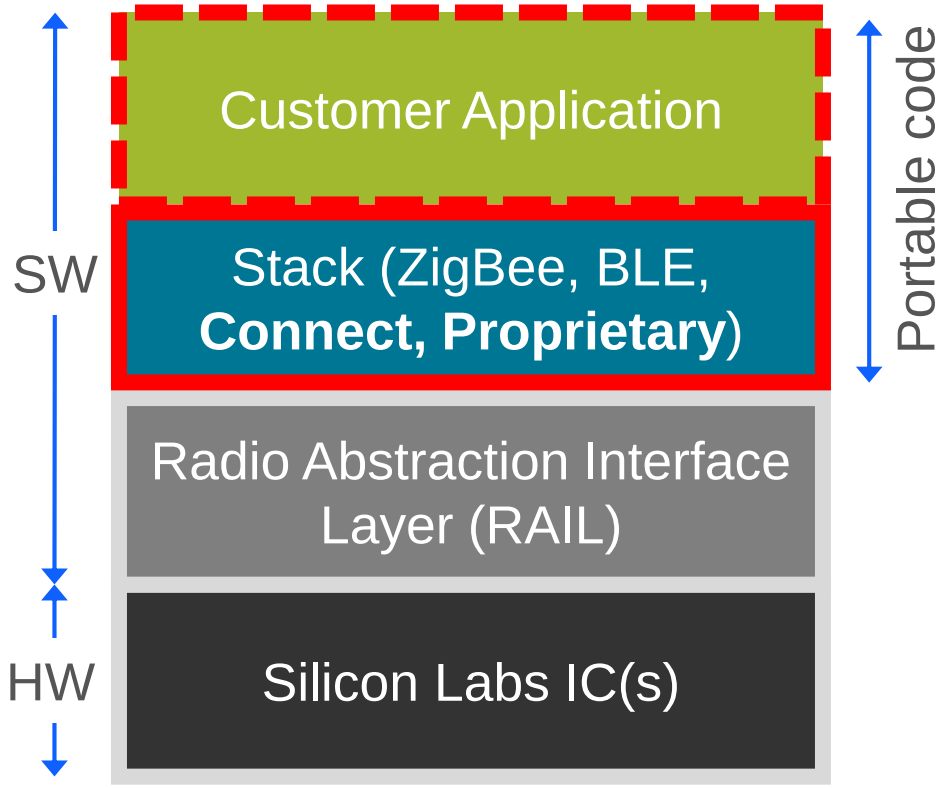
Garage Door Openers

Public Infrastructure

Agriculture

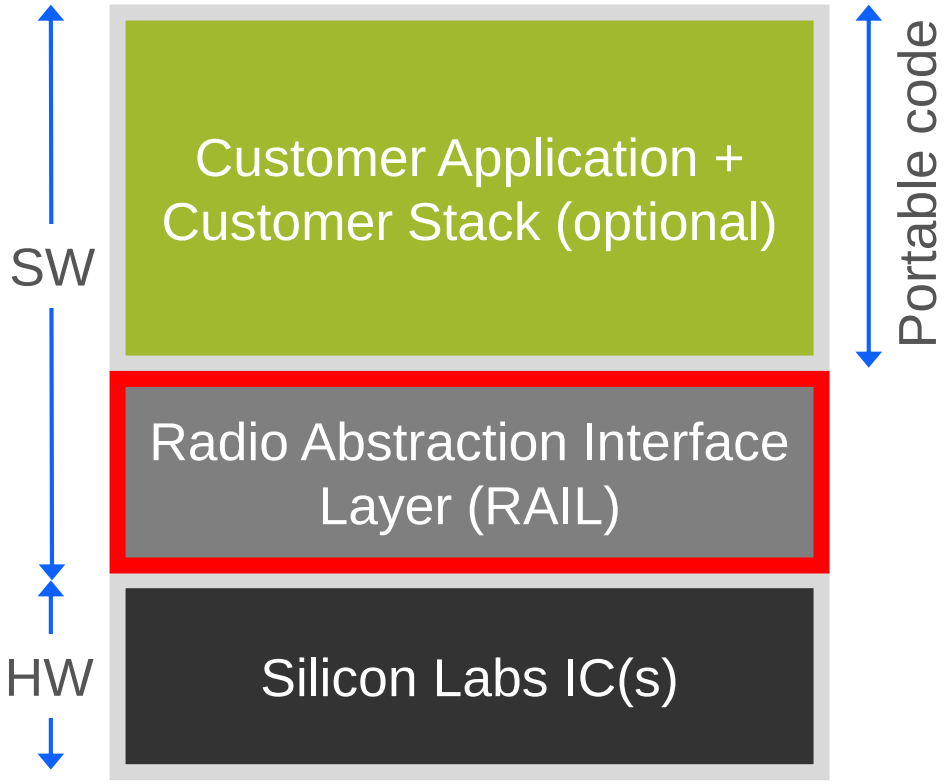
Asset Tracking & ESL

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

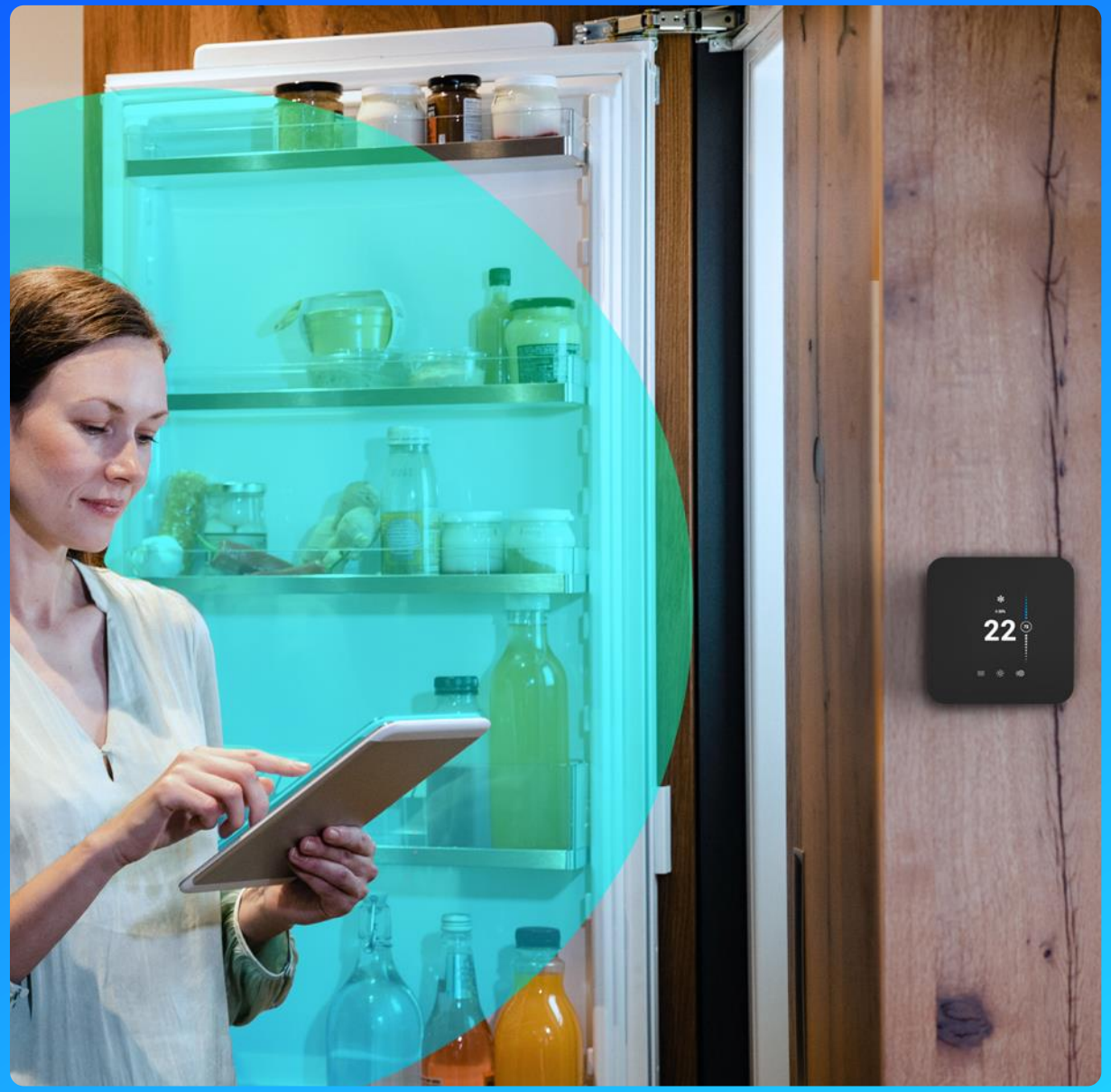
- Pros:
  - Very flexible
  - Support legacy proprietary systems
  - Can be based on custom PHY configuration
  - More efficient resource footprint
- Cons:
  - No network layer, so no multi-hop support
  - No application features like OTA
  - Security must be done in application

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

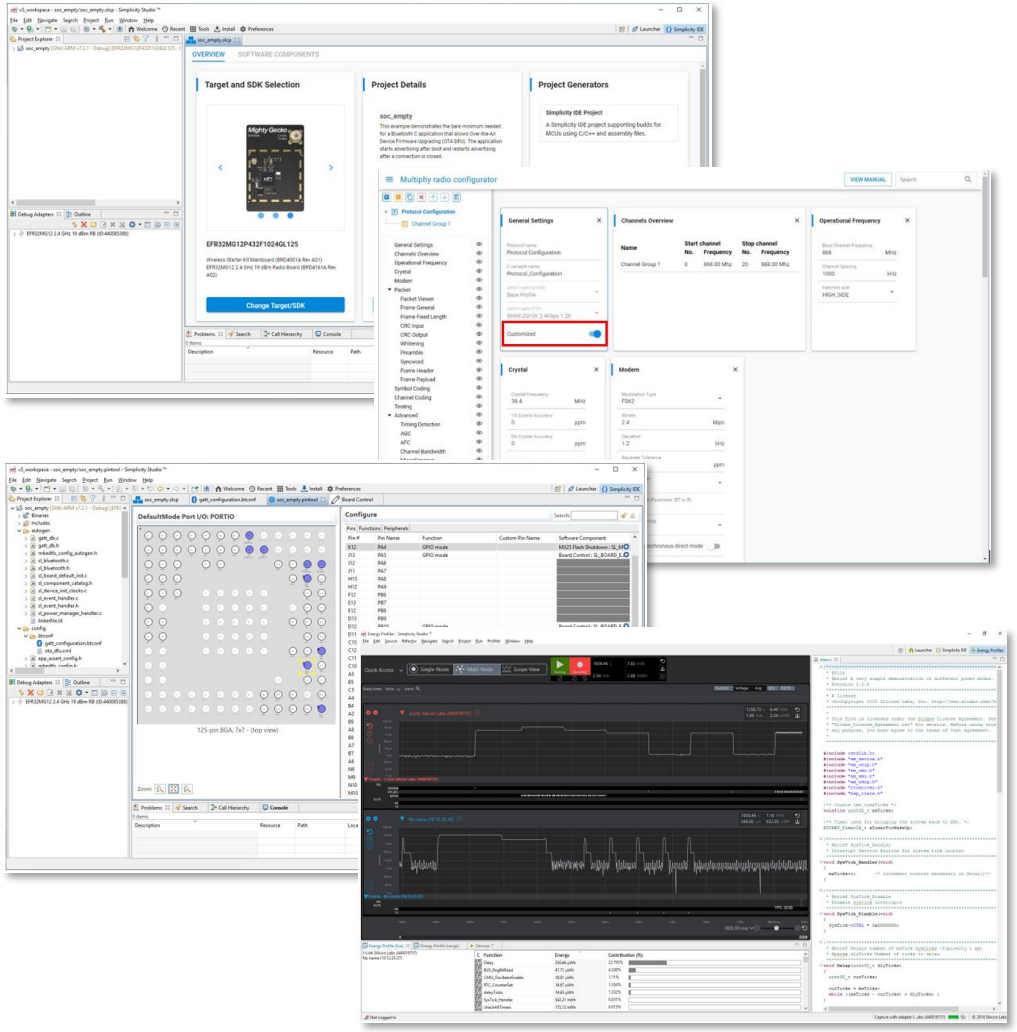
Creating a RAIL Project

In Simplicity Studio v5



Tools and API

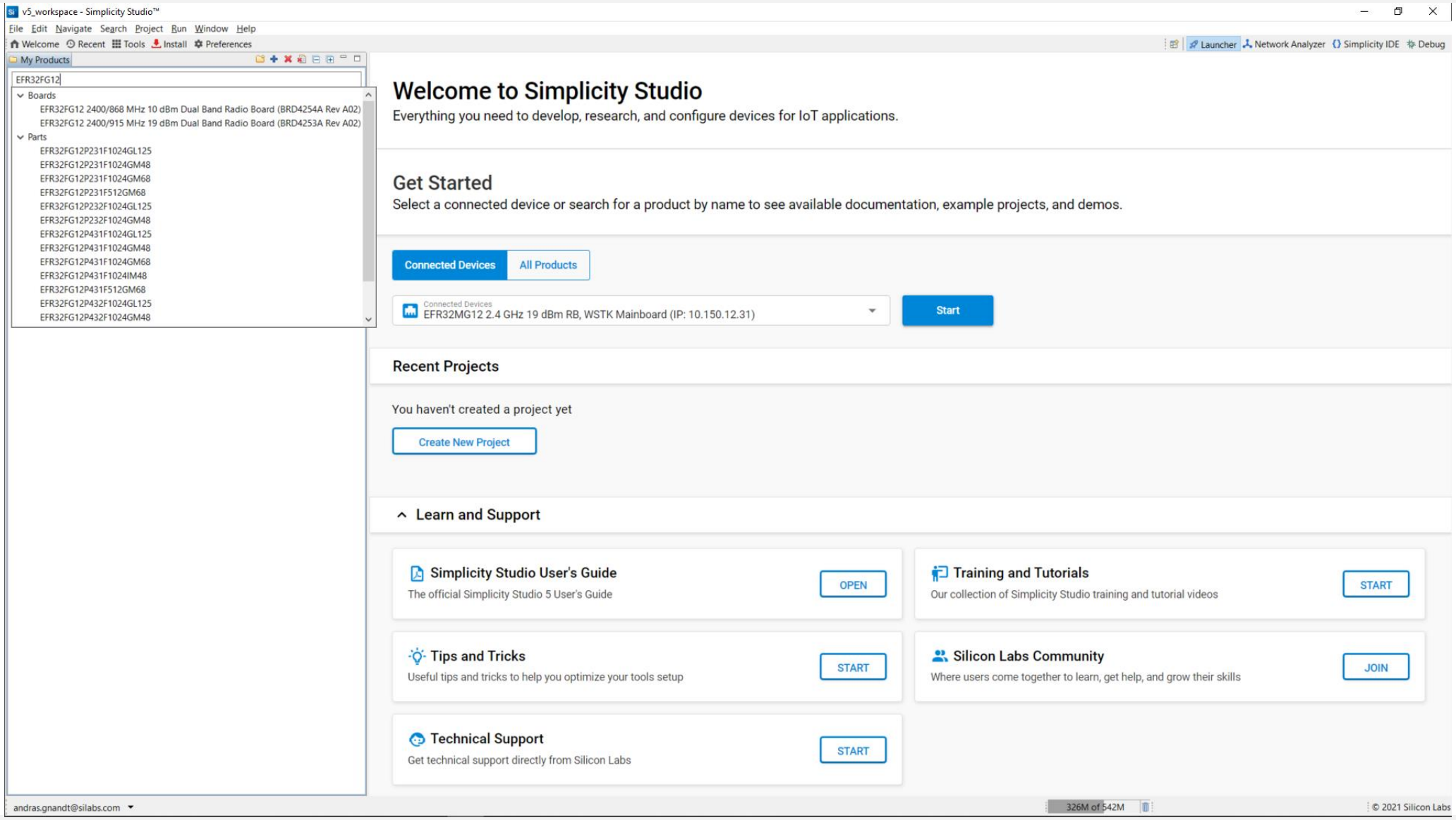
SSv5 Tools



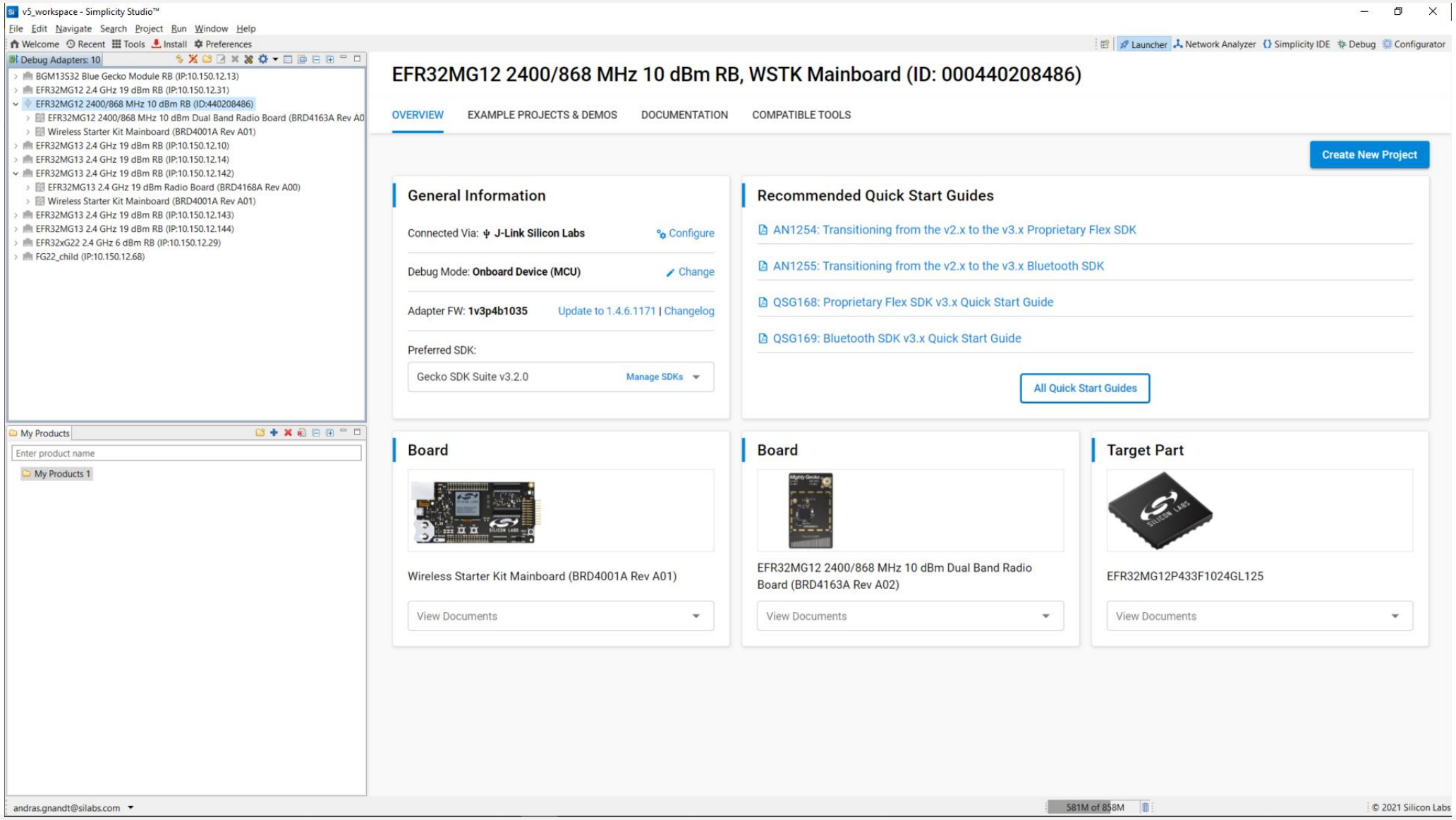
RAIL API

- 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

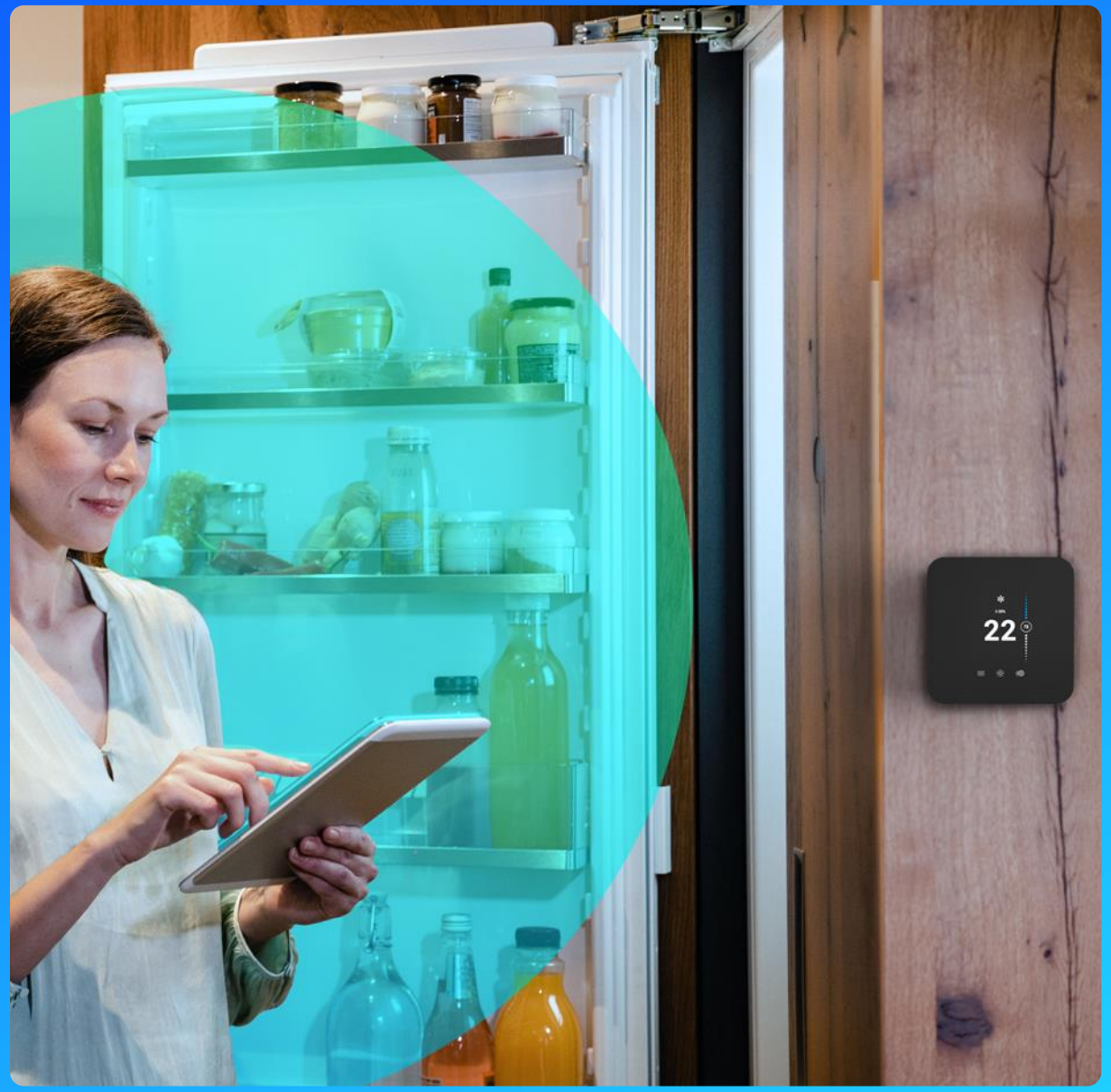


Studio v5 – Starting up with a connected device

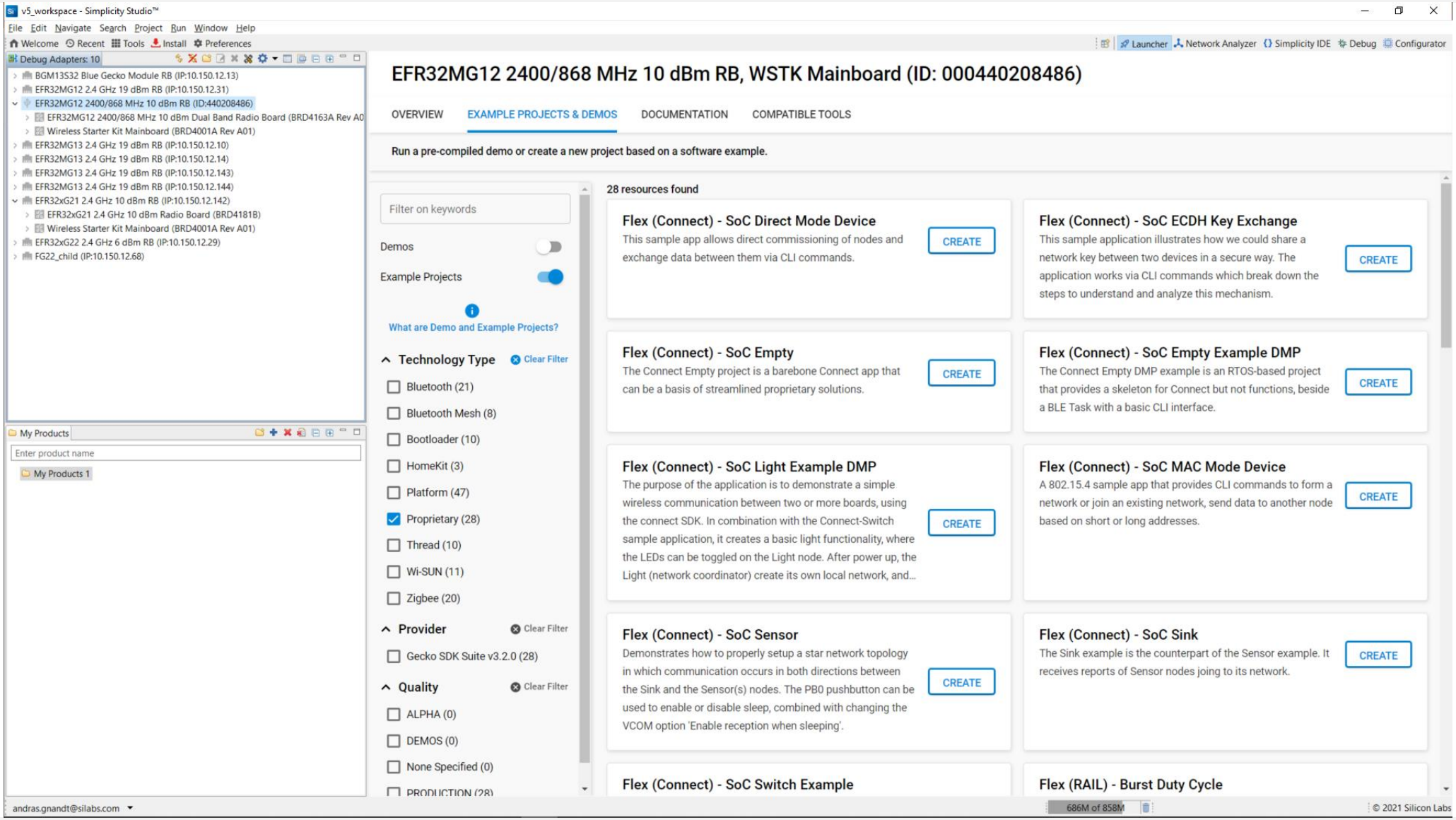


RAIL SimpleTRX Walkthrough

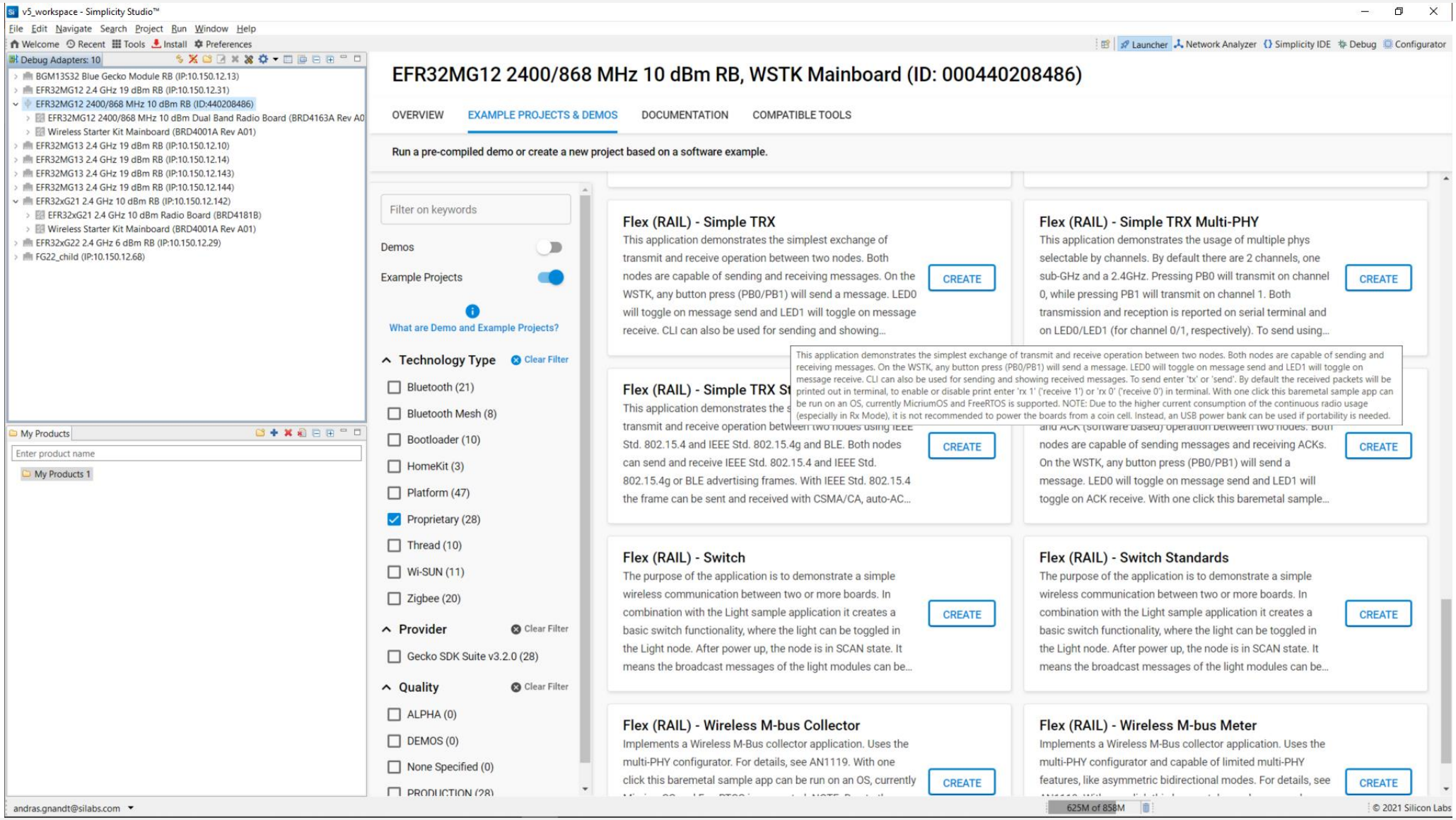
Create Project in SSV5



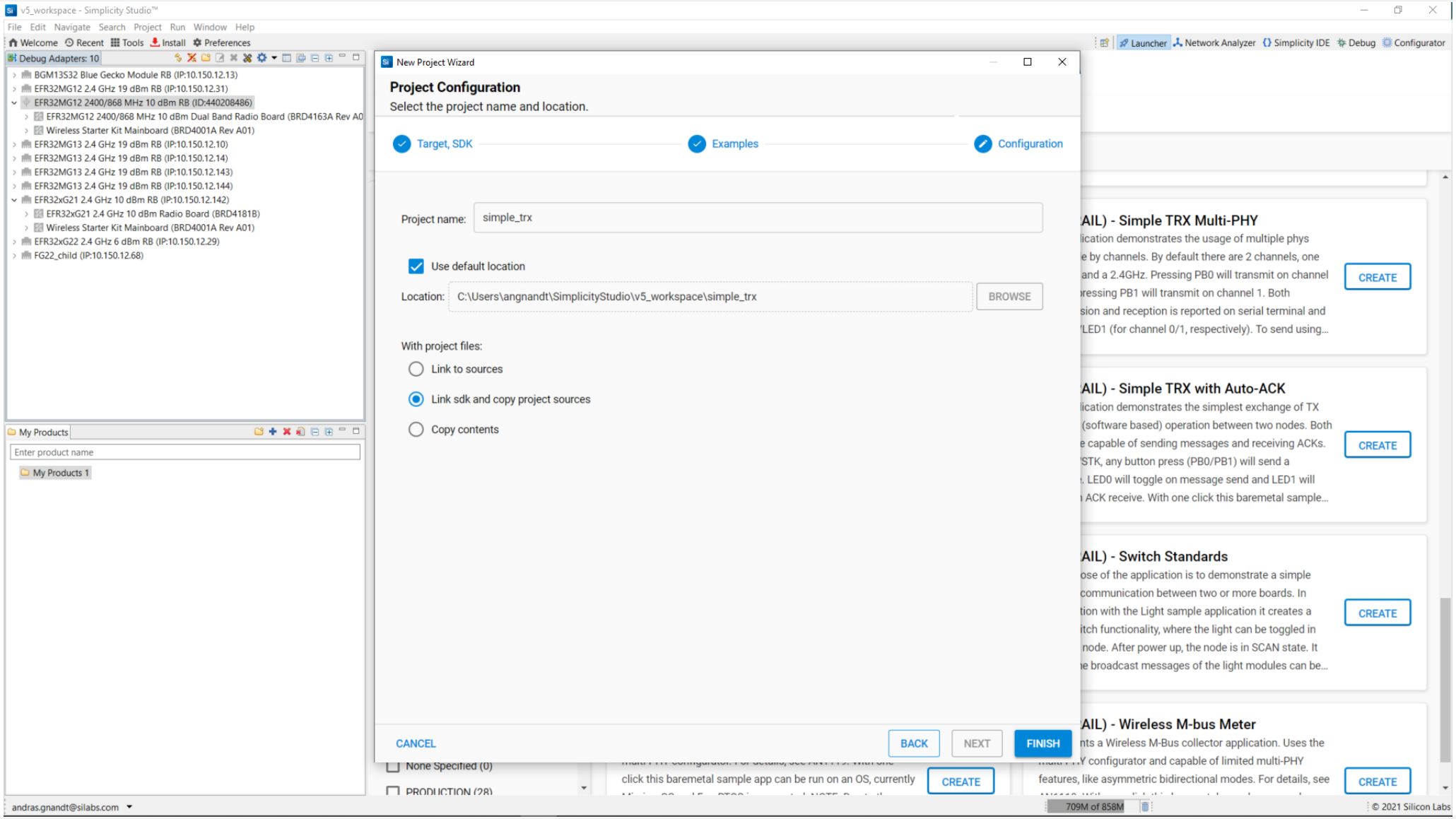
Studio v5 – Example Projects for Proprietary



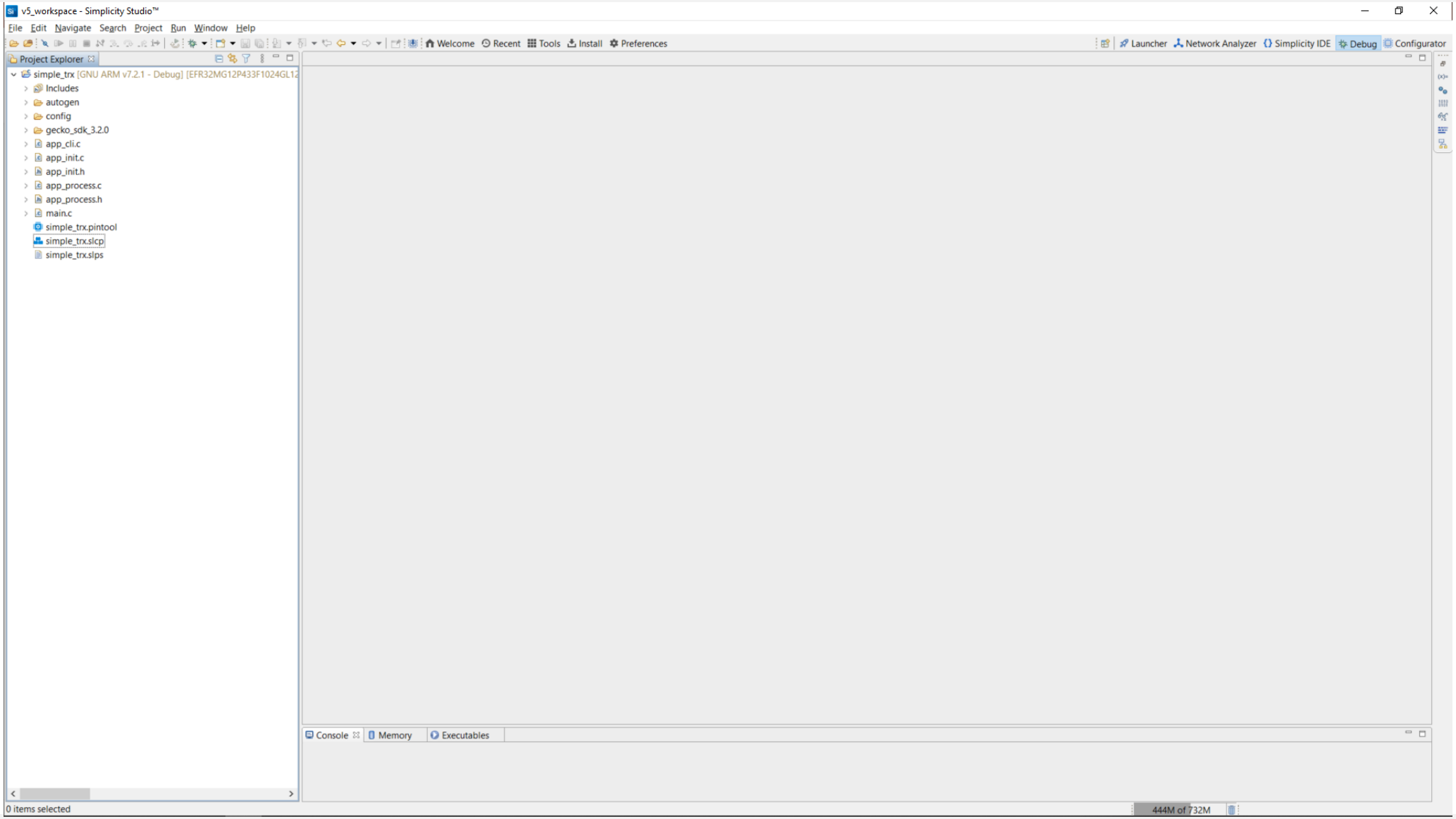
Studio v5 – SimpleTRX Example Project



Create SimpleTRX Sample Project

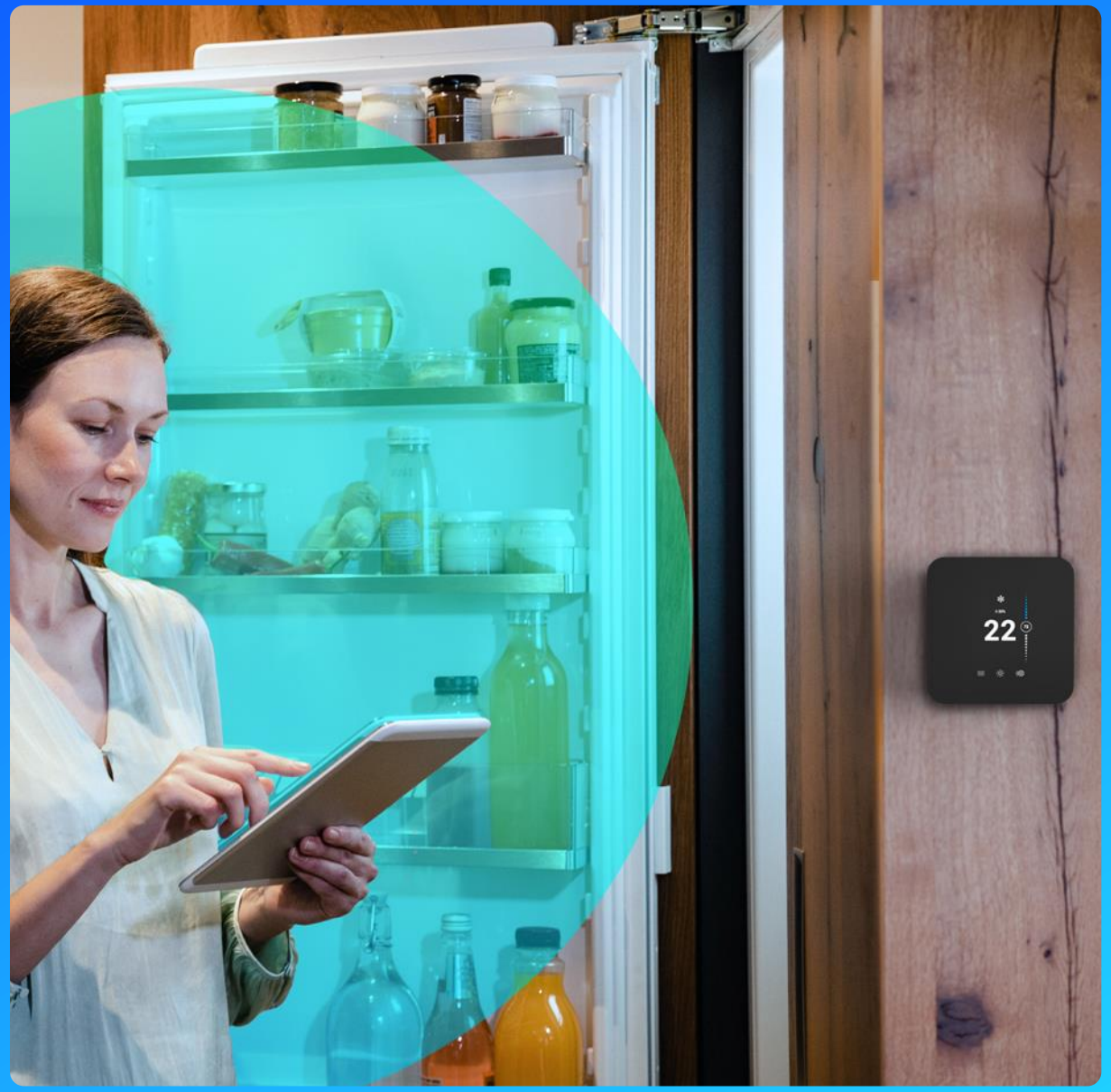


SimpleTRX Project Components

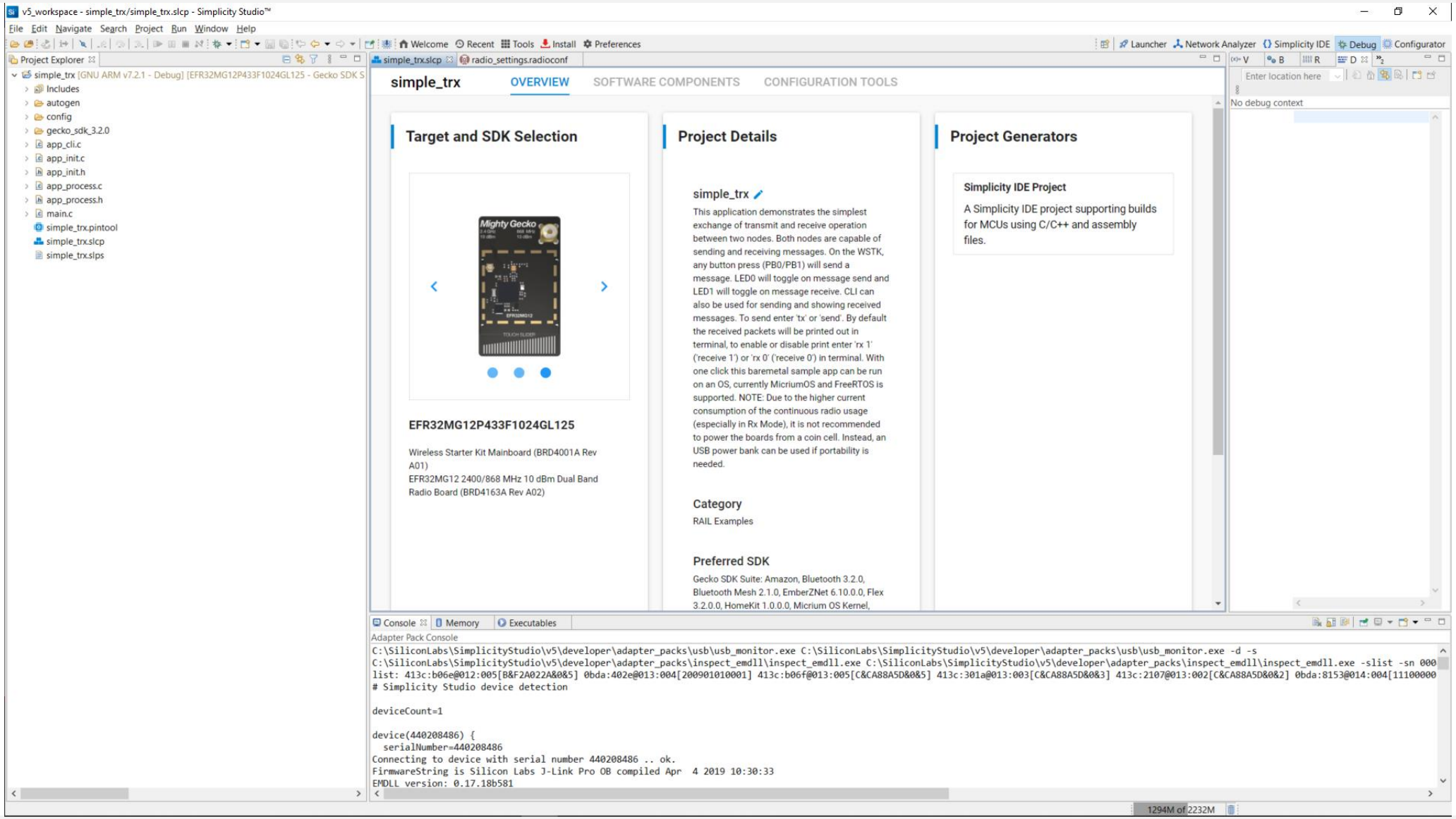


Software Components

Demo



SimpleTRX Project Details



Drivers – USART vcom

v5_workspace - simple_trx/simple_trx.slcp - Simplicity Studio™

File Edit Navigate Search Project Run Window Help

Welcome Recent Tools Install Preferences | Launcher Network Analyzer Simplicity IDE Debug Configurator

Project Explorer

- simple_trx [GNU ARM v7.2.1 - Debug] [EFR32MG12P433F1024GL12
  - 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.slcp
  - simple_trx.slps

simple_trx.slcp | radio_settings.radioconf

simple_trx — OVERVIEW | SOFTWARE COMPONENTS | CONFIGURATION TOOLS

Filter : Configurable Components ☐ Installed Components ☐ Components Installed by You ☐ Search keywords, component's name

- IO
- IO Stream
  - IO Stream: USART
    - vcom
  - IO Stream
  - IO Stream VUART
  - IO Stream debug
  - IO Stream: LEUART
  - IO Stream: LEUART Core
  - IO Stream: RTT
  - IO Stream: Retarget STDIO
  - IO Stream: STDLIB Configuration
  - IO Stream: SWO
  - IO Stream: USART Core
- Interrupt
- NVM3
- Simulated EEPROM
- System Setup
- Token Manager

vcom — Configure

Function Documentation

sl_iostream_usart_init()

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

USART Stream init.

Parameters

[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.

✕ Uninstall | + Add New Instances | View Dependencies & Instances

Console | Memory | Executables

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 000440208486
```

893M of 2232M

Drivers – Simple button

v5_workspace - simple_trx/simple_trx.slcp - Simplicity Studio™

File Edit Navigate Search Project Run Window Help

Welcome Recent Tools Install Preferences | Launcher Network Analyzer Simplicity IDE Debug Configurator

Project Explorer

- simple_trx [GNU ARM v7.2.1 - Debug] [EFR32MG12P433F1024GL12
  - 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.slcp
  - simple_trx.slps

simple_trx.slcp | radio_settings.radioconf

simple_trx — OVERVIEW | SOFTWARE COMPONENTS | CONFIGURATION TOOLS

Filter : Configurable Components ☐ Installed Components ☐ Components Installed by You ☐ Search keywords, component's name

- OpenThread
- Platform
  - Board
  - Board Drivers
  - Bootloader
  - CMSIS
  - Common
  - Device
  - Driver
    - Button
      - Simple Button
        - btn0
        - btn1
      - Generic Button API
    - I2C
    - LED
    - PWM
    - SPI
    - UART
    - Coulomb Counter

Simple Button — Configure

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".

```
// sl_simple_button_instances.c

#include "sl_simple_button.h"
#include "sl_simple_button_inst0_config.h"

sl_simple_button_context_t simple_inst0_context = {
  .state = 0,
  .history = 0,
  .port = SL_SIMPLE_BUTTON_INST0_PORT,
  .pin = SL_SIMPLE_BUTTON_INST0_PIN,
  .mode = SL_SIMPLE_BUTTON_INST0_MODE,
};

const sl_button_t sl_button_inst0 = {
  .context = &simple_inst0_context,
  .init = sl_simple_button_init,
  .get_state = sl_simple_button_get_state,
  .poll = sl_simple_button_poll_step,
};

const sl_button_t *sl_simple_button_array[] = {&sl_button_inst0};
const uint8_t simple_button_count = 1;

void sl_simple_button_init_instances(void)
{
```

✕ Uninstall | + Add New Instances | View Dependencies & Instances

Console | Memory | Executables

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 000440208486
```

1339M of 2232M

Drivers - USART

v5_workspace - simple_trx/simple_trx.slcp - Simplicity Studio™

File Edit Navigate Search Project Run Window Help

Welcome Recent Tools Install Preferences Launcher Network Analyzer Simplicity IDE Debug Configurator

Project Explorer

simple_trx [GNU ARM v7.2.1 - Debug] [EFR32MG12P433F1024GL12]
- 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.slcp
- simple_trx.slps

simple_trx.slcp | radio_settings.radioconf

simple_trx OVERVIEW SOFTWARE COMPONENTS CONFIGURATION TOOLS

Filter : Configurable Components ☐ Installed Components ☐ Components Installed by You ☐ Search keywords, component's name

- OpenThread
- Platform
  - Board
  - Board Drivers
  - Bootloader
  - CMSIS
  - Common
  - Device
  - Driver
    - Button
    - I2C
    - LED
    - PWM
    - SPI
    - UART
      - UARTDRV Core
      - UARTDRV LEUART
      - UARTDRV USART
    - Coulomb Counter
    - Cycle Counter

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

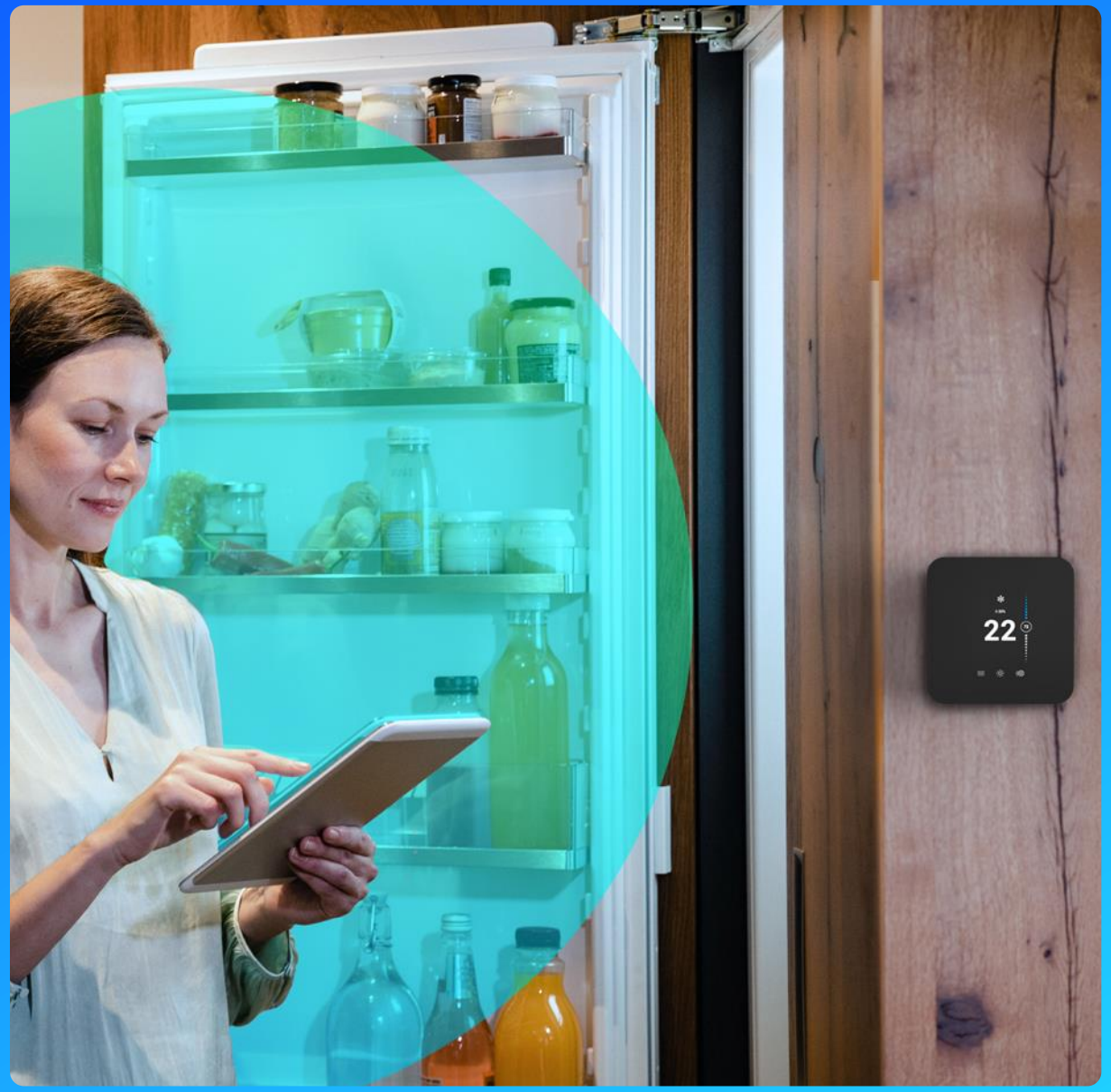
Adapter Pack Console

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

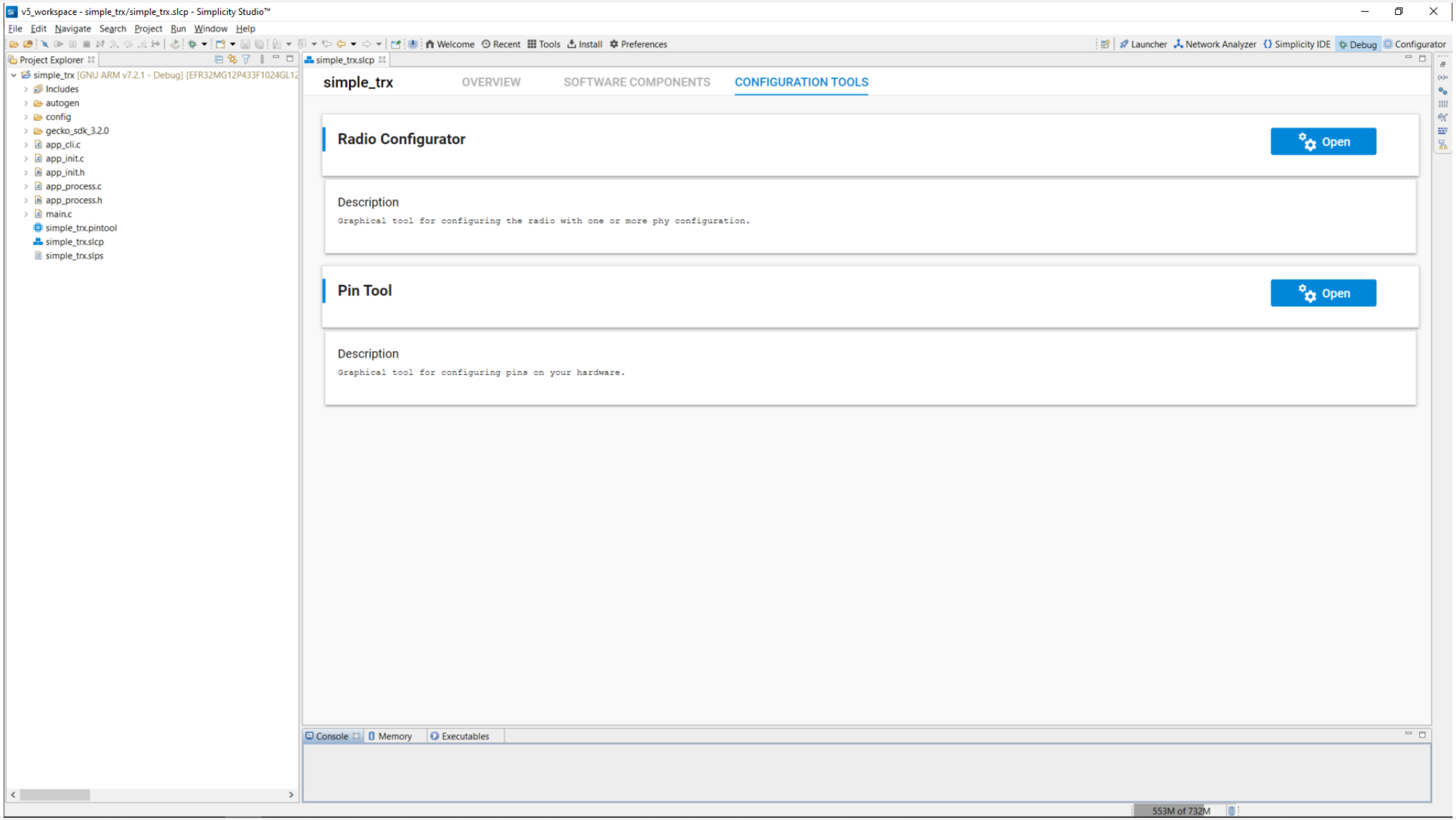
1189M of 2232M

Radio Configurator Flow

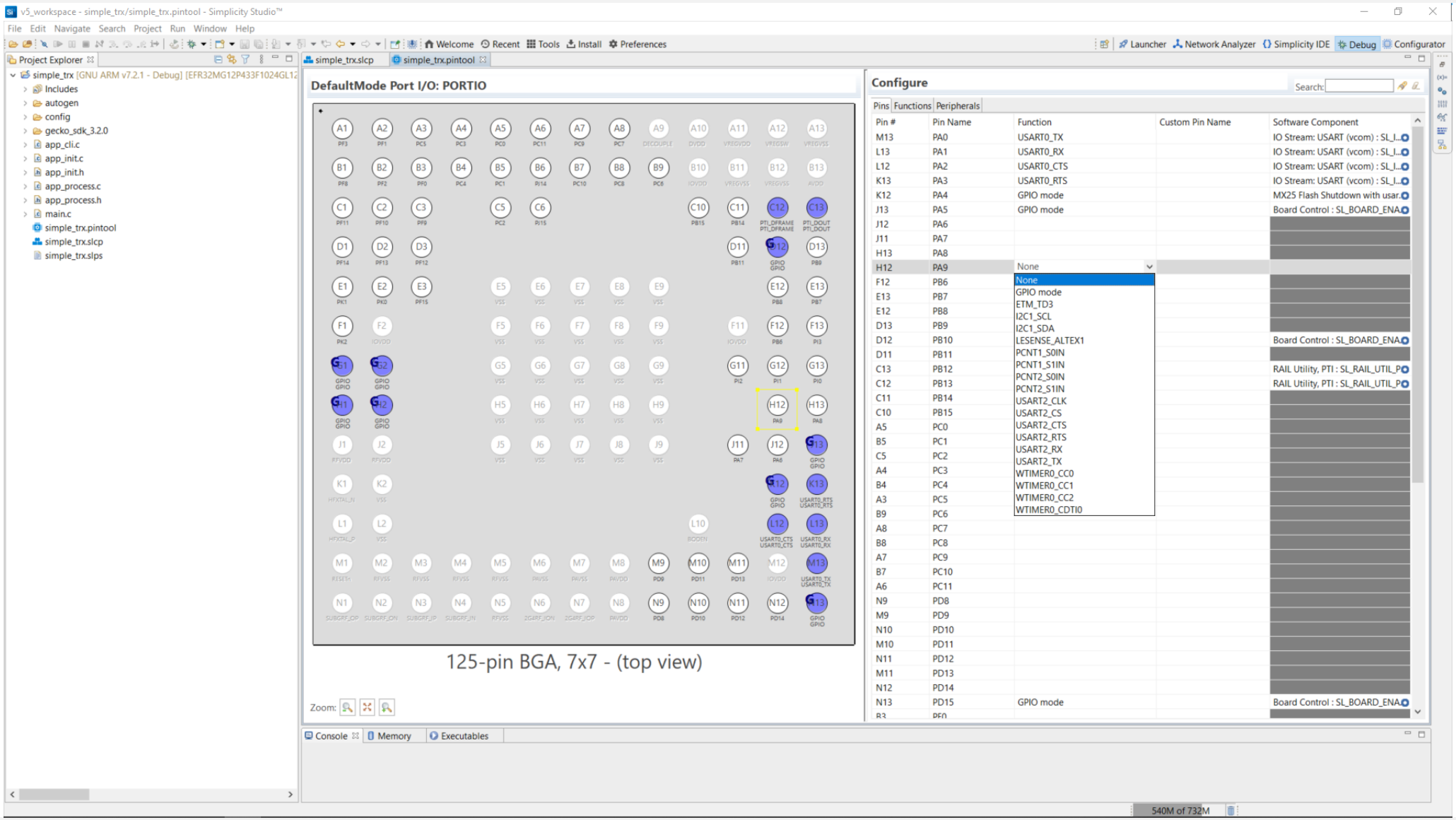
Demo



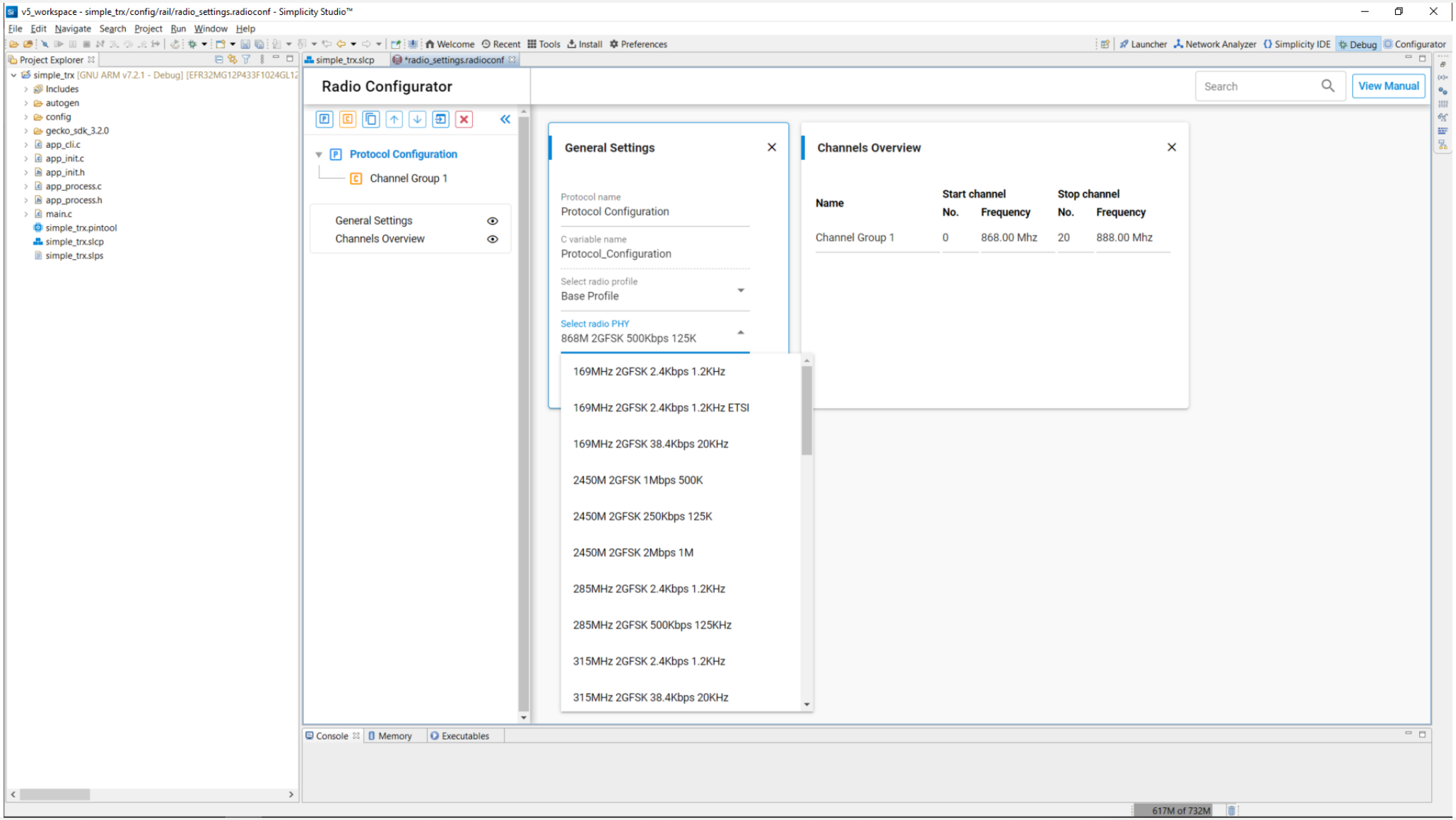
Configuration Tools



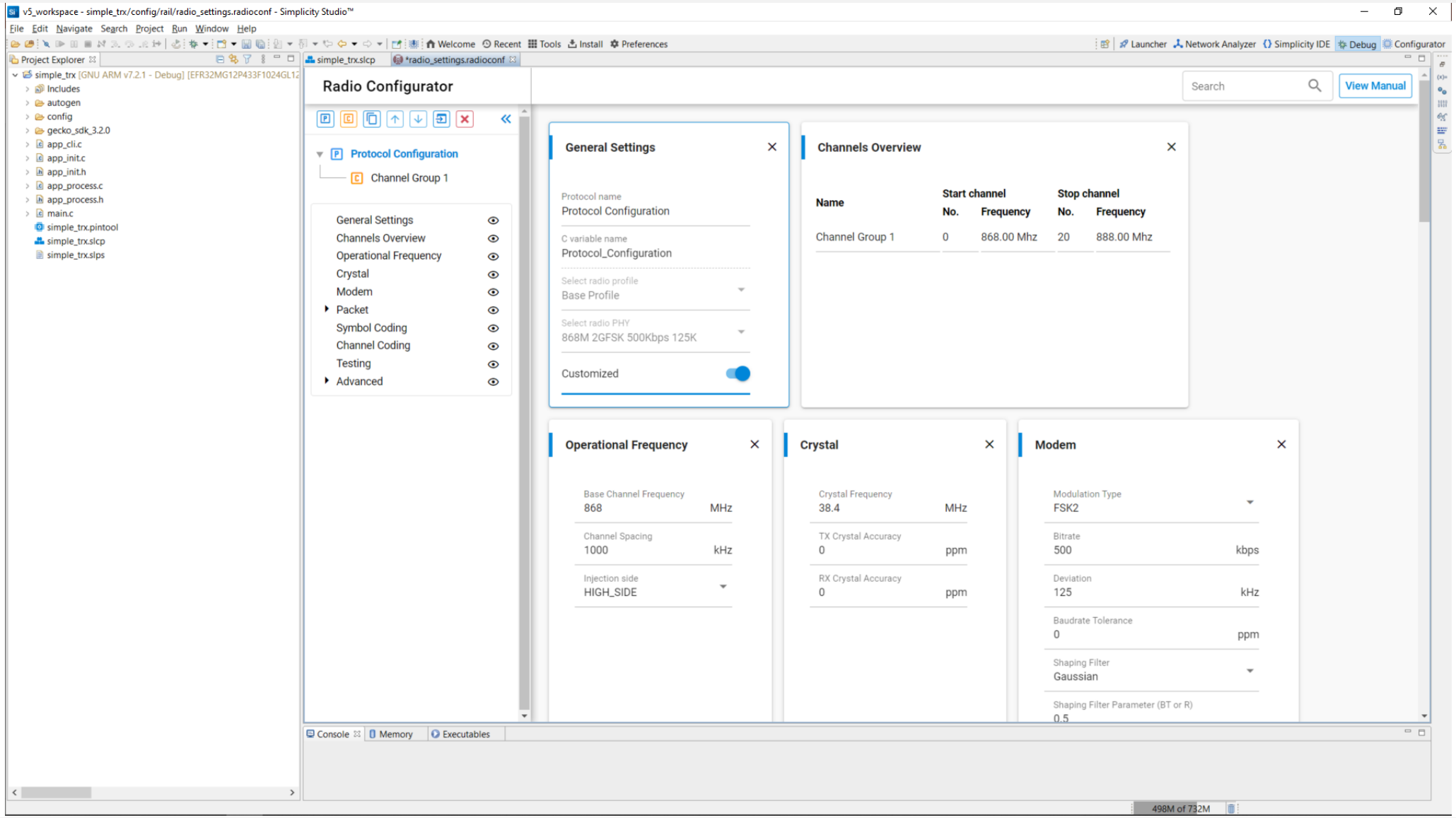
Configuration Tools – Pin Tool



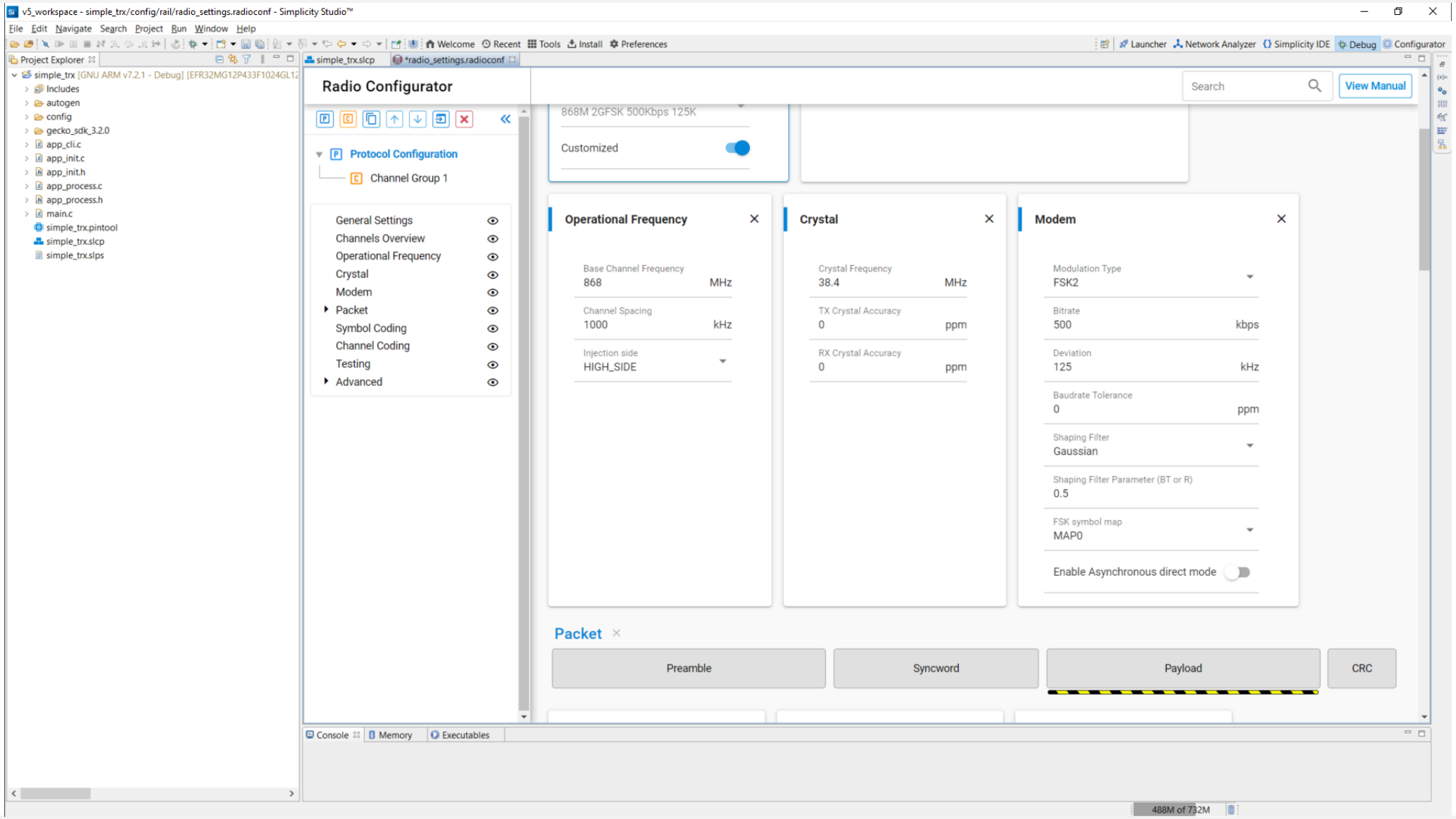
Configuration Tool – Radio Configurator



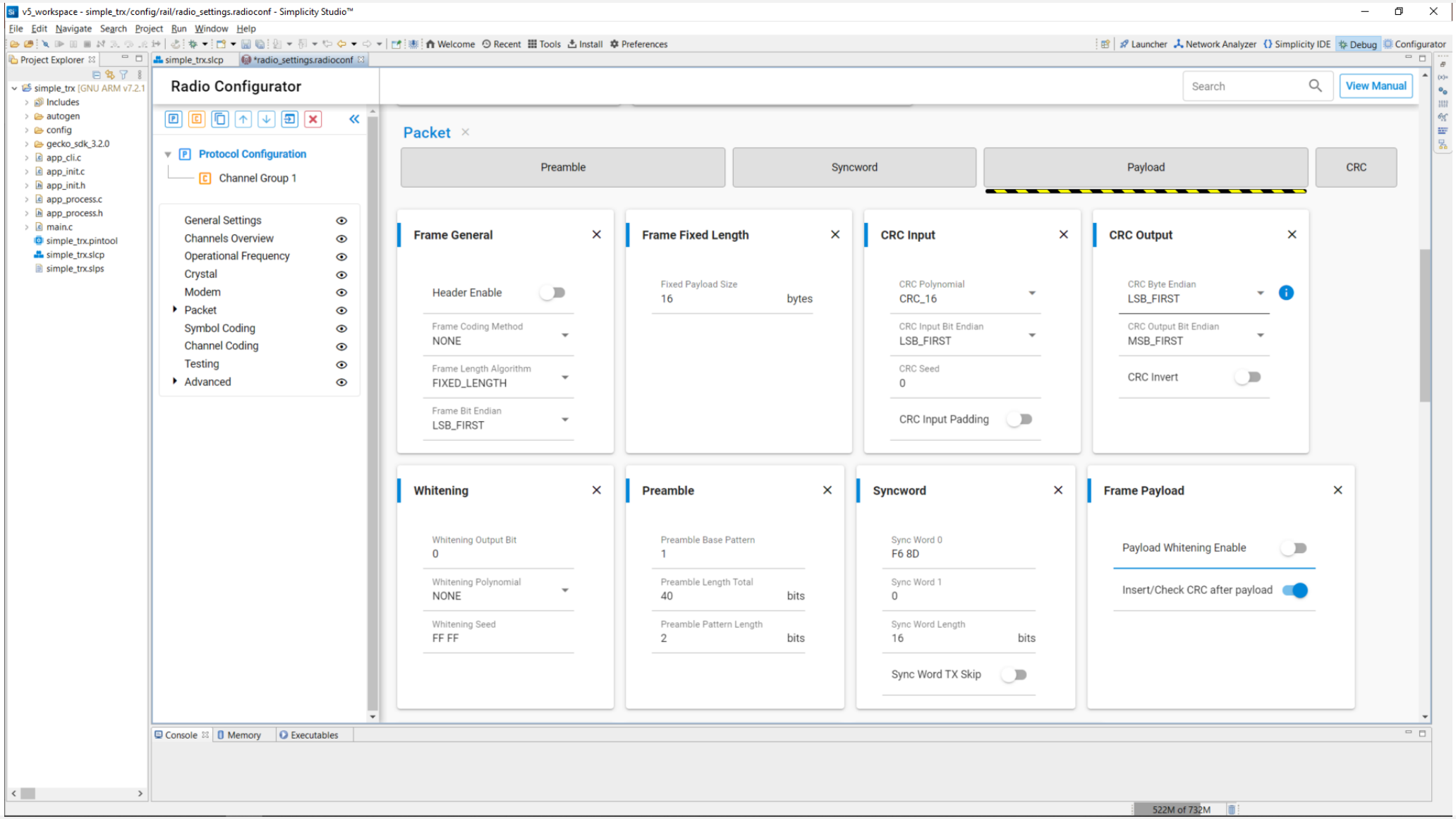
Radio Configurator - Customized



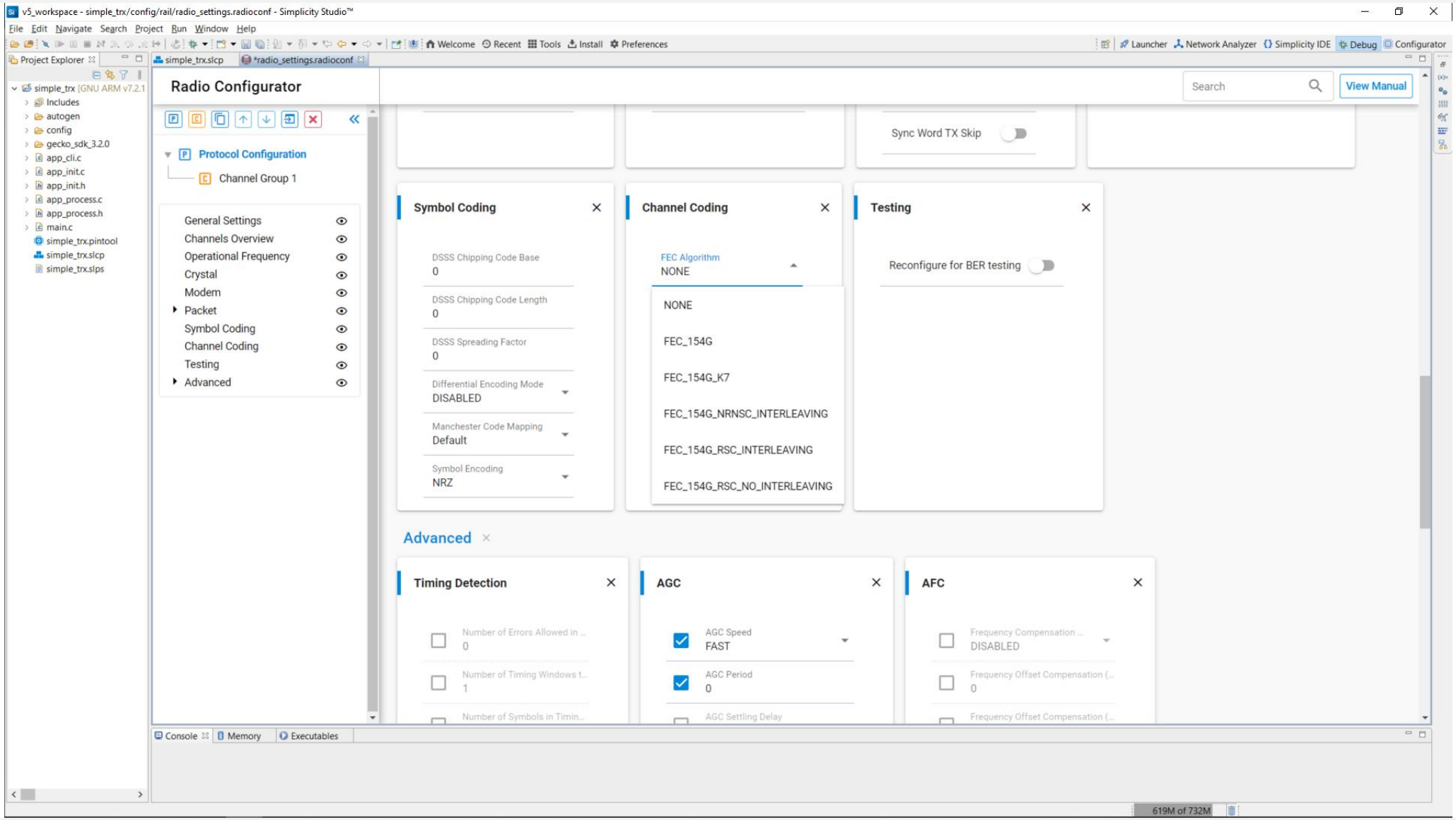
Radio Configurator – Modem settings



Radio Configurator – Packet structure settings

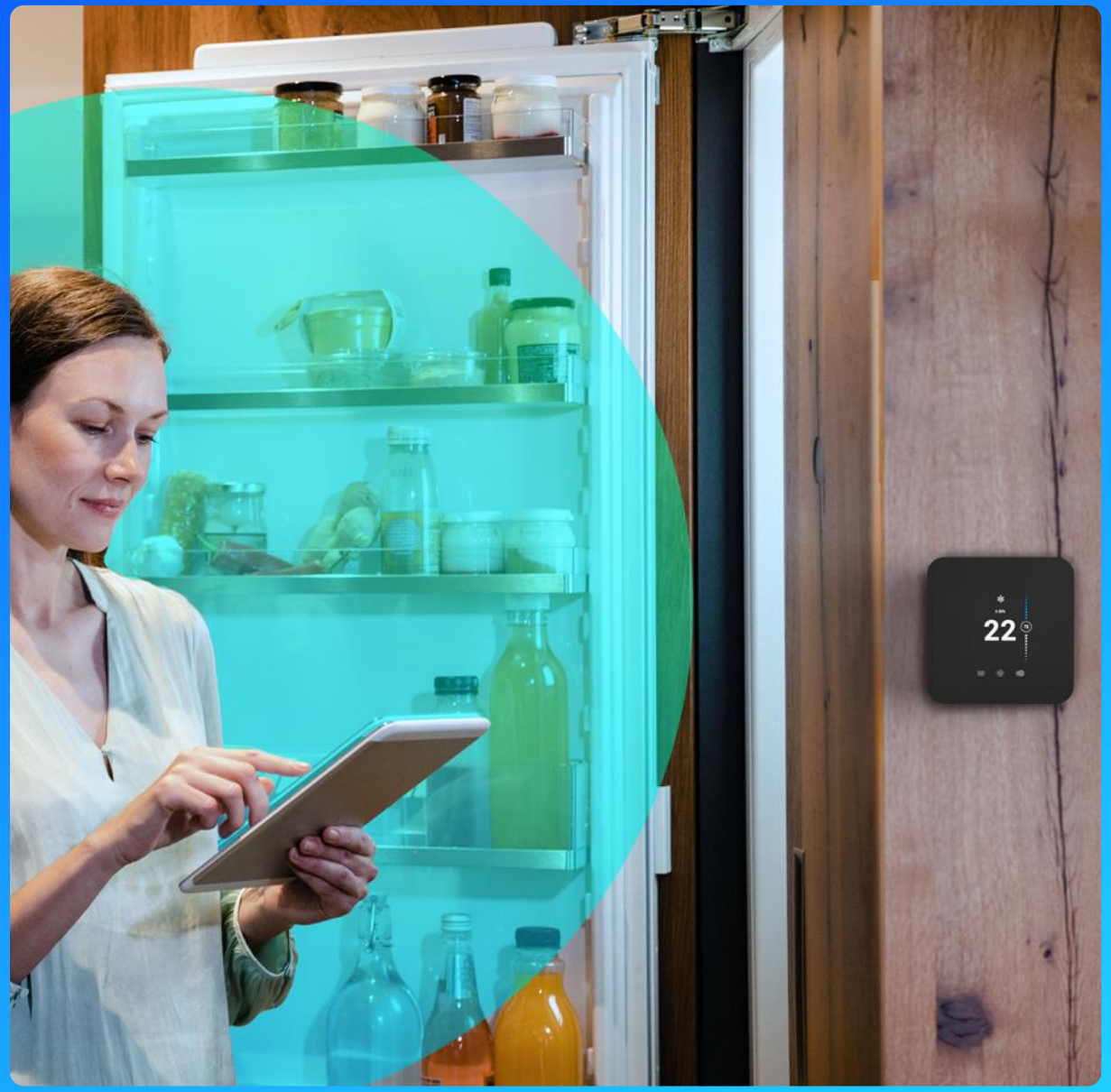


Radio Configurator – Packet structure settings



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

Welcome Recent Tools Install Preferences Launcher Network Analyzer Simplicity IDE Debug Configurator

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.slcp
  - simple_trx.slps

simple_trx.slcp | *radio_settings.radioconf | main.c

```c
// -----------------------------------------------------------------------------
//                                   Static Variables
// -----------------------------------------------------------------------------
#if !defined(SL_CATALOG_KERNEL_PRESENT)
/// A static handle of a RAIL instance
static RAIL_Handle_t rail_handle;
#endif
// -----------------------------------------------------------------------------
//                          Public Function Definitions
// -----------------------------------------------------------------------------
/*******************************************************************************
 * Main function
 ******************************************************************************/
int main(void)
{
  // Initialize Silicon Labs device, system, service(s) and protocol stack(s).
  // Note that if the kernel is present, processing task(s) will be created by
  // this call.
  sl_system_init();

  // Initialize the application. For example, create periodic timer(s) or
  // task(s) if the kernel is present.
#if defined(SL_CATALOG_KERNEL_PRESENT)
  app_task_init();
#else
  rail_handle = app_init();
#endif

#if defined(SL_CATALOG_KERNEL_PRESENT)
  // Start the kernel. Task(s) created in app_init() will start running.
  sl_system_kernel_start();
#else // SL_CATALOG_KERNEL_PRESENT
  while (1) {
    // Do not remove this call: Silicon Labs components process action routine
    // must be called from the super loop.
    sl_system_process_action();

    // Application process.
    app_process_action(rail_handle);

#if defined(SL_CATALOG_POWER_MANAGER_PRESENT)
    // Let the CPU go to sleep if the system allows it.
    sl_power_manager_sleep();
#endif
  }
#endif // SL_CATALOG_KERNEL_PRESENT
}

// -----------------------------------------------------------------------------
//                          Static Function Definitions
// -----------------------------------------------------------------------------
```

Console | Memory | Executables

Writable | Smart Insert | 1 : 1 : 0 | 623M of 732M

SimpleTRX – RAIL App Init

v5_workspace - simple_trx/app_init.c - Simplicity Studio™

File Edit Source Refactor Navigate Search Project Run Window Help

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.slcp
  - simple_trx.slps

simple_trx.slcp | *radio_settings.radioconf | main.c | app_process.c | app_init.c

```c
// -----------------------------------------------------------------------------
//                          Public Function Definitions
// -----------------------------------------------------------------------------
/******************************************************************************
 * Print sample app name
 ******************************************************************************/
SL_WEAK void print_sample_app_name(const char* app_name)
{
  app_log_info("%s\n", app_name);
}

/******************************************************************************
 * The function is used for some basic initialization related to the app.
 ******************************************************************************/
RAIL_Handle_t app_init(void)
{
  validation_check();

  // Get RAIL handle, used later by the application
  RAIL_Handle_t rail_handle = sl_rail_util_get_handle(SL_RAIL_UTIL_HANDLE_INST0);

  set_up_tx_fifo(rail_handle);

  // Turn OFF LEDs
  sl_led_turn_off(&sl_led_led0);
#if defined(SL_CATALOG_LED1_PRESENT)
  sl_led_turn_off(&sl_led_led1);
#endif
  // Start reception
  RAIL_Status_t status = RAIL_StartRx(rail_handle, CHANNEL, NULL);
  if (status != RAIL_STATUS_NO_ERROR) {
    app_log_warning("After initialization RAIL_StartRx() result:%d ", status);
  }

  // CLI info message
  print_sample_app_name("Simple TRX");

  return rail_handle;
}

// -----------------------------------------------------------------------------
//                          Static Function Definitions
// -----------------------------------------------------------------------------
/******************************************************************************
 * Checks phy setting to avoid errors at packet sending
 ******************************************************************************/
static void validation_check(void)
{
  _Static_assert(SL_RAIL_UTIL_INIT_PROTOCOL_INST0_DEFAULT == SL_RAIL_UTIL_PROTOCOL_PROPRIETARY,
                 "Please use the Flex (RAIL) - Simple TRX Standards sample app instead, which is designed to show the protocol usage.");
}
```

Console | Memory | Executables

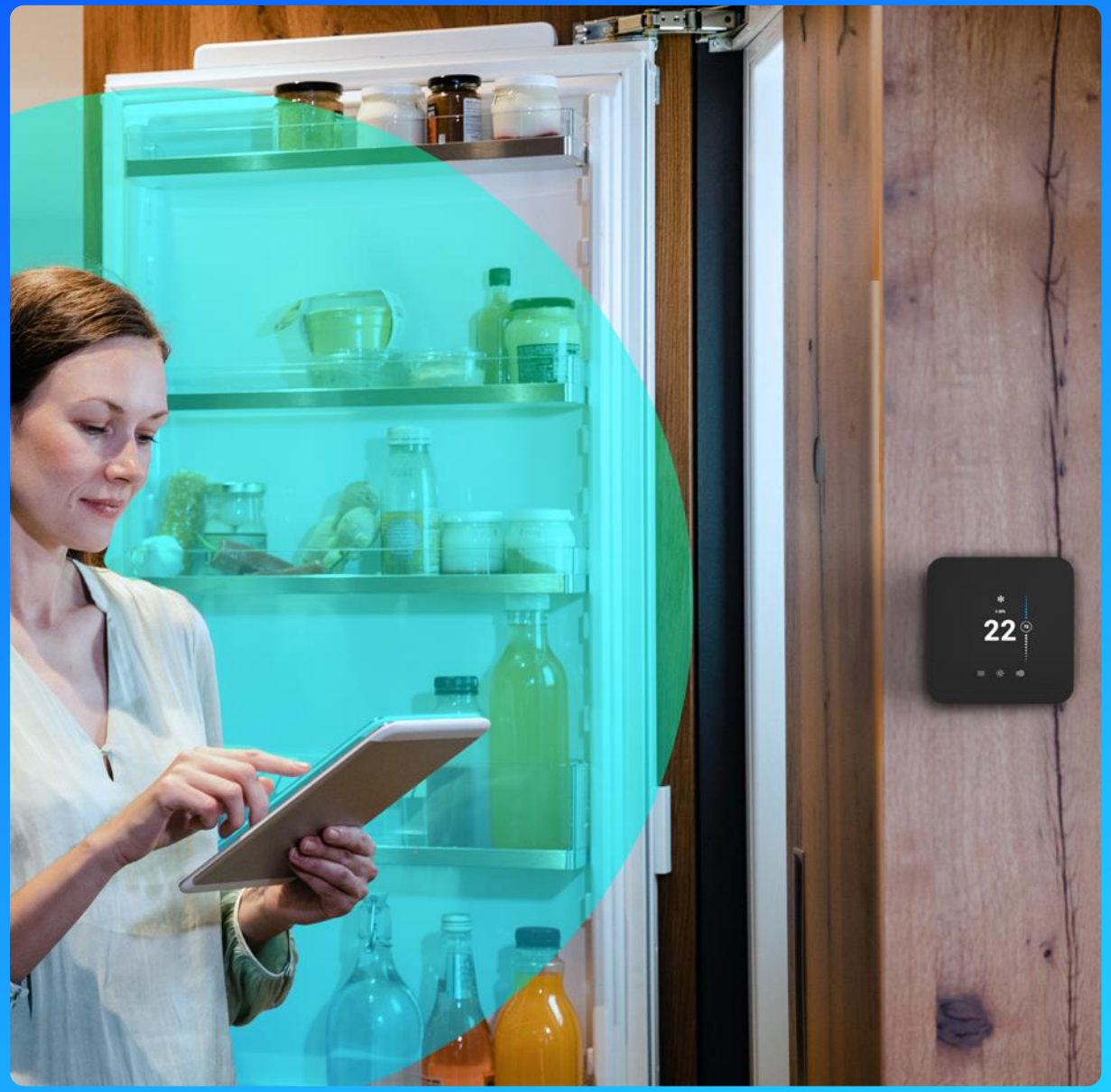
Writable | Smart Insert | 1 : 1 : 0 | 626M of 732M

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 trough app_init:
  - Tx fifo (required for tx only)
  - Start rx (sets starting radio state)

RAIL SimpleTRX Walkthrough

Process



SimpleTRX – Process - Functions

v5_workspace - simple_trx/app_process.c - Simplicity Studio™

File Edit Source Refactor Navigate Search Project Run Window Help

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.slcp
  - simple_trx.slps

simple_trx.slcp | *radio_settings.radioconf | main.c | app_process.c

```c
// -----------------------------------------------------------------------------
//                                Macros and Typedefs
// -----------------------------------------------------------------------------
/// Size of RAIL RX/TX FIFO
#define RAIL_FIFO_SIZE (256u)
/// Transmit data length
#define TX_PAYLOAD_LENGTH (16u)

/// State machine of simple_trx
typedef enum {
  S_PACKET_RECEIVED,
  S_PACKET_SENT,
  S_RX_PACKET_ERROR,
  S_TX_PACKET_ERROR,
  S_CALIBRATION_ERROR,
  S_IDLE,
} state_t;

// -----------------------------------------------------------------------------
//                           Static Function Declarations
// -----------------------------------------------------------------------------
/******************************************************************************//**
 * The function printfs the received rx message.
 *
 * @param rx_buffer Msg buffer
 * @param length How many bytes should be printed out
 * @returns None
 ******************************************************************************/
static void printf_rx_packet(const uint8_t * const rx_buffer, uint16_t length);

/******************************************************************************
 * The API helps to unpack the received packet, point to the payload and returns the length.
 *
 * @param rx_destination Where should the full packet be unpacked
 * @param packet_information Where should all the information of the packet stored
 * @param start_of_payload Pointer where the payload starts
 * @return The length of the received payload
 ******************************************************************************/
static uint16_t unpack_packet(uint8_t *rx_destination, const RAIL_RxPacketInfo_t *packet_information, uint8_t **start_of_payload);

/******************************************************************************
 * The API prepares the packet for sending and load it in the RAIL TX FIFO
 *
 * @param rail_handle Which rail handlers should be used for the TX FIFO writing
 * @param out_data The payload buffer
 * @param length The length of the payload
 ******************************************************************************/
static void prepare_package(RAIL_Handle_t rail_handle, uint8_t *out_data, uint16_t length);

// -----------------------------------------------------------------------------
//                                Global Variables
// -----------------------------------------------------------------------------
/// Flag, indicating transmit request (button has pressed / CLI transmit request has occured)
```

Console | Memory | Executables

Writable | Smart Insert | 78 : 1 : 3076 | 588M of 732M

SimpleTRX – Process – State Machine

v5_workspace - simple_trx/app_process.c - Simplicity Studio™

File Edit Source Refactor Navigate Search Project Run Window Help

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.slcp
  - simple_trx.slps

simple_trx.slcp | *radio_settings.radioconf | main.c | app_process.c

```c
172
173   switch (state) {
174     case S_PACKET_RECEIVED:
175       // Packet received:
176       //  - Check whether RAIL_HoldRxPacket() was successful, i.e. packet handle is valid
177       //  - Copy it to the application FIFO
178       //  - Free up the radio FIFO
179       //  - Return to IDLE state i.e. RAIL Rx
180       rx_packet_handle = RAIL_GetRxPacketInfo(rail_handle, RAIL_RX_PACKET_HANDLE_OLDEST_COMPLETE, &packet_info);
181       while (rx_packet_handle != RAIL_RX_PACKET_HANDLE_INVALID) {
182         uint8_t *start_of_packet = 0;
183         uint16_t packet_size = unpack_packet(rx_fifo, &packet_info, &start_of_packet);
184         rail_status = RAIL_ReleaseRxPacket(rail_handle, rx_packet_handle);
185         if (rail_status != RAIL_STATUS_NO_ERROR) {
186           app_log_warning("RAIL_ReleaseRxPacket() result:%d", rail_status);
187         }
188         if (rx_requested) {
189           printf_rx_packet(start_of_packet, packet_size);
190         }
191         sl_led_toggle(&sl_led_led0);
192         rx_packet_handle = RAIL_GetRxPacketInfo(rail_handle, RAIL_RX_PACKET_HANDLE_OLDEST_COMPLETE, &packet_info);
193       }
194       state = S_IDLE;
195       break;
196     case S_PACKET_SENT:
197       app_log_info("Packet has been sent\n");
198 #if defined(SL_CATALOG_LED1_PRESENT)
199       sl_led_toggle(&sl_led_led1);
200 #else
201       sl_led_toggle(&sl_led_led0);
202 #endif
203       state = S_IDLE;
204       break;
205     case S_RX_PACKET_ERROR:
206       // Handle Rx error
207       app_log_error("Radio RX Error occurred\nEvents: %llX\n", error_code);
208       state = S_IDLE;
209       break;
210     case S_TX_PACKET_ERROR:
211       // Handle Tx error
212       app_log_error("Radio TX Error occurred\nEvents: %llX\n", error_code);
213       state = S_IDLE;
214       break;
215     case S_IDLE:
216       if (tx_requested) {
217         prepare_package(rail_handle, out_packet, sizeof(out_packet));
218         rail_status = RAIL_StartTx(rail_handle, CHANNEL, RAIL_TX_OPTIONS_DEFAULT, NULL);
219         if (rail_status != RAIL_STATUS_NO_ERROR) {
220           app_log_warning("RAIL_StartTx() result:%d ", rail_status);
221         }
222         tx_requested = false;
223       }
224       break;
```

Console | Memory | Executables

Writable | Smart Insert | 78 : 1 : 3076 | 547M of 732M

SimpleTRX – Process - RAIL

v5_workspace - simple_trx/app_process.c - Simplicity Studio™

File Edit Source Refactor Navigate Search Project Run Window Help

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.slcp
  - simple_trx.slps

simple_trx.slcp | *radio_settings.radioconf | main.c | app_process.c

```c
/*****************************************************************************
 * RAIL callback, called if a RAIL event occurs.
 *****************************************************************************/
void sl_rail_util_on_event(RAIL_Handle_t rail_handle, RAIL_Events_t events)
{
  error_code = events;
  // Handle Rx events
  if ( events & RAIL_EVENTS_RX_COMPLETION ) {
    if (events & RAIL_EVENT_RX_PACKET_RECEIVED) {
      // Keep the packet in the radio buffer, download it later at the state machine
      RAIL_HoldRxPacket(rail_handle);
      packet_recieved = true;
    } else {
      // Handle Rx error
      rx_error = true;
    }
  }
  // Handle Tx events
  if ( events & RAIL_EVENTS_TX_COMPLETION) {
    if (events & RAIL_EVENT_TX_PACKET_SENT) {
      packet_sent = true;
    } else {
      // Handle Tx error
      tx_error = true;
    }
  }

  // Perform all calibrations when needed
  if ( events & RAIL_EVENT_CAL_NEEDED ) {
    calibration_status = RAIL_Calibrate(rail_handle, NULL, RAIL_CAL_ALL_PENDING);
    if (calibration_status != RAIL_STATUS_NO_ERROR) {
      cal_error = true;
    }
  }
#if defined(SL_CATALOG_KERNEL_PRESENT)
  app_task_notify();
#endif
}

/*****************************************************************************
 * Button callback, called if any button is pressed or released.
 *****************************************************************************/
void sl_button_on_change(const sl_button_t *handle)
{
  if (sl_button_get_state(handle) == SL_SIMPLE_BUTTON_PRESSED) {
    tx_requested = true;
  }
#if defined(SL_CATALOG_KERNEL_PRESENT)
  app_task_notify();
#endif
}
```

Console | Memory | Executables

Writable | Smart Insert | 78 : 1 : 3076 | 644M of 732M

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

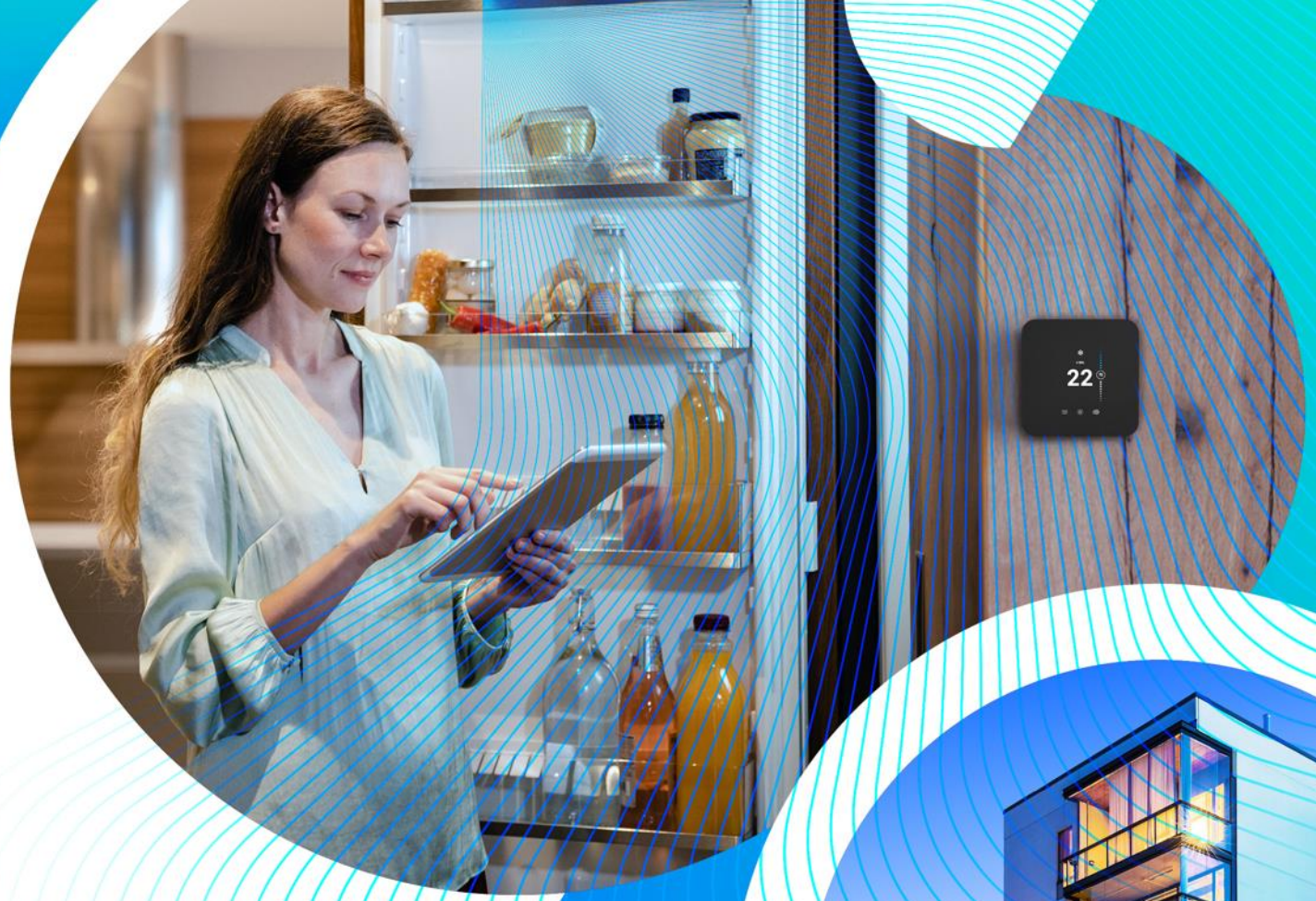
Resources

- 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



SILICON LABS