SILICON LABS

tech talks

WELCOME

Uncover Sub-GHz and Proprietary Solutions within Simplicity Studio v5

Chris League

Agenda

- High Level Overview
  - 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

Typical Proprietary Wireless Solutions

- Smart Meters

- Home Automation and Security

- Garage Door Openers

- Public Infrastructure

- Agriculture

- Asset Tracking & ESL

When is Proprietary Wireless Appropriate?

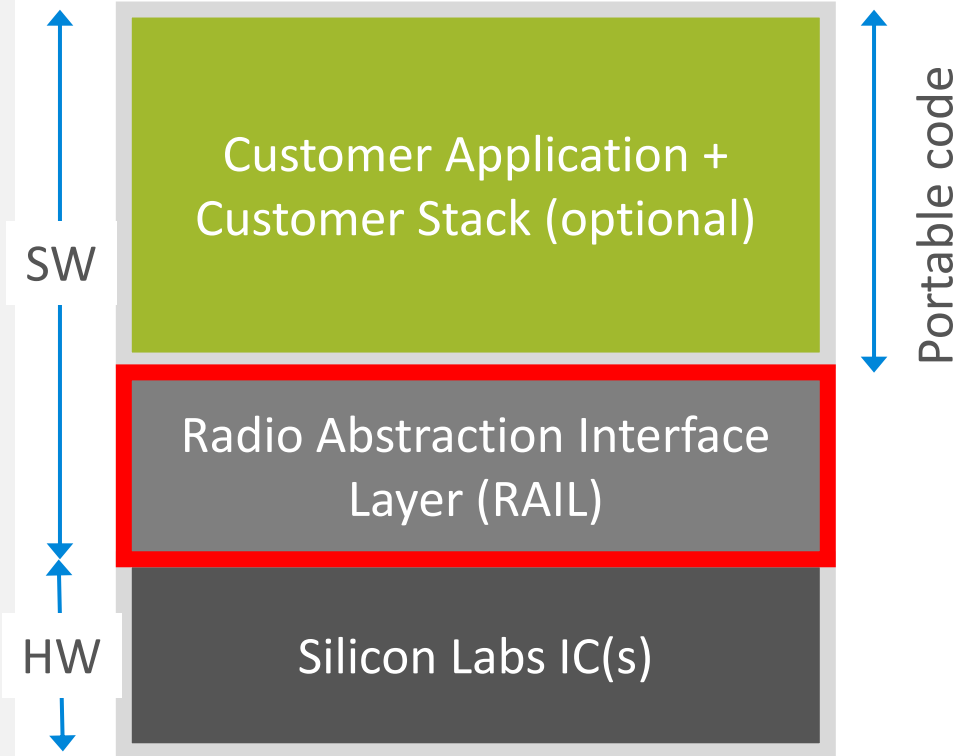
When the application demands:

+ 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:

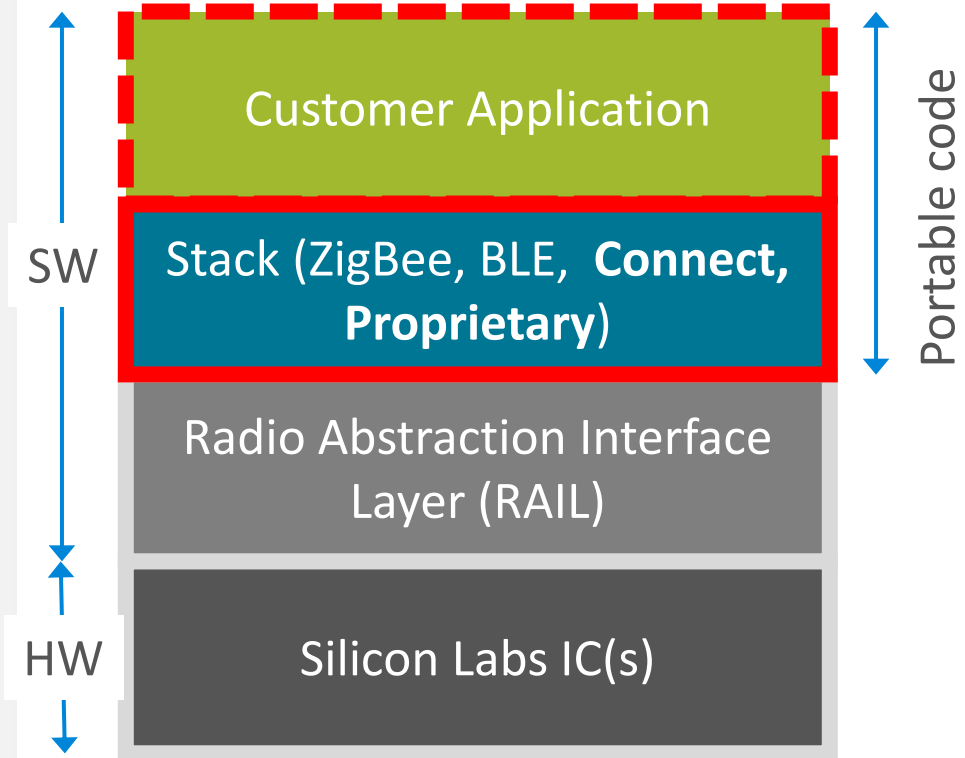
– More difficult development, longer “time to market”
– Incompatibility with existing/future non-proprietary infrastructures
– Security holes that can remain hidden for a long time due to the difficulty of the analysis
  – But once discovered, exploiting them is usually easy (high obfuscation, not necessarily high security)

EFR32 Development Path #1: 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

EFR32 Development Path #2: 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

Best Path Depends on the Application Needs

Connect

+ Full featured stack, including network layer
+ Task Scheduler
+ OTA bootloader
+ MAC provides 15.4 security

– 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

RAIL

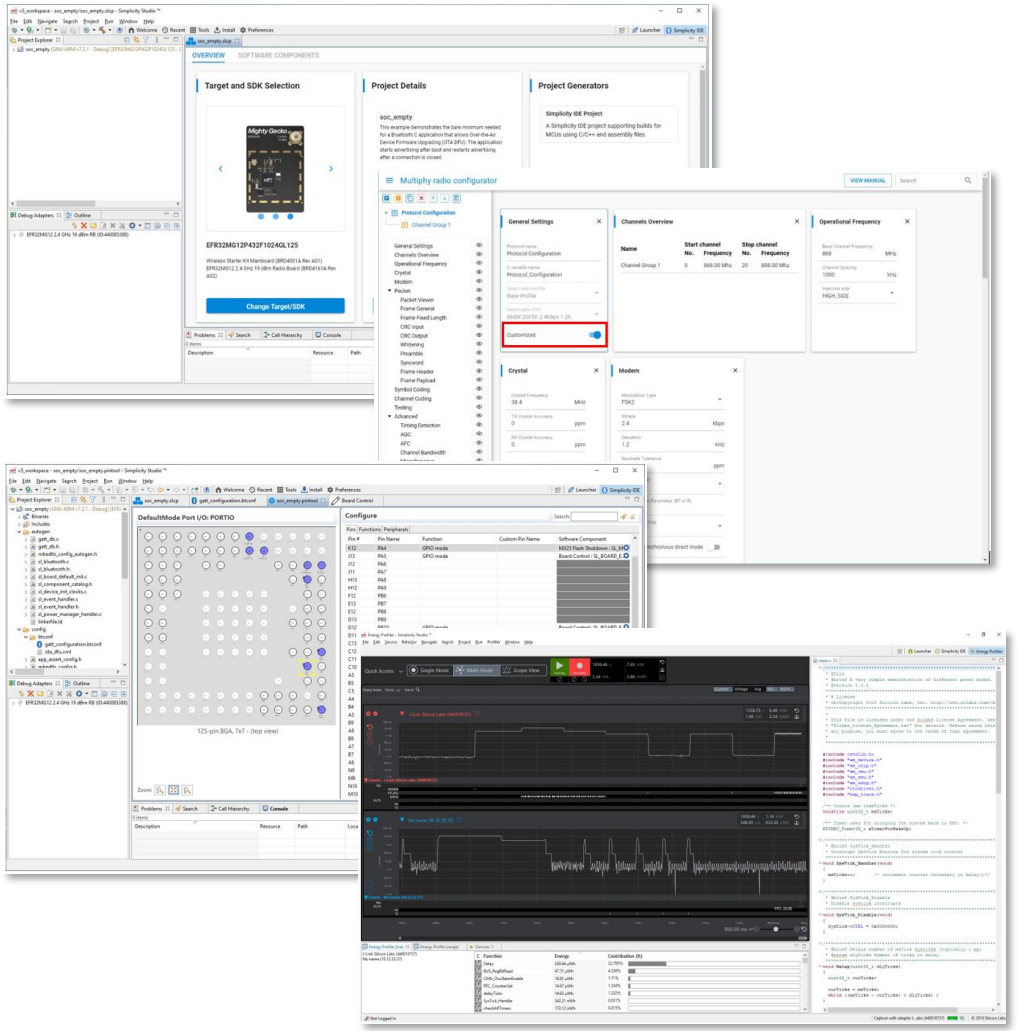
+ Very flexible
+ Support legacy proprietary systems
+ Accelerated MAC is usually enough for single hop networks
+ More efficient resource footprint

– No network layer, so no multi-hop support
– No application features like OTA
– Security must be done in application

Starting a RAIL Project in Studio5

Tools and API

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, but doesn't support CCA without retransmission
- 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

RAIL SimpleTRX walkthrough: Create Project in SSv5

Radio Configurator Flow: Demo

RAIL SimpleTRX walkthrough: Init

RAIL SimpleTRX init

- Initialized through components:
  - Clocks, DCDC, 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

RAIL SimpleTRX 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

Support Documentation

- 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

SILICON LABS

tech talks

THANK YOU