**RF Sense Protocols for Silicon Labs EFR32G14 Board**

**Hope Dargan (**[**hoped@mit.edu**](mailto:hoped@mit.edu)**)**

**July 2018**

**Introduction to RF sense:**

The EFR32G14 can detect radiofrequency (RF) energy above -20 dBm even in low power modes. When the EFR32G14 detects this energy for a certain amount of time, it will trigger a wakeup and two boards can then communicate with each other by sending signals. This allows for both energy conservation and the creation of a proximity sensor at a range of up to around 8 meters.

This document explains the specific protocols that have been created for the EFR32G14 with the intent that this code could be used for the tracking of animal social interactions based on proximity. It is not necessary to understand the code in order to run the system- simple instructions have been provided to run the code on the EFR32G14 at the end of this document. However, this document is also written to help future developers that wish to modify the code[[1]](#footnote-1).

**RF sense program overview:**

1. **Wake Mode:**

Wake mode is the high power mode that enables packet transmission and reception. Wake mode lasts for a period of 10 milliseconds by setting a timer.[[2]](#footnote-2) When the timer expires, the EFR32G14 board goes back to sleep.

1. **Sleep mode:**

Sleep mode is the low energy power mode that conserves energy. Sleep mode lasts on average for a period of 1 second by setting a timer.[[3]](#footnote-3) Sleep mode ends in one of two ways. Either the timer expires and sends the board into wake mode or the board detects RF energy for a period[[4]](#footnote-4) and sends the board into wake mode.

If a board did not communicate (receive another board’s ID packet and transmit its in return) during the wake cycle, it will sleep randomly for a period of sleep time plus or minus one or two times wake time (between .98 and 1.02) seconds. If a board successfully communicated the previous wake cycle, it will sleep for a period of 1 second. This prevents synchronization with other boards. If two boards are synchronized and wakeup at the exact same time they can never wake each other using RF energy in order to exchange ID packets.

Snooze mode is an optional feature of sleep mode, which can be enabled by setting SNOOZE\_ENABLED to 1[[5]](#footnote-5) within the main.c file. If snooze mode is enabled, the board tracks the number of times it was woken by RF sense without receiving any ID packets.[[6]](#footnote-6) If this happens sixty times in a row, the board is put to sleep for five minutes in snooze mode.[[7]](#footnote-7) While in snooze mode, the board can only be woken by the sleep timer running out and sensing RF energy will be disabled. This protocol is intended to save battery by preventing unnecessary RF Sense wakeups due to proximity to cell towers or the like. Snooze mode is disabled by default.

1. **PacketTx mode:**

This is the transmission mode where packets are sent. The radio is first put into an idle state and then transmits the desired packet. Each wakeup cycle starts with the transmission of a wakeup packet. All other packets sent during the wake cycle will be ID packets.

The wakeup packet is a long packet designed to trigger the RF sense detection of any other boards in low power mode in the vicinity and wake them up. The ID packets are short and contain an 8 byte header as well as an 8 byte ID number.

If a wakeup was triggered due to RF energy, the wakeup cycle will start with the transmission of a wakeup packet followed by an ID packet. The board that caused the wakeup will then be able to respond to the ID packet by sending its own ID packet. If wakeup was triggered due to the sleep timer, the board will wait to receive an ID packet. This protocol helps ensure communication. Because a board cannot both transmit and receive simultaneously, having an ID packet always follow a wakeup packet would present the possibility that both boards would be transmitting ID packets at the same time and neither board would receive them.

Note that when transmitting ID packets, the transmission energy is reduced to a minimum level to limit the possibility that an awake board outside of RFsense range would receive an ID packet and log that as a communication. Further testing is needed to determine the difference between RFsense range and packet reception range.

1. **PacketRx mode:**

In a simple sense, this mode idles the radio and then prepares to receive a packet during wake mode. This mode is entered automatically following the successful transmission of a packet.

When a packet is successfully received, the program confirms that the packet is an ID packet by checking its length and header.

RFsense Experiment protocol: If the received packet is identified as an ID packet, the program checks if the ID has already been received this wakeup session. If not, it stores the ID and time stamp in a temporary location before adding it to the log entries at the end of the wake cycle. This generates a log entry every wake cycle.

RFsensev7 protocol: RFsensev7 has a more complicated procedure when handling received ID packets. RFsensev7 tracks interactions- that is the packetArray tracks information about a single ID for a period of five minutes before storing the information into the packetLog. This greatly reduces the number of log entries and compacts the data.

1. **Testing Mode:**

Testing mode is an additional mode created specifically for use with Simplicity Studio and the Silicon Labs development kit board EFR32G14. Testing mode records various types of data and displays it to make experimentation easier. Testing mode can be enabled or disabled in the same way that Snooze mode can be enabled or disabled.

While testing, you may find it useful to enable buttons by changing the value of BUTTONS\_ENABLED. With buttons enabled, the board starts in wake mode with no timer enabled, allowing for testing of transmission by pressing the buttons on the board – button 0 (BP0) sends the board into transmit mode (as long as it is not currently sleeping) and button 1 (BP1) sends the board into sleep mode in order to start the sleep / wake up cycle.

Note that with the console enabled, the board will consume more energy. With testing mode disabled and no console, the board consumes an average of 280 micro-Amps of current per second. With testing mode enabled and displaying data to the console, the EFR32G14 consumes an average of 3.2 milli-Amps of current.

**Main To Dos:**

* Write packetLog Data to external flash. Using NVM3 protocol is suggested. See PDFs in Writing to Flash folder for more information about non-volatile data storage. For now, testing mode prints the data to the screen along with other statistics during testing.
* Make packetLog timestamp more accurate. Use RTC or RTCC driver. Using RAIL\_time is not practical for real world applications because it loops every hour or so. See AN0005: EFM32 Real Time Counters.
* Work with biologists to adjust contents of recorded data (what is written to the packetLog).

**Note About Simplicity Studios:** First, **all of the code I have written is contained in the main.c file**. The two most important parts of the project are the main.c file and the RFsense.isc file. The .isc file generates the radio configuration, the main file enables the protocols.

In order to copy the project, go to simplicity studios and copy and paste the desired project, then go into the .isc file and click generate so the radio configuration code is generated and then you can debug and test the project.

In order to run the project, just right click the project folder and then select Debug As… This will load the project onto the board of your choice. You can disconnect the board from the debugger once the code is loaded and use other applications like the energy profiler but you cannot use both at the same time.

In order to see testing mode data: You can see the data when the board is running by going to debug adapters (clearly visible on the left hand side from launcher perspective), right clicking one of the boards, selecting launch console and then hitting enter in the serial 1 display. If working the data should appear one line at a time. If the time stamp seems wrong (i.e. not incrementing by seconds) re-run the project in debug mode and try again.

A note about Energy profile – sometimes using the console affects the Energy profile and it will show a blank screen. Steps to prevent this: Enter Energy profile mode before launching the console. If the console has been launched first, starting energy capture will sometimes just show a blank screen for the board that has its console open. The way to fix this is by closing simplicity studios, unplugging the boards from the computer, plugging them in again and then opening Simplicity studios. The boards will still have the same applications uploaded even if they lose power. Then go back to energy profiler first and start energy capture before launching the console again.

**Useful documentation resources for Silicon Studios:** <file:///C:/SiliconLabs/SimplicityStudio/v4/developer/sdks/gecko_sdk_suite/v2.3/protocol/flex/documentation/index.html>

Most useful links include: [**Getting Started with the Silicon Labs Flex SDK for the Wireless Gecko (EFR32) Portfolio**](file:///C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\v2.3\protocol\flex\documentation\qsg138-flex-efr32.pdf)**,** [**EFR32 Radio Configurator Guide**](file:///C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\v2.3\protocol\flex\documentation\an971-efr32-radio-configurator-guide.pdf) **-- AN971,** [**Using RAIL Test**](file:///C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\v2.3\protocol\flex\documentation\RailTestUserGuide.html) **(describes RF sense functionality),** [**Silicon Labs RAIL API Reference Guide**](file:///C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\v2.3\protocol\flex\documentation\API_RAIL_HTML\index.html)

1. It is recommended to use Simplicity Studio for developing as it has various helpful tools like the Energy Profiler and the easy to use Radio Configuration GUI. The code for this project can be found in the main.c file and Radio Configuration can be modified in RFsense Radio Config.isc. [↑](#footnote-ref-1)
2. In the code, WAKE\_TIME\_MS value defines how long wake mode lasts. [↑](#footnote-ref-2)
3. In the code, SLEEP\_TIME\_MS value defines how long average sleep mode lasts. [↑](#footnote-ref-3)
4. In the code, RFSENSE\_TIME defines this value in micro seconds. [↑](#footnote-ref-4)
5. Setting SNOOZE\_ENABLED to 0 will disable snooze mode [↑](#footnote-ref-5)
6. Using rfWakeCount variable [↑](#footnote-ref-6)
7. In the code, the variable wakeLimit and SNOOZE\_TIME\_MS controls these values. [↑](#footnote-ref-7)