System for Laboratory

and

Wildlife Mouse Tracking

A Design Project Report

Presented to the School of Electrical and Computer Engineering of Cornell University

In Partial Fulfillment of the Requirements for the Degree of

Master of Engineering, Electrical and Computer engineering

Submitted by

Nathan Shen

MEng Field Advisor: Joe Skovira

MEng Outside Advisor: Michael Sheehan

Degree Date: January 2018

Abstract

Master of Engineering Program

School of Electrical and Computer Engineering

Cornell University

Design Project Report

Project Title: System for Laboratory and Wildlife Mouse Tracking

Author: Nathan Shen

Abstract: A major challenge for wildlife radio tracking is in obtaining high-accuracy position data of multiple individuals in a population. Systems currently exist that use multiple antennas to create an estimate of location based on the received strength of a signal from three or more receiver towers. However, these systems require bulky and expensive receiver modules, resulting in imprecise location data. Our goal is to create a small (350mg), energy efficient radio tag capable of communicating with the receiver network. For this project, we propose an Angle of Arrival (AOA) based automated radio telemetry system. The basic components of the direction finding (DF) system will use 2 dipole antennas to receive radio frequency (RF) packets from radio tags transmitting at a carrier frequency of ~150 Mhz. The other aspects of this DF system include a low noise amplifier, mixer, and local oscillator (LO) to boost the received signal and mix it with the LO, an RF demodulator to enable the estimation of differences in phase between the received signals, and ADC ICs; they are all typical components of this kind of system. Our system will use the CC1310 wireless-microcontroller (MCU) attached to Raspberry Pi's to collect the RF data and run the triangulation algorithm. We present a low cost, weight, and power system that can operate in short range (100-300m), cluttered environments with high spatial accuracy (~5m) triangulation results for approximately 50 transmitters.

Executive Summary:

The System for Laboratory and Wildlife Mouse Tracking is concerned with developing a system that can track multiple small organisms in simulated and natural habitats. There are many methods currently utilized by researchers today. However, they are either highly inefficient because they animals need to be manually tracked or the tags are too big for the animal.

The aim of this project is to create a low-cost system machine-based tracking system using a microcontroller and Raspberry Pi that can analyze groups of interacting animals. With the advent of low cost active RF and microcontroller (MCU) current consumption, in addition to flexible low-power modes, we can design our own direction finding system. This requires using a radio finding (RF) architecture that can operate in short range (100-300m), cluttered environments with high spatial accuracy (~5m) triangulation results for approximately 50 transmitters.

We have proposed an Angle-of-Arrival (AOA) RF solution for a low-cost system that is currently in its first stage of development. Accomplishments made during this project include creating a communication protocol between multiple receivers and transmitters while utilizing power saving techniques. This project is planned to be completed by December 2018. This document covers all the design choices and decision made during the development.

Table of Content:

1. First Semester: Wildlife Mouse Tracking 5
    1.1 Introduction 5
    1.2 Problem to be solved 5
    1.3 Design Decisions 5
    1.4 Implementation 6
    1.5 Results 7
2. Second Semester: Localization 8
    2.1 Introduction 8
    2.2 Initial Radio Finding System Design 9
    2.3 Current Radio Finding System Design 10
    2.4 Mobile-node to ground-node Protocol 11
    2.5 Implementation 11
    2.6 Setting up the Code 12
    2.7 CC1310 Launchpad 13
3. Conclusion 13
4. Future Work 14
5. References 14
6. Appendix 15
    Code Block 1: BaseStationLogger.py 15
    Code Block 2: DataServer.py 17
    Code Block 3: DataClient.py 18
    Code Block 4: rfPacketRx.c 20
    Code Block 6: rfWakeOnRadioTx.c 26
    Code Block 7: rfWakeOnRadioRx.c 31

1. First Semester: Wildlife Mouse Tracking

1.1 Introduction

The study of animal social behavior in laboratory settings has tremendous impact on a wide range of research fields. Laboratory animals such as mice, display a wide range of social behaviors that can be quantitatively measured with laboratory techniques. Manual tracking of these behaviors tends to require a large time investment in terms of both training and analysis, and is prone to error. To circumvent these human constraints, there is a need for machine-based tracking that is applicable toward the analysis of socially interacting groups of animals. Automatic tracking of interacting animals could thus overcome the limitations of manual tracking methods. There is a proposed method that simultaneously and continuously tracks the identity and spatial position of ≥ 10 mice, without the need for visual tagging, by integrating video and RFID tracking data sets, which are time-synchronized and then fused by means of machine-based algorithm [1].

1.2 Problem to be solved

We wish to adapt the existing TABER system [6] to recreate the proposed system in article [1] for the requirements for a laboratory based mouse habitat and eventually a wildlife mouse habitat. The base station software that currently works with multiple tags using a single antenna will be adapted to read multiple tags from multiple antennas and process movement of individuals throughout the habitat. The design requires:

- Adapting methods for tracking with RFID tags to use the TABER radio tags.
- Expanding the system using multiple base stations to allow the use of multiple RF receivers.
- Develop a coordinated solution to record tag data for multiple individuals from many receivers and base stations.
- Process radio tag location data from multiple individuals.
- Characterize the limits of tracking:
    - Number of simultaneous individuals
    - Limits of accuracy
    - Spatial and temporal precision
- Display tag tracking data in real time on a map of the habitat
- System must be developed with extensibility in mind:
    - Can more individuals be added?
- Can the habitat footprint scale?

1.3 Design Decisions

The first decision of the project is to decide between using Radio or RFID tags. After advising from Joe Skovira, we presented a comparison to Michael Sheehan between Radio vs. RFID tags. We decided to pursue Radio tags for the following reasons:

1. Radio tags have further receiver-transmitter range (up to 1 mile) compared to a range of 13-inches for RFID. This is important for future project development for the wildlife mouse habitat. By having a farther detection range, we will be able to track the mice's movements in a bigger area.

2. Radio tags can be easily adapted in the future to also record biometrics such as body temperature, heartbeat, etc. We can do this because we added an onboard SI1060 processor to the radio tags.
3. Radio tags are already developed by TABER for outdoor use.

We will test our TABER mouse system in an 8x8ft indoor laboratory based mouse habitat. As a design decision, we decided on a 3x3 grid array which means a total of 9 USB dongles will be used for each grid space. Each Raspberry Pi 3 has 4 USB ports; therefore, we will assign 3 USB dongles to each of the 3 Raspberry Pi's that we will be using.

1.4 Implementation

The first step is to get the baseline code for the transmission and receiver working from the previous M. Eng. project. With the help of Rich Gabrielson, I received a working RaspberryPi 2 Model B with a working Kernel of the project. This kernel would boot up with a fully automatic setup. After it displays the IP, the tag recording program is automatically started. It is a blocking call – so it will not allow other functions to be executed. After working with Rich to find a workaround, we found a solution: we need to connect the Raspberry Pi via Ethernet to another computer and SSH into the Pi after reconfiguring our IP address to 169.254.113.50 and Netmask to 255.255.0.0. Now that we have access into the files and programs, we can begin adjusting the code to our needs.

The next step is to port the software from Raspberry Pi 2 Model B to the Raspberry Pi 3. The BCM2837 on the Raspberry Pi 3 has 2 UARTs (as did Raspberry Pi 2). However, to support the Bluetooth functionality, the PL011 UART was moved from the header pins to the Bluetooth chip and the mini UART made available on header pins 8 & 10. As a result, /dev/ttyAMA0 previously used to access the UART now connects to Bluetooth. The solution I used was to replace /dev/ttyAMA0 with /dev/S0 to use the same software on the Pi3 and earlier models. Another small wrinkle is that the GPIO serial port is disabled by default. We need to enable it by editing the config.txt

sudo nano /boot/config.txt

and add the line at the bottom:

enable_uart=1

Now that we have the baseline code for the transmission and receiver working on the Raspberry Pi 3, the next step is to configure our Raspberry Pi's to communicate with each other. This way, the three different Raspberry Pi's can send data to a "master" Raspberry Pi that will log and keep track of our 3x3 array of mouse tag data. The communication network that we decided to implement is a TCP socket server and a corresponding TCP socket client. There are plenty of examples and tutorials online on how to do this in detail. I followed the one from [7]. Basically, we import the socket library and create a socket object. On the server side, we bind the servers own IP address and any arbitrary unused port to the socket, then we start listening for incoming requests. On the client side, we first create a corresponding socket, but then connect it to the listening remote servers IP address and port. We then monitor our GPIO in a loop and send the data using sendall. When we are finished, we can close the connection. By the end of the first semester, we have 2 client Raspberry Pi sending collected radio tag data over to a single server Raspberry Pi. Figure 1 illustrates our design and solution.

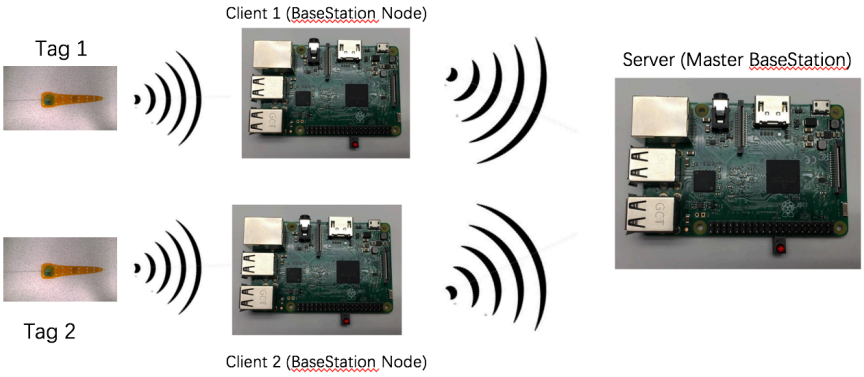

Figure 1: 2 Client Raspberry Pi transmitting tag data to a Server Raspberry Pi

1.5 Results

```
================================================================================
Basestation start up on Apr 20 2017 at 04:27
================================================================================
2A524B2A,04:27:25,2017 Apr 20
2A524B2A,04:27:31,2017 Apr 20
2A524B2A,04:27:33,2017 Apr 20
2A524B2A,04:27:37,2017 Apr 20
2A524B2A,04:27:39,2017 Apr 20
2A524B2A,04:27:43,2017 Apr 20
2A524B2A,04:27:46,2017 Apr 20
2A524B2A,04:27:48,2017 Apr 20
3FFFFE00,04:27:50,2017 Apr 20
================================================================================
Basestation start up on Apr 20 2017 at 05:16
================================================================================
2A524B2A,05:16:34,2017 Apr 20
================================================================================
Basestation start up on Apr 21 2017 at 14:05
================================================================================
================================================================================
Basestation start up on Apr 21 2017 at 14:18
================================================================================
================================================================================
Basestation start up on Apr 21 2017 at 14:31
================================================================================
2A524B2A,14:31:06,2017 Apr 21
================================================================================
Basestation start up on Apr 21 2017 at 14:32
================================================================================
2A524B2A,14:33:02,2017 Apr 21
2A524B2A,14:33:12,2017 Apr 21
2A524B2A,14:33:25,2017 Apr 21
```

Figure 2: Received transmitter tags stored in tags.txt

Figure 3: BaseStation Setup with Raspberry Pi

2. Second Semester: Localization

2.1 Introduction

During the summer, a group of undergraduate students helped advance my M. Eng. project. While they were testing the system within the lab enclosure, they realized that the TABER wildlife tags have a radio frequency reception range as large as a few kilometers; for the mouse system, the reception distance must be limited to within a few feet. To solve this problem, they modified the antennas by building small loop antennas with radii less than one tenth of the emitted wavelength to reduce the reception distance between the tags and the base station. However, the small loop antenna design on the tag was proven to be noneffective.

At the same time, the TABER group decided to switch from the SI1060 to TI's CC1310. The Mouse Habitat also decided to switch after being persuaded by the TABER group and Julian Kapoor, a Post-Doctorate research of animal behavior. Because of this switch, we needed to resign the radio tags that I had been working with in the previous semester. This meant that I would no longer working on the original scope of the project outlined earlier. With the approval of my M. Eng. field advisor Joe Skovira and M. Eng. Outside Advisor Michael Sheehan, I joined the Localization team lead by Julian.

The localization team consists of a total of 5 people: 2 undergraduate ECE students (Mei Yang, Russell Silva), a ECE M.Eng. Student (Peidong Qi), the team lead (Julian Kapoor), and me. We were advised by Professor Joe Skovira and Edwin Kan. Mei was tasked with developing the testing interface for the system, Russell was tasked with the RF hardware design, Peidong was tasked with the CC1310 and Raspberry Pi communication interface, and I was tasked with the mobile-node (CC1310) to ground-node (CC1310) communication protocol. An overview of the system is shown below in Figure 4.

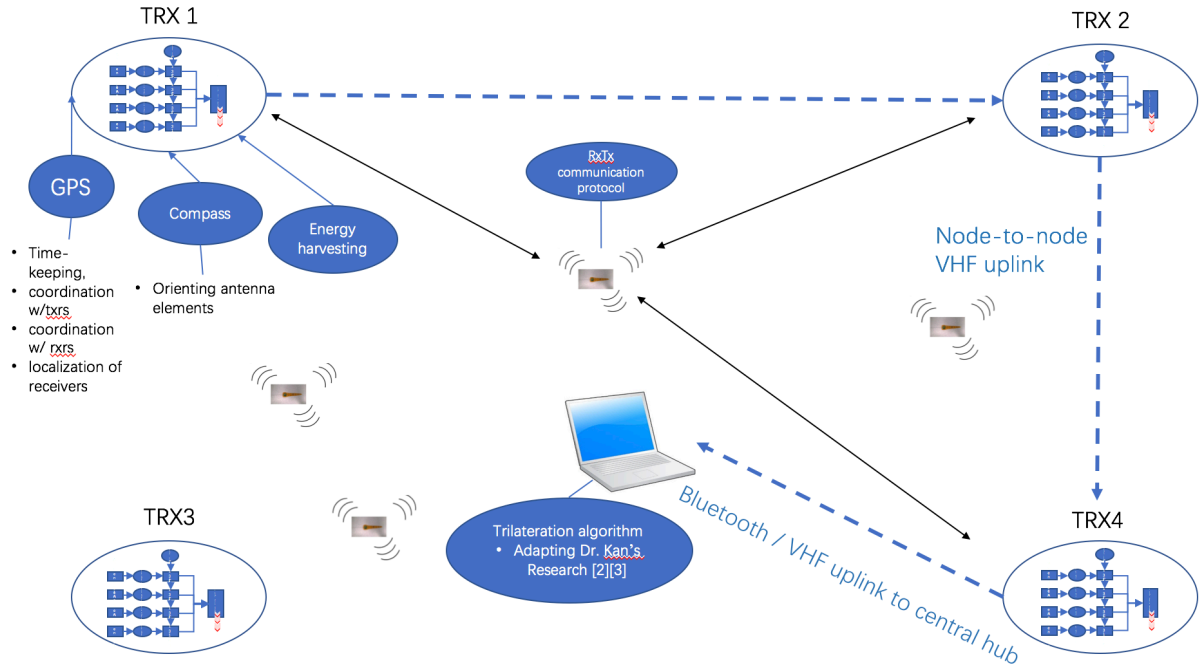

Figure 4: Julian Kapoor's proposed automated phase based telemetry receiver system [8]

2.2 Initial Radio Finding System Design

In our first Localization team meeting, we brainstormed on what kind of direction finding system to create. Some ideas that were thrown around include Doppler, Time Distance of Arrival (TDOA), Watson-Watt, and Angle of Arrival (AOA). After much consideration, the team decided to proceed with phase interferometry to estimate the Angle of Arrival (AOA) of radio signals. The advantages of an AOA system are as follows:

1. No need for time, phase, or frequency synchronization between measuring units (Watson-Watt Pair).
2. The simplicity of the antenna system (only need two measuring units for 2D),
3. Electronic switching can be performed between measuring units.
4. Bearing calculations can be done in the background (not time sensitive)

The main drawback of an AOA system is the susceptibility to multipath interference. Section 2.3 provides a proposed architecture to mitigate this problem.

Our first proposed AOA system architecture is shown below in Figure 5. We were not sure if the CC1310 obscured away in-phase and quadrature data, therefore we were relatively conservative and came up with a system architecture that was known to work before with the previous tags (SI1060) that TABER had worked with. As shown in the figure, the CC1310 would only be used to trigger the ADC along with the packet information, but would provide no direction-finding solutions; all the radio direction finding would be done in hardware.

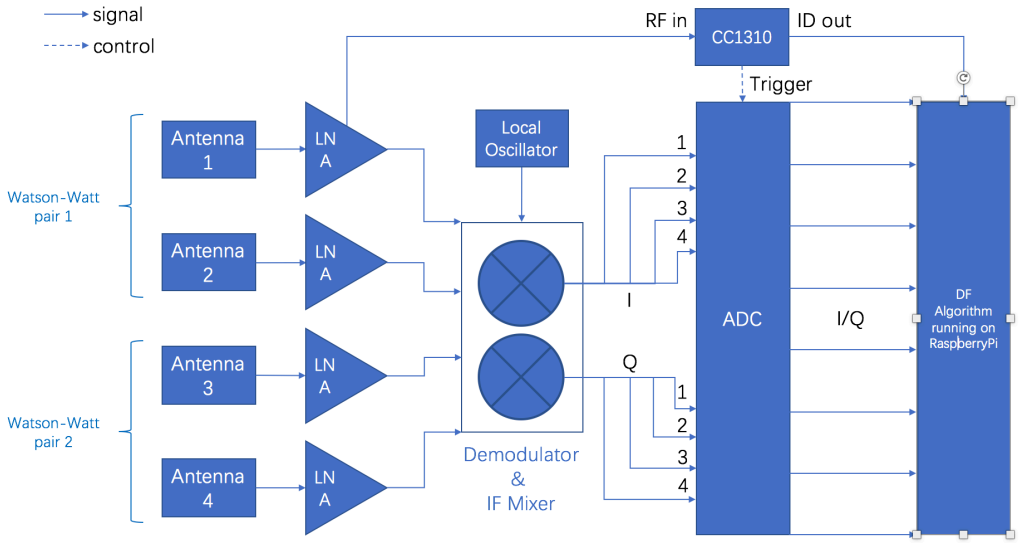

Figure 5: Julian Kapoor's proposed Angle of Arrival (AOA) System Architecture [8]

Requirements and constraints outlined by Julian Kapoor:

1. Short range (100-300 m between receivers)
2. Extremely simple transmitter design (lightweight, low power)
3. System can operate in cluttered environments (multipath interference)
4. System can operate with ~50 Txers
5. High spatial accuracy (~5 m) triangulation results
6. Low cost receivers (COTS components)
7. Low power consumption of receivers

2.3 Current Radio Finding System Design

One of the big challenges with a new project is understanding and designing a new system and figuring out the constraints. We began our approach with the help of Dr. Kan's existing indoor localization [2][3]. Julian Kapoor outlined many terms and concepts in the paper that we needed to understand such as triangulation vs. trilateration, phase cycle integer ambiguity, coherent detection, digital beamforming, heuristic optimized sparse multi-frequency ranging and why it can help resolve phase cycle integer ambiguity. After much discussion and debate, we decided to proceed with the following architecture for our radio finding system:

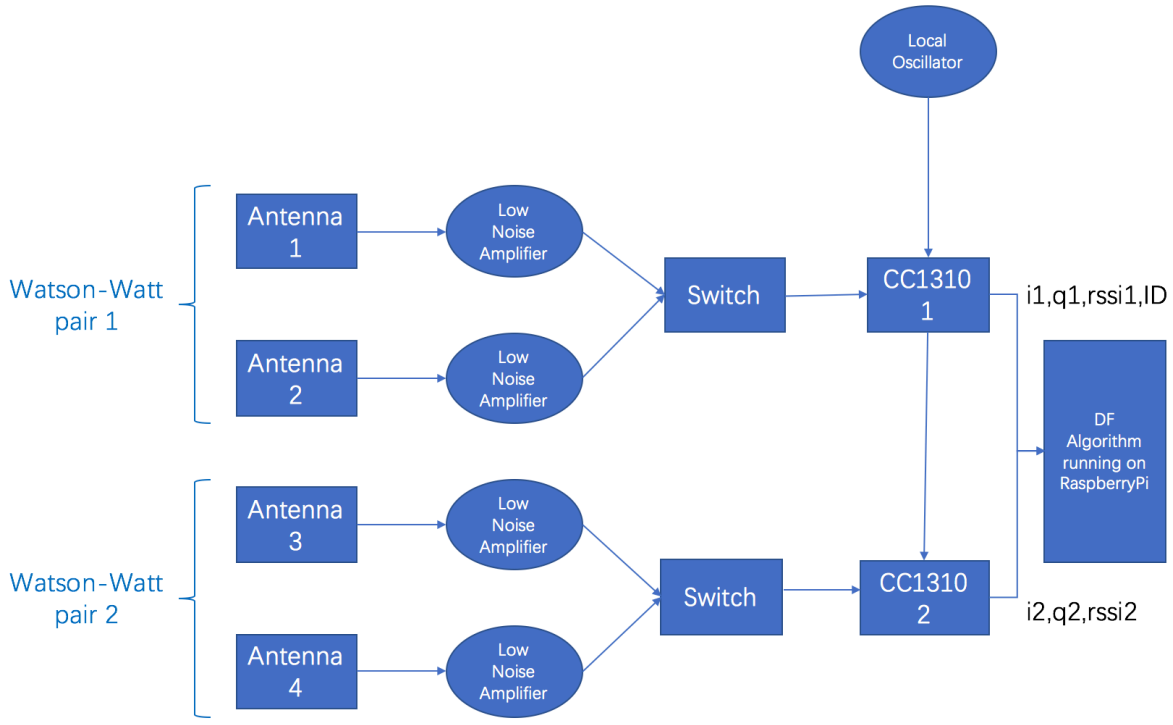

Figure 6: Localization team's proposed RF system architecture [8]

We employed techniques found in [2][3] to come up with our system. Because we can extract I and Q data from the CC1310, it simplifies our design by not needing to include a Low-Noise Amplifier, ADC, Demodulator, or IF Mixer that was in Figure 5. We are taking an RF switch-based approach to receiving phase-difference information on two antennas with one CC1310. By using this approach, we can split the signal into two in-phase signals, feed them into the CC1310, and then create code to read the I and Q data into two buffers (one for each channel), and calculate phase differences. Another concept that we will include from Dr. Kan's paper is heuristic multi-frequency continuous wave (HMFCW) ranging method. By using this method, we can reduce multi-path induced phase error which was a big concern with using an AOA system.

Included in Figure 6 are Watson-Watt pairs proposed by Julian Kapoor. Watson-Watt is a technique to obtain information by using two Adcock antenna pairs to calculate an amplitude difference. The advantage of using this configuration is that it doesn't pick up any horizontal interference and allows for a fast calculation time to obtain the bearings.

2.4 Mobile-node to ground-node Protocol

The tags will be solar powered by using energy-harvesting solutions developed by TABER [6]; therefore, we will have to account for energy consumption. While in active mode, MCU's such as the CC1310 that we are using, consume a lot of power. As a result, it will be important that the tags transmit only when the receiver is in range. In order to do so, we will need to take advantage of sleep mode but we also need to make sure that we do not miss a tag by oversleeping.

Another concern is transmitting and receiving signals so that multiple tags do not interfere with each other. To solve this problem, we developed a similar protocol to time-division multiplexing (TDM) which is a method of transmitting and receiving independent signals by means of synchronized switches at each end of the transmission line so that each signal appears on the line only a fraction of time in an alternating pattern.

Figure 8 illustrates the specifications given to us by Julian Kapoor and is as follows:

1. Mobile nodes should sleep just under 5s.
2. Wake long enough to receive a single countdown signal from a ground node.
3. Once they have received a countdown signal, and the checksum demonstrates a good link, update global time.
4. Listen for rest of commands.
5. Once the commands are received: use global time, received command, and pre-programmed transmission time delay to send ID during the ground node's ~5 min RX period.

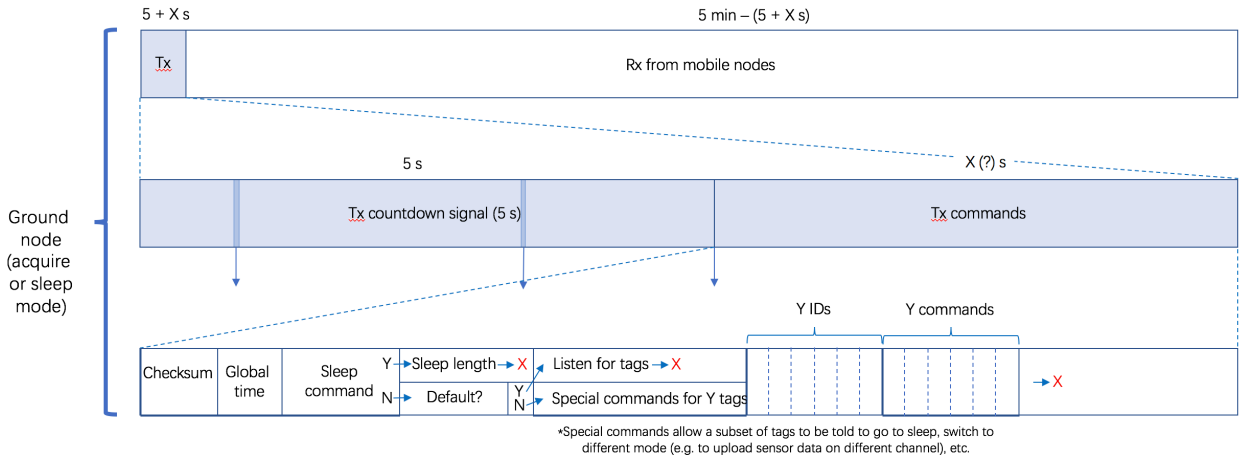

Figure 8: Mobile-Node to Ground-Node Protocol [8]

2.5 Implementation

The following code found in the appendix is adapted from CodeComposerStudio's Resource Explorer [4] to fit the design of our system. The codes were used with minor parameter changes to investigate the capabilities of the devices and are included here to emphasize what sample elements were used for this initial testing.

To tackle the first and second specifications outlined in section 2.4.3, I found example code in TI Code Composer Studio (CCS). The folders were rfWakeOnRadioRx and rfWakeonRadioTx. Run the example on one of the boards, this will be the RX board. Board_LED0 will blink on this board for every wakeup. Start the rfWakeOnRadioTx companion example on another board (TX board) and press Board_BUTTON0 on that board to send a packet. Board_LED1 on the RX board should now toggle for every button press. These examples use the sleep and wakeup functionality of the CC1310 by scheduling automatic wake-ups in the future. The default code schedules a wakeup every 0.5 seconds. To change the wakeup time to meet the specification of 5s, we want to change line 58 in both rfWakeOnRadioRx.c and rfWakeOnRadioTx.c to 0.2. This makes it so that the transceiver will wake up once every 5 seconds.

The TI documentation outlines a typical radio physical layer packet format [4].

Preamble	Sync Word	Length Byte	Payload	CRC
4 byte	4 byte	1 byte	X bytes	2 bytes

From research, the preamble is almost always 101010... with a length that can change (we will use 4 bytes until we have a reason to change the length). The idea is that the receiver can "hear" this, and set the gain and other things correctly. However, this is not technically not needed for bit synchronization. The sync word is given to us by TI and unless there is a very compelling reason, we should not change it since TI has tested the sync word to reduce the likelihood of false positives. In the default code, the payload length is 30. This can be set in rfWakeOnRadioTx on line 61. To change the payload, this can be done in rfWakeOnRadioTx on lines 198-206. To receive the payload can be found in rfWakeOnRadioRX lines 381-387. Currently, I have not added in any payload commands or information. The code section that takes care and makes sure that the receiver wakes up long enough is found in line 330-348.

TO DO:
1. Add in payload information (lines 198-206 in rfWakeOnRadioTx and lines 381-387 in rfWakeOnRadioRX).
2. Add in another state in rfWakeOnRadioRx that sleeps for a pre-programmed transmission time delay (Specification 6). Currently the receivers will always sleep for 5s. The code that controls the sleep time is: RF_cmdPropRxSniff.startTime += WOR_WAKE_UP_INTERVAL_RAT_TICKS(WOR_WAKEUPS_PER_SECOND);
3. Sync and update Global Time.

While working on the previous specifications, I stumbled upon rfListenBeforeTalk, which is also found in TI CCS. This project outlines how to check if the channel is busy before transmission. I was thinking on incorporating this code with rfWakeOnRadioTx/Rx as a fail-safe, in case our countdown signal is incorrect or too long; it will not interfere with another signal.

2.6 Setting up the Code

1. Download TI Code Composer Studio (CCS) Integrated Development Environment (IDE) http://www.ti.com/tool/CCSTUDIO
2. Download TI Real Time Operating System (RTOS)

http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/tirtos/index.html

3. Open CCS
4. View -> Resource Explorer -> Software -> SimpleLink CC13x0 SDK -> Examples -> Development Tools -> CC1310 LaunchPad -> TI Drivers
5. Open the project you want to run (i.e. rFWakeOnRadioRx, rfWakeOnRadioTx)
6. For this example, we'll open rFPacketRX -> TI-RTOS -> CCS Compiler
7. Select rfPacketRx. Near the top right, select the "Import to IDE" button.
8. The rFPacketRX project will be imported to the right under Project Explorer. Select the project so that it is [Active].
9. Connect the CC1310 Launchpad to your computer and select Run -> Debug
10. The code should now be running on a CC1310.
11. To program rFPacketTX, repeat steps 4-9.

2.7 CC1310 Launchpad

To aide our development, we were given 2 CC1310 Launchpads. This gives us access to the integrated development environment (IDE) for development and rapid prototyping. A benefit of this launchpad is the availability of Sub-1GHz radio for wireless applications and an integrated PCB trace antenna.

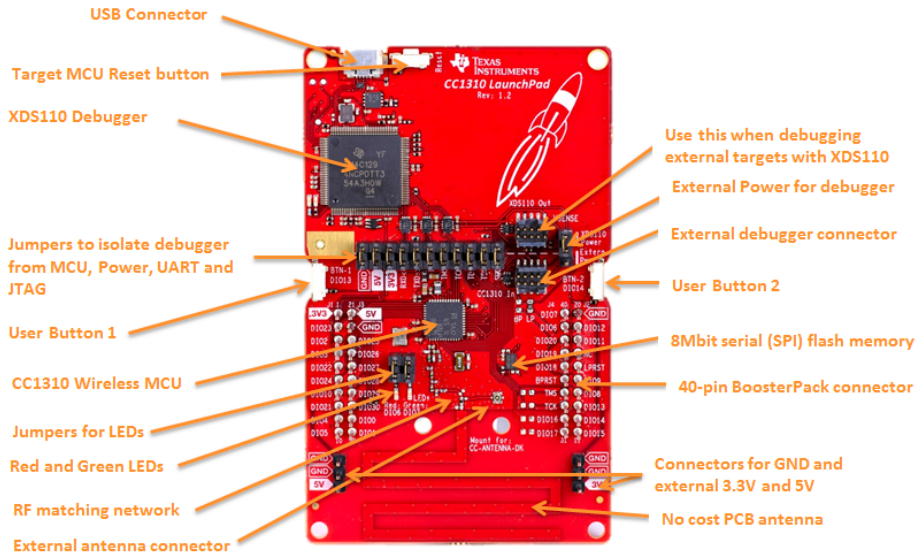

Figure 9:CC1310 Hardware Description [5]

3. Conclusion

My M.Eng. project can be divided into two parts. I first began with the goal of adapting the existing TABER system [6] for the requirements for a laboratory based mouse habitat. I made progress towards this goal and by the end of the first semester. Achievements including porting software from the Raspberry Pi 2B to the Raspberry Pi3 and having 2 client Raspberry Pi sending tag data over to a single server Raspberry Pi. During the summer, a couple of undergraduates advanced my project but ran into a problem with reception distance; the base stations were recording tag data from too large of a reception range. In parallel to the

development during the summer, the TABER group decided to switch from the SI1060 to the CC1310.

The second part of my M.Eng. project started when I came back for the second semester. I transitioned onto the Localization team lead by Julian Kapoor. The reason for this transition is due to the lack of future support of the system by TABER as well as the difficulties of our previous system that was discovered during the summer. This team's goal is to create a telemetry system that can obtain high-accuracy position data of multiple individuals in a population. I was tasked with the mobile-node to ground-node communication protocol. Achievements include adapting the baseline rfWakeOnRadio and rfListenBeforeTalk code for our system as well as working with the team to draft up preliminary system design for our radio finding hardware.

4. Future Work

The localization team expects this project to be completed in one to two years and has laid out four milestones. For the first milestone, we want to be able to calculate an Angle of Arrival (AOA) from a received signal. The second milestone involves beamforming to calculate a direction from the signal. Milestone 3 is where we calculate the range of the signal. Lastly, milestone 4, consists of completing any further refinements to our system. This project is still in stage one of development that is scheduled to be completed by the end of Spring 2018.

5. References

[1] Weissbrod, Aharon et al. "Automated Long-Term Tracking And Social Behavioural Phenotyping Of Animal Colonies Within A Semi-Natural Environment". *Nature Communications* 4 (2013): n. pag. Web. 18 May 2017.

[2] Y. Ma, X. Hui and E. C. Kan, "3D real-time indoor localization via nonlinear backscatter in passive devices with centimeter precision," ACM MobiCom, New York City, NY, Oct. 2016.

[3] Y. Ma and E. C. Kan, "Passive ranging by low-directivity antennas with quality estimate", IEEE MTT 2015 International Microwave Symp. (IMS), Pheonix, AZ, May 18 – 22, 2015.

[4] CC13x0 Proprietary RF User's Guide:
http://dev.ti.com/tirex/content/simplelink_cc13x0_sdk_1_30_00_06/docs/proprietary-rf/html/cc13x0/index.html

[5] CC1310 Description and Features
http://www.ti.com/tool/LAUNCHXL-CC1310

[6] Taber Work:
https://courses.cit.cornell.edu/ece6930/ECE6930_Spring16_Final_MEng_Reports/Bird_Tag_Team/Bird_Tags.pdf

[7] RPi TCP/IP Communication Tutorial

http://www.python-exemplary.com/index_en.php?inhalt_links=navigation_en.inc.php&inhalt_mitte=raspi/en/communication.inc.php

[8] Localization Github
https://github.com/jakapoor/AMRUPT

6. Appendix

Code Block 1: BaseStationLogger.py

```
#!/usr/bin/env python3

#================================================================================
# BaseStationLogger.py
#================================================================================
#
# Code devloped by:
#   Taylor Pritchard (tjp79@cornell.edu)
#
# Code Modified by:
#   Gautham Ponnu (gapo)
#   Nathan  Shen  (nds64)
#--------------------------------------------------------------------------------

import re
import sys
import datetime
import os

from time import sleep, strftime

from RPi import GPIO

from serial            import *
from serial.serialutil import SerialException

TAG_FILE = '/dev/ttyS0'
TAG_BAUD = 115200
SYS_LOG = '/home/pi/tag.txt'

class BaseStationLogger():

    #--------------------------------------------------------------------------------
    # Initialization
    #--------------------------------------------------------------------------------

    def __init__( self ):

        GPIO.setmode( GPIO.BCM )
        GPIO.setup( 24, GPIO.OUT )

        GPIO.output( 24, 0 )
        GPIO.setwarnings(False)

        self.time = 0

        self.tag_reader = Serial(TAG_FILE,TAG_BAUD,timeout=1)

        self.write_log(
            '='*80 + '\n' + \
```

```
            ' Basestation start up on ' + strftime('%b %d %Y') + \
            ' at ' + strftime('%H:%M') + '\n' + \
            '='*80 + '\n'
        )
    #---------------------------------------------------------------------------
    # Logging
    #---------------------------------------------------------------------------

    def write_log(self, msg ):
            # self.lock.acquire()
            if os.path.exists("/home/pi/tag.txt") == True:
                # then the file exist and everything is alright
                log_file = open(SYS_LOG, 'a+')
            else:
                # then file does not exist
                log_file = open(SYS_LOG, 'w+')
            log_file.write( msg )
            log_file.close()
            # self.lock.release()

    def write_empty(self):
            # Empty file writer
            # this function should check whether a file is present, ...
            #... if not just create a file
            if os.path.exists("/home/pi/tag.txt") == False:
                log_file = open(SYS_LOG, 'w+')
                log_file.write( 'No tag found' )
                log_fole.close()

    #---------------------------------------------------------------------------
    # Main Class Loop
    #---------------------------------------------------------------------------

    def run ( self ):

            last_tag = self.tag_reader.readline().rstrip().decode('utf-8','strict')
            date_str = strftime('%Y %b %d')
            time_str = strftime('%H:%M:%S')

            if ( last_tag != ""  ):
                print (last_tag)
                self.blink()
                self.write_log( last_tag + ',' + time_str + ',' + date_str + '\n' )
            else:
                # Here create a flag and then write an empty msg for the first time
alone
                # maybe do a file check - that can be flag and then can write based
on that
                self.write_empty
            # except Exception:
            #     GPIO.output( 24, 0 )
            #     pass

    #---------------------------------------------------------------------------
    # Blink LED on BCM GPIO24
    #---------------------------------------------------------------------------

    def blink( self ):
        GPIO.output( 24, 1 )
        sleep( 0.05 )
        GPIO.output( 24, 0 )
        # sleep( 0.05 )
        # GPIO.output( 24, 1 )
        # sleep( 0.05 )
        # GPIO.output( 24, 0 )
```

```
#=====================================================
# Main (for testing purposes )
#=====================================================

if __name__ == "__main__":
    basestation = BaseStationLogger()
    while 1:
        basestation.run()
```

Code Block 2: DataServer.py

```
# DataServer.py

from threading import Thread
import socket
import time
import RPi.GPIO as GPIO

VERBOSE = False
IP_PORT = 22000
P_BUTTON = 24 # adapt to your wiring

def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(P_BUTTON, GPIO.IN, GPIO.PUD_UP)

def debug(text):
    if VERBOSE:
        print "Debug:---", text

# ---------------------- class SocketHandler ------------------------
class SocketHandler(Thread):
    def __init__(self, conn):
        Thread.__init__(self)
        self.conn = conn

    def run(self):
        global isConnected
        debug("SocketHandler started")
        while True:
            cmd = ""
            try:
                debug("Calling blocking conn.recv()")
                cmd = self.conn.recv(1024)
            except:
                debug("exception in conn.recv()")
                # happens when connection is reset from the peer
                break
            debug("Received cmd: " + cmd + " len: " + str(len(cmd)))
            if len(cmd) == 0:
                break
            self.executeCommand(cmd)
        conn.close()
        print "Client disconnected. Waiting for next client..."
        isConnected = False
        debug("SocketHandler terminated")

    def executeCommand(self, cmd):
```

```
        debug("Calling executeCommand() with  cmd: " + cmd)
        if cmd[:-1] == "go":  # remove trailing "\0"
            if GPIO.input(P_BUTTON) == GPIO.LOW:
                state = "Button pressed"
            else:
                state = "Button released"
            print "Reporting current state:", state
            self.conn.sendall(state + "\0")
# ----------------- End of SocketHandler -----------------------

setup()
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# close port when process exits:
serverSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
debug("Socket created")
HOSTNAME = "" # Symbolic name meaning all available interfaces
try:
    serverSocket.bind((HOSTNAME, IP_PORT))
except socket.error as msg:
    print "Bind failed", msg[0], msg[1]
    sys.exit()
serverSocket.listen(10)

print "Waiting for a connecting client..."
isConnected = False
while True:
    debug("Calling blocking accept()...")
    conn, addr = serverSocket.accept()
    print "Connected with client at " + addr[0]
    isConnected = True
    socketHandler = SocketHandler(conn)
    # necessary to terminate it at program termination:
    socketHandler.setDaemon(True)
    socketHandler.start()
    t = 0
    while isConnected:
        print "Server connected at", t, "s"
        time.sleep(10)
        t += 10
```

Code Block 3: DataClient.py

```
# DataClient.py

from threading import Thread
import socket, time

VERBOSE = False
IP_ADDRESS = "192.168.0.17"
IP_PORT = 22000

def debug(text):
    if VERBOSE:
        print "Debug:---", text

# ------------------------ class Receiver --------------------------
class Receiver(Thread):
    def run(self):
        debug("Receiver thread started")
```

```
        while True:
            try:
                rxData = self.readServerData()
            except:
                debug("Exception in Receiver.run()")
                isReceiverRunning = False
                closeConnection()
                break
        debug("Receiver thread terminated")

    def readServerData(self):
        debug("Calling readResponse")
        bufSize = 4096
        data = ""
        while data[-1:] != "\0": # reply with end-of-message indicator
            try:
                blk = sock.recv(bufSize)
                if blk != None:
                    debug("Received data block from server, len: " + \
                        str(len(blk)))
                else:
                    debug("sock.recv() returned with None")
            except:
                raise Exception("Exception from blocking sock.recv()")
            data += blk
        print "Data received:", data
# ----------------------- End of Receiver --------------------

def startReceiver():
    debug("Starting Receiver thread")
    receiver = Receiver()
    receiver.start()

def sendCommand(cmd):
    debug("sendCommand() with cmd = " + cmd)
    try:
        # append \0 as end-of-message indicator
        sock.sendall(cmd + "\0")
    except:
        debug("Exception in sendCommand()")
        closeConnection()

def closeConnection():
    global isConnected
    debug("Closing socket")
    sock.close()
    isConnected = False

def connect():
    global sock
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    debug("Connecting...")
    try:
        sock.connect((IP_ADDRESS, IP_PORT))
    except:
        debug("Connection failed.")
        return False
    startReceiver()
    return True
```

```
sock = None
isConnected = False

if connect():
    isConnected = True
    print "Connection established"
    time.sleep(1)
    while isConnected:
        print "Sending command: go..."
        sendCommand("go")
        time.sleep(2)
else:
    print "Connection to %s:%d failed" % (IP_ADDRESS, IP_PORT)
print "done"
```

Code Block 4: rfPacketRx.c

```
/*
 * Copyright (c) 2015-2016, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * *  Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * *  Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/***** Includes *****/
#include <stdlib.h>
#include <xdc/std.h>
#include <xdc/cfg/global.h>
#include <xdc/runtime/System.h>

#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Task.h>

/* Drivers */
#include <ti/drivers/rf/RF.h>
#include <ti/drivers/PIN.h>
```

```
#include <driverlib/rf_prop_mailbox.h>

/* Board Header files */
#include "Board.h"

#include "RFQueue.h"
#include "smartrf_settings/smartrf_settings.h"

#include <stdlib.h>

/* Pin driver handle */
static PIN_Handle ledPinHandle;
static PIN_State ledPinState;

/*
 * Application LED pin configuration table:
 *   - All LEDs board LEDs are off.
 */
PIN_Config pinTable[] =
{
    Board_LED2 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL | PIN_DRVSTR_MAX,
    PIN_TERMINATE
};


/***** Defines *****/
#define RX_TASK_STACK_SIZE 1024
#define RX_TASK_PRIORITY   2

/* Packet RX Configuration */
#define DATA_ENTRY_HEADER_SIZE 8  /* Constant header size of a Generic Data Entry */
#define MAX_LENGTH             30 /* Max length byte the radio will accept */
#define NUM_DATA_ENTRIES       2  /* NOTE: Only two data entries supported at the
moment */
#define NUM_APPENDED_BYTES     2  /* The Data Entries data field will contain:
                                   * 1 Header byte (RF_cmdPropRx.rxConf.bIncludeHdr
= 0x1)
                                   * Max 30 payload bytes
                                   * 1 status byte
(RF_cmdPropRx.rxConf.bAppendStatus = 0x1) */



/***** Prototypes *****/
static void rxTaskFunction(UArg arg0, UArg arg1);
static void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e);

/***** Variable declarations *****/
static Task_Params rxTaskParams;
Task_Struct rxTask;    /* not static so you can see in ROV */
static uint8_t rxTaskStack[RX_TASK_STACK_SIZE];

static RF_Object rfObject;
static RF_Handle rfHandle;

/* Buffer which contains all Data Entries for receiving data.
 * Pragmas are needed to make sure this buffer is 4 byte aligned (requirement from
the RF Core) */
#if defined(__TI_COMPILER_VERSION__)
    #pragma DATA_ALIGN (rxDataEntryBuffer, 4);
        static uint8_t
rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES,
                                                  MAX_LENGTH,
                                                  NUM_APPENDED_BYTES)
];
```

```
#elif defined(__IAR_SYSTEMS_ICC__)
    #pragma data_alignment = 4
        static uint8_t
rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES,
                                                  MAX_LENGTH,
                                                  NUM_APPENDED_BYTES)
];
#elif defined(__GNUC__)
        static uint8_t rxDataEntryBuffer
[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES,
            MAX_LENGTH, NUM_APPENDED_BYTES)] __attribute__ ((aligned (4)));
#else
    #error This compiler is not supported.
#endif

/* Receive dataQueue for RF Core to fill in data */
static dataQueue_t dataQueue;
static rfc_dataEntryGeneral_t* currentDataEntry;
static uint8_t packetLength;
static uint8_t* packetDataPointer;

static PIN_Handle pinHandle;

static uint8_t packet[MAX_LENGTH + NUM_APPENDED_BYTES - 1]; /* The length byte is
stored in a separate variable */


/***** Function definitions *****/
void RxTask_init(PIN_Handle ledPinHandle) {
    pinHandle = ledPinHandle;

    Task_Params_init(&rxTaskParams);
    rxTaskParams.stackSize = RX_TASK_STACK_SIZE;
    rxTaskParams.priority = RX_TASK_PRIORITY;
    rxTaskParams.stack = &rxTaskStack;
    rxTaskParams.arg0 = (UInt)1000000;

    Task_construct(&rxTask, rxTaskFunction, &rxTaskParams, NULL);
}

static void rxTaskFunction(UArg arg0, UArg arg1)
{
    RF_Params rfParams;
    RF_Params_init(&rfParams);

    if( RFQueue_defineQueue(&dataQueue,
                            rxDataEntryBuffer,
                            sizeof(rxDataEntryBuffer),
                            NUM_DATA_ENTRIES,
                            MAX_LENGTH + NUM_APPENDED_BYTES))
    {
        /* Failed to allocate space for all data entries */
        while(1);
    }

    /* Modify CMD_PROP_RX command for application needs */
    RF_cmdPropRx.pQueue = &dataQueue;           /* Set the Data Entity queue for
received data */
    RF_cmdPropRx.rxConf.bAutoFlushIgnored = 1;  /* Discard ignored packets from Rx
queue */
    RF_cmdPropRx.rxConf.bAutoFlushCrcErr = 1;   /* Discard packets with CRC error
from Rx queue */
    RF_cmdPropRx.maxPktLen = MAX_LENGTH;        /* Implement packet length filtering
to avoid PROP_ERROR_RXBUF */
    RF_cmdPropRx.pktConf.bRepeatOk = 1;
```

```
    RF_cmdPropRx.pktConf.bRepeatNok = 1;

    /* Request access to the radio */
    rfHandle = RF_open(&rfObject, &RF_prop,
(RF_RadioSetup*)&RF_cmdPropRadioDivSetup, &rfParams);

    /* Set the frequency */
    RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);

    /* Enter RX mode and stay forever in RX */
    RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropRx, RF_PriorityNormal, &callback,
IRQ_RX_ENTRY_DONE);

    while(1);
}

void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e)
{
    if (e & RF_EventRxEntryDone)
    {
        /* Toggle pin to indicate RX */
        PIN_setOutputValue(pinHandle, Board_LED2,!PIN_getOutputValue(Board_LED2));

        /* Get current unhandled data entry */
        currentDataEntry = RFQueue_getDataEntry();

        /* Handle the packet data, located at &currentDataEntry->data:
         * - Length is the first byte with the current configuration
         * - Data starts from the second byte */
        packetLength      = *(uint8_t*)(&currentDataEntry->data);
        packetDataPointer = (uint8_t*)(&currentDataEntry->data + 1);

        /* Copy the payload + the status byte to the packet variable */
        memcpy(packet, packetDataPointer, (packetLength + 1));

        RFQueue_nextEntry();
    }
}

/*
 *  ======== main ========
 */
int main(void)
{
    /* Call board init functions. */
    Board_initGeneral();

    /* Open LED pins */
    ledPinHandle = PIN_open(&ledPinState, pinTable);
    if(!ledPinHandle)
    {
        System_abort("Error initializing board LED pins\n");
    }

    /* Initialize task */
    RxTask_init(ledPinHandle);

    /* Start BIOS */
    BIOS_start();

    return (0);
}
```

Code Block 5: rfPacketTx.c

```c
/*
 * Copyright (c) 2015-2016, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * *  Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * *  Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/***** Includes *****/
#include <stdlib.h>
#include <xdc/std.h>
#include <xdc/runtime/System.h>

#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Task.h>

/* Drivers */
#include <ti/drivers/rf/RF.h>
#include <ti/drivers/PIN.h>

/* Board Header files */
#include "Board.h"

#include "smartrf_settings/smartrf_settings.h"

/* Pin driver handle */
static PIN_Handle ledPinHandle;
static PIN_State ledPinState;

/*
 * Application LED pin configuration table:
 *   - All LEDs board LEDs are off.
 */
PIN_Config pinTable[] =
{
    Board_LED1 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL | PIN_DRVSTR_MAX,
    PIN_TERMINATE
};


/***** Defines *****/
```

```
#define TX_TASK_STACK_SIZE 1024
#define TX_TASK_PRIORITY   2

/* Packet TX Configuration */
#define PAYLOAD_LENGTH      30
#define PACKET_INTERVAL     (uint32_t)(4000000*0.5f) /* Set packet interval to 500ms
*/



/***** Prototypes *****/
static void txTaskFunction(UArg arg0, UArg arg1);



/***** Variable declarations *****/
static Task_Params txTaskParams;
Task_Struct txTask;    /* not static so you can see in ROV */
static uint8_t txTaskStack[TX_TASK_STACK_SIZE];

static RF_Object rfObject;
static RF_Handle rfHandle;

uint32_t time;
static uint8_t packet[PAYLOAD_LENGTH];
static uint16_t seqNumber;
static PIN_Handle pinHandle;


/***** Function definitions *****/
void TxTask_init(PIN_Handle inPinHandle)
{
    pinHandle = inPinHandle;

    Task_Params_init(&txTaskParams);
    txTaskParams.stackSize = TX_TASK_STACK_SIZE;
    txTaskParams.priority = TX_TASK_PRIORITY;
    txTaskParams.stack = &txTaskStack;
    txTaskParams.arg0 = (UInt)1000000;

    Task_construct(&txTask, txTaskFunction, &txTaskParams, NULL);
}

static void txTaskFunction(UArg arg0, UArg arg1)
{
    uint32_t time;
    RF_Params rfParams;
    RF_Params_init(&rfParams);

    RF_cmdPropTx.pktLen = PAYLOAD_LENGTH;
    RF_cmdPropTx.pPkt = packet;
    RF_cmdPropTx.startTrigger.triggerType = TRIG_ABSTIME;
    RF_cmdPropTx.startTrigger.pastTrig = 1;
    RF_cmdPropTx.startTime = 0;

    /* Request access to the radio */
    rfHandle = RF_open(&rfObject, &RF_prop,
(RF_RadioSetup*)&RF_cmdPropRadioDivSetup, &rfParams);

    /* Set the frequency */
    RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);

    /* Get current time */
    time = RF_getCurrentTime();
    while(1)
```

```
    {
        /* Create packet with incrementing sequence number and random payload */
        packet[0] = (uint8_t)(seqNumber >> 8);
        packet[1] = (uint8_t)(seqNumber++);
        uint8_t i;
        for (i = 2; i < PAYLOAD_LENGTH; i++)
        {
            packet[i] = rand();
        }

        /* Set absolute TX time to utilize automatic power management */
        time += PACKET_INTERVAL;
        RF_cmdPropTx.startTime = time;

        /* Send packet */
        RF_EventMask result = RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTx,
RF_PriorityNormal, NULL, 0);
        if (!(result & RF_EventLastCmdDone))
        {
            /* Error */
            while(1);
        }

        PIN_setOutputValue(pinHandle, Board_LED1,!PIN_getOutputValue(Board_LED1));
    }
}

/*
 *  ======== main ========
 */
int main(void)
{
    /* Call board init functions. */
    Board_initGeneral();

    /* Open LED pins */
    ledPinHandle = PIN_open(&ledPinState, pinTable);
    if(!ledPinHandle)
    {
        System_abort("Error initializing board LED pins\n");
    }

    /* Initialize task */
    TxTask_init(ledPinHandle);

    /* Start BIOS */
    BIOS_start();

    return (0);
}
```

Code Block 6: rfWakeOnRadioTx.c

```
/*
 * Copyright (c) 2016, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * *  Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above copyright
```

```
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * *  Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/***** Includes *****/
#include <stdlib.h>
#include <xdc/std.h>
#include <xdc/cfg/global.h>
#include <xdc/runtime/System.h>

#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Task.h>

/* Drivers */
#include <ti/drivers/rf/RF.h>
#include <ti/drivers/PIN.h>

/* Display */
#include <ti/mw/display/Display.h>

/* Board Header files */
#include "Board.h"

/* RF settings */
#include "smartrf_settings/smartrf_settings.h"


/***** Defines *****/
/* Wake-on-Radio configuration */
#define WOR_WAKEUPS_PER_SECOND 2

/* TX number of random payload bytes */
#define PAYLOAD_LENGTH 30

/* WOR Example configuration defines */
#define WOR_PREAMBLE_TIME_RAT_TICKS(x) \
    ((uint32_t)(4000000*(1.0f/(x))))

/* TX task stack size and priority */
#define TX_TASK_STACK_SIZE 1024
#define TX_TASK_PRIORITY   2


/***** Prototypes *****/
static void txTaskFunction(UArg arg0, UArg arg1);
static void initializeTxAdvCmdFromTxCmd(rfc_CMD_PROP_TX_ADV_t* RF_cmdPropTxAdv,
rfc_CMD_PROP_TX_t* RF_cmdPropTx);
```

```
/***** Variable declarations *****/
/* TX task objects and task stack */
static Task_Params txTaskParams;
Task_Struct txTask;    /* not static so you can see in ROV */
static uint8_t txTaskStack[TX_TASK_STACK_SIZE];

/* TX packet payload (length +1 to fit length byte) and sequence number */
static uint8_t packet[PAYLOAD_LENGTH +1];
static uint16_t seqNumber;

/* RF driver objects and handles */
static RF_Object rfObject;
static RF_Handle rfHandle;

/* Pin driver objects and handles */
static PIN_Handle ledPinHandle;
static PIN_Handle buttonPinHandle;
static PIN_State ledPinState;
static PIN_State buttonPinState;

/* TX Semaphore */
static Semaphore_Struct txSemaphore;
static Semaphore_Handle txSemaphoreHandle;

/* Advanced TX command for sending long preamble */
static rfc_CMD_PROP_TX_ADV_t RF_cmdPropTxAdv;

/*
 * Application LED pin configuration table:
 *   - All LEDs board LEDs are off.
 */
PIN_Config pinTable[] =
{
    Board_LED0 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL | PIN_DRVSTR_MAX,
    Board_LED1 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL | PIN_DRVSTR_MAX,
    PIN_TERMINATE
};

/*
 * Application button pin configuration table:
 *   - Buttons interrupts are configured to trigger on falling edge.
 */
PIN_Config buttonPinTable[] = {
    Board_BUTTON0 | PIN_INPUT_EN | PIN_PULLUP | PIN_IRQ_NEGEDGE,
    PIN_TERMINATE
};


/***** Function definitions *****/
/* Pin interrupt Callback function board buttons configured in the pinTable. */
void buttonCallbackFunction(PIN_Handle handle, PIN_Id pinId) {

    /* Simple debounce logic, only toggle if the button is still pushed (low) */
    CPUdelay((uint32_t)((48000000/3)*0.050f));
    if (!PIN_getInputValue(pinId)) {
        /* Post TX semaphore to TX task */
        Semaphore_post(txSemaphoreHandle);
    }
}

/* TX task initialization function. Runs once from main() */
void txTaskInit()
{
    /* Initialize TX semaphore */
    Semaphore_construct(&txSemaphore, 0, NULL);
```

```
    txSemaphoreHandle = Semaphore_handle(&txSemaphore);

    /* Initialize and create TX task */
    Task_Params_init(&txTaskParams);
    txTaskParams.stackSize = TX_TASK_STACK_SIZE;
    txTaskParams.priority = TX_TASK_PRIORITY;
    txTaskParams.stack = &txTaskStack;
    Task_construct(&txTask, txTaskFunction, &txTaskParams, NULL);
}

/* TX task function. Executed in Task context by TI-RTOS when the scheduler starts.
*/
static void txTaskFunction(UArg arg0, UArg arg1)
{
    /* Initialize the display and try to open both UART and LCD types of display. */
    Display_Params params;
    Display_Params_init(&params);
    params.lineClearMode = DISPLAY_CLEAR_BOTH;
    Display_Handle uartDisplayHandle = Display_open(Display_Type_UART, &params);
    Display_Handle lcdDisplayHandle = Display_open(Display_Type_LCD, &params);

    /* Print initial display content on both UART and any LCD present */
    Display_print0(uartDisplayHandle, 0, 0, "Wake-on-Radio TX");
    Display_print1(uartDisplayHandle, 0, 0, "Pkts sent: %u", seqNumber);
    Display_print0(lcdDisplayHandle, 0, 0, "Wake-on-Radio TX");
    Display_print1(lcdDisplayHandle, 1, 0, "Pkts sent: %u", seqNumber);

    /* Setup callback for button pins */
    if (PIN_registerIntCb(buttonPinHandle, &buttonCallbackFunction) != 0) {
        System_abort("Error registering button callback function");
    }

    /* Initialize the radio */
    RF_Params rfParams;
    RF_Params_init(&rfParams);

    /* Initialize TX_ADV command from TX command */
    initializeTxAdvCmdFromTxCmd(&RF_cmdPropTxAdv, &RF_cmdPropTx);

    /* Set application specific fields */
    RF_cmdPropTxAdv.pktLen = PAYLOAD_LENGTH +1; /* +1 for length byte */
    RF_cmdPropTxAdv.pPkt = packet;
    RF_cmdPropTxAdv.preTrigger.triggerType = TRIG_REL_START;
    RF_cmdPropTxAdv.preTime = WOR_PREAMBLE_TIME_RAT_TICKS(WOR_WAKEUPS_PER_SECOND);

    /* Request access to the radio */
    rfHandle = RF_open(&rfObject, &RF_prop,
(RF_RadioSetup*)&RF_cmdPropRadioDivSetup, &rfParams);

    /* Set the frequency */
    RF_runCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);

    /* Enter main TX loop */
    while(1)
    {
        /* Wait for a button press */
        Semaphore_pend(txSemaphoreHandle, BIOS_WAIT_FOREVER);

        /* Create packet with incrementing sequence number and random payload */
        packet[0] = PAYLOAD_LENGTH;
        packet[1] = (uint8_t)(seqNumber >> 8);
        packet[2] = (uint8_t)(seqNumber++);
        uint8_t i;
        for (i = 3; i < PAYLOAD_LENGTH +1; i++)
        {
```

```
            packet[i] = rand();
        }

        /* Send packet */
        RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTxAdv, RF_PriorityNormal, NULL, 0);

        /* Update display */
        Display_print1(uartDisplayHandle, 0, 0, "Pkts sent: %u", seqNumber);
        Display_print1(lcdDisplayHandle, 1, 0, "Pkts sent: %u", seqNumber);

        /* Toggle LED */
        PIN_setOutputValue(ledPinHandle,
Board_LED1, !PIN_getOutputValue(Board_LED1));
    }
}

/* Copy the basic RX configuration from CMD_PROP_RX to CMD_PROP_RX_SNIFF command. */
static void initializeTxAdvCmdFromTxCmd(rfc_CMD_PROP_TX_ADV_t* RF_cmdPropTxAdv,
rfc_CMD_PROP_TX_t* RF_cmdPropTx)
{
    #define RADIO_OP_HEADER_SIZE 14

    /* Copy general radio operation header from TX commmand to TX_ADV */
    memcpy(RF_cmdPropTxAdv, RF_cmdPropTx, RADIO_OP_HEADER_SIZE);

    /* Set command to CMD_PROP_TX_ADV */
    RF_cmdPropTxAdv->commandNo = CMD_PROP_TX_ADV;

    /* Copy over relevant parameters */
    RF_cmdPropTxAdv->pktConf.bFsOff = RF_cmdPropTx->pktConf.bFsOff;
    RF_cmdPropTxAdv->pktConf.bUseCrc = RF_cmdPropTx->pktConf.bUseCrc;
    RF_cmdPropTxAdv->syncWord = RF_cmdPropTx->syncWord;
}


/*
 *  ======== main ========
 */
int main(void)
{
    /* Call board init functions. */
    Board_initGeneral();

    /* Open LED pins */
    ledPinHandle = PIN_open(&ledPinState, pinTable);
    if (!ledPinHandle)
    {
        System_abort("Error initializing board LED pins\n");
    }

    /* Open Button pins */
    buttonPinHandle = PIN_open(&buttonPinState, buttonPinTable);
    if (!buttonPinHandle) {
        System_abort("Error initializing button pins\n");
    }

    /* Initialize task */
    txTaskInit();
    /* Start BIOS */
    BIOS_start();
    return (0);
}
```

Code Block 7: rfWakeOnRadioRx.c

```
/*
 * Copyright (c) 2016, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * *  Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * *  Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
/***** Includes *****/
#include <stdlib.h>
#include <xdc/std.h>
#include <xdc/cfg/global.h>
#include <xdc/runtime/System.h>

#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Task.h>

/* Drivers */
#include <ti/drivers/rf/RF.h>
#include <ti/drivers/PIN.h>
#include <ti/drivers/pin/PINCC26XX.h>

/* Board Header files */
#include "Board.h"

/* RF settings, defines and helper files */
#include "RFQueue.h"
#include "smartrf_settings/smartrf_settings.h"
#include "driverlib/rf_prop_mailbox.h"


/***** Defines *****/
/* Wake-on-Radio wakeups per second */
#define WOR_WAKEUPS_PER_SECOND  2

/* Wake-on-Radio mode. Can be:
 * - RSSI only
 * - PQT, preamble detection
 * - Both, first RSSI and then PQT if RSSI  */
#define WOR_MODE CarrierSenseMode_RSSIandPQT
```

```
/* Threshold for RSSI based Carrier Sense in dBm */
#define WOR_RSSI_THRESHOLD      ((int8_t)(-111))

/* Macro used to set actual wakeup interval */
#define WOR_WAKE_UP_MARGIN_S 0.005f
#define WOR_WAKE_UP_INTERVAL_RAT_TICKS(x) \
    ((uint32_t)(4000000*(1.0f/(x) - (WOR_WAKE_UP_MARGIN_S))))

/* TI-RTOS Task configuration */
#define RX_TASK_STACK_SIZE 1024
#define RX_TASK_PRIORITY   2

/* TX Configuration */
#define DATA_ENTRY_HEADER_SIZE 8  /* Constant header size of a Generic Data Entry */
#define MAX_LENGTH             31 /* Max length byte the radio will accept */
#define NUM_DATA_ENTRIES       2  /* NOTE: Only two data entries supported at the
moment */
#define NUM_APPENDED_BYTES     1  /* Length byte included in the stored packet */

/***** Type declarations *****/
/* General wake-on-radio RX statistics */
struct WorStatistics {
  uint32_t doneIdle;
  uint32_t doneIdleTimeout;
  uint32_t doneRxTimeout;
  uint32_t doneOk;
};

/* Modes of carrier sense possible */
enum CarrierSenseMode {
    CarrierSenseMode_RSSI,
    CarrierSenseMode_PQT,
    CarrierSenseMode_RSSIandPQT,
};


/***** Variable declarations *****/
/* TX task objects and task stack */
static Task_Params rxTaskParams;
static Task_Struct rxTask;
static uint8_t rxTaskStack[RX_TASK_STACK_SIZE];

/* RF driver object and handle */
static RF_Object rfObject;
static RF_Handle rfHandle;

/* Pin driver object and handle */
static PIN_Handle ledPinHandle;
static PIN_State ledPinState;

/* General wake-on-radio sniff status statistics and statistics from the RF Core
about received packets */
static volatile struct WorStatistics worStatistics;
static rfc_propRxOutput_t rxStatistics;

/*
 * Application LED pin configuration table:
 *   - All LEDs board LEDs are off.
 */
static PIN_Config pinTable[] =
{
    Board_LED0 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL | PIN_DRVSTR_MAX,
    Board_LED1 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL | PIN_DRVSTR_MAX,
    PIN_TERMINATE
```

```
};

/* Buffer which contains all Data Entries for receiving data.
 * Pragmas are needed to make sure this buffer is 4 byte aligned (requirement from
the RF Core) */
#if defined(__TI_COMPILER_VERSION__)
    #pragma DATA_ALIGN (rxDataEntryBuffer, 4);
        static uint8_t
rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES,
                                                  MAX_LENGTH,
                                                  NUM_APPENDED_BYTES)
];
#elif defined(__IAR_SYSTEMS_ICC__)
    #pragma data_alignment = 4
        static uint8_t
rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES,
                                                  MAX_LENGTH,
                                                  NUM_APPENDED_BYTES)
];
#elif defined(__GNUC__)
        static uint8_t rxDataEntryBuffer
[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES,
            MAX_LENGTH, NUM_APPENDED_BYTES)] __attribute__ ((aligned (4)));
#else
    #error This compiler is not supported.
#endif

/* RX Data Queue and Data Entry pointer to read out received packets */
static dataQueue_t dataQueue;
static rfc_dataEntryGeneral_t* currentDataEntry;

/* Received packet's length and pointer to the payload */
static uint8_t packetLength;
static uint8_t* packetDataPointer;

/* Sniff command for doing combined Carrier Sense and RX*/
static rfc_CMD_PROP_RX_SNIFF_t RF_cmdPropRxSniff;


/***** Prototypes *****/
static void rxTaskFunction(UArg arg0, UArg arg1);
static void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e);
static void initializeSniffCmdFromRxCmd(rfc_CMD_PROP_RX_SNIFF_t* rxSniffCmd,
rfc_CMD_PROP_RX_t* rxCmd);
static void configureSniffCmd(rfc_CMD_PROP_RX_SNIFF_t* rxSniffCmd, enum
CarrierSenseMode mode, uint32_t datarate, uint8_t wakeupPerSecond);
static uint32_t calculateSymbolRate(uint8_t prescaler, uint32_t rateWord);


/***** Function definitions *****/
/* RX task initialization function. Runs once from main() */
void rxTaskInit()
{
    Task_Params_init(&rxTaskParams);
    rxTaskParams.stackSize = RX_TASK_STACK_SIZE;
    rxTaskParams.priority = RX_TASK_PRIORITY;
    rxTaskParams.stack = &rxTaskStack;

    Task_construct(&rxTask, rxTaskFunction, &rxTaskParams, NULL);
}

/* RX task function. Executed in Task context by TI-RTOS when the scheduler starts.
*/
static void rxTaskFunction(UArg arg0, UArg arg1)
{
```

```
    RF_Params rfParams;
    RF_Params_init(&rfParams);

    /* Route out LNA active pin to LED1 */
    PINCC26XX_setMux(ledPinHandle, Board_LED0, PINCC26XX_MUX_RFC_GPO0);

    /* Create queue and data entries */
    if (RFQueue_defineQueue(&dataQueue,
                            rxDataEntryBuffer,
                            sizeof(rxDataEntryBuffer),
                            NUM_DATA_ENTRIES,
                            MAX_LENGTH + NUM_APPENDED_BYTES))
    {
        /* Failed to allocate space for all data entries */
        while(1);
    }

    /* Copy all RX options from the SmartRF Studio exported RX command to the RX
Sniff command */
    initializeSniffCmdFromRxCmd(&RF_cmdPropRxSniff, &RF_cmdPropRx);

    /* Configure RX part of RX_SNIFF command */
    RF_cmdPropRxSniff.pQueue    = &dataQueue;
    RF_cmdPropRxSniff.pOutput   = (uint8_t*)&rxStatistics;
    RF_cmdPropRxSniff.maxPktLen = MAX_LENGTH;

    /* Discard ignored packets and CRC errors from Rx queue */
    RF_cmdPropRxSniff.rxConf.bAutoFlushIgnored = 1;
    RF_cmdPropRxSniff.rxConf.bAutoFlushCrcErr  = 1;

    /* Calculate datarate from prescaler and rate word */
    uint32_t datarate =
calculateSymbolRate(RF_cmdPropRadioDivSetup.symbolRate.preScale,
                                            RF_cmdPropRadioDivSetup.symbolRate.rateWor
d);

    /* Configure Sniff-mode part of the RX_SNIFF command */
    configureSniffCmd(&RF_cmdPropRxSniff, WOR_MODE, datarate,
WOR_WAKEUPS_PER_SECOND);

    /* Request access to the radio */
    rfHandle = RF_open(&rfObject, &RF_prop,
(RF_RadioSetup*)&RF_cmdPropRadioDivSetup, &rfParams);

    /* Set frequency */
    RF_runCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, &callback, 0);

    /* Save the current radio time */
    RF_cmdPropRxSniff.startTime = RF_getCurrentTime();

    /* Enter main loop */
    while(1)
    {
        /* Set next wakeup time in the future */
        RF_cmdPropRxSniff.startTime +=
WOR_WAKE_UP_INTERVAL_RAT_TICKS(WOR_WAKEUPS_PER_SECOND);

        /* Schedule RX */
        RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropRxSniff, RF_PriorityNormal,
&callback, RF_EventRxEntryDone);

        /* Log RX_SNIFF status */
        switch(RF_cmdPropRxSniff.status) {
            case PROP_DONE_IDLE:
                /* Idle based on RSSI */
```

```
                worStatistics.doneIdle++;
                break;
            case PROP_DONE_IDLETIMEOUT:
                /* Idle based on PQT */
                worStatistics.doneIdleTimeout++;
                break;
            case PROP_DONE_RXTIMEOUT:
                /* Got valid preamble on the air, but did not find sync word */
                worStatistics.doneRxTimeout++;
                break;
            case PROP_DONE_OK:
                /* Received packet */
                worStatistics.doneOk++;
                break;
            default:
                /* Unhandled status */
                break;
        };
    }
}

/* Calculates datarate from prescaler and rate word */
static uint32_t calculateSymbolRate(uint8_t prescaler, uint32_t rateWord)
{
    /* Calculate datarate according to TRM Section 23.7.5.2:
     * f_baudrate = (R * f_ref)/(p * 2^20)
     *   - R = rateWord
     *   - f_ref = 24Mhz
     *   - p = prescaler */
    uint64_t numerator = rateWord*24000000ULL;
    uint64_t denominator = prescaler*1048576ULL;
    uint32_t result = (uint32_t)(numerator/denominator);
    return result;
}

/* Copies all RX options from the SmartRF Studio exported RX command to the RX Sniff
command */
static void initializeSniffCmdFromRxCmd(rfc_CMD_PROP_RX_SNIFF_t* rxSniffCmd,
rfc_CMD_PROP_RX_t* rxCmd)
{

    /* Copy RX configuration from RX command */
    memcpy(rxSniffCmd, rxCmd, sizeof(rfc_CMD_PROP_RX_t));

    /* Change to RX_SNIFF command from RX command */
    rxSniffCmd->commandNo = CMD_PROP_RX_SNIFF;
}

/* Configures Sniff-mode part of the RX_SNIFF command based on mode, datarate and
wakeup interval */
static void configureSniffCmd(rfc_CMD_PROP_RX_SNIFF_t* rxSniffCmd, enum
CarrierSenseMode mode, uint32_t datarate, uint8_t wakeupPerSecond)
{
    /* Enable or disable RSSI */
    if ((mode == CarrierSenseMode_RSSI) || (mode == CarrierSenseMode_RSSIandPQT)) {
        rxSniffCmd->csConf.bEnaRssi         = 1;
    } else {
        rxSniffCmd->csConf.bEnaRssi         = 0;
    }

    /* Enable or disable PQT */
    if ((mode == CarrierSenseMode_PQT) || (mode == CarrierSenseMode_RSSIandPQT)) {
        rxSniffCmd->csConf.bEnaCorr         = 1;
        rxSniffCmd->csEndTrigger.triggerType  = TRIG_REL_START;
    } else {
```

```
        rxSniffCmd->csConf.bEnaCorr        = 0;
        rxSniffCmd->csEndTrigger.triggerType  = TRIG_NEVER;
    }

    /* General Carrier Sense configuration */
    rxSniffCmd->csConf.operation       = 1; /* Report Idle if RSSI reports Idle to
quickly exit if not above
                                             RSSI threshold */
    rxSniffCmd->csConf.busyOp          = 0; /* End carrier sense on channel Busy
(the receiver will continue when
                                             carrier sense ends, but it will
then not end if channel goes Idle) */
    rxSniffCmd->csConf.idleOp          = 1; /* End on channel Idle */
    rxSniffCmd->csConf.timeoutRes      = 1; /* If the channel is invalid, it will
return PROP_DONE_IDLE_TIMEOUT */

    /* RSSI configuration */
    rxSniffCmd->numRssiIdle            = 1; /* One idle RSSI samples signals that
the channel is idle */
    rxSniffCmd->numRssiBusy            = 1; /* One busy RSSI samples signals that
the channel is busy */
    rxSniffCmd->rssiThr    = (int8_t)WOR_RSSI_THRESHOLD; /* Set the RSSI threshold
in dBm */

    /* PQT configuration */
    rxSniffCmd->corrConfig.numCorrBusy = 1;   /* One busy PQT samples signals that
the channel is busy */
    rxSniffCmd->corrConfig.numCorrInv  = 1;   /* One busy PQT samples signals that
the channel is busy */

    /* Calculate basic timing parameters */
    uint32_t symbolLengthUs  = 1000000UL/datarate;
    uint32_t preambleSymbols = (1000000UL/wakeupPerSecond)/symbolLengthUs;
    uint8_t syncWordSymbols  = RF_cmdPropRadioDivSetup.formatConf.nSwBits;

    /* Calculate sniff mode parameters */
    #define US_TO_RAT_TICKS 4
    #define CORR_PERIOD_SYM_MARGIN 16
    #define RX_END_TIME_SYM_MARGIN 8
    #define CS_END_TIME_MIN_TIME_SYM 30
    #define CS_END_TIME_MIN_TIME_STATIC_US 150

    /* Represents the time in which we need to receive corrConfig.numCorr*
correlation peaks to detect preamble.
     * When continously checking the preamble quality, this period has to be wide
enough to also contain the sync
     * word, with a margin. If it is not, then there is a chance the SNIFF command
will abort while receiving the
     * sync word, as it no longer detects a preamble. */
    uint32_t correlationPeriodUs = (syncWordSymbols +
CORR_PERIOD_SYM_MARGIN)*symbolLengthUs;

    /* Represents the time where we will force a check if preamble is present (only
done once).
     * The main idea is that his should be shorter than "correlationPeriodUs" so
that if we get RSSI valid, but
     * there is not a valid preamble on the air, we will leave RX as quickly as
possible. */
    uint32_t csEndTimeUs = (CS_END_TIME_MIN_TIME_SYM*symbolLengthUs +
CS_END_TIME_MIN_TIME_STATIC_US);

    /* Represents the maximum time from the startTrigger to when we expect a sync
word to be received. */
    uint32_t rxEndTimeUs = (preambleSymbols + syncWordSymbols +
RX_END_TIME_SYM_MARGIN)*symbolLengthUs;
```

```
    /* Set sniff mode timing configuration in sniff command in RAT ticks */
    rxSniffCmd->corrPeriod = (uint16_t)(correlationPeriodUs * US_TO_RAT_TICKS);
    rxSniffCmd->csEndTime  = (uint32_t)(csEndTimeUs * US_TO_RAT_TICKS);
    rxSniffCmd->endTime    = (uint32_t)(rxEndTimeUs * US_TO_RAT_TICKS);

    /* Set correct trigger types */
    rxSniffCmd->endTrigger.triggerType   = TRIG_REL_START;
    rxSniffCmd->startTrigger.triggerType = TRIG_ABSTIME;
    rxSniffCmd->startTrigger.pastTrig    = 1;
}

/* Called for every received packet and command done */
void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e)
{
    /* If we've received a new packet and it's available to read out */
    if (e & RF_EventRxEntryDone)
    {
        do
        {
            /* Toggle LED on RX */
            PIN_setOutputValue(ledPinHandle,
Board_LED1, !PIN_getOutputValue(Board_LED1));

            /* Get current unhandled data entry */
            currentDataEntry = RFQueue_getDataEntry();

            /* Handle the packet data, located at &currentDataEntry->data:
             * - Length is the first byte with the current configuration
             * - Data starts from the second byte */
            packetLength      = *(uint8_t*)(&currentDataEntry->data);
            packetDataPointer = (uint8_t*)(&currentDataEntry->data + 1);

            /* This code block is added to avoid a compiler warning.
            * Normally, an application will reference these variables for
            * useful data. */
            volatile uint8_t dummy;
            dummy = packetLength;
            dummy = dummy + packetDataPointer[0];

        } while(RFQueue_nextEntry() == DATA_ENTRY_FINISHED);
    }
}
/*
 *  ======== main ========
 */
int main(void)
{
    /* Call board init functions. */
    Board_initGeneral();

    /* Open LED pins */
    ledPinHandle = PIN_open(&ledPinState, pinTable);
    if(!ledPinHandle)
    {
        System_abort("Error initializing board LED pins\n");
    }
    /* Initialize task */
    rxTaskInit();
    /* Start BIOS */
    BIOS_start();
    return (0);
}
```