

Communication between Wireless Sensor Devices and GNU Radio

Sriram Sanka and Gaurav Konchady¹
Department of Electrical and Computer Engineering
Cleveland State University
Cleveland, OH 44115

May 4, 2009

Abstract

Wireless sensor network nodes (motes) are available in different hardware platforms. Radio is a key component in a mote. Not all mote platforms are equipped with the same radio transceiver which limits communication between different mote platforms. Certain wireless sensor network applications need a combination of mote platforms depending on the sensing, computing, power and communication requirements. Thus there is a need for a device that can handle signals from a variety of mote platforms. This can greatly simplify the base-station design and also enable communication between heterogeneous mote platforms. We have used a universal software radio peripheral (USRP) device to demonstrate communication between MICA2 and TelosB mote platforms. Our project is based on work done by Thomas Schmid at UCLA.

1. Introduction

A wireless sensor network consists of sensor motes that monitor physical parameters. A typical wireless sensor network (WSN) is based on a two-tiered architecture. One tier contains spatially distributed sensing motes and the other tier contains a base-station node. A sensing mote periodically samples designated physical parameters and sends the collected data to base-station. A base-station can be a mote or any device capable of communicating with the motes in the network. A base-station may act as a relay to base-stations of other networks or may be connected to a computer. The computer stores and performs offline processing of the collected data. The collected sensor data can be published to a repository using web technology, thus enabling remote access to users.

Software-defined radios can be used as base-stations in a wireless sensor network. The USRP device supports a wide frequency range and can send or receive data using different modulation schemes. Hence the USRP can communicate with different mote platforms and serve as a powerful base-station in a wireless sensor network. In our project we implement a wireless sensor network scenario containing MICA2 [1], MICAz [2] and TelosB [3] motes with the USRP [4] acting as a base-station. To demonstrate bi-directional communication we send out data packets from the MICA2 motes. These packets are intercepted by the USRP that converts the received data packets to an IEEE 802.15.4 [5] compliant format. The packets are then relayed to the TelosB and MICAz alike. The motes are programmed in the TinyOS [6] 2.x software platform.

In Section 2, we describe the related work and provide background information on mote platforms, TinyOS, GNU Radio [7] and the USRP. Section 3 covers information about the radio used in the MICA2 mote. In Section 4, we discuss about the IEEE 802.15.4 standard. Section 5 covers the project's experiment setup. In Section 6, we describe the migration of code from SOS to TinyOS platform. Section 7 covers the experiment results and in Section 8, we conclude and present the future scope of the project.

¹ Email: {s.sanka99, g.konchady}@csuohio.edu

2. Background

2.1. Mote platforms and TinyOS

Motes are the smallest individual unit of a wireless sensor network. A typical mote contains a radio transceiver chip, microcontroller, sensors and expansion connectors. Motes can be powered using batteries, USB ports and expansion connectors. Popular mote platforms include the MICA2, MICAz, TelosB, IRIS [8], iMote [9], eKo [10] and the most recent Epic family [11]. For our project we have used the Crossbow MICA2, MICAz and TelosB motes. Table 1 shows a comparison of these three mote platforms.

	MICA2	MICAz	TelosB
Processor	Atmel ATmega128L	Atmel ATmega128L	TI MSP430
Radio Transceiver	CC1000	CC2420	CC2420
IEEE 802.15.4 support	No	Yes	Yes
Center Frequency	433/868/916 MHz	2400 to 2483.5 MHz	2400 to 2483.5 MHz
On-board sensors	None	None	Light, Temperature and Humidity
Programming interface	51-pin expansion connector	51-pin expansion connector	USB

Table 1: Comparison of MICA2, MICAz and TelosB motes; data obtained from [1][2][3]

TinyOS is an open-source component based software development platform for motes. Motes execute programs written in the network embedded systems C (nesC) language for TinyOS. TinyOS offers modular programming, multi-tasking and hardware abstractions which provide an easy programming interface for resource constrained motes. TinyOS currently supports many different platforms and sensor boards. Other popular software development platforms for motes include SOS [12], Contiki [13] and MANTIS [14].

2.2. GNU Radio and USRP

We will implement software-defined radio using GNU Radio, an open-source signal processing toolkit [7]. Applications for the GNU Radio are written by creating graphs using the python scripting language. The graphs contain an interconnection of different signal processing blocks implemented in C++. An interface compiler, SWIG is used to bind C++ blocks with python. The diverse range of signal processing blocks provided by GNU Radio makes it a powerful tool for development of complex digital signal processing applications on a desktop computer when connected to radio peripherals. A large number of applications have been implemented in the GNU Radio. Some of them include a HDTV transmitter and receiver, different modulators and demodulators and concurrent multichannel receivers.

USRP acts as the hardware interface for the GNU Radio. USRP is a radio system consisting of a signal processing motherboard and supports two transmit and two receive daughterboards. A USRP motherboard consists of an ADC/DAC module and a user programmable FPGA. A USB interface is available for programming the USRP motherboard. A recent version of the USRP, the USRP2 contains a more powerful FPGA and a faster programming interface via an Ethernet connection.

Figure 1 shows the block diagram of a typical USRP application. On the sender side, USRP is programmed by the sender using the USB interface. Signals sent to the USRP via USB are in 16-bit I-Q format [16]. FPGA up converts the signal to be transmitted to the desired frequency. DAC converts the samples to analog form and this signal is routed to the RF front end present on the daughterboard. On the receiver side, signals received by the daughterboard through the RF front end is converted to digital form and down converted by ADC and FPGA respectively. Signals can be finally received by the computer at

receiver end via USB connection for subsequent processing. In our project we are using a single USRP with two daughterboards for simultaneous transmission and reception.

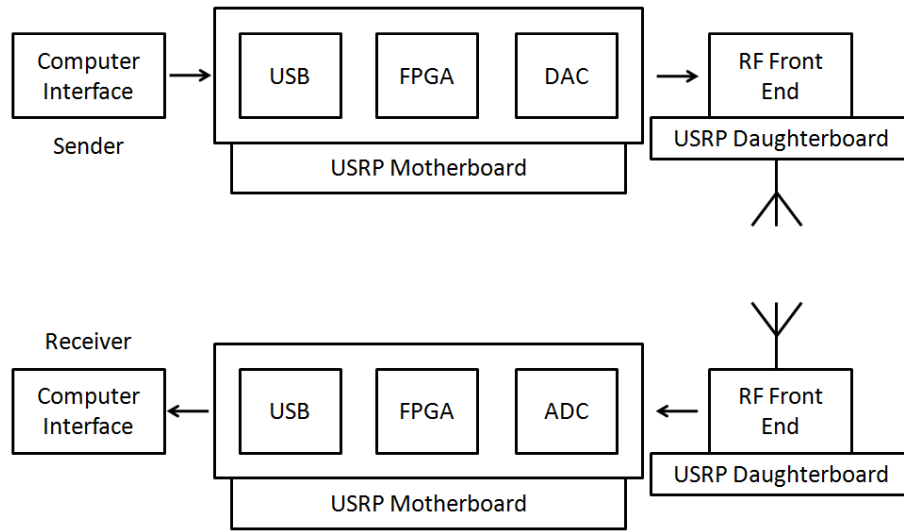


Figure 1: Block diagram of a typical USRP application. Sender programs USRP for transmission. Transmitted signals received by a computer through another USRP on the receiver side. Adapted from [15]

2.4. Related Work

GNU Radio encoding and decoding blocks for IEEE 802.15.4 and CC1000 radios were developed by Thomas Schmid at UCLA [16]. C++ and Python blocks developed by Schmid were intended to operate with the SOS software platform for embedded network devices. Our work extends the same for the TinyOS software platform. The reason for choosing TinyOS is that our other related projects are based on this platform.

3. CC1000 RF transceiver

As mentioned in Table 1, the MICA2 mote uses the CC1000 RF transceiver chip for communication. CC1000 can operate over a frequency range of 300 to 1000 MHz [17]. CC1000 uses FSK modulation with Manchester encoding and supports data rates up to 76.8 kBaud. CC1000 features an integrated bit-synchronizer for reliable data communication. Figure 2, illustrates a receive cycle for CC1000.

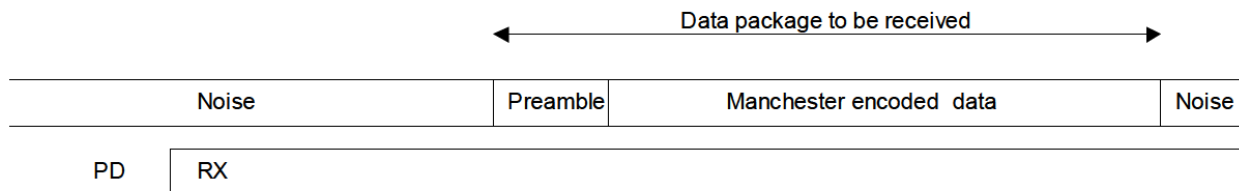


Figure 2: CC1000 bit-synchronization mechanism for Manchester encoded data. At receiver side, CC1000 looks for the preamble field before detecting the encoded data. Taken from [17]

For TinyOS, the frame format for a CC1000 based MICA2 mote differs from that of the SOS platform. Figure 3 shows the frame format of a CC1000 packet in TinyOS. The packet header consists of destination and source address, group id and active message type (AM type). The maximum packet size for CC1000 radio is 128 bytes which limits the maximum data payload to 119 bytes. The packet footer

consists of a 2-byte CRC field. The Python block for CC1000 appends a 64-bit synchronization header, padding bits and access code to TinyOS frame prior to transmission.

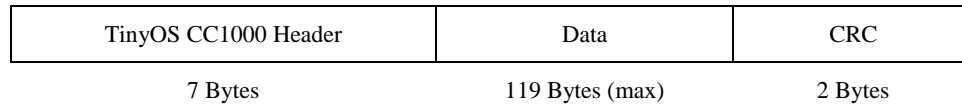


Figure 3: TinyOS CC1000 frame format for MICA2 mote.

4. IEEE 802.15.4 standard

The IEEE 802.15.4 standard defines the MAC and physical layer protocol for short-distance, low data rate and low-power communication between wireless devices operating in the RF range [5]. To establish communication between USRP and motes, we shall implement the 802.15.4 decoder block using GNU Radio for TinyOS. The 802.15.4 compliant motes will transmit data in the form of physical protocol data unit (PPDU) to the software-radio. Figure 4 shows the PPDU format. The GNU Radio application constantly looks for the PPDU frame in the incoming bit-stream. When a PPDU frame is detected, payload will be extracted and displayed on the computer.

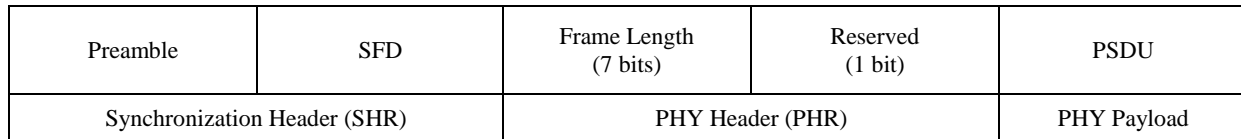


Figure 4: PPDU Format; redraw of [5]

The TinyOS 802.15.4 frame format [18] differs from the original IEEE 802.15.4 format. There are two types of TinyOS frame formats, a T-frame and an I-frame format. T-frame is used in a network not shared with other types of networking architectures. Figure 5 shows the TinyOS 802.15.4 T-frame format. To decode TinyOS frames sent by motes to USRP, we will have to make changes to the 802.15.4 decoder block of the GNU Radio. This will be implemented by making changes in the Python and C++ languages modules for 802.15.4 decoder.

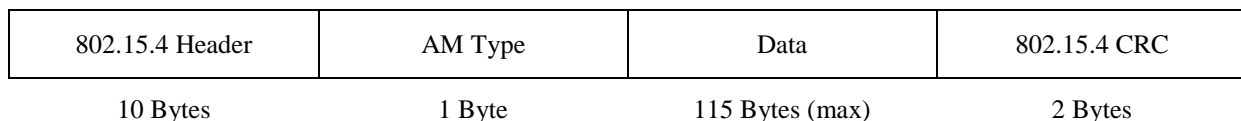


Figure 5: TinyOS T-Frame format adapted from [18]. AM Type is not part of the IEEE 802.15.4 specification.

5. Experimental setup

The MICA2 mote transmits data at 433 MHz, which is received by the FLEX400 [19] daughterboard connected to side-B of a USRP. RFX2400 [20] daughterboard is connected to the same USRP on side-A which relays the data sent by MICA2 mote to a TelosB mote. We used the latest stable version of GNU Radio 3.1.3 because it contains all the blocks necessary for the execution of the Python scripts. The MIB510 programmer board is used to program the MICA2 mote through a serial cable. TelosB mote is programmed via a USB cable. A base-station Java application running on a computer logs the transmitted and received data for verification. Figure 6 shows the block diagram of our project. The Java based base-station application continuously monitors the medium for any packets and prints the header and payload information on the screen.

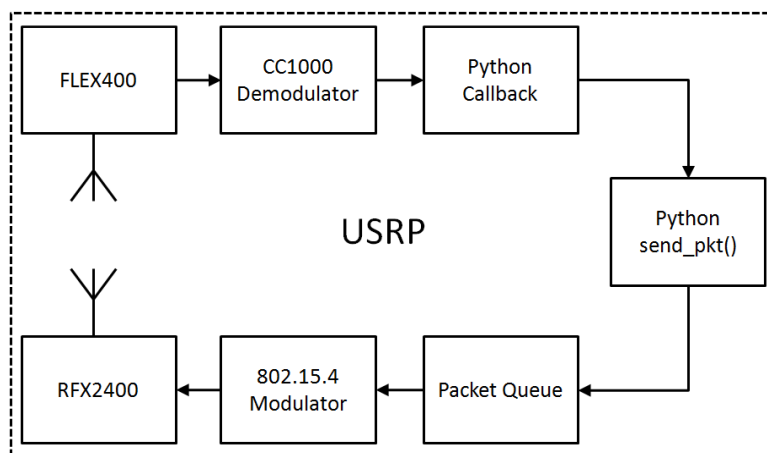


Figure 6: Project block diagram. Packets received from MICA2 mote via FLEX400 daughterboard and relayed to TelosB mote via RFX2400 daughterboard.

Data received on the FLEX400 daughterboard is routed to the CC1000 demodulator block implemented in Python and C++. The CC1000 FSK demodulator block consists of an FM demodulator, correlator and a packet queue. The FM demodulator and correlator detect the received data sequence and strip the synchronization and preamble information. The packets with the header information are then assembled in a message queue. A Python callback function then detects the header information of each packet and passes the payload data to the 802.15.4 encoding block.

The 802.15.4 encoding block accepts the payload data in a *send_pkt()* Python routine where the payload is collected in a message queue and the 802.15.4 MAC header and footer information is appended to the packet. This is followed by the addition of the physical layer synchronization header information. Each frame is then passed to the 802.15.4 modulator block. Inside the modulator block the payload is converted from bytes to a chip sequence. The chips are converted to symbols which are then modulated by the O-QPSK technique.

6. SOS to TinyOS migration

Using *ucla_zigbee_phy* [21] source code as the base for our work, we modified the Python scripts for communication between MICA2 and USRP, and between TelosB and USRP. The Python scripts modified to achieve above mentioned communication are *cc1k_rxtest.py*, *cc1k_txttest.py*, *cc1k_sos_pkt.py*, *cc2420_rxtest.py* and *cc2420_txttest.py*. We created a Python script named *cc1k_tos_pkt.py* derived from an existing Python script *cc1k_sos_pkt.py* for the encoding and decoding of MICA2 packets. We changed the header information and CRC calculation procedure for a TinyOS packet. To implement the 802.15.4 packet encoding and decoding mechanism we had to change the channel number to the default TinyOS channel number as well as make changes to the header information.

Based on successful implementation of the above python communication modules, we proceeded to the main program of our project i.e. data relay from MICA2 to TelosB via USRP. This required making changes to the *cc1k_to_cc2420_relay.py* Python code. In the first part of our program, we had problems detecting the FLEX400 daughterboard. Hence we had to manually select the receiving daughterboard in our script. In the second part we had to extract the MICA2 header information using the *rx_callback* function and add the necessary TinyOS 802.15.4 header information in the Python *send_pkt()* function. Apart from this we had to change the channel number to the default channel number. Appendix A contains the *cc1k_tos_pkt.py* and the new data relay *cc1k_to_cc2420_tosrelay.py* scripts.

7. Results

Figure 9 shows the results obtained from running the setup as described in Section 6. The highlighted portions are the payload bytes. Figure 9 (a) shows the packets received by a MICA2 mote running the Java base-station program. The packets sent by the transmitting MICA2 mote consist of 4-byte payloads. The MICA2 packets are detected by the USRP using the FLEX daughterboard. Through the Python blocks, the payload of a MICA2 packet is extracted and inserted into an 802.15.4 frame along with its header and footer fields. Since MICA2 packets do not contain sequence number, a randomly generated sequence number is used. Figure 9 (b) shows the payload extracted from the MICA2. Figure 9 (c) shows the base-station output from a TelosB mote.

```
(a) 00 00 11 00 01 04 00 06 00 01 04 9B
(b) Mica2 payload: ['0x0', '0x1', '0x4', '0x9b']
    Relaying payload to TelosB...
(c) 00 FF FF 01 00 03 00 00 01 04 9B
    Address          Payload
```

Figure 9: (a) MICA2 base-station output showing MICA2 packet, (b) USRP output showing payload sent by MICA2 and received using FLEX400 daughterboard (c) TelosB base-station output showing 802.15.4 packets relayed by USRP and transmitted using RFX2400 daughterboard. Payload highlighted by red box.

8. Conclusion and future work

We successfully implemented CC1000 FSK encoding and decoding and IEEE 802.15.4 encoding and decoding for the TinyOS software platform. We demonstrated MICA2 to TelosB mote communication via USRP by using the above mentioned communication blocks. In future we plan to implement bidirectional communication between MICA2 and TelosB motes. We also plan to extend this work to other mote platforms and implement communication stacks for Bluetooth. We intend to establish wireless sensor motes communication with Bluetooth devices and evaluate the performance of custom built Bluetooth radio based sensor motes in a wireless sensor network scenario.

References

- [1] Crossbow MICA2 datasheet, http://www.xbow.com/products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf
- [2] Crossbow MICAz datasheet, http://www.xbow.com/products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf
- [3] Crossbow TelosB datasheet, http://www.xbow.com/products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf
- [4] USRP User's and Developer's Guide, www.olifantasia.com/gnuradio/usrp/files/usrp_guide.pdf
- [5] IEEE 802.15.4-2006 standard, <http://standards.ieee.org/getieee802/802.15.html>
- [6] TinyOS open-source operating system, <http://tinysos.net>
- [7] GNU Radio, <http://gnuradio.org>
- [8] Crossbow IRIS datasheet, http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/IRIS_Datasheet.pdf
- [9] iMote Datasheet, <http://www.princeton.edu/~wolf/EECS579/imotes/data%20sheet.pdf>

- [10] Crossbow eKo datasheet, http://www.xbow.com/eko/Images/eKo_Pro_datasheet.pdf
- [11] P. Dutta, J. Taneja, J. Jeong, X. Jiang and D. Culler, *A Building Block Approach to Sensor Network Systems*, ACM SenSys, 2008
- [12] SOS Embedded OS homepage, <https://projects.nesl.ucla.edu/public/sos-2x/doc/>
- [13] Contiki OS homepage, <http://www.sics.se/contiki/>
- [14] MANTIS OS homepage, <http://mantis.cs.colorado.edu>
- [15] Dr. Chansu Yu, Mobile Computing course website, <http://academic.csuohio.edu/yuc/mobile09>
- [16] Thomas Schmid, *GNU Radio 802.15.4 En-and Decoding*, http://nesl.ee.ucla.edu/fw/thomas/thomas_project_report.pdf
- [17] CC1000 datasheet, http://www.cse.ohio-state.edu/siefast/nest/nest_webpage/datasheet/Chipcon%20-%20CC1000%20Data%20Sheet%20v2.1.pdf
- [18] TEP 125 – TinyOS 802.15.4 Frames, http://tinysos.cvs.sourceforge.net/*checkout*/tinysos/tinysos-2.x/doc/html/tep125.html
- [19] RFX/FLEX400 daughterboard datasheet, http://www.ettus.com/downloads/transceiver_dbrds_v3b.pdf
- [20] RFX2400 daughterboard datasheet, http://www.ettus.com/downloads/er_ds_transceiver_dbrds_v5b.pdf
- [21] Thomas Schmid, UCLA ZigBee PHY project website, <https://www.cgran.org/wiki/UCLAZigBee>

Appendix A

Source Code

cc1k_to_cc2420_tosrelay.py

```
#!/usr/bin/env python

#
# Copyright 2004,2006 Free Software Foundation, Inc.
#
# This file is part of GNU Radio
#
# GNU Radio is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2, or (at your option)
# any later version.
#
# GNU Radio is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with GNU Radio; see the file COPYING. If not, write to
# the Free Software Foundation, Inc., 59 Temple Place - Suite 330,
# Boston, MA 02111-1307, USA.
#

#
# This test example relays messages from the CC1K to the CC2420.
# The default center frequencies are 434.845MHz for CC1K receive and
```

```

# channel 26 (2480MHz) for send.    ## default channel for TinyOS
#
# Modified by: Thomas Schmid
# Modified by: Sriram Sanka and Gaurav Konchady for TinyOS (April 2009)
#

from gnuradio import gr, eng_notation
from gnuradio import usrp
from gnuradio import ucla
from gnuradio.ucla_blks import ieee802_15_4_pkt
from gnuradio.ucla_blks import cclk_tos_pkt
from gnuradio.eng_option import eng_option
from optparse import OptionParser
import math, struct, time
import random          ## for sequence number generation

def pick_subdevice(u):
    """
    The user didn't specify a subdevice on the command line.
    If there's a daughterboard on A, select A.
    If there's a daughterboard on B, select B.
    Otherwise, select A.
    """
    if u.db[0][0].dbid() >= 0:      # dbid is < 0 if there's no d'board or a
problem
        return (0, 0)
    if u.db[1][0].dbid() >= 0:
        return (1, 0)
    return (0, 0)

class stats(object):
    """
    This class is used to keep statistics for received
    packets.
    """
    def __init__(self):
        self.npkts = 0
        self.nright = 0

## CC1000 receive routine
class fsk_rx_graph (gr.flow_graph):
    st = stats()

    def __init__(self, options, rx_callback):
        gr.flow_graph.__init__(self)

        # -----
        self.data_rate = 38400
        self.samples_per_symbol = 8
        self.usrp_decim = int (64e6 / self.samples_per_symbol /
self.data_rate)
        self.fs = self.data_rate * self.samples_per_symbol
        payload_size = 128          # bytes

        print "data_rate = ", eng_notation.num_to_str(self.data_rate)
        print "samples_per_symbol = ", self.samples_per_symbol

```

```

print "usrp_decim = ", self.usrp_decim
print "fs = ", eng_notation.num_to_str(self.fs)

max_deviation = self.data_rate / 4

u = usrp.source_c (0, self.usrp_decim)
if options.rx_subdev_spec is None:
    options.rx_subdev_spec = (1, 0)    ## select subdevice receiver
on FLEX400
u.set_mux(usrp.determine_rx_mux_value(u, options.rx_subdev_spec))

subdev = usrp.selected_subdev(u, options.rx_subdev_spec)
print "Using RX d'board %s" % (subdev.side_and_name(),)

u.tune(0, subdev, options.cordic_freq_rx)
u.set_pga(0, options.gain)
u.set_pga(1, options.gain)

filter_taps = gr.firdes.low_pass (1,                # gain
                                  self.fs,          # sampling
                                  rate,              # cutoff
                                  width,             # trans
                                  gr.firdes.WIN_HANN)

print "len = ", len (filter_taps)

# receiver
gain_mu = 0.002*self.samples_per_symbol
self.packet_receiver = cclink_tos_pkt.cclink_demod_pkts(self,
                                                         callback=rx_callback,
                                                         sps=self.samples_per_symbol,
                                                         symbol_rate=self.data_rate,
                                                         p_size=payload_size,
                                                         threshold=-1)

self.connect(u, self.packet_receiver)

if 0 and not(options.no_gui):
    fft_input = fftsink.fft_sink_c (self, panel, title="Input",
    fft_size=512, sample_rate=self.fs)
    self.connect (u, fft_input)
    vbox.Add (fft_input.win, 1, wx.EXPAND)

## CC2420 transmit routine
class transmit_path(gr.top_block):
    def __init__(self, options):
        gr.top_block.__init__(self)
        self.normal_gain = 8000

        self.u = usrp.sink_c()
        dac_rate = self.u.dac_rate();

```

```

self._data_rate = 2000000
self._spb = 2
self._interp = int(128e6 / self._spb / self._data_rate)
self.fs = 128e6 / self._interp

self.u.set_interp_rate(self._interp)

# determine the daughterboard subdevice we're using
if options.tx_subdev_spec is None:
    options.tx_subdev_spec = usrp.pick_tx_subdevice(self.u)
self.u.set_mux(usrp.determine_tx_mux_value(self.u,
options.tx_subdev_spec))
self.subdev = usrp.selected_subdev(self.u, options.tx_subdev_spec)
print "Using TX d'board %s" % (self.subdev.side_and_name(),)

self.u.tune(self.subdev._which, self.subdev, options.cordic_freq_tx)
self.u.set_pga(0, options.gain)
self.u.set_pga(1, options.gain)

# transmitter
self.packet_transmitter =
ieee802_15_4_pkt.ieee802_15_4_mod_pkts(self, spb=self._spb, msgq_limit=2)
self.gain = gr.multiply_const_cc (self.normal_gain)

self.connect(self.packet_transmitter, self.gain, self.u)

self.set_gain(self.subdev.gain_range()[1]) # set max Tx gain
self.set_auto_tr(True) # enable Auto
Transmit/Receive switching

def set_gain(self, gain):
    self.gain = gain
    self.subdev.set_gain(gain)

def set_auto_tr(self, enable):
    return self.subdev.set_auto_tr(enable)

def send_pkt(self, payload='', eof=False):
    ## generate random sequence number between 0 and 254
    return self.packet_transmitter.send_pkt(random.randint(0,
254),struct.pack("7B", 0x22, 0x0, 0xFF,0xFF, 0x1, 0x0, 0x06),payload, eof)
    ## create TinyOS 802.15.4 packet

def main ():

def rx_callback(ok, dest, source, group, am_type, payload, crc):
    st.npkts += 1
    if ok:
        st.nright += 1
        print "MICA2 payload: " + str(map(hex, map(ord, payload)))

        print "Relaying to TelosB...\n"
        fgtx.send_pkt(payload) ## strip MICA2 header, send only payload

```

```

    parser = OptionParser (option_class=eng_option)
    parser.add_option("-R", "--rx-subdev-spec", type="subdev", default=None,
                    help="select USRP Rx side A or B (default=first one
with a daughterboard)")
    parser.add_option("-T", "--tx-subdev-spec", type="subdev", default=None,
                    help="select USRP Tx side A or B (default=first one
with a daughterboard)")

    parser.add_option ("-t", "--cordic-freq-tx", type="eng_float",
default=2480000000,
                    help="set Tx cordic frequency to FREQ",
metavar="FREQ")

    parser.add_option ("-c", "--cordic-freq-rx", type="eng_float",
default=434845200,
                    help="set rx cordic frequency to FREQ",
metavar="FREQ")
    parser.add_option ("-r", "--data-rate", type="eng_float",
default=2000000)
    parser.add_option ("-f", "--filename", type="string",
                    default="rx.dat", help="write data to FILENAME")
    parser.add_option ("-g", "--gain", type="eng_float", default=0,
                    help="set Rx PGA gain in dB [0,20]")
    parser.add_option ("-N", "--no-gui", action="store_true", default=False)

    (options, args) = parser.parse_args ()

    print "cordic_freq_rx = %s" % (eng_notation.num_to_str
(options.cordic_freq_rx))
    print "cordic_freq_tx = %s" % (eng_notation.num_to_str
(options.cordic_freq_tx))

    st = stats()

    fgtx = transmit_path(options)
    fgtx.start()

    fgrx = fsk_rx_graph(options, rx_callback)
    fgrx.start()

    fgrx.wait()
    fgtx.wait()

if __name__ == '__main__':
    main ()

```

cclk_tos_pkt.py

```

#
# Copyright 2005 Free Software Foundation, Inc.
#
# This file is part of GNU Radio
#
# GNU Radio is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2, or (at your option)
# any later version.
#
# GNU Radio is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with GNU Radio; see the file COPYING. If not, write to
# the Free Software Foundation, Inc., 59 Temple Place - Suite 330,
# Boston, MA 02111-1307, USA.
#

# This is derived from gmsk2_pkt.py.
#
# Modified by: Thomas Schmid
# Modified by: Sriram Sanka and Gaurav Konchady for TinyOS (April 2009)
#

from math import pi
import Numeric

from gnuradio import gr, packet_utils
from gnuradio import ucla
import crc8
import gnuradio.gr.gr_threading as _threading
import cclk
import struct

HEADER_SIZE = 8
MAX_PKT_SIZE = 128 - HEADER_SIZE

"""
typedef nx_struct CC1KHeader {
    nx_am_addr_t dest;
    nx_am_addr_t source;
    nx_uint8_t length;
    nx_am_group_t group;
    nx_am_id_t type;
} cc1000_header_t;
"""

def make_tos_packet(dest, source, group, am_type, payload, spb, access_code,
pad_for_usrp=True):
    """
    Build a TinyOS packet

```

```

@param dest:
@param source:
@param group:
@param am_type:
@param payload:
@param sbp:
@param access_code:
@param pad_for_usrp:
"""

if len(payload) > MAX_PKT_SIZE:
    raise ValueError, "len(payload) must be in [0, %d]" %(MAX_PKT_SIZE)

header = ''.join((chr(dest&0xFF), chr((dest >> 8) & 0xFF),
chr(source&0xFF), chr((source >> 8) & 0xFF), chr((len(payload) & 0xFF)),
chr(group&0xFF), chr(am_type&0xFF)))

crcClass = crc8.crc8()
crc = struct.pack('H', crcClass.crc(header[0:]+payload))

# create the packet with the synchronization header of 100x '10' in front.
pkt = ''.join((20*struct.pack('B', 0xaa), access_code, header, payload,
crc, 20*struct.pack('B', 0xaa)))

return pkt

class cclk_mod_pkts(gr.hier_block):
    """
    CC1K modulator that is a GNU Radio source.

    Send packets by calling send_pkt
    """
    def __init__(self, fg, access_code=None, msgq_limit=2, pad_for_usrp=True,
*args, **kwargs):
        """
        Hierarchical block for the CC1K fsk modulation.

        Packets to be sent are enqueued by calling send_pkt.
        The output is the complex modulated signal at baseband.

        @param fg: flow graph
        @type fg: flow graph
        @param access_code: 64-bit sync code
        @type access_code: string of length 8
        @param msgq_limit: maximum number of messages in message queue
        @type msgq_limit: int
        @param pad_for_usrp: If true, packets are padded such that they end
up a multiple of 128 samples

        See cclk_mod for remaining parameters
        """
        self.pad_for_usrp = pad_for_usrp
        if access_code is None:
            #this is 0x33CC
            access_code = struct.pack('BB', 0x33, 0xCC)

        #if not isinstance(access_code, str) or len(access_code) != 8:

```

```

#     raise ValueError, "Invalid access_code '%r'" % (access_code,)
self._access_code = access_code

# accepts messages from the outside world
self.pkt_input = gr.message_source(gr.sizeof_char, msgq_limit)
self.cclk_mod = cclk.cclk_mod(fg, *args, **kwargs)
fg.connect(self.pkt_input, self.cclk_mod)
gr.hier_block.__init__(self, fg, None, self.cclk_mod)

def send_pkt(self, dest, source, group, am_type, payload='', eof=False):
    """
    Send the payload.

    @param payload: data to send
    @type payload: string
    """
    if eof:
        msg = gr.message(1) # tell self.pkt_input we're not sending any
more packets
    else:
        pkt = make_tos_packet(dest,
                               source,
                               group,
                               am_type,
                               payload,
                               self.cclk_mod.spb,
                               self._access_code,
                               self.pad_for_usrp)
        msg = gr.message_from_string(pkt)
        self.pkt_input.msgq().insert_tail(msg)

class cclk_demod_pkts(gr.hier_block):
    """
    cclk demodulator that is a GNU Radio sink.

    The input is complex baseband.  When packets are demodulated, they are
passed to the
    app via the callback.
    """

    def __init__(self, fg, access_code=None, callback=None, threshold=-1,
*args, **kwargs):
        """
        Hierarchical block for binary FSK demodulation.

        The input is the complex modulated signal at baseband.
        Demodulated packets are sent to the handler.

        @param fg: flow graph
        @type fg: flow graph
        @param access_code: 64-bit sync code
        @type access_code: string of length 8
        @param callback: function of two args: ok, payload
        @type callback: ok: bool; payload: string
        @param threshold: detect access_code with up to threshold bits wrong
(-1 -> use default)

```

```

    @type threshold: int

    See cclk_demod for remaining parameters.
    """

    if access_code is None:
        #this is 0x999999995a5aa5a5
        access_code = chr(153) + chr(153) + chr(153) + chr(153) + chr(90)
+ chr(90) + chr(165) + chr(165)
    if not isinstance(access_code, str) or len(access_code) != 8:
        raise ValueError, "Invalid access_code '%r' len '%r'" %
(access_code, len(access_code),)
    self._access_code = access_code

    self._rcvd_pktq = gr.msg_queue()          # holds packets from the
PHY
    self.cclk_demod = cclk.cclk_demod(fg, *args, **kwargs)
    self._packet_sink = ucla.sos_packet_sink(map(ord, access_code),
self._rcvd_pktq, threshold)

    fg.connect(self.cclk_demod, self._packet_sink)

    gr.hier_block.__init__(self, fg, self.cclk_demod, None)
    self._watcher = _queue_watcher_thread(self._rcvd_pktq, callback)

def carrier_sensed(self):
    """
    Return True if we detect carrier.
    """
    return self._packet_sink.carrier_sensed()

class _queue_watcher_thread(_threading.Thread):
    def __init__(self, rcvd_pktq, callback):
        _threading.Thread.__init__(self)
        self.setDaemon(1)
        self.rcvd_pktq = rcvd_pktq
        self.callback = callback
        self.keep_running = True
        self.start()

    def run(self):
        while self.keep_running:
            msg = self.rcvd_pktq.delete_head()
            ok = 1
            payload = msg.to_string()

            dest = ord(payload[0])*256 + ord(payload[1])
            source = ord(payload[2])*256 + ord(payload[3])
            length = ord(payload[4])
            group = ord(payload[5])
            am_type = ord(payload[6])
            msg_payload = payload[7:7+length]
            crc = ord(payload[-1])

            crcClass = crc8.crc8()

```

```
        crcCheck = crcClass.crc(payload[0:7+length]) & 0xFF
        ok = (crc == crcCheck)
        if self.callback:
            self.callback(ok, dest, source, group, am_type, msg_payload,
crc)
```