Degree's Final Project

Bachelor's Degree in Industrial Technology Engineering

Development of a Python application for monitoring RF messages using one NRF24L01 board and a USB-MPSSE cable

ANNEX A: **Application's code**
ANNEX B: **first_level_class.py documentation screenshots**
ANNEX C: **Packet sending - Manual mode code**
ANNEX D: **Packet sending – Automatic/Multireceiving mode code**
ANNEX E: **File sending code**
ANNEX F: **Keyboard sending code**
ANNEX G: **CRC full calculation**

Author: Joan Ràfols Bellés
Directors/s: Juan Manuel Moreno Eguilaz

Call: June 2016

School of Industrial Engineering of Barcelona

Annex A: Application's code

A.1 first_level_class.py

```
"""
mpsse - python library for FTDI mpsse mode
verson 1.0
2013/3/29
"""

import d2xx
import time

# SPI mode
MODE0 = 0  # data captured on the clock's rising edge and propagated on a falling edge
MODE1 = 1  # data captured on the clock's falling edge and propagated on a rising edge
MODE2 = 2  # data captured on the clock's falling edge and propagated on a rising edge
MODE3 = 3  # data captured on the clock's rising edge and propagated on a falling edge

# endianess:
MSB = 0
LSB = 8

# pins:
SK = 1
DO = 2
DI = 4
CS = 8
GPIOL0 = 16
GPIOL1 = 32
GPIOL2 = 64
GPIOL3 = 128
GPIOH0 = 256 + 1
GPIOH1 = 256 + 2
GPIOH2 = 256 + 4
GPIOH3 = 256 + 8
GPIOH4 = 256 + 16
GPIOH5 = 256 + 32
GPIOH6 = 256 + 64
GPIOH7 = 256 + 128

# frequency
ONE_HUNDRED_KHZ = 100000
FOUR_HUNDRED_KHZ = 400000
ONE_MHZ = 1000000
TWO_MHZ = 2000000
THREE_MHZ = 3000000
FIVE_MHZ = 5000000
```

```
SIX_MHZ = 6000000
TEN_MHZ = 10000000
TWELVE_MHZ = 12000000
FIFTEEN_MHZ = 15000000
TWENTY_MHZ = 20000000
THIRTY_MHZ = 30000000

_negwrite = 0x01
_negread = 0x04

_dowrite = 0x10
_doread = 0x20

_bitmode = 2

_low = 0
_high = 1

_input = 0
_output = 1


class ftdi(object):


    def __init__(self):
        """no parameters are necessary"""
        self.d2xx = d2xx.open(0)
        self.info = []
        self._port = {'high': 0, 'low': 0}
        self._dir = {'high': 0, 'low': 0}

    def open(self, description):
        """
        :param str description: ftdi's device name.
        :return: 'OK' if the open function was successful, otherwise return 'no device was
found'.
        Looks for the device. If it is not found returns None and prints "no device was found".
        If the device is found, it configures it.
        """
        device_not_found = True
        n = d2xx.createDeviceInfoList()

        for i in range(n):
            self.info.append(d2xx.getDeviceInfoDetail(i))
            if self.info[i]['description'] == description:
                device_not_found = False
        self.d2xx.resetDevice()
        self.d2xx.setLatencyTimer(1)  # 1ms
        if n == 0 or device_not_found or not self.setmpsse():
```

```
            return "no device was found"

        if (self.d2xx.getDeviceInfo())['type'] >= 6:
            self.write('\x8A')  # disable div by 5
        return 'OK'

    def setmpsse(self):
        """
        :return: True if the communication works, otherwise returns False.
        sets mpsse bit mode and checks if it works by sending a wrong command and expecting a
wrong command answer."""
        self.d2xx.setBitMode(0, 2)  # d2xx.BITMODE_MPSSE) #all gpio pin input
        self.d2xx.write('\xAA')
        time.sleep(0.5)  # wait for a while
    #   self.d2xx.getQueueStatus()
        if self.d2xx.getQueueStatus() != 2:  # res "\xFA\xAA"
            return False

        x = self.d2xx.read(2)
        if x != "\xFA\xAA":
            return False

        return True

    def get_device_info(self):
        """prints information of every ftdi device connected."""
        print self.info

    def buildgpiocmd(self):
        """
        :return: built gpio command.
        :rtype: bytes string
        Builds gpio commands to be sent to the ftdi cable."""
        raw = (0x82, self._port['high'], self._dir['high'], 0x80, self._port['low'], self._dir['low'])
        return ''.join(chr(i) for i in raw)

    def gpio_commit(self):
        """Sends buildgpiocmd"""
        self.write(self.buildgpiocmd())

    def gpio_setbit(self, pin, bit, dirset):
        """
        :param int pin: can be chosen from the pin variables defined at the beginning of the file.
        :param int bit: from the variables defined above, _high or _low.
        :param int dirset: from the variables defined above, _output or _input.
        sets gpio cable connection on a high or low state.
        """
        if pin > 256:
            pin -= 256
            self._port['high'] = self._port['high'] | pin if bit else self._port['high'] & (~pin)
```

```
            self._dir['high'] = self._dir['high'] | pin if dirset else self._dir['high'] & (~pin)
        else:
            self._port['low'] = self._port['low'] | pin if bit else self._port['low'] & (~pin)
            self._dir['low'] = self._dir['low'] | pin if dirset else self._dir['low'] & (~pin)

    def setfrequency(self, freq):
        """Sets ftdi cable frequency."""
        maxfreq = 30000000 if (self.d2xx.getDeviceInfo())['type'] >= 6 else 6000000
        div = maxfreq / freq - 1
        low = div & 0xFF
        high = (div >> 8) & 0xFF
        self.write(''.join(('\x86', chr(low), chr(high))))  # set clk divisor

    def write(self, data):
        """Sends data to the ftdi cable."""
        self.d2xx.write(data)

    def read(self, size):
        """
        :param int size: 1 byte=0 ; 2bytes=1 ; 3 bytes=2 ; ...
        :return: incoming data.
        :rtype: str (bytes' string).
        Returns incoming data from the ftdi cable."""
        return self.d2xx.read(size)

    def clearBuffer(self):
        """It reads everything in the buffer so that there is nothing left."""
        self.d2xx.read(self.d2xx.getQueueStatus())


class SPI(object):
    def __init__(self, name, mode, frequency, endianness=MSB, cs=CS):
        """Initializes SPI configuration."""
        self._name = name
        self.mode = mode
        self.freq = frequency
        self.endia = endianness
        self.cs = cs
        self.ftdi = ftdi()
        self.setupmode()

        if self.ftdi.open(self._name) != 'OK':
            raise Exception('No FTDI device was found!')

        # init GPIO
        self.ftdi.gpio_setbit(cs, _high, _output)
        self.ftdi.gpio_setbit(SK, self.skidle, _output)
        self.ftdi.gpio_setbit(DO, _low, _output)
        self.ftdi.gpio_setbit(DI, _low, _input)
        self.ftdi.gpio_commit()
```

```
        self.ftdi.setfrequency(frequency)

    def setupmode(self):
        """Configures when data is captured and propagated."""

        if self.mode == MODE0:
            self.skidle = _low
            self.sksetup = _low
            self.txcmd = _dowrite | self.endia | _negwrite
            self.rxcmd = _doread | self.endia
            self.txrxcmd = _dowrite | _doread | self.endia | _negwrite

        elif self.mode == MODE1:
            self.skidle = _low
            self.sksetup = _high
            self.txcmd = _dowrite | self.endia
            self.rxcmd = _doread | self.endia | _negread
            self.txrxcmd = _dowrite | _doread | self.endia | _negread

        elif self.mode == MODE2:
            self.skidle = _high
            self.sksetup = _high
            self.txcmd = _dowrite | self.endia
            self.rxcmd = _doread | self.endia | _negread
            self.txrxcmd = _dowrite | _doread | self.endia | _negread

        elif self.mode == MODE3:
            self.skidle = _high
            self.sksetup = _low
            self.txcmd = _dowrite | self.endia | _negwrite
            self.rxcmd = _doread | self.endia
            self.txrxcmd = _dowrite | _doread | self.endia | _negwrite
        else:
            raise Exception('Incorrect mode')

    def startcmd(self):
        """
        :return: built init gpio command.
        :rtype: bytes string.
        Initialize SK."""
        self.ftdi.gpio_setbit(SK, self.sksetup, _output)
        return self.ftdi.buildgpiocmd()

    def endcmd(self):
        """
        :return: built end gpio command.
        :rtype: bytes string.
        Ends SK."""
        self.ftdi.gpio_setbit(SK, self.skidle, _output)
        return self.ftdi.buildgpiocmd()
```

```
    def write(self, data):
        """
        :param str data: supports bytes' string **only**.
        One byte through the TDO cable connection from the ftdi cable.

        .. note:: This function doesn't convert data into a bytes' string because it would
complicate
         the code unnecessarily as some data will be already converted.
        """
        xx = len(data) - 1
        cmd = chr(self.txcmd) + chr(xx & 0xFF) + chr((xx >> 8) & 0xFF)
        self.ftdi.write(self.startcmd() + cmd + data + self.endcmd())

    def read(self):
        """Reads one byte through the TDI cable connection from the ftdi cable"""
        self.ftdi.clearBuffer()
        size = 0x0000
        readcmd = chr(self.rxcmd) + chr(size & 0xFF) + chr((size >> 8) & 0xFF)
        self.ftdi.write(self.startcmd() + readcmd + self.endcmd() + '\x87')
        return self.ftdi.read(1)

    def setgpio(self, gpio, bit, direction):
        """
        :param int pin: can be chosen from the pin variables defined at the beginning of the file.
        :param int bit: from the variables defined above, _high or _low
        :param int direction: from the variables defined above, _output or _input
        Sets high or low state through gpio cable connection from the ftdi cable"""
        self.ftdi.gpio_setbit(gpio, bit, direction)
        self.ftdi.gpio_commit()
```

A.2 second_level_class.py

```
"""NRF24L01 driver for Micro Python
"""

from first_level_class import *

# nRF24L01+ registers
CONFIG = 0x00  # General Configuration Register
EN_AA = 0x01  # Enable Auto Acknowledgment Function
EN_RXADDR = 0x02  # Enabled RX Addresses
SETUP_AW = 0x03  # Setup of Address Widths
SETUP_RETR = 0x04  # Setup of Automatic Retransmission
RF_CH = 0x05  # RF Channel
RF_SETUP = 0x06  # RF Setup Register
STATUS = 0x07  # nRF24L01 STATUS REGISTER
OBSERVE_TX = 0x08  # Transmit observe register
CD = 0x09  # Carrier Detect register
DYNPD = 0x1C  # Enable dynamic payload length
FEATURE = 0x1D  # Feature Register

RX_ADDR_P0 = 0x0A  # Receive Address for Data Pipe 0  (3, 4 or 5 byte lenght)
# (used in RX Mode and TX Mode as Auto Acknowledgment Receiving Channel)
RX_ADDR_P1 = 0x0B  # Receive Address for Data Pipe 1  (3, 4 or 5 byte lenght) (used in
RX Mode)
RX_ADDR_P2 = 0x0C  # Receive Address for Data Pipe 2  (1 byte) (used in RX Mode)
RX_ADDR_P3 = 0x0D  # Receive Address for Data Pipe 3  (1 byte) (used in RX Mode)
RX_ADDR_P4 = 0x0E  # Receive Address for Data Pipe 4  (1 byte) (used in RX Mode)
RX_ADDR_P5 = 0x0F  # Receive Address for Data Pipe 5  (1 byte) (used in RX Mode)
TX_ADDR = 0x10  # Transmit Address   (3, 4 or 5 byte lenght) (used in TX Mode)

RX_PW_P0 = 0x11  # RX FIFO Memory Width reserved for Pipe 0
RX_PW_P1 = 0x12  # RX FIFO Memory Width reserved for Pipe 1
RX_PW_P2 = 0x13  # RX FIFO Memory Width reserved for Pipe 2
RX_PW_P3 = 0x14  # RX FIFO Memory Width reserved for Pipe 3
RX_PW_P4 = 0x15  # RX FIFO Memory Width reserved for Pipe 4
RX_PW_P5 = 0x16  # RX FIFO Memory Width reserved for Pipe 5
FIFO_STATUS = 0x17  # FIFO Status Register

# CONFIG register
EN_CRC = 0x08  # enable CRC
CRCO = 0x04  # CRC encoding scheme; 0=1 byte, 1=2 bytes
PWR_UP = 0x02  # 1=power up, 0=power down
PRIM_RX = 0x01  # RX/TX control; 0=PTX, 1=PRX

# RF_SETUP register
POWER_0 = 0x00  # -18 dBm
POWER_1 = 0x02  # -12 dBm
POWER_2 = 0x04  # -6 dBm
POWER_3 = 0x06  # 0 dBm
```

```
SPEED_1M = 0x00
SPEED_2M = 0x08
SPEED_250K = 0x20


# FIFO_STATUS register
RX_EMPTY = 0x01  # 1 if RX FIFO is empty

# constants for instructions
R_RX_PL_WID = 0x60   # read RX payload width
R_RX_PAYLOAD = 0x61  # read RX payload
W_TX_PAYLOAD = 0xa0  # write TX payload
FLUSH_TX = 0xe1      # flush TX FIFO
FLUSH_RX = 0xe2      # flush RX FIFO
NOP = 0xff           # use to read STATUS register

#  nRF24L01 STATUS Register BIT Definition
#Name              Bit          Description
RX_DR    =  0b01000000     # Data Ready RX FIFO interrupt => Set high when new data arrives (Write 1 to clear bit)
TX_DS    =  0b00100000     # Data Sent TX FIFO interrupt => Set high when packet sent on TX (Write 1 to clear bit)
MAX_RT   =  0b00010000     # Maximum number of TX retries interrupt (Write 1 to clear bit)
RX_P_NO  =  0b00001110     # Data Pipe Number of the Payload Available for reading in the RX FIFO
PRIM_RX  =  0b00000001     # TX FIFO full flag

# Direction
_input = 0
_output = 1


# ------------------------------------------------------------------------------


class NRF24L01(object):
    """
    :param int cs: The pin in wich the Chip Select is located.
    :param int ce: The pin in wich the Chip Enable is located.
    :param int channel:
    :param int payload_size:
    Pins are stored and reset, SPI Class is initialized, reg_read and reg_write functions are checked,
    communication parameters are set and buffers are flushed.
    .. note:: If reg_read and/or reg_write don't work properly, an error is raised.
    """
    def __init__(self, cs, ce, channel=64, payload_size=16):

        assert payload_size <= 32
```

```
        # store the pins
        self.spi = SPI('C232HM-DDHSL-0',MODE0, ONE_HUNDRED_KHZ)
        self.cs = cs
        self.ce = ce

        # reset everything
        self.low(self.ce)
        self.high(self.cs)
        self.payload_size = payload_size
        self.pipe0_read_addr = None
        self.reg_write(0x03, 0b10)
        if self.reg_read(SETUP_AW) != 0b10:
            pass
            raise OSError("nRF24L01+ Hardware not responding")
        else:
            print("nRF24L01' has been identified successfully")


        # disable dynamic payloads
        self.reg_write(DYNPD, 0x00)

        # auto retransmit delay: 1750us
        # auto retransmit count: 8
        self.reg_write(SETUP_RETR, (6 << 4) | 8)

        # set rf power and speed
        self.set_power_speed(POWER_3, SPEED_1M)  # Best for point to point links

        # init CRC
        self.set_crc(2)

        # clear status flags
        self.reg_write(STATUS, RX_DR | TX_DS | MAX_RT)

        # set channel
        self.set_channel(channel)

        # flush buffers
        self.flush_rx()
        self.flush_tx()

    def low(self, pin):
        """
        :param int pin:
        Sets a pin in the low value"""
        self.spi.setgpio(pin, 0, _output)

    def high(self, pin):
        """
```

```
        :param int pin:
        Sets a pin in the high value"""
        self.spi.setgpio(pin, 1, _output)

    def reg_write(self, reg, buf):
        """
        :param int reg: The register where data must be written
        .. warning:: **reg** variable must be a an integer **only**.
        :param (int/bytes' string) buf: Data to be written in the register reg.
        Writes data (buf) in the register (reg).
        """
        self.low(self.cs)
        self.spi.write(chr(0x20 | reg))
        if isinstance(buf,(int,long)):
            buf = chr(buf)
            self.spi.write(buf)
        else:
            self.spi.write(buf)
        self.high(self.cs)


    def reg_read(self, reg):
        """
        :param int reg: The register where data must be written
        .. warning:: **reg** variable must be a an integer **only**.
        :return: Data read from reg.
        Reads data from the register (reg).
        """
        self.low(self.cs)
        self.spi.write(chr(reg))
        buf = ord(self.spi.read())
        self.high(self.cs)
        return buf


    def flush_rx(self):
        """Flushes RX buffer"""
        self.low(self.cs)
        self.spi.write(chr(FLUSH_RX))
        self.high(self.cs)
        time.sleep(0.000001)

    def flush_tx(self):
        """Flushes TX buffer"""
        self.low(self.cs)
        self.spi.write(chr(FLUSH_TX))
        self.high(self.cs)
        time.sleep(0.000001)

    def set_power_speed(self, power, speed):
```

```
        """
        :param int power: from the variable above (RF_SETUP register).
        :param int speed: from the variable above (RF_SETUP register).
        Sets power and speed"""
        setup = self.reg_read(RF_SETUP) & 0b11010001
        self.reg_write(RF_SETUP, setup | power | speed)

    def set_crc(self, length):
        """
        :param int length: 0(no CRC), 1 or 2.
        Sets CRC (verification code).
        """
        config = self.reg_read(CONFIG) & ~(CRCO | EN_CRC)
        if length == 0:
            pass
        elif length == 1:
            config |= EN_CRC
        else:
            config |= EN_CRC | CRCO
        self.reg_write(CONFIG, config)
        print self.reg_read(CONFIG)

    def set_channel(self, channel):
        """
        :param int channel: from 0 to 125
        Sets channel according to the formula F = 2400+RF_CH[MHz]
        """
        self.reg_write(RF_CH, min(channel, 125))
        # address should be a bytes object 5 bytes long

    def open_tx_pipe(self, address):
        """
        .. warning:: Adress must be from 3 to 5 bytes long bytes object.
        Opens TX pipe"""
        self.reg_write(RX_ADDR_P0, address)
        self.reg_write(TX_ADDR, address)
        self.reg_write(RX_PW_P0, self.payload_size)

    # address should be a bytes object 5 bytes long

    def open_rx_pipe(self, pipe_id, address):
        """
        :param int pipe_id: from 0 to 5.
        :param bytes address: Variables defined above.
        .. warning:: pipe 0 and 1 have a 5 bytes address.
        .. note:: pipes 2-5 use same 4 most-significant bytes as pipe 1, plus 1 extra byte.
        Enables the pipe and set address to the pipe (note that if there are other pipes enabled, it
keeps them enabled).
        """
        assert 0 <= pipe_id <= 5
```

```
        if pipe_id == 0:
            self.pipe0_read_addr = address
        if pipe_id < 2:
            self.reg_write(RX_ADDR_P0 + pipe_id, address)
        else:
            self.reg_write(RX_ADDR_P0 + pipe_id, address[0])
        self.reg_write(RX_PW_P0 + pipe_id, self.payload_size)
        self.reg_write(EN_RXADDR, (1 << pipe_id))
        self.reg_write(EN_AA, 1 << pipe_id)

    def start_listening(self):
        """
        It sets the nRF24L01 in a listening state until stop_listening function is called.
        """
        self.reg_write(CONFIG, self.reg_read(CONFIG) | PWR_UP | PRIM_RX)
        self.reg_write(STATUS, RX_DR | TX_DS | MAX_RT)
        self.flush_rx()
        self.flush_tx()
        self.high(self.ce)
        time.sleep(0.000130)
        #nuevo
        p=[]
        self.low(self.cs)
        self.spi.write(chr(RX_ADDR_P0))
        for _ in range (5):
            p.append(self.spi.read())
        self.high(self.cs)
        print p

        z=[15,63,63,2,3,64,7,14,0,0,178,17,0,0,0,0,231,9,9,1,1,1,1,17,0,0,0,0,0,0]

        f=[15, 63, 3, 3, 104, 64, 7, 14, 0, 0, 136, 194, 195, 196, 197, 198, 231, 5, 0, 0, 0, 0, 0, 17,
0, 0, 0, 0, 0, 0]
        for i in range(30):
            print i, hex(self.reg_read(i))

    def stop_listening(self):
        """
        Stops listening by setting CE low and returns to the standby mode. It also flushes both
FIFOs as well.
        """
        self.low(self.ce)
        self.reg_write(CONFIG, self.reg_read(CONFIG) & ~PWR_UP)
        self.flush_tx()
        self.flush_rx()
    def any(self):
        """
        :return: True if any data is available to be received.
        """
        return not bool(self.reg_read(FIFO_STATUS) & RX_EMPTY)
```

```
    def recv(self):
        """
        When a packet has been received this method reads the message which has been saved in
the RX FIFO.
        :return: the data available.
        :rtype: list of integers"""
        list=[]
        self.low(self.cs)
        self.spi.write(chr(R_RX_PAYLOAD))
        for i in range(self.payload_size):
            list.append(int('0x'+self.spi.read().encode('HEX'),16))
        self.high(self.cs)
        # clear RX ready flag
        self.reg_write(STATUS, RX_DR)
        return list


    # non-blocking tx
    def start_sending(self, msg):
        """
        :param list of int s: message to send
        When a packet has been received this method reads the message that has been saved in
the RX FIFO.
        """

        self.reg_write(CONFIG, (self.reg_read(CONFIG) | PWR_UP) & ~PRIM_RX)
        time.sleep(0.000150)
        self.low(self.cs)
        self.spi.write(chr(16))
        p=[]
        for _ in range (5):
            p.append(self.spi.read())
        self.high(self.cs)
        print p
        # send the data
        self.low(self.cs)
        self.spi.write(chr(W_TX_PAYLOAD))
        for i in msg:
            self.spi.write(chr(int(i,16)))
        self.high(self.cs)

        # enable the chip so it can send the data
        self.high(self.ce)
        time.sleep(0.5)
        time.sleep(0.000015)  # needs to be >10us
        self.low(self.ce)
        time.sleep(0.002)
        #self.flush_tx()
```

A.3 app.py

```
from second_level_class import *
from manual_mode import *
from automatic_mode import *
from multireceiving_mode import *
from CRC import crc_
import sys
from bitstring import BitArray

state = 'Disconnected'

first_exec = True


class GeneralModeFunctions(QtGui.QMainWindow):
    def __init__(self, mode, txt_title):
        super(GeneralModeFunctions, self).__init__()
        self.txt_title = txt_title
        self.islistening = False
        self.parameters_applied = False
        self.record = ''
        self.filerecord = ''
        self.path = ''
        self.filemode = False
        self.focused = False
        self.param = {'Speed': '', 'Power': '', 'Channel': '', 'Payload size': '', 'CRC': '', 'TX
Address': '',
                      'RX Address': '', 'RX Pipe id': ''}
        self.trans_p_s = {'-18 dBm': 0, '-12 dBm': 2, '-6 dBm': 4, '0 dBm': 6, '1 Mbps': 0, '2
Mbps': 8,
                          '250 Kbps': 0x20}
        self.listen = ListeningLoop()
        self.move_window = Move()
        self.getconnection = GetConnection()
        self.checkconnection = CheckConnection()
        self.app = mode
        self.app.setupUi(self)
        self.app.listen.clicked.connect(self.listening)
        self.app.stopB.clicked.connect(self.stop_listening)
        self.app.applychangesB.clicked.connect(self.apply_new_parameters)
        self.app.movingB.pressed.connect(self.move_window.start)
        self.app.pushButton_3.clicked.connect(lambda: app.quit())
        self.app.pushButton_4.clicked.connect(lambda: myapp.showMinimized())
        self.app.ex2.clicked.connect(lambda: self.change_mode("left"))
        self.app.ex3.clicked.connect(lambda: self.change_mode("right"))
        self.connect(self.getconnection, QtCore.SIGNAL('connection()'),
self.end_Get_CheckConnection)
        self.connect(self.checkconnection, QtCore.SIGNAL('connection()'),
self.end_Get_CheckConnection)
```

```
        self.connect(self.listen, QtCore.SIGNAL('unexpected_disconnection()'),
                     lambda: QtGui.QMessageBox.warning(self, 'Error!', 'An unexpected
disconnection occurred'))
        self.connect(self.listen, QtCore.SIGNAL('data_received()'), self.read)
        self.app.statusBar.showMessage(state)
        if state == 'Connected':
            self.app.statusBar.setStyleSheet("color: rgb(0, 255, 20);")
        if state == 'Disconnected':
            self.getconnection.start()
        else:
            self.checkconnection.start()

    def listening(self):
        """
        It is triggered when clicking the *Listen* button. This function is a very important one
not only because it
        sets the NRF24L01 in a listening state but because in order to do that, it has to call
another class which is a
        thread. The reason of being of this Thread is the need of a loop when listening. If the
loop was written in this
        function, the program would get stuck and would not respond. Therefore, the loop must
be executed in another
        class that runs simultaneously together with one of these three main classes. As it was
mentioned in the
        ``apply_parameters()``, if the method fails execution, a message will let the user know
whether it is due tu a loss
        of connection or because the parameters have not been applied yet.
        """
        if self.parameters_applied:
            try:
                nrf.start_listening()
                self.add_to_record('NRF24L01 listening...')
                self.listen.start()
                self.islistening = True
            except:
                global state
                state = 'Disconnected'
                myapp.app.statusBar.showMessage(state)
                myapp.getconnection.start()
                QtGui.QMessageBox.warning(self, "Error!", 'Device couldn\'t be found')
        else:
            QtGui.QMessageBox.warning(self, "Error!", 'You should apply your parameters
first')

    def stop_listening(self):
        """
        It runs when the *Stop* button is clicked. It makes the NRF24L01 stop listening. A
message confirming the
        execution of this method is shown on the browser. As always, if the method fails it will
display a message box
```

```
        saying that the device was not found (as the error will always be due to a loss of
connection).
        """
        self.focused = False
        try:
            self.listen.terminate()
            nrf.stop_listening()
            self.add_to_record('NRF24L01 stopped listening')
            self.islistening = False
        except:
            global state
            state = 'Disconnected'
            self.app.statusBar.showMessage(state)
            self.getconnection.start()
            QtGui.QMessageBox.warning(self, "Error!", 'Device couldn\'t be found')

    def end_Get_CheckConnection(self):
        """
        It executes when *connection()* from ``GetConnection`` class finds the device. It
updates the
        status bar, terminates the thread and starts ``CheckConnection`` class
        """
        myapp.app.statusBar.showMessage(state)
        if state == 'Connected':
            self.getconnection.terminate()
            self.app.statusBar.setStyleSheet("color: rgb(0, 255, 20);")
            self.add_to_record('The device has been successfully connected')
            self.checkconnection.start()
        else:
            self.getconnection.terminate()
            self.app.statusBar.setStyleSheet("color: rgb(0, 0, 0);")
            self.getconnection.start()

    def add_to_record(self, s):
        """
        :param str s: information to display
        Normally, this method displays information to the user via the text browser and saves
the displayed information in a text file
        named *Manual mode.txt*. Nevertheless if focus feature (from Automatic mode) is
activated, concatenates the messages without
        line breaks. If path is not a null string (meaning there is an incoming text file), the
messages are saved in a
        variable

        """
        if not self.filemode:
            if self.focused == False:
                self.record += s + '\n'
            else:
                self.record = self.record[:-1]+ s
```

```
            myapp.app.textBrowser.setText(self.record)

self.app.textBrowser.verticalScrollBar().setValue(self.app.textBrowser.verticalScrollBar().m
aximum())
            file = open(self.txt_title, 'w')
            file.write(self.record)
            file.close()
        else:

            if s == 'end  file\n':
                print 'ya ha acabado'
                self.record = self.record[:-5]+ '100%]\nFile downloaded\nListening...\n'
                myapp.app.textBrowser.setText(self.record)

self.app.textBrowser.verticalScrollBar().setValue(self.app.textBrowser.verticalScrollBar().m
aximum())
                self.path = QtGui.QFileDialog.getSaveFileName(caption='Save file', filter='Text
file (*.txt)')
                print 'path: '+self.path
                file = open(self.path, 'w')
                file.write(self.filerecord)
                file.close()
                self.filerecord=''
                self.listen.terminate()
                nrf.stop_listening()
                self.islistening= False
                self.listofthreads = [packet_validation(BitArray('')) for _ in range(10)]
                self.apply_new_parameters()
                self.filemode = False
                self.last_pid = ''
                self.last_crc = ''
                self.listen.start()
                nrf.start_listening()


            else:
                self.record = self.record[:-4]+str(100*len(self.filerecord)/self.file_size)+'%]'
                myapp.app.textBrowser.setText(self.record)

self.app.textBrowser.verticalScrollBar().setValue(self.app.textBrowser.verticalScrollBar().m
aximum())
                self.filerecord = self.filerecord[:-1]+ s
                print self.filerecord

    def change_mode(self, button):
        """
        :param button: the button that was clicked (left; right)
        This method is executed when the mode buttons are clicked. It closes the actual window
and opens another one containing the selected mode.
        """
```

```
        global myapp, window_xpos, window_ypos
        self.listen.terminate()
        self.getconnection.terminate()
        self.checkconnection.terminate()
        myapp.hide()
        if button + self.txt_title[:2] == 'leftMa':
            myapp = AutomaticMode()
        elif button + self.txt_title[:2] == 'rightMa':
            myapp = MultireceivingMode()
        elif button + self.txt_title[:2] == 'leftAu':
            myapp = ManualMode()
        elif button + self.txt_title[:2] == 'rightAu':
            myapp = MultireceivingMode()
        elif button + self.txt_title[:2] == 'leftMu':
            myapp = ManualMode()
        elif button + self.txt_title[:2] == 'rightMu':
            myapp = AutomaticMode()
        myapp.move(window_xpos, window_ypos)
        myapp.show()


class ManualMode(GeneralModeFunctions):
    """
    This class calls the inherited class ``GeneralModeFunctions`` passing as a parameters the
    ``Manual`` class from which the components of the GUI will be taken and it runs
    its ``setupUI()`` method. This step applies for every "mode" class. From now on this class has access to the GUI components. Buttons' triggers and thread's
    signal triggers are established coupled with some variables that are necessary for the well-functioning of the app
    but lack importance.

    .. note:: Parameters are stored in a dictionary type variable which provide an easy access to all the variables at once coupled with an easy printing when informing the user. Once again, this applies for every "mode" class.
    """

    def __init__(self):
        super(ManualMode, self).__init__(Manual(), 'Manual_mode.txt')

        global first_exec
        self.app.send.clicked.connect(self.tx_message)
        if first_exec == True:
            self.intr = Intro()
            self.connect(self.intr, QtCore.SIGNAL('end_intro()'), self.app.intro.hide)
            self.intr.start()
            first_exec = False
        else:
            self.app.intro.hide()
```

```
    def apply_new_parameters(self):
        """
        It tries to apply the input parameters chosen by the user into the NRF24L01. If it fails a
message box will tell
        the user whether is it due to an unacceptable input or because the app lost connection
with the chip (if this
        is the case, ``GetConnection()`` thread will be executed waiting for the device to be
connected while status bar
        changes to "Device not found"). If it succeeds parameters will appear on the text
browser together with a
        message confirming the success. It also checks if the user has clicked on stop_listening
after clicking on
        listening. If not, ``stop_listening()`` method is executed.
        """

        if self.islistening:
            self.stop_listening()

        if state == 'Connected':
            try:
                self.param['Speed'] = myapp.app.speed.currentText()
                self.param['Power'] = myapp.app.power.currentText()
                self.param['Channel'] = myapp.app.channel.text()
                self.param['Payload size'] = myapp.app.payload_size.text()
                self.param['CRC'] = myapp.app.crc.currentText()
                self.param['TX Address'] = myapp.app.tx_adress.text()
                self.param['RX Address'] = myapp.app.rx_adress.text()
                self.param['RX Pipe id'] = myapp.app.rx_id.currentText()

nrf.set_power_speed(self.trans_p_s[str(self.app.power.currentText())],self.trans_p_s[str(self.
app.speed.currentText())])
                nrf.set_channel(int(self.param['Channel']))

                if 3 <= len(str(self.param['RX Address']).split(' ')) <= 5:
                    nrf.reg_write(DYNPD, 0)
                    nrf.reg_write(FEATURE, 0)
                    nrf.reg_write(SETUP_AW, len(self.param['RX Address'].split(' ')) - 2)
                    assert 1 <= int(self.param['Payload size']) <= 32
                    nrf.payload_size = int(self.param['Payload size'])
                    l = []
                    for i in str(self.param['RX Address']).split(' ')[::-1]:
                        l.append(int(i, base=16))
                    nrf.open_rx_pipe(int(self.param['RX Pipe id']), str(bytearray(l)))

                if 3 <= len(str(self.param['TX Address']).split(' ')) <= 5:
                    nrf.reg_write(SETUP_AW, len(self.param['TX Address'].split(' ')) - 2)
                    nrf.reg_write(DYNPD, 0b0)
                    nrf.reg_write(FEATURE, 0b0)
                    l = []
                    for i in str(self.param['TX Address']).split(' ')[::-1]:
```

```
                l.append(int(i, base=16))
            nrf.open_tx_pipe(str(bytearray(l)))
        if not 3 <= len(str(self.param['RX Address']).split(' ')) <= 5 and not 3 <= len(
            str(self.param['TX Address']).split(' ')) <= 5:
            QtGui.QMessageBox.warning(self, "Error!",'Incorrect TX or RX Address.
Remember that it must be between 3 and 5 bytes and written in hexadecimal format.
Use a space as a separator')

        nrf.set_crc(int(self.param['CRC']))
        nrf.reg_write(EN_AA, 1 << int(self.param['RX Pipe id']))

        nrf.reg_write(EN_RXADDR, 1 << int(self.param['RX Pipe id']))

        for h in self.param:
            self.add_to_record(h + ': ' + self.param[h])
        self.add_to_record('\n')
        self.parameters_applied = True
      except:
        QtGui.QMessageBox.warning(self, "Error!", 'All parameters must be properly
fulfilled')
    else:
      QtGui.QMessageBox.warning(self, "Error!", 'Device couldn\'t be found')


  def read(self):
    """
    This class reads the RX FIFO from the chip through the method recv() from the
``NRF24L01`` class inside
    second_level_class.py which returns a list of numbers which are converted into a string
of numbers before being
    sent as a parameter to ``add_to_record(s)`` method.
    """
    data = nrf.recv()
    self.add_to_record(' '.join(str(hex(e))[2:] for e in data))

  def tx_message(self):
    """
    Opens an input box where the user can enter the message that wants to send. When
clicking ok, ``sending()`` method is called.
    """
    if self.parameters_applied and self.param['TX Address'] != '':
      text, ok = QtGui.QInputDialog.getText(self, 'Message', 'Write your message:')

      if ok:
        msg = str(text).split(' ')
        self.sending(msg)
    else:
      QtGui.QMessageBox.warning(self, "Error!", 'You should apply your parameters
first')
```

```
    def sending(self, msg):
        """
        :param list msg: the message that wants to be sent.
        Sends the message and prints the sending state on the text browser.
        """

        if self.parameters_applied:

            try:

                self.add_to_record('NRF24L01 sending message...')

                nrf.start_sending(msg)

                self.add_to_record('The message was sent successfully')

            except:
                global state
                state = 'Disconnected'
                myapp.app.statusBar.showMessage(state)
                myapp.getconnection.start()
                QtGui.QMessageBox.warning(self, "Error!", 'Device couldn\'t be found')
        else:
            QtGui.QMessageBox.warning(self, "Error!", 'You should apply your parameters
first')


class AutomaticMode(GeneralModeFunctions):
    """
    It is important to draw attention to the list of threads that are initialised in the
constructor, as this class
    needs various threads running at the same time to process all the information with a
decent agility.
    """

    def __init__(self):
        super(AutomaticMode, self).__init__(Automatic(), 'Automatic_mode.txt')
        self.address = []
        self.last_pid = 0b1111
        self.last_crc = ''
        self.focused = False
        self.next_thread = 0
        self.app.focus.clicked.connect(self.focus)

    def focus(self):
        """
        This method sets focused attribute True if it was False and removes the break line that
``add_to_record()`` created
        """
        if self.focused == False:
```

```
            self.record = self.record[:-2]
            self.add_to_record('\n\nFocusing on address 0x'+str(self.data[:self.address_length
* 8].hex)+':\n')
            self.focused = True
            self.fast_reception(len(self.crc) / 8, str(self.data[:self.address_length * 8].hex))
        else:
            self.add_to_record('\n\nFocus disabled\n')
            self.listen.terminate()
            nrf.stop_listening()
            self.islistening= False
            self.listofthreads = [packet_validation(BitArray('')) for _ in range(10)]
            self.apply_new_parameters()
            self.focused = False
            self.listen.start()
            nrf.start_listening()


    def apply_new_parameters(self):
        """
        It configures the device according to this mode.
        """
        self.listofthreads = [packet_validation(BitArray('')) for _ in range(10)]
        if self.islistening:
            self.stop_listening()

        if state == 'Connected':
            try:

                self.param['Speed'] = myapp.app.speed.currentText()
                self.param['Power'] = myapp.app.power.currentText()
                self.param['Channel'] = myapp.app.channel.text()
                self.param['Payload size'] = '32'
                self.param['CRC'] = '0'
                self.param['TX Address'] = myapp.app.tx_adress.text()
                self.param['RX Address'] = myapp.app.rx_adress.text()
                self.param['RX Pipe id'] = myapp.app.rx_id.currentText()

                assert 3 <= len(self.param['RX Address'].split(' ')) <= 5

                nrf.reg_write(SETUP_AW, len(self.param['RX Address'].split(' ')) - 2)
                nrf.reg_write(CONFIG, 0)
                nrf.reg_write(DYNPD, 0)
                nrf.reg_write(FEATURE, 0)  # Cambio
                nrf.set_power_speed(self.trans_p_s[str(self.app.power.currentText())],
                                    self.trans_p_s[str(self.app.speed.currentText())])
                nrf.set_channel(int(self.param['Channel']))
                nrf.payload_size = int(self.param['Payload size'])
                nrf.set_crc(int(self.param['CRC']))
                self.address = []
                for i in str(self.param['RX Address']).split(' ')[::-1]:
```

```
                self.address.append(int(i, base=16))
            nrf.open_rx_pipe(0, str(bytearray(self.address)))
            nrf.reg_write(EN_AA, 0)  # Notice that nrf.open_rx_pipe() modifies the EN_AA
register.
            if not self.filemode and not self.focused:
                for h in self.param:
                    self.add_to_record(h + ': ' + self.param[h])
                self.add_to_record('\n')
            self.parameters_applied = True
        except:
            QtGui.QMessageBox.warning(self, "Error!", 'All parameters must be properly
fulfilled')
    else:
        QtGui.QMessageBox.warning(self, "Error!", 'Device couldn\'t be found')

def fast_reception(self, crc, address):
    """
    :param BitArray crc:
    :param BitArray address:
    Having identified all the parameters from the transmitters configuration. This method
changes the mode
    adapting it to receive and process the packages right from the NRF24L01 board.
    """
    self.listofthreads=[]
    self.listen.terminate()
    nrf.stop_listening()
    nrf.reg_write(DYNPD, 0b111111)
    nrf.reg_write(FEATURE, 0b100)
    nrf.reg_write(EN_AA, 0b1)  # distinto
    nrf.set_crc(crc)
    nrf.reg_write(SETUP_AW,len(address) / 2-2)
    a = []
    l = []
    for i in range(len(address) / 2):
        a.append(address[i * 2:i * 2 + 2])
    for i in a[::-1]:
        l.append(int(i, base=16))
    nrf.open_rx_pipe(0, str(bytearray(l)))
    nrf.reg_write(1,0x3f)
    nrf.reg_write(2,0x3f)
    self.listen.start()
    nrf.start_listening()


def read(self):
    """
    Normally, it converts the data from the RX FIFO into a string of bits and starts
``packet_validation`` class passing this
    string as a parameter. However, if the file or focus features are enabled, the data from
the RX FIFO will be
```

```
        the message itself so it will just show the message on the screen.
        """

        if self.filemode == True:
            nrf.payload_size = nrf.reg_read(0b01100000)
            data = nrf.recv()
            print data
            self.add_to_record(''.join(str(hex(e))[2:].decode("hex") for e in data)+'\n')
        elif self.focused == True:
            nrf.payload_size = nrf.reg_read(0b01100000)
            data = nrf.recv()
            print data
            if data!=[0]:
                self.add_to_record(str(hex(data[0]))[2:].decode("hex")+'\n')
         #   else:
             #   self.record = self.record[:-1]
        else:
            addr = BitArray(bytearray(self.address[::-1]))
            data = nrf.recv()
            data = BitArray(bytearray(data))
            data.prepend(addr)
            data = BitArray(data)

            self.listofthreads[self.next_thread] = packet_validation(data)
            self.connect(self.listofthreads[self.next_thread], QtCore.SIGNAL('v'),
self.filter_duplicities)
            self.listofthreads[self.next_thread].start()
            self.next_thread += 1
            if self.next_thread == 9:
                self.next_thread = 0


    def filter_duplicities(self, data, crc, address_length, msg_start_bit, msg_end_bit):
        """
        :param str data:
        :param str crc:
        :param int address_length:
        :param int msg_start_bit:
        :param int msg_end_bit:
        It filters retransmitted packets. Usually, if it is a new packet, this method prints the
message on the text
        browser. Nevertheless, if the message converted into strings equals "text file"
``fast_reception()`` will be
        executed and a Saving file QFileDialog will pop up. This will return a path stored in
self.path. If self.focused
        is enabled fast_reception will be executed.
        """
        pid = data.bin[msg_start_bit - 3:msg_end_bit - 1]
        if pid != self.last_pid and crc != self.last_crc:
            self.crc = crc
```

```
            self.data = data
            self.address_length = address_length
            l = str(data[msg_start_bit:msg_end_bit].hex).decode("hex")
            print l[:12]
            if l[:5] == 'text 'and self.filemode==False:

                self.file_size=int(l[5:9].strip('0'))
                self.add_to_record('Downloading text file...[   0%]')
                self.filemode = True
                self.fast_reception(len(crc) / 8, str(data[:address_length * 8].hex))

            else:
                self.add_to_record("This was sent from address " + str(data[:address_length *
8]) + ":\n" + str(
                    data[msg_start_bit:msg_end_bit].hex))
        self.last_pid = pid
        self.last_crc = crc

class MultireceivingMode(GeneralModeFunctions):
    """
    It adds more parameters as more pipes are enabled.
    """

    def __init__(self):
        super(MultireceivingMode, self).__init__(Multireceiving(),
'Multireceiving_mode.txt')
        self.param = {'Speed': '', 'Power': '', 'Channel': '', 'CRC': '', 'Address P0': '',
'Address P1': '',
                      'Address P2': '', 'Address P3': '', 'Address P4': '', 'Address P5': ''}

    def apply_new_parameters(self):
        """
        The chip is configured to fit this mode.
        """
        if self.islistening:
            self.stop_listening()

        try:

            nrf.reg_write(EN_AA, 0b111111)  # distinto
            nrf.reg_write(EN_RXADDR, 0b111111)  # enable rx addresses
            nrf.reg_write(CONFIG, 0b1100)
            nrf.reg_write(DYNPD, 0b111111)
            nrf.reg_write(FEATURE, 0b100)
            self.param['Speed'] = myapp.app.speed.currentText()
            self.param['Power'] = myapp.app.power.currentText()
            self.param['Channel'] = myapp.app.channel.text()
            self.param['CRC'] = myapp.app.crc.currentText()
            self.param['Address P0'] = myapp.app.rx_ad_p0.text()
            self.param['Address P1'] = myapp.app.rx_adress_p1.text()
```

```
            self.param['Address P2'] = myapp.app.p2.text()
            self.param['Address P3'] = myapp.app.p3.text()
            self.param['Address P4'] = myapp.app.p4.text()
            self.param['Address P5'] = myapp.app.p5.text()
            nrf.reg_write(SETUP_AW, len(self.param['Address P0'].split(' ')) - 2)
            try:
                nrf.set_power_speed(self.trans_p_s[str(self.app.power.currentText())],
                                    self.trans_p_s[str(self.app.speed.currentText())])
                nrf.set_channel(int(self.param['Channel']))
                nrf.set_crc(int(self.param['CRC']))

                for n in range(6):
                    l = []
                    for i in str(self.param['Address P' + str(n)]).split(' ')[::-1]:
                        l.append(int(i, base=16))
                    nrf.reg_write(10 + n, str(bytearray(l)))
                if not self.filemode:
                    for h in self.param:
                        self.add_to_record(h + ': ' + self.param[h])
                self.add_to_record('\n')
                self.parameters_applied = True
            except:
                QtGui.QMessageBox.warning(self, "Error!", 'All parameters must be properly
fulfilled')
        except:
            global state
            state = 'Disconnected'
            self.app.statusBar.showMessage(state)
            self.getconnection.start()
            QtGui.QMessageBox.warning(self, "Error!", 'Device couldn\'t be found')

    def read(self):
        """
        It gets the pipe from which the NRF24L01 has received the information. This can be
obtained from the STATUS
        register. Afterwards, the received message is displayed on the text browser together
with the corresponding pipe.
        """
        self.add_to_record('from pipe ' + str((nrf.reg_read(STATUS) & 15) >> 1) + ':')
        nrf.payload_size = nrf.reg_read(0b01100000)
        data = nrf.recv()
        self.add_to_record(' '.join(str(hex(e))[2:] for e in data))



class packet_validation(QtCore.QThread):
    def __init__(self, data):
        super(packet_validation, self).__init__()
        self.data = data
```

```
    def run(self):
        """
        :param bitstring data:
        Looks for the CRC in the packet, if it finds it, then looks for the payload length and
returns a signal with the
        parameters extracted from the packet to the AutomaticMode class if it finds it.
        """
        print self.data.bin
        for i in range(29):

            a = time.time()
            if crc_(self.data[:41 + 8 * i], 1) == self.data.bin[41 + 8 * i:41 + 8 * i + 8]:
                msg_end_bit, crc = 41 + 8 * i, self.data.bin[41 + 8 * i:41 + 8 * i + 8]
                for a in range(3, 6):
                    if int(self.data.bin[a * 8:a * 8 + 6], 2) == len(
                            self.data.bin[a * 8 + 9:msg_end_bit]) / 8 and a * 8 + 9 != msg_end_bit:
                        self.emit(QtCore.SIGNAL('v'), self.data, crc, a, a * 8 + 9, msg_end_bit)
                        self.terminate()

            elif crc_(self.data[:41 + 8 * i], 2) == self.data.bin[41 + 8 * i:41 + 8 * i + 16]:
                msg_end_bit, crc = 41 + 8 * i, self.data.bin[41 + 8 * i:41 + 8 * i + 16]
                for a in range(3, 6):
                    if int(self.data.bin[a * 8:a * 8 + 6], 2) == len(
                            self.data.bin[a * 8 + 9:msg_end_bit]) / 8 and a * 8 + 9 != msg_end_bit:
                        self.emit(QtCore.SIGNAL('v'), self.data, crc, a, a * 8 + 9, msg_end_bit)
                        self.terminate()

        self.terminate()


class ListeningLoop(QtCore.QThread):
    def __init__(self, parent=None):
        super(ListeningLoop, self).__init__(parent)

    def run(self):
        """
        It runs a loop in which it checks if the ``any()`` function from the NRF24L01 class
returns True and, if so,
        sends a signal (*data_received()*) to the current class when something is received.
        """
        while 1:
            time.sleep(0.2)
            try:
                if nrf.any():
                    self.emit(QtCore.SIGNAL('data_received()'))
            except:
                self.emit(QtCore.SIGNAL('unexpected_disconnection()'))
                break
```

```
class Move(QtCore.QThread):
    def __init__(self, parent=None):
        super(Move, self).__init__(parent)

    def run(self):
        """
        This method updates the the current position of the window when this thread is
triggered. It maintains the
        relative distance between the cursor and the window so that the window moves where
the cursor goes.
        """
        global window_xpos, window_ypos
        pos = QtGui.QCursor.pos()
        xini = pos.x()
        yini = pos.y()
        geo = myapp.geometry()
        while app.mouseButtons():
            time.sleep(0.001)
            pos = QtGui.QCursor.pos()
            myapp.move(geo.x() + pos.x() - xini, geo.y() + pos.y() - yini)
        window_xpos = geo.x() + pos.x() - xini
        window_ypos = geo.y() + pos.y() - yini


class GetConnection(QtCore.QThread):
    def __init__(self, parent=None):
        super(GetConnection, self).__init__(parent)

    def run(self):
        """
        This method is triggered when this file is executed right at the beginning, in the if
statement and every
        time connection with the device is lost. It is basically a loop that checks if the device
has been
        connected to the PC.
        """
        global nrf, state

        while 1:
            try:
                nrf = NRF24L01(CS, GPIOL1)
                state = 'Connected'
                self.emit(QtCore.SIGNAL('connection()'))
                break
            except:
                time.sleep(0.5)


class CheckConnection(QtCore.QThread):
    def __init__(self, parent=None):
```

```
        super(CheckConnection, self).__init__(parent)

    def run(self):
        """
        Checks whether the connection between the *cable* and the PC still exists.
        """
        global nrf, state
        while 1:
            n = d2xx.createDeviceInfoList()
            if n == 1:
                time.sleep(2)
            else:
                state = 'Disconnected'
                self.emit(QtCore.SIGNAL('connection()'))
                break


class Intro(QtCore.QThread):
    def __init__(self, parent=None):
        super(Intro, self).__init__(parent)

    def run(self):
        """
        Waits for 3 seconds before sending the signal that will remove the introduction image
from the GUI.
        """
        time.sleep(3)
        self.emit(QtCore.SIGNAL('end_intro()'))


if __name__ == '__main__':
    app = QtGui.QApplication(sys.argv)
    myapp = ManualMode()
    myapp.show()
    geo = myapp.geometry()
    window_xpos = geo.x()
    window_ypos = geo.y()
    myapp.getconnection.start()
    sys.exit(app.exec_())
```

A.4 manual_mode.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'ex1.ui'
#
# Created by: PyQt4 UI code generator 4.11.4
#
# WARNING! All changes made in this file will be lost!

from PyQt4 import QtCore, QtGui

try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    def _fromUtf8(s):
        return s

try:
    _encoding = QtGui.QApplication.UnicodeUTF8
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig, _encoding)
except AttributeError:
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig)

class Manual(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName(_fromUtf8("MainWindow"))
        MainWindow.setMaximumSize(QtCore.QSize(740, 435))
        MainWindow.setMinimumSize(QtCore.QSize(740, 435))
        MainWindow.setWindowFlags(QtCore.Qt.FramelessWindowHint)
        MainWindow.setToolTip(_fromUtf8(""))
        MainWindow.setStatusTip(_fromUtf8(""))
        MainWindow.setWhatsThis(_fromUtf8(""))
        MainWindow.setAccessibleName(_fromUtf8(""))
        MainWindow.setAccessibleDescription(_fromUtf8(""))
        MainWindow.setUnifiedTitleAndToolBarOnMac(False)
        self.label = QtGui.QLabel(MainWindow)
        self.label.setGeometry(QtCore.QRect(0, 0, 749, 450))
        self.centralWidget = QtGui.QWidget(MainWindow)
        self.centralWidget.setMinimumSize(QtCore.QSize(749, 440))
        self.centralWidget.setObjectName(_fromUtf8("centralWidget"))
        self.verticalLayoutWidget = QtGui.QWidget(self.centralWidget)
        self.verticalLayoutWidget.setGeometry(QtCore.QRect(-22, -50, 794, 471))
        self.verticalLayoutWidget.setObjectName(_fromUtf8("verticalLayoutWidget"))
        self.verticalLayout_3 = QtGui.QVBoxLayout(self.verticalLayoutWidget)
        self.verticalLayout_3.setMargin(11)
        self.verticalLayout_3.setSpacing(0)
```

```
self.verticalLayout_3.setObjectName(_fromUtf8("verticalLayout_3"))
self.horizontalLayout_7 = QtGui.QHBoxLayout()
self.horizontalLayout_7.setMargin(11)
self.horizontalLayout_7.setSpacing(0)
self.horizontalLayout_7.setObjectName(_fromUtf8("horizontalLayout_7"))
self.movingB = QtGui.QPushButton(self.verticalLayoutWidget)
self.movingB.setMinimumSize(QtCore.QSize(590, 80))
self.movingB.setMaximumSize(QtCore.QSize(670, 16777215))
self.movingB.setText(_fromUtf8(""))
self.movingB.setObjectName(_fromUtf8("movingB"))
self.horizontalLayout_7.addWidget(self.movingB)
self.movingB.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
self.pushButton_4 = QtGui.QPushButton(self.verticalLayoutWidget)
self.pushButton_4.setMinimumSize(QtCore.QSize(37, 35))
self.pushButton_4.setMaximumSize(QtCore.QSize(37, 16777215))
self.pushButton_4.setText(_fromUtf8(""))
self.pushButton_4.setObjectName(_fromUtf8("pushButton_4"))
self.horizontalLayout_7.addWidget(self.pushButton_4)
self.pushButton_4.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
self.pushButton_4.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
self.pushButton_3 = QtGui.QPushButton(self.verticalLayoutWidget)
self.pushButton_3.setMinimumSize(QtCore.QSize(45, 35))
self.pushButton_3.setMaximumSize(QtCore.QSize(45, 16777215))
self.pushButton_3.setText(_fromUtf8(""))
self.pushButton_3.setObjectName(_fromUtf8("pushButton_3"))
self.horizontalLayout_7.addWidget(self.pushButton_3)
self.pushButton_3.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
self.pushButton_3.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
self.verticalLayout_3.addLayout(self.horizontalLayout_7)
self.horizontalLayout_10 = QtGui.QHBoxLayout()
self.horizontalLayout_10.setMargin(10)
self.horizontalLayout_10.setSpacing(0)
self.horizontalLayout_10.setObjectName(_fromUtf8("horizontalLayout_10"))
self.verticalLayout_2 = QtGui.QVBoxLayout()
self.verticalLayout_2.setSizeConstraint(QtGui.QLayout.SetDefaultConstraint)
self.verticalLayout_2.setMargin(10)
self.verticalLayout_2.setSpacing(0)
self.verticalLayout_2.setObjectName(_fromUtf8("verticalLayout_2"))
self.displaydataB = QtGui.QLabel(self.verticalLayoutWidget)
self.displaydataB.setMaximumSize(QtCore.QSize(100, 16777215))
font = QtGui.QFont()
font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
self.displaydataB.setFont(font)
self.displaydataB.setText(_fromUtf8(""))
self.displaydataB.setObjectName(_fromUtf8("displaydataB"))
self.verticalLayout_2.addWidget(self.displaydataB)
self.textBrowser = QtGui.QTextBrowser(self.verticalLayoutWidget)
self.textBrowser.setEnabled(True)
self.textBrowser.setMinimumSize(QtCore.QSize(330, 315))
self.textBrowser.setMaximumSize(QtCore.QSize(100, 315))
```

```
self.textBrowser.setObjectName(_fromUtf8("textBrowser"))
self.verticalLayout_2.addWidget(self.textBrowser)
self.horizontalLayout_10.addLayout(self.verticalLayout_2)
self.verticalLayout = QtGui.QVBoxLayout()
self.verticalLayout.setMargin(11)
self.verticalLayout.setSpacing(6)
self.verticalLayout.setObjectName(_fromUtf8("verticalLayout"))
self.horizontalLayout_8 = QtGui.QHBoxLayout()
self.horizontalLayout_8.setMargin(11)
self.horizontalLayout_8.setSpacing(6)
self.horizontalLayout_8.setObjectName(_fromUtf8("horizontalLayout_8"))
self.ex2 = QtGui.QPushButton(self.verticalLayoutWidget)
self.ex2.setMinimumSize(QtCore.QSize(0, 40))
self.ex2.setBaseSize(QtCore.QSize(0, 0))
self.ex2.setText(_fromUtf8(""))
self.ex2.setObjectName(_fromUtf8("ex2"))
self.horizontalLayout_8.addWidget(self.ex2)
self.ex2.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
self.ex2.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
self.ex3 = QtGui.QPushButton(self.verticalLayoutWidget)
self.ex3.setMinimumSize(QtCore.QSize(0, 40))
self.ex3.setText(_fromUtf8(""))
self.ex3.setObjectName(_fromUtf8("ex3"))
self.horizontalLayout_8.addWidget(self.ex3)
self.ex3.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
self.ex3.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
self.verticalLayout.addLayout(self.horizontalLayout_8)
self.horizontalLayout = QtGui.QHBoxLayout()
self.horizontalLayout.setMargin(11)
self.horizontalLayout.setSpacing(20)
self.horizontalLayout.setObjectName(_fromUtf8("horizontalLayout"))
self.speedL = QtGui.QLabel(self.verticalLayoutWidget)
self.speedL.setMaximumSize(QtCore.QSize(60, 16777215))
self.speedL.setMaximumSize(QtCore.QSize(60, 16777215))
font = QtGui.QFont()
font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
self.speedL.setFont(font)
self.speedL.setObjectName(_fromUtf8("speedL"))
self.horizontalLayout.addWidget(self.speedL)
self.speed = QtGui.QComboBox(self.verticalLayoutWidget)
self.speed.setMinimumSize(QtCore.QSize(60, 16777215))
self.speed.setObjectName(_fromUtf8("speed"))
self.speed.addItem(_fromUtf8(""))
self.speed.addItem(_fromUtf8(""))
self.speed.addItem(_fromUtf8(""))
self.horizontalLayout.addWidget(self.speed)
self.powerL = QtGui.QLabel(self.verticalLayoutWidget)
self.powerL.setMaximumSize(QtCore.QSize(16777215, 16777215))
font = QtGui.QFont()
font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
```

```
        self.powerL.setFont(font)
        self.powerL.setObjectName(_fromUtf8("powerL"))
        self.horizontalLayout.addWidget(self.powerL)
        self.power = QtGui.QComboBox(self.verticalLayoutWidget)
        self.power.setMinimumSize(QtCore.QSize(70, 16777215))
        self.power.setMaximumSize(QtCore.QSize(70, 16777215))
        self.power.setObjectName(_fromUtf8("power"))
        self.power.addItem(_fromUtf8(""))
        self.power.addItem(_fromUtf8(""))
        self.power.addItem(_fromUtf8(""))
        self.power.addItem(_fromUtf8(""))
        self.horizontalLayout.addWidget(self.power)
        self.channelL = QtGui.QLabel(self.verticalLayoutWidget)
        self.channelL.setMaximumSize(QtCore.QSize(1000, 16777215))
        font = QtGui.QFont()
        font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
        self.channelL.setFont(font)
        self.channelL.setObjectName(_fromUtf8("channelL"))
        self.horizontalLayout.addWidget(self.channelL)
        self.channel = QtGui.QLineEdit(self.verticalLayoutWidget)
        self.channel.setMaximumSize(QtCore.QSize(30, 16777215))
        self.channel.setText(_fromUtf8(""))
        self.channel.setObjectName(_fromUtf8("channel"))
        self.horizontalLayout.addWidget(self.channel)
        spacerItem = QtGui.QSpacerItem(40, 20, QtGui.QSizePolicy.Expanding,
QtGui.QSizePolicy.Minimum)
        self.horizontalLayout.addItem(spacerItem)
        self.verticalLayout.addLayout(self.horizontalLayout)
        self.horizontalLayout_4 = QtGui.QHBoxLayout()
        self.horizontalLayout_4.setMargin(0)
        self.horizontalLayout_4.setSpacing(52)
        self.horizontalLayout_4.setObjectName(_fromUtf8("horizontalLayout_4"))
        self.payloadsizeL = QtGui.QLabel(self.verticalLayoutWidget)
        self.payloadsizeL.setMaximumSize(QtCore.QSize(40, 16777215))
        font = QtGui.QFont()
        font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
        self.payloadsizeL.setFont(font)
        self.payloadsizeL.setObjectName(_fromUtf8("payloadsizeL"))
        self.horizontalLayout_4.addWidget(self.payloadsizeL)
        self.payload_size = QtGui.QLineEdit(self.verticalLayoutWidget)
        self.payload_size.setMaximumSize(QtCore.QSize(30, 16777215))
        self.payload_size.setText(_fromUtf8(""))
        self.payload_size.setObjectName(_fromUtf8("payload_size"))
        self.horizontalLayout_4.addWidget(self.payload_size)
        self.crcL = QtGui.QLabel(self.verticalLayoutWidget)
        self.crcL.setMaximumSize(QtCore.QSize(55, 16777215))
        font = QtGui.QFont()
        font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
        self.crcL.setFont(font)
        self.crcL.setObjectName(_fromUtf8("crcL"))
```

```
        self.horizontalLayout_4.addWidget(self.crcL)
        self.verticalLayout_4 = QtGui.QVBoxLayout()
        self.verticalLayout_4.setMargin(11)
        self.verticalLayout_4.setSpacing(6)
        self.verticalLayout_4.setObjectName(_fromUtf8("verticalLayout_4"))
        self.horizontalLayout_4.addLayout(self.verticalLayout_4)
        self.crc = QtGui.QComboBox(self.verticalLayoutWidget)
        self.crc.setMaximumSize(QtCore.QSize(30, 16777215))
        self.crc.setObjectName(_fromUtf8("crc"))
        self.crc.addItem(_fromUtf8(""))
        self.crc.addItem(_fromUtf8(""))
        self.horizontalLayout_4.addWidget(self.crc)
        #spacerItem1 = QtGui.QSpacerItem(40, 20, QtGui.QSizePolicy.Expanding,
QtGui.QSizePolicy.Minimum)
        # self.horizontalLayout_4.addItem(spacerItem1)
        self.verticalLayout.addLayout(self.horizontalLayout_4)
        self.horizontalLayout_2 = QtGui.QHBoxLayout()
        self.horizontalLayout_2.setMargin(11)
        self.horizontalLayout_2.setSpacing(6)
        self.horizontalLayout_2.setObjectName(_fromUtf8("horizontalLayout_2"))
        self.rx_adressL = QtGui.QLabel(self.verticalLayoutWidget)
        font = QtGui.QFont()
        font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
        self.rx_adressL.setFont(font)
        self.rx_adressL.setObjectName(_fromUtf8("rx_adressL"))
        self.horizontalLayout_2.addWidget(self.rx_adressL)
        self.rx_adress = QtGui.QLineEdit(self.verticalLayoutWidget)
        self.rx_adress.setObjectName(_fromUtf8("rx_adress"))
        self.horizontalLayout_2.addWidget(self.rx_adress)
        self.rx_adress.setMaximumSize(QtCore.QSize(150, 16777215))
        self.rx_idL = QtGui.QLabel(self.verticalLayoutWidget)
        font = QtGui.QFont()
        font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
        self.rx_idL.setFont(font)
        self.rx_idL.setObjectName(_fromUtf8("rx_idL"))
        self.horizontalLayout_2.addWidget(self.rx_idL)
        self.rx_id = QtGui.QComboBox(self.verticalLayoutWidget)
        self.rx_id.setMaximumSize(QtCore.QSize(30, 16777215))
        self.rx_id.setObjectName(_fromUtf8("rx_id"))
        self.rx_id.addItem(_fromUtf8(""))
        self.rx_id.addItem(_fromUtf8(""))
        self.rx_id.addItem(_fromUtf8(""))
        self.rx_id.addItem(_fromUtf8(""))
        self.rx_id.addItem(_fromUtf8(""))
        self.rx_id.addItem(_fromUtf8(""))
        self.horizontalLayout_2.addWidget(self.rx_id)
        self.verticalLayout.addLayout(self.horizontalLayout_2)
        self.horizontalLayout_9 = QtGui.QHBoxLayout()
        self.horizontalLayout_9.setMargin(0)
        self.horizontalLayout_9.setSpacing(47)
```

```
        self.horizontalLayout_9.setObjectName(_fromUtf8("horizontalLayout_9"))
        self.tx_adressL = QtGui.QLabel(self.verticalLayoutWidget)
        self.tx_adressL.setMaximumSize(QtCore.QSize(60, 16777215))
        font = QtGui.QFont()
        font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
        self.tx_adressL.setFont(font)
        self.tx_adressL.setObjectName(_fromUtf8("tx_adressL"))
        self.horizontalLayout_9.addWidget(self.tx_adressL)
        self.tx_adress = QtGui.QLineEdit(self.verticalLayoutWidget)
        self.tx_adress.setMinimumSize(QtCore.QSize(150, 0))
        self.tx_adress.setMaximumSize(QtCore.QSize(150, 16777215))
        self.tx_adress.setObjectName(_fromUtf8("tx_adress"))
        self.horizontalLayout_9.addWidget(self.tx_adress)
        self.ackL = QtGui.QLabel(self.verticalLayoutWidget)
        self.ackL.setMaximumSize(QtCore.QSize(30, 16777215))
        self.ackL.setFont(font)
        self.horizontalLayout_9.addWidget(self.ackL)
        self.acknowledgement = QtGui.QCheckBox(self.verticalLayoutWidget)
        self.acknowledgement.hide()
        self.horizontalLayout_9.addWidget(self.acknowledgement)
        spacerItem2 = QtGui.QSpacerItem(40, 20, QtGui.QSizePolicy.Expanding,
QtGui.QSizePolicy.Minimum)
        self.horizontalLayout_9.addItem(spacerItem2)
        self.verticalLayout.addLayout(self.horizontalLayout_9)
        self.horizontalLayout_5 = QtGui.QHBoxLayout()
        self.horizontalLayout_5.setMargin(11)
        self.horizontalLayout_5.setSpacing(6)
        self.horizontalLayout_5.setObjectName(_fromUtf8("horizontalLayout_5"))
        self.applychangesB = QtGui.QPushButton(self.verticalLayoutWidget)
        self.applychangesB.setMinimumSize(QtCore.QSize(0, 50))
        self.applychangesB.setMaximumSize(QtCore.QSize(16777215, 16777215))
        self.applychangesB.setText(_fromUtf8(""))
        self.applychangesB.setIconSize(QtCore.QSize(16, 32))
        self.applychangesB.setAutoDefault(False)
        self.applychangesB.setDefault(False)
        self.applychangesB.setFlat(False)
        self.applychangesB.setObjectName(_fromUtf8("applychangesB"))
        self.applychangesB.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.applychangesB.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
        self.horizontalLayout_5.addWidget(self.applychangesB)
        self.listen = QtGui.QPushButton(self.verticalLayoutWidget)
        self.listen.setMinimumSize(QtCore.QSize(0, 50))
        self.listen.setBaseSize(QtCore.QSize(0, 0))
        self.listen.setText(_fromUtf8(""))
        self.listen.setObjectName(_fromUtf8("listen"))
        self.listen.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.listen.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
        self.horizontalLayout_5.addWidget(self.listen)
        self.verticalLayout.addLayout(self.horizontalLayout_5)
        spacerIt = QtGui.QSpacerItem(0, 20, QtGui.QSizePolicy.Expanding,
```

```
QtGui.QSizePolicy.Minimum)
        self.verticalLayout.addItem(spacerIt)
        self.horizontalLayout_3 = QtGui.QHBoxLayout()
        self.horizontalLayout_3.setMargin(11)
        self.horizontalLayout_3.setSpacing(6)
        self.horizontalLayout_3.setObjectName(_fromUtf8("horizontalLayout_3"))
        self.send = QtGui.QPushButton(self.verticalLayoutWidget)
        self.send.setMinimumSize(QtCore.QSize(0, 50))
        self.send.setText(_fromUtf8(""))
        self.send.setObjectName(_fromUtf8("pushButton"))
        self.send.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.send.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
        self.horizontalLayout_3.addWidget(self.send)
        self.stopB = QtGui.QPushButton(self.verticalLayoutWidget)
        self.stopB.setMinimumSize(QtCore.QSize(0, 50))
        self.stopB.setText(_fromUtf8(""))
        self.stopB.setObjectName(_fromUtf8("stopB"))
        self.stopB.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.stopB.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
        self.horizontalLayout_3.addWidget(self.stopB)
        self.verticalLayout.addLayout(self.horizontalLayout_3)
        self.horizontalLayout_10.addLayout(self.verticalLayout)
        self.verticalLayout_3.addLayout(self.horizontalLayout_10)
        MainWindow.setCentralWidget(self.centralWidget)
        self.statusBar = QtGui.QStatusBar(MainWindow)
        self.statusBar.setObjectName(_fromUtf8("statusBar"))
        MainWindow.setStatusBar(self.statusBar)
        font = QtGui.QFont()
        font.setPixelSize(15)
        self.statusBar.setFont(font)
        self.fons = QtGui.QPixmap('manual_mode.png')
        self.label.setPixmap(self.fons)
        self.label.setAlignment(QtCore.Qt.AlignCenter)
        self.intro = QtGui.QLabel(MainWindow)
        self.intro.setGeometry(QtCore.QRect(0, 0, 740, 435))
        self.intro.setStyleSheet("background-color: rgba( 0,0,0, 100% );" )
        self.pixmap=QtGui.QPixmap('icono.png')
        self.intro.setPixmap(self.pixmap)
        self.intro.setAlignment(QtCore.Qt.AlignCenter)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        MainWindow.setWindowTitle(_translate("MainWindow", "Sniffer app", None))
        self.speedL.setText(_translate("MainWindow", "          ", None))
        self.speed.setItemText(0, _translate("MainWindow", "1 Mbps", None))
        self.speed.setItemText(1, _translate("MainWindow", "2 Mbps", None))
        self.speed.setItemText(2, _translate("MainWindow", "250 Kbps", None))
        self.powerL.setText(_translate("MainWindow", "          ", None))
```

```
        self.power.setItemText(0, _translate("MainWindow", "0 dBm", None))
        self.power.setItemText(1, _translate("MainWindow", "-6 dBm", None))
        self.power.setItemText(2, _translate("MainWindow", "-12 dBm", None))
        self.power.setItemText(3, _translate("MainWindow", "-18 dBm", None))
        self.channelL.setText(_translate("MainWindow", "
", None))
        self.channel.setProperty("default value", _translate("MainWindow", "9", None))
        self.payloadsizeL.setText(_translate("MainWindow", "  ", None))
        self.crcL.setText(_translate("MainWindow", "                                ", None))
        self.crc.setItemText(0, _translate("MainWindow", "1", None))
        self.crc.setItemText(1, _translate("MainWindow", "2", None))
        self.rx_adressL.setText(_translate("MainWindow", "                          ", None))
        self.rx_idL.setText(_translate("MainWindow", "", None))
        self.rx_id.setItemText(0, _translate("MainWindow", "0", None))
        self.rx_id.setItemText(1, _translate("MainWindow", "1", None))
        self.rx_id.setItemText(2, _translate("MainWindow", "2", None))
        self.rx_id.setItemText(3, _translate("MainWindow", "3", None))
        self.rx_id.setItemText(4, _translate("MainWindow", "4", None))
        self.rx_id.setItemText(5, _translate("MainWindow", "5", None))
        self.tx_adressL.setText(_translate("MainWindow", "                  ", None))
        self.applychangesB.setShortcut(_translate("MainWindow", "Return", None))
        self.listen.setShortcut(_translate("MainWindow", "L", None))
        self.send.setShortcut(_translate("MainWindow", "S", None))
        self.stopB.setShortcut(_translate("MainWindow", "Space", None))
```

A.5 multireceiving_mode.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'ex1.ui'
#
# Created by: PyQt4 UI code generator 4.11.4
#
# WARNING! All changes made in this file will be lost!

from PyQt4 import QtCore, QtGui

try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    def _fromUtf8(s):
        return s

try:
    _encoding = QtGui.QApplication.UnicodeUTF8
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig, _encoding)
except AttributeError:
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig)

class Multireceiving(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName(_fromUtf8("MainWindow"))
        MainWindow.resize(749, 490)
        MainWindow.setMaximumSize(QtCore.QSize(740, 435))
        MainWindow.setMinimumSize(QtCore.QSize(740, 435))
        MainWindow.setWindowFlags(QtCore.Qt.FramelessWindowHint)
        MainWindow.setToolTip(_fromUtf8(""))
        MainWindow.setStatusTip(_fromUtf8(""))
        MainWindow.setWhatsThis(_fromUtf8(""))
        MainWindow.setAccessibleName(_fromUtf8(""))
        MainWindow.setAccessibleDescription(_fromUtf8(""))
        MainWindow.setUnifiedTitleAndToolBarOnMac(False)
        self.label = QtGui.QLabel(MainWindow)
        self.label.setGeometry(QtCore.QRect(0, 0, 749, 450))
        self.centralWidget = QtGui.QWidget(MainWindow)
        self.centralWidget.setMinimumSize(QtCore.QSize(749, 440))
        self.centralWidget.setObjectName(_fromUtf8("centralWidget"))
        self.verticalLayoutWidget = QtGui.QWidget(self.centralWidget)
        self.verticalLayoutWidget.setGeometry(QtCore.QRect(-22, -50, 794, 471))
        self.verticalLayoutWidget.setObjectName(_fromUtf8("verticalLayoutWidget"))
        self.verticalLayout_3 = QtGui.QVBoxLayout(self.verticalLayoutWidget)
        self.verticalLayout_3.setMargin(11)
        self.verticalLayout_3.setSpacing(0)
```

```
self.verticalLayout_3.setObjectName(_fromUtf8("verticalLayout_3"))
self.horizontalLayout_7 = QtGui.QHBoxLayout()
self.horizontalLayout_7.setMargin(11)
self.horizontalLayout_7.setSpacing(0)
self.horizontalLayout_7.setObjectName(_fromUtf8("horizontalLayout_7"))
self.movingB = QtGui.QPushButton(self.verticalLayoutWidget)
self.movingB.setMinimumSize(QtCore.QSize(590, 80))
self.movingB.setMaximumSize(QtCore.QSize(670, 16777215))
self.movingB.setText(_fromUtf8(""))
self.movingB.setObjectName(_fromUtf8("movingB"))
self.horizontalLayout_7.addWidget(self.movingB)
self.movingB.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
self.pushButton_4 = QtGui.QPushButton(self.verticalLayoutWidget)
self.pushButton_4.setMinimumSize(QtCore.QSize(37, 35))
self.pushButton_4.setMaximumSize(QtCore.QSize(37, 16777215))
self.pushButton_4.setText(_fromUtf8(""))
self.pushButton_4.setObjectName(_fromUtf8("pushButton_4"))
self.horizontalLayout_7.addWidget(self.pushButton_4)
self.pushButton_4.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
self.pushButton_4.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
self.pushButton_3 = QtGui.QPushButton(self.verticalLayoutWidget)
self.pushButton_3.setMinimumSize(QtCore.QSize(45, 35))
self.pushButton_3.setMaximumSize(QtCore.QSize(45, 16777215))
self.pushButton_3.setText(_fromUtf8(""))
self.pushButton_3.setObjectName(_fromUtf8("pushButton_3"))
self.horizontalLayout_7.addWidget(self.pushButton_3)
self.pushButton_3.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
self.pushButton_3.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
self.verticalLayout_3.addLayout(self.horizontalLayout_7)
self.horizontalLayout_10 = QtGui.QHBoxLayout()
self.horizontalLayout_10.setMargin(10)
self.horizontalLayout_10.setSpacing(0)
self.horizontalLayout_10.setObjectName(_fromUtf8("horizontalLayout_10"))
self.verticalLayout_2 = QtGui.QVBoxLayout()
self.verticalLayout_2.setSizeConstraint(QtGui.QLayout.SetDefaultConstraint)
self.verticalLayout_2.setMargin(10)
self.verticalLayout_2.setSpacing(0)
self.verticalLayout_2.setObjectName(_fromUtf8("verticalLayout_2"))
self.displaydataB = QtGui.QLabel(self.verticalLayoutWidget)
self.displaydataB.setMaximumSize(QtCore.QSize(100, 16777215))
font = QtGui.QFont()
font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
self.displaydataB.setFont(font)
self.displaydataB.setText(_fromUtf8(""))
self.displaydataB.setObjectName(_fromUtf8("displaydataB"))
self.verticalLayout_2.addWidget(self.displaydataB)
self.textBrowser = QtGui.QTextBrowser(self.verticalLayoutWidget)
self.textBrowser.setEnabled(True)
self.textBrowser.setMinimumSize(QtCore.QSize(330, 315))
self.textBrowser.setMaximumSize(QtCore.QSize(100, 315))
```

```
self.textBrowser.setObjectName(_fromUtf8("textBrowser"))
self.verticalLayout_2.addWidget(self.textBrowser)
self.horizontalLayout_10.addLayout(self.verticalLayout_2)
self.verticalLayout = QtGui.QVBoxLayout()
self.verticalLayout.setMargin(11)
self.verticalLayout.setSpacing(6)
self.verticalLayout.setObjectName(_fromUtf8("verticalLayout"))
self.horizontalLayout_8 = QtGui.QHBoxLayout()
self.horizontalLayout_8.setMargin(11)
self.horizontalLayout_8.setSpacing(6)
self.horizontalLayout_8.setObjectName(_fromUtf8("horizontalLayout_8"))
self.ex2 = QtGui.QPushButton(self.verticalLayoutWidget)
self.ex2.setMinimumSize(QtCore.QSize(0, 40))
self.ex2.setBaseSize(QtCore.QSize(0, 0))
self.ex2.setText(_fromUtf8(""))
self.ex2.setObjectName(_fromUtf8("ex2"))
self.horizontalLayout_8.addWidget(self.ex2)
self.ex2.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
self.ex2.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
self.ex3 = QtGui.QPushButton(self.verticalLayoutWidget)
self.ex3.setMinimumSize(QtCore.QSize(0, 40))
self.ex3.setText(_fromUtf8(""))
self.ex3.setObjectName(_fromUtf8("ex3"))
self.horizontalLayout_8.addWidget(self.ex3)
self.ex3.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
self.ex3.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
self.verticalLayout.addLayout(self.horizontalLayout_8)
self.horizontalLayout = QtGui.QHBoxLayout()
self.horizontalLayout.setMargin(11)
self.horizontalLayout.setSpacing(20)
self.horizontalLayout.setObjectName(_fromUtf8("horizontalLayout"))
self.speedL = QtGui.QLabel(self.verticalLayoutWidget)
self.speedL.setMaximumSize(QtCore.QSize(60, 16777215))
self.speedL.setMaximumSize(QtCore.QSize(60, 16777215))
font = QtGui.QFont()
font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
self.speedL.setFont(font)
self.speedL.setObjectName(_fromUtf8("speedL"))
self.horizontalLayout.addWidget(self.speedL)
self.speed = QtGui.QComboBox(self.verticalLayoutWidget)
self.speed.setMinimumSize(QtCore.QSize(60, 16777215))
self.speed.setObjectName(_fromUtf8("speed"))
self.speed.addItem(_fromUtf8(""))
self.speed.addItem(_fromUtf8(""))
self.speed.addItem(_fromUtf8(""))
self.horizontalLayout.addWidget(self.speed)
self.powerL = QtGui.QLabel(self.verticalLayoutWidget)
self.powerL.setMaximumSize(QtCore.QSize(16777215, 16777215))
font = QtGui.QFont()
font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
```

```
        self.powerL.setFont(font)
        self.powerL.setObjectName(_fromUtf8("powerL"))
        self.horizontalLayout.addWidget(self.powerL)
        self.power = QtGui.QComboBox(self.verticalLayoutWidget)
        self.power.setMinimumSize(QtCore.QSize(70, 16777215))
        self.power.setMaximumSize(QtCore.QSize(70, 16777215))
        self.power.setObjectName(_fromUtf8("power"))
        self.power.addItem(_fromUtf8(""))
        self.power.addItem(_fromUtf8(""))
        self.power.addItem(_fromUtf8(""))
        self.power.addItem(_fromUtf8(""))
        self.horizontalLayout.addWidget(self.power)
        self.channelL = QtGui.QLabel(self.verticalLayoutWidget)
        self.channelL.setMaximumSize(QtCore.QSize(1000, 16777215))
        font = QtGui.QFont()
        font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
        self.channelL.setFont(font)
        self.channelL.setObjectName(_fromUtf8("channelL"))
        self.horizontalLayout.addWidget(self.channelL)
        self.channel = QtGui.QLineEdit(self.verticalLayoutWidget)
        self.channel.setMaximumSize(QtCore.QSize(30, 16777215))
        self.channel.setText(_fromUtf8(""))
        self.channel.setObjectName(_fromUtf8("channel"))
        self.horizontalLayout.addWidget(self.channel)
        spacerItem = QtGui.QSpacerItem(40, 20, QtGui.QSizePolicy.Expanding,
QtGui.QSizePolicy.Minimum)
        self.horizontalLayout.addItem(spacerItem)
        self.verticalLayout.addLayout(self.horizontalLayout)
        self.horizontalLayout_4 = QtGui.QHBoxLayout()
        self.horizontalLayout_4.setMargin(0)
        self.horizontalLayout_4.setSpacing(18)
        self.horizontalLayout_4.setObjectName(_fromUtf8("horizontalLayout_4"))
        self.payloadsizeL = QtGui.QLabel(self.verticalLayoutWidget)
        self.payloadsizeL.setMaximumSize(QtCore.QSize(90, 16777215))
        font = QtGui.QFont()
        font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
        self.payloadsizeL.setFont(font)
        self.payloadsizeL.setObjectName(_fromUtf8("payloadsizeL"))
        self.horizontalLayout_4.addWidget(self.payloadsizeL)
        self.rx_ad_p0 = QtGui.QLineEdit(self.verticalLayoutWidget)
        self.rx_ad_p0.setMinimumSize(QtCore.QSize(130, 16777215))
        self.rx_ad_p0.setMaximumSize(QtCore.QSize(130, 16777215))
        self.rx_ad_p0.setText(_fromUtf8(""))
        self.rx_ad_p0.setObjectName(_fromUtf8("rx_ad_p0"))
        self.horizontalLayout_4.addWidget(self.rx_ad_p0)
        self.crcL = QtGui.QLabel(self.verticalLayoutWidget)
        self.crcL.setMaximumSize(QtCore.QSize(55, 16777215))
        font = QtGui.QFont()
        font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
        self.crcL.setFont(font)
```

```
        self.crcL.setObjectName(_fromUtf8("crcL"))
        self.crcL.setMaximumSize(QtCore.QSize(10, 16777215))
        self.horizontalLayout_4.addWidget(self.crcL)
        self.verticalLayout_4 = QtGui.QVBoxLayout()
        self.verticalLayout_4.setMargin(11)
        self.verticalLayout_4.setSpacing(6)
        self.verticalLayout_4.setObjectName(_fromUtf8("verticalLayout_4"))
        self.horizontalLayout_4.addLayout(self.verticalLayout_4)
        self.crc = QtGui.QComboBox(self.verticalLayoutWidget)
        self.crc.setMaximumSize(QtCore.QSize(30, 16777215))
        self.crc.setObjectName(_fromUtf8("crc"))
        self.crc.addItem(_fromUtf8(""))
        self.crc.addItem(_fromUtf8(""))
        self.horizontalLayout_4.addWidget(self.crc)
        #spacerItem1 = QtGui.QSpacerItem(40, 20, QtGui.QSizePolicy.Expanding,
QtGui.QSizePolicy.Minimum)
        # self.horizontalLayout_4.addItem(spacerItem1)
        self.verticalLayout.addLayout(self.horizontalLayout_4)
        self.horizontalLayout_2 = QtGui.QHBoxLayout()
        self.horizontalLayout_2.setMargin(11)
        self.horizontalLayout_2.setSpacing(6)
        self.horizontalLayout_2.setObjectName(_fromUtf8("horizontalLayout_2"))
        self.rx_adressL = QtGui.QLabel(self.verticalLayoutWidget)
        font = QtGui.QFont()
        font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
        self.rx_adressL.setFont(font)
        self.rx_adressL.setObjectName(_fromUtf8("rx_adressL"))
        self.horizontalLayout_2.addWidget(self.rx_adressL)
        self.rx_adress_p1 = QtGui.QLineEdit(self.verticalLayoutWidget)
        self.rx_adress_p1.setObjectName(_fromUtf8("rx_adress"))
        self.horizontalLayout_2.addWidget(self.rx_adress_p1)
        self.rx_adress_p1.setMaximumSize(QtCore.QSize(130, 16777215))
        self.rx_idL = QtGui.QLabel(self.verticalLayoutWidget)
        font = QtGui.QFont()
        font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
        self.rx_idL.setFont(font)
        self.rx_idL.setObjectName(_fromUtf8("rx_idL"))
        self.horizontalLayout_2.addWidget(self.rx_idL)



        self.verticalLayout.addLayout(self.horizontalLayout_2)
        self.horizontalLayout_9 = QtGui.QHBoxLayout()
        self.horizontalLayout_9.setMargin(0)
        self.horizontalLayout_9.setSpacing(25)
        self.horizontalLayout_9.setObjectName(_fromUtf8("horizontalLayout_9"))
        self.p2L = QtGui.QLabel(self.verticalLayoutWidget)
        self.p2L.setMaximumSize(QtCore.QSize(70, 16777215))
        self.horizontalLayout_9.addWidget(self.p2L)
        self.p2 = QtGui.QLineEdit(self.verticalLayoutWidget)
```

```
        self.p2.setMaximumSize(QtCore.QSize(25, 16777215))
        self.p2.setObjectName(_fromUtf8("p2"))
        self.horizontalLayout_9.addWidget(self.p2)
        self.p3 = QtGui.QLineEdit(self.verticalLayoutWidget)
        self.p3.setMaximumSize(QtCore.QSize(25, 16777215))
        self.horizontalLayout_9.addWidget(self.p3)
        self.p4 = QtGui.QLineEdit(self.verticalLayoutWidget)
        self.p4.setMaximumSize(QtCore.QSize(25, 16777215))
        self.horizontalLayout_9.addWidget(self.p4)
        self.p5 = QtGui.QLineEdit(self.verticalLayoutWidget)
        self.p5.setMaximumSize(QtCore.QSize(25, 16777215))
        self.horizontalLayout_9.addWidget(self.p5)
       # spacerItem2 = QtGui.QSpacerItem(40, 20, QtGui.QSizePolicy.Expanding,
QtGui.QSizePolicy.Minimum)
      #  self.horizontalLayout_9.addItem(spacerItem2)
        self.verticalLayout.addLayout(self.horizontalLayout_9)
        self.horizontalLayout_5 = QtGui.QHBoxLayout()
        self.horizontalLayout_5.setMargin(11)
        self.horizontalLayout_5.setSpacing(6)
        self.horizontalLayout_5.setObjectName(_fromUtf8("horizontalLayout_5"))
        self.applychangesB = QtGui.QPushButton(self.verticalLayoutWidget)
        self.applychangesB.setMinimumSize(QtCore.QSize(0, 50))
        self.applychangesB.setMaximumSize(QtCore.QSize(16777215, 16777215))
        self.applychangesB.setText(_fromUtf8(""))
        self.applychangesB.setIconSize(QtCore.QSize(16, 32))
        self.applychangesB.setAutoDefault(False)
        self.applychangesB.setDefault(False)
        self.applychangesB.setFlat(False)
        self.applychangesB.setObjectName(_fromUtf8("applychangesB"))
        self.applychangesB.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.applychangesB.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
        self.horizontalLayout_5.addWidget(self.applychangesB)
        self.listen = QtGui.QPushButton(self.verticalLayoutWidget)
        self.listen.setMinimumSize(QtCore.QSize(0, 50))
        self.listen.setBaseSize(QtCore.QSize(0, 0))
        self.listen.setText(_fromUtf8(""))
        self.listen.setObjectName(_fromUtf8("listen"))
        self.listen.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.listen.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
        self.horizontalLayout_5.addWidget(self.listen)
        self.verticalLayout.addLayout(self.horizontalLayout_5)
        spacerIt = QtGui.QSpacerItem(0, 20, QtGui.QSizePolicy.Expanding,
QtGui.QSizePolicy.Minimum)
        self.verticalLayout.addItem(spacerIt)
        self.horizontalLayout_3 = QtGui.QHBoxLayout()
        self.horizontalLayout_3.setMargin(11)
        self.horizontalLayout_3.setSpacing(6)
        self.horizontalLayout_3.setObjectName(_fromUtf8("horizontalLayout_3"))
        self.pushButton = QtGui.QPushButton(self.verticalLayoutWidget)
        self.pushButton.setMinimumSize(QtCore.QSize(0, 50))
```

```
        self.pushButton.setText(_fromUtf8(""))
        self.pushButton.setObjectName(_fromUtf8("pushButton"))
        self.pushButton.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
        self.horizontalLayout_3.addWidget(self.pushButton)
        self.stopB = QtGui.QPushButton(self.verticalLayoutWidget)
        self.stopB.setMinimumSize(QtCore.QSize(0, 50))
        self.stopB.setText(_fromUtf8(""))
        self.stopB.setObjectName(_fromUtf8("stopB"))
        self.stopB.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.stopB.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
        self.horizontalLayout_3.addWidget(self.stopB)
        self.verticalLayout.addLayout(self.horizontalLayout_3)
        self.horizontalLayout_10.addLayout(self.verticalLayout)
        self.verticalLayout_3.addLayout(self.horizontalLayout_10)
        MainWindow.setCentralWidget(self.centralWidget)
        self.statusBar = QtGui.QStatusBar(MainWindow)
        self.statusBar.setObjectName(_fromUtf8("statusBar"))
        MainWindow.setStatusBar(self.statusBar)
        font = QtGui.QFont()
        font.setPixelSize(15)
        self.statusBar.setFont(font)
        self.fons = QtGui.QPixmap('multireceiving_mode.png')
        self.label.setPixmap(self.fons)
        self.label.setAlignment(QtCore.Qt.AlignCenter)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        MainWindow.setWindowTitle(_translate("MainWindow", "Sniffer app", None))
        self.speedL.setText(_translate("MainWindow", "          ", None))
        self.speed.setItemText(0, _translate("MainWindow", "1 Mbps", None))
        self.speed.setItemText(1, _translate("MainWindow", "2 Mbps", None))
        self.speed.setItemText(2, _translate("MainWindow", "250 Kbps", None))
        self.powerL.setText(_translate("MainWindow", "                ", None))
        self.power.setItemText(0, _translate("MainWindow", "0 dBm", None))
        self.power.setItemText(1, _translate("MainWindow", "-6 dBm", None))
        self.power.setItemText(2, _translate("MainWindow", "-12 dBm", None))
        self.power.setItemText(3, _translate("MainWindow", "-18 dBm", None))
        self.power.hide()
        self.channelL.setText(_translate("MainWindow", "                      ", None))
        self.channel.setProperty("default value", _translate("MainWindow", "9", None))
        self.payloadsizeL.setText(_translate("MainWindow", "
", None))
        self.crcL.setText(_translate("MainWindow", "                          ", None))
        self.crc.setItemText(0, _translate("MainWindow", "1", None))
        self.crc.setItemText(1, _translate("MainWindow", "2", None))
        self.rx_adressL.setText(_translate("MainWindow", "                      ", None))
        self.rx_idL.setText(_translate("MainWindow", "", None))
        self.p2L.setText(_translate("MainWindow", "          ", None))
```

```
self.applychangesB.setShortcut(_translate("MainWindow", "Return", None))
self.listen.setShortcut(_translate("MainWindow", "L", None))
self.pushButton.setShortcut(_translate("MainWindow", "S", None))
self.stopB.setShortcut(_translate("MainWindow", "Space", None))
```

A.6 automatic_mode.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'ex1.ui'
#
# Created by: PyQt4 UI code generator 4.11.4
#
# WARNING! All changes made in this file will be lost!

from PyQt4 import QtCore, QtGui

try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    def _fromUtf8(s):
        return s

try:
    _encoding = QtGui.QApplication.UnicodeUTF8
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig, _encoding)
except AttributeError:
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig)

class Automatic(object):
    def setupUi(self, MainWindow):
        print 1
        MainWindow.setObjectName(_fromUtf8("MainWindow"))
        MainWindow.resize(749, 490)
        MainWindow.setMaximumSize(QtCore.QSize(740, 435))
        MainWindow.setMinimumSize(QtCore.QSize(740, 435))
        MainWindow.setWindowFlags(QtCore.Qt.FramelessWindowHint)
        MainWindow.setToolTip(_fromUtf8(""))
        MainWindow.setStatusTip(_fromUtf8(""))
        MainWindow.setWhatsThis(_fromUtf8(""))
        MainWindow.setAccessibleName(_fromUtf8(""))
        MainWindow.setAccessibleDescription(_fromUtf8(""))
        MainWindow.setUnifiedTitleAndToolBarOnMac(False)
        self.label = QtGui.QLabel(MainWindow)
        self.label.setGeometry(QtCore.QRect(0, 0, 749, 450))
        self.centralWidget = QtGui.QWidget(MainWindow)
        self.centralWidget.setMinimumSize(QtCore.QSize(749, 440))
        self.centralWidget.setObjectName(_fromUtf8("centralWidget"))
        self.verticalLayoutWidget = QtGui.QWidget(self.centralWidget)
        self.verticalLayoutWidget.setGeometry(QtCore.QRect(-22, -50, 794, 471))
        self.verticalLayoutWidget.setObjectName(_fromUtf8("verticalLayoutWidget"))
        self.verticalLayout_3 = QtGui.QVBoxLayout(self.verticalLayoutWidget)
        self.verticalLayout_3.setMargin(11)
```

```
self.verticalLayout_3.setSpacing(0)
self.verticalLayout_3.setObjectName(_fromUtf8("verticalLayout_3"))
self.horizontalLayout_7 = QtGui.QHBoxLayout()
self.horizontalLayout_7.setMargin(11)
self.horizontalLayout_7.setSpacing(0)
self.horizontalLayout_7.setObjectName(_fromUtf8("horizontalLayout_7"))
self.movingB = QtGui.QPushButton(self.verticalLayoutWidget)
self.movingB.setMinimumSize(QtCore.QSize(590, 80))
self.movingB.setMaximumSize(QtCore.QSize(670, 16777215))
self.movingB.setText(_fromUtf8(""))
self.movingB.setObjectName(_fromUtf8("movingB"))
self.horizontalLayout_7.addWidget(self.movingB)
self.movingB.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
self.pushButton_4 = QtGui.QPushButton(self.verticalLayoutWidget)
self.pushButton_4.setMinimumSize(QtCore.QSize(37, 35))
self.pushButton_4.setMaximumSize(QtCore.QSize(37, 16777215))
self.pushButton_4.setText(_fromUtf8(""))
self.pushButton_4.setObjectName(_fromUtf8("pushButton_4"))
self.horizontalLayout_7.addWidget(self.pushButton_4)
self.pushButton_4.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
self.pushButton_4.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
self.pushButton_3 = QtGui.QPushButton(self.verticalLayoutWidget)
self.pushButton_3.setMinimumSize(QtCore.QSize(45, 35))
self.pushButton_3.setMaximumSize(QtCore.QSize(45, 16777215))
self.pushButton_3.setText(_fromUtf8(""))
self.pushButton_3.setObjectName(_fromUtf8("pushButton_3"))
self.horizontalLayout_7.addWidget(self.pushButton_3)
self.pushButton_3.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
self.pushButton_3.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
self.verticalLayout_3.addLayout(self.horizontalLayout_7)
self.horizontalLayout_10 = QtGui.QHBoxLayout()
self.horizontalLayout_10.setMargin(10)
self.horizontalLayout_10.setSpacing(0)
self.horizontalLayout_10.setObjectName(_fromUtf8("horizontalLayout_10"))
self.verticalLayout_2 = QtGui.QVBoxLayout()
self.verticalLayout_2.setSizeConstraint(QtGui.QLayout.SetDefaultConstraint)
self.verticalLayout_2.setMargin(10)
self.verticalLayout_2.setSpacing(0)
self.verticalLayout_2.setObjectName(_fromUtf8("verticalLayout_2"))
self.displaydataB = QtGui.QLabel(self.verticalLayoutWidget)
self.displaydataB.setMaximumSize(QtCore.QSize(100, 16777215))
font = QtGui.QFont()
font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
self.displaydataB.setFont(font)
self.displaydataB.setText(_fromUtf8(""))
self.displaydataB.setObjectName(_fromUtf8("displaydataB"))
self.verticalLayout_2.addWidget(self.displaydataB)
self.textBrowser = QtGui.QTextBrowser(self.verticalLayoutWidget)
self.textBrowser.setEnabled(True)
self.textBrowser.setMinimumSize(QtCore.QSize(330, 315))
```

```
self.textBrowser.setMaximumSize(QtCore.QSize(100, 315))
self.textBrowser.setObjectName(_fromUtf8("textBrowser"))
self.verticalLayout_2.addWidget(self.textBrowser)
self.horizontalLayout_10.addLayout(self.verticalLayout_2)
self.verticalLayout = QtGui.QVBoxLayout()
self.verticalLayout.setMargin(11)
self.verticalLayout.setSpacing(6)
self.verticalLayout.setObjectName(_fromUtf8("verticalLayout"))
self.horizontalLayout_8 = QtGui.QHBoxLayout()
self.horizontalLayout_8.setMargin(11)
self.horizontalLayout_8.setSpacing(6)
self.horizontalLayout_8.setObjectName(_fromUtf8("horizontalLayout_8"))
self.ex2 = QtGui.QPushButton(self.verticalLayoutWidget)
self.ex2.setMinimumSize(QtCore.QSize(0, 40))
self.ex2.setBaseSize(QtCore.QSize(0, 0))
self.ex2.setText(_fromUtf8(""))
self.ex2.setObjectName(_fromUtf8("ex2"))
self.horizontalLayout_8.addWidget(self.ex2)
self.ex2.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
self.ex2.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
self.ex3 = QtGui.QPushButton(self.verticalLayoutWidget)
self.ex3.setMinimumSize(QtCore.QSize(0, 40))
self.ex3.setText(_fromUtf8(""))
self.ex3.setObjectName(_fromUtf8("ex3"))
self.horizontalLayout_8.addWidget(self.ex3)
self.ex3.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
self.ex3.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
self.verticalLayout.addLayout(self.horizontalLayout_8)
self.horizontalLayout = QtGui.QHBoxLayout()
self.horizontalLayout.setMargin(11)
self.horizontalLayout.setSpacing(20)
self.horizontalLayout.setObjectName(_fromUtf8("horizontalLayout"))
self.speedL = QtGui.QLabel(self.verticalLayoutWidget)
self.speedL.setMaximumSize(QtCore.QSize(60, 16777215))
self.speedL.setMaximumSize(QtCore.QSize(60, 16777215))
font = QtGui.QFont()
font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
self.speedL.setFont(font)
self.speedL.setObjectName(_fromUtf8("speedL"))
self.horizontalLayout.addWidget(self.speedL)
self.speed = QtGui.QComboBox(self.verticalLayoutWidget)
self.speed.setMinimumSize(QtCore.QSize(60, 16777215))
self.speed.setObjectName(_fromUtf8("speed"))
self.speed.addItem(_fromUtf8(""))
self.speed.addItem(_fromUtf8(""))
self.speed.addItem(_fromUtf8(""))
self.horizontalLayout.addWidget(self.speed)
self.powerL = QtGui.QLabel(self.verticalLayoutWidget)
self.powerL.setMaximumSize(QtCore.QSize(16777215, 16777215))
font = QtGui.QFont()
```

```
        font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
        self.powerL.setFont(font)
        self.powerL.setObjectName(_fromUtf8("powerL"))
        self.horizontalLayout.addWidget(self.powerL)
        self.power = QtGui.QComboBox(self.verticalLayoutWidget)
        self.power.setMinimumSize(QtCore.QSize(70, 16777215))
        self.power.setMaximumSize(QtCore.QSize(70, 16777215))
        self.power.setObjectName(_fromUtf8("power"))
        self.power.addItem(_fromUtf8(""))
        self.power.addItem(_fromUtf8(""))
        self.power.addItem(_fromUtf8(""))
        self.power.addItem(_fromUtf8(""))
        self.horizontalLayout.addWidget(self.power)
        self.power.hide()
        self.channelL = QtGui.QLabel(self.verticalLayoutWidget)
        self.channelL.setMaximumSize(QtCore.QSize(1000, 16777215))
        font = QtGui.QFont()
        font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
        self.channelL.setFont(font)
        self.channelL.setObjectName(_fromUtf8("channelL"))
        self.horizontalLayout.addWidget(self.channelL)
        self.channel = QtGui.QLineEdit(self.verticalLayoutWidget)
        self.channel.setMaximumSize(QtCore.QSize(30, 16777215))
        self.channel.setText(_fromUtf8(""))
        self.channel.setObjectName(_fromUtf8("channel"))
        self.horizontalLayout.addWidget(self.channel)
        spacerItem = QtGui.QSpacerItem(40, 20, QtGui.QSizePolicy.Expanding,
QtGui.QSizePolicy.Minimum)
        self.horizontalLayout.addItem(spacerItem)
        self.verticalLayout.addLayout(self.horizontalLayout)
        self.horizontalLayout_4 = QtGui.QHBoxLayout()
        self.horizontalLayout_4.setMargin(0)
        self.horizontalLayout_4.setSpacing(52)
        self.horizontalLayout_4.setObjectName(_fromUtf8("horizontalLayout_4"))
        self.payloadsizeL = QtGui.QLabel(self.verticalLayoutWidget)
        self.payloadsizeL.setMaximumSize(QtCore.QSize(75, 16777215))
        font = QtGui.QFont()
        font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
        self.payloadsizeL.setFont(font)
        self.payloadsizeL.setObjectName(_fromUtf8("payloadsizeL"))
        self.horizontalLayout_4.addWidget(self.payloadsizeL)
        self.payload_size = QtGui.QLineEdit(self.verticalLayoutWidget)
        self.payload_size.setMaximumSize(QtCore.QSize(30, 16777215))
        self.payload_size.setText(_fromUtf8(""))
        self.payload_size.setObjectName(_fromUtf8("payload_size"))
        self.payload_size.hide()
        self.horizontalLayout_4.addWidget(self.payload_size)
        self.crcL = QtGui.QLabel(self.verticalLayoutWidget)
        self.crcL.setMaximumSize(QtCore.QSize(30, 16777215))
        font = QtGui.QFont()
```

```
        font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
        self.crcL.setFont(font)
        self.crcL.setObjectName(_fromUtf8("crcL"))
        self.horizontalLayout_4.addWidget(self.crcL)
        self.verticalLayout_4 = QtGui.QVBoxLayout()
        self.verticalLayout_4.setMargin(11)
        self.verticalLayout_4.setSpacing(6)
        self.verticalLayout_4.setObjectName(_fromUtf8("verticalLayout_4"))
        self.horizontalLayout_4.addLayout(self.verticalLayout_4)
        self.crc = QtGui.QComboBox(self.verticalLayoutWidget)
        self.crc.setMaximumSize(QtCore.QSize(30, 16777215))
        self.crc.setObjectName(_fromUtf8("crc"))
        self.crc.addItem(_fromUtf8(""))
        self.crc.hide()
        self.horizontalLayout_4.addWidget(self.crc)
        #spacerItem1 = QtGui.QSpacerItem(40, 20, QtGui.QSizePolicy.Expanding,
QtGui.QSizePolicy.Minimum)
        # self.horizontalLayout_4.addItem(spacerItem1)
        self.verticalLayout.addLayout(self.horizontalLayout_4)
        self.horizontalLayout_2 = QtGui.QHBoxLayout()
        self.horizontalLayout_2.setMargin(11)
        self.horizontalLayout_2.setSpacing(6)
        self.horizontalLayout_2.setObjectName(_fromUtf8("horizontalLayout_2"))
        self.rx_adressL = QtGui.QLabel(self.verticalLayoutWidget)
        font = QtGui.QFont()
        font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
        self.rx_adressL.setFont(font)
        self.rx_adressL.setObjectName(_fromUtf8("rx_adressL"))
        self.horizontalLayout_2.addWidget(self.rx_adressL)
        self.rx_adress = QtGui.QLineEdit(self.verticalLayoutWidget)
        self.rx_adress.setObjectName(_fromUtf8("rx_adress"))
        self.horizontalLayout_2.addWidget(self.rx_adress)
        self.rx_adress.setMaximumSize(QtCore.QSize(150, 16777215))
        self.rx_idL = QtGui.QLabel(self.verticalLayoutWidget)
        font = QtGui.QFont()
        font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
        self.rx_idL.setFont(font)
        self.rx_idL.setObjectName(_fromUtf8("rx_idL"))
        self.horizontalLayout_2.addWidget(self.rx_idL)
        self.rx_id = QtGui.QComboBox(self.verticalLayoutWidget)
        self.rx_id.setMaximumSize(QtCore.QSize(30, 16777215))
        self.rx_id.setObjectName(_fromUtf8("rx_id"))
        self.rx_id.addItem(_fromUtf8(""))
        self.rx_id.addItem(_fromUtf8(""))
        self.rx_id.addItem(_fromUtf8(""))
        self.rx_id.addItem(_fromUtf8(""))
        self.rx_id.addItem(_fromUtf8(""))
        self.rx_id.addItem(_fromUtf8(""))
        self.rx_id.hide()
        self.horizontalLayout_2.addWidget(self.rx_id)
```

```
        self.verticalLayout.addLayout(self.horizontalLayout_2)
        self.horizontalLayout_9 = QtGui.QHBoxLayout()
        self.horizontalLayout_9.setMargin(0)
        self.horizontalLayout_9.setSpacing(0)
        self.horizontalLayout_9.setObjectName(_fromUtf8("horizontalLayout_9"))
        self.tx_adressL = QtGui.QLabel(self.verticalLayoutWidget)
        font = QtGui.QFont()
        font.setFamily(_fromUtf8("Arial Rounded MT Bold"))
        self.tx_adressL.setFont(font)
        self.tx_adressL.setObjectName(_fromUtf8("tx_adressL"))
        self.horizontalLayout_9.addWidget(self.tx_adressL)
        self.tx_adressL.setMaximumSize(QtCore.QSize(500, 15151))
        self.tx_adress = QtGui.QLineEdit(self.verticalLayoutWidget)
        self.tx_adress.hide()
        self.tx_adress.setMinimumSize(QtCore.QSize(150, 0))
        self.tx_adress.setMaximumSize(QtCore.QSize(150, 15151))
        self.tx_adress.setObjectName(_fromUtf8("tx_adress"))
        self.horizontalLayout_9.addWidget(self.tx_adress)
        spacerItem2 = QtGui.QSpacerItem(0, 20, QtGui.QSizePolicy.Expanding,
QtGui.QSizePolicy.Minimum)
        self.horizontalLayout_9.addItem(spacerItem2)
        self.verticalLayout.addLayout(self.horizontalLayout_9)
        self.horizontalLayout_5 = QtGui.QHBoxLayout()
        self.horizontalLayout_5.setMargin(11)
        self.horizontalLayout_5.setSpacing(6)
        self.horizontalLayout_5.setObjectName(_fromUtf8("horizontalLayout_5"))
        self.applychangesB = QtGui.QPushButton(self.verticalLayoutWidget)
        self.applychangesB.setMinimumSize(QtCore.QSize(0, 50))
        self.applychangesB.setMaximumSize(QtCore.QSize(16777215, 16777215))
        self.applychangesB.setText(_fromUtf8(""))
        self.applychangesB.setIconSize(QtCore.QSize(16, 32))
        self.applychangesB.setAutoDefault(False)
        self.applychangesB.setDefault(False)
        self.applychangesB.setFlat(False)
        self.applychangesB.setObjectName(_fromUtf8("applychangesB"))
        self.applychangesB.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.applychangesB.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
        self.horizontalLayout_5.addWidget(self.applychangesB)
        self.listen = QtGui.QPushButton(self.verticalLayoutWidget)
        self.listen.setMinimumSize(QtCore.QSize(0, 50))
        self.listen.setBaseSize(QtCore.QSize(0, 0))
        self.listen.setText(_fromUtf8(""))
        self.listen.setObjectName(_fromUtf8("listen"))
        self.listen.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.listen.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
        self.horizontalLayout_5.addWidget(self.listen)
        self.verticalLayout.addLayout(self.horizontalLayout_5)
        spacerIt = QtGui.QSpacerItem(0, 20, QtGui.QSizePolicy.Expanding,
QtGui.QSizePolicy.Minimum)
        self.verticalLayout.addItem(spacerIt)
```

```
        self.horizontalLayout_3 = QtGui.QHBoxLayout()
        self.horizontalLayout_3.setMargin(11)
        self.horizontalLayout_3.setSpacing(6)
        self.horizontalLayout_3.setObjectName(_fromUtf8("horizontalLayout_3"))
        self.focus = QtGui.QPushButton(self.verticalLayoutWidget)
        self.focus.setMinimumSize(QtCore.QSize(0, 50))
        self.focus.setText(_fromUtf8(""))
        self.focus.setObjectName(_fromUtf8("focus"))
        self.focus.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.focus.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
        self.horizontalLayout_3.addWidget(self.focus)
        self.stopB = QtGui.QPushButton(self.verticalLayoutWidget)
        self.stopB.setMinimumSize(QtCore.QSize(0, 50))
        self.stopB.setText(_fromUtf8(""))
        self.stopB.setObjectName(_fromUtf8("stopB"))
        self.stopB.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.stopB.setStyleSheet("background-color: rgba( 255, 255, 255, 0% );" )
        self.horizontalLayout_3.addWidget(self.stopB)
        self.verticalLayout.addLayout(self.horizontalLayout_3)
        self.horizontalLayout_10.addLayout(self.verticalLayout)
        self.verticalLayout_3.addLayout(self.horizontalLayout_10)
        MainWindow.setCentralWidget(self.centralWidget)
        self.statusBar = QtGui.QStatusBar(MainWindow)
        self.statusBar.setObjectName(_fromUtf8("statusBar"))
        MainWindow.setStatusBar(self.statusBar)
        font = QtGui.QFont()
        font.setPixelSize(15)
        self.statusBar.setFont(font)
        self.fons = QtGui.QPixmap('automatic_mode.png')
        self.label.setPixmap(self.fons)
        self.label.setAlignment(QtCore.Qt.AlignCenter)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        MainWindow.setWindowTitle(_translate("MainWindow", "Sniffer app", None))
        self.speedL.setText(_translate("MainWindow", "          ", None))
        self.speed.setItemText(0, _translate("MainWindow", "1 Mbps", None))
        self.speed.setItemText(1, _translate("MainWindow", "2 Mbps", None))
        self.speed.setItemText(2, _translate("MainWindow", "250 Kbps", None))
        self.powerL.setText(_translate("MainWindow", "          ", None))
        self.power.setItemText(0, _translate("MainWindow", "0 dBm", None))
        self.power.setItemText(1, _translate("MainWindow", "-6 dBm", None))
        self.power.setItemText(2, _translate("MainWindow", "-12 dBm", None))
        self.power.setItemText(3, _translate("MainWindow", "-18 dBm", None))
        self.channelL.setText(_translate("MainWindow", "
", None))
        self.channel.setProperty("default value", _translate("MainWindow", "9", None))
        self.payloadsizeL.setText(_translate("MainWindow", "                    ", None))
```

```
self.crcL.setText(_translate("MainWindow", "                ", None))
self.crc.setItemText(0, _translate("MainWindow", "1", None))
self.crc.setItemText(1, _translate("MainWindow", "2", None))
self.rx_adressL.setText(_translate("MainWindow", "                        ", None))
self.rx_idL.setText(_translate("MainWindow", "", None))
self.rx_id.setItemText(0, _translate("MainWindow", "0", None))
self.rx_id.setItemText(1, _translate("MainWindow", "1", None))
self.rx_id.setItemText(2, _translate("MainWindow", "2", None))
self.rx_id.setItemText(3, _translate("MainWindow", "3", None))
self.rx_id.setItemText(4, _translate("MainWindow", "4", None))
self.rx_id.setItemText(5, _translate("MainWindow", "5", None))
self.tx_adressL.setText(_translate("MainWindow", "                          ", None))
self.applychangesB.setShortcut(_translate("MainWindow", "Return", None))
self.listen.setShortcut(_translate("MainWindow", "L", None))
self.focus.setShortcut(_translate("MainWindow", "F", None))
self.stopB.setShortcut(_translate("MainWindow", "Space", None))
```

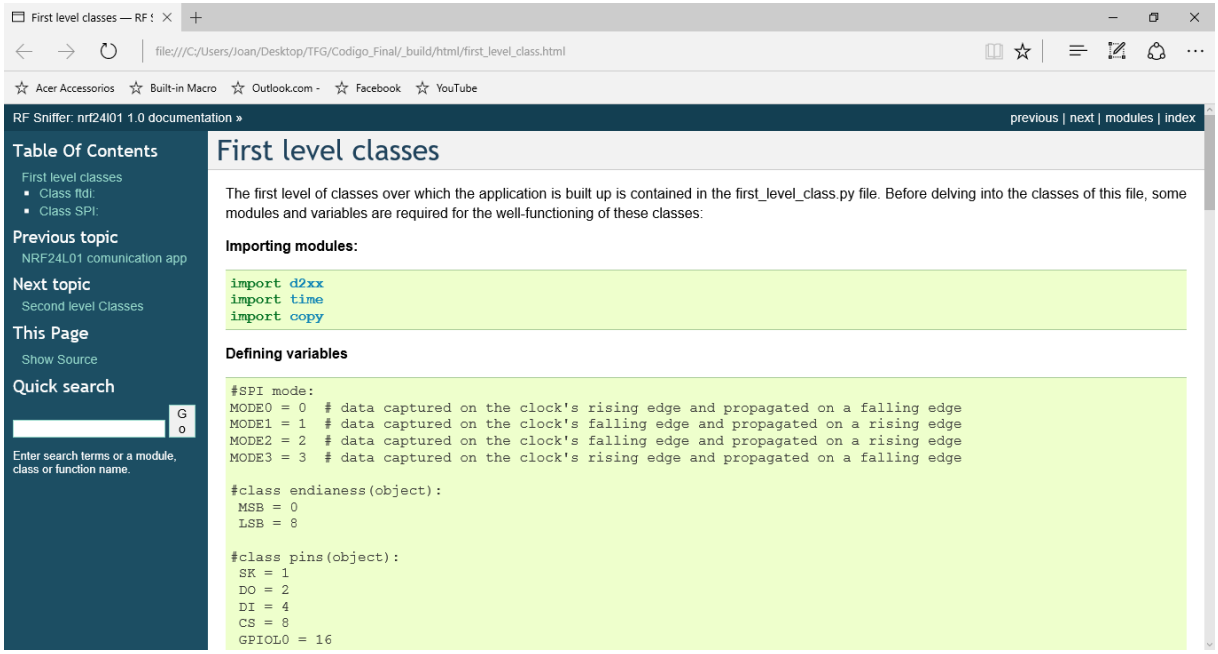
A.7 CRC.py

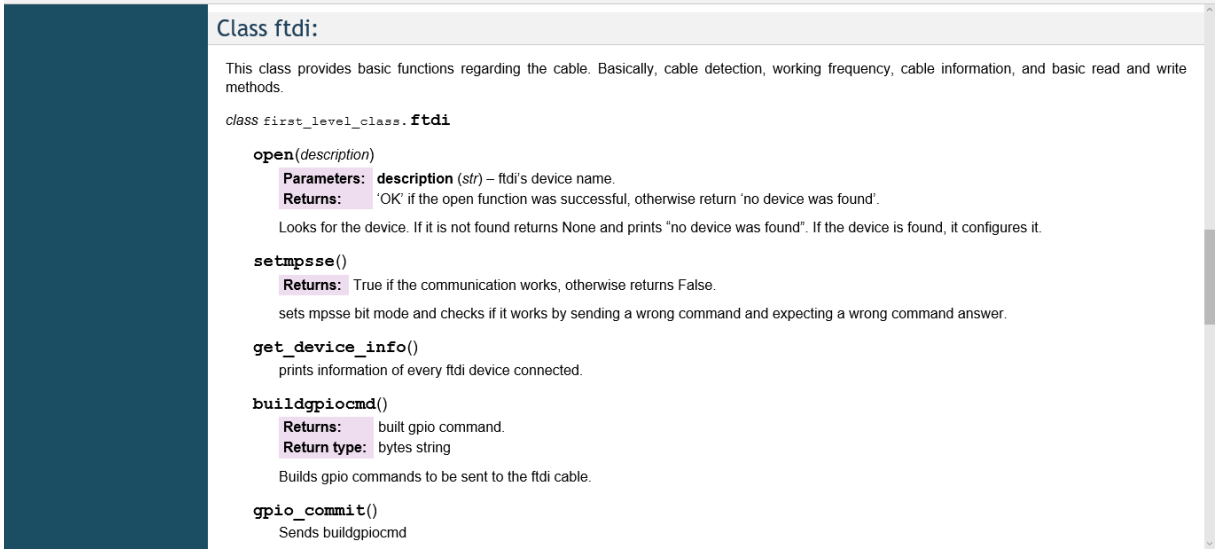
```
from bitstring import BitArray
import time
initial_value_1=BitArray('0xff')
initial_value_2=BitArray('0xffff')
poly_1=BitArray('0b100000111')
poly_2=BitArray('0b10001000000100001')
def crc_(msg,case):
    """
    Optimized version of the NRF24L01's CRC calculation.
    :param BitArray msg: string of bits over wich the CRC is going to be calculated.
    :param int case: 1 or 2 depending on the CRC length.
    :return str: CRC
    """
    if case==1:
        count=0
        value=msg.copy()
        value.append(len(poly_1)-1)
        result = value[:8]^initial_value_1
        value.overwrite(result,0)
        for i in range(len(value)):
            if value[i]:
                del value[0:count]
                break
            else:
                count+=1


        while len(value)>8:
            count=0
            result = value[:9]^poly_1
            value.overwrite(result,0)
            for i in range(len(value)):
                if value[i]:
                    del value[0:count]
                    break
                else:
                    count+=1
        value.prepend(8-len(value))
        return value.bin

    else:
        count=0
        value=msg.copy()
        value.append(len(poly_2)-1)
        result = value[:16]^initial_value_2
        value.overwrite(result,0)
        for i in range(len(value)):
            if value[i]:
```

```
            del value[0:count]
            break
        else:
            count+=1


    while len(value)>16:
        count=0
        result = value[:17]^poly_2
        value.overwrite(result,0)
        for i in range(len(value)):
            if value[i]:
                del value[0:count]
                break
            else:
                count+=1
    value.prepend(16-len(value))
    return value.bin


def crc_func(msg,case):
    """
    :param msg
    :param case
    old version but easier to print and understand how it works
    """
    if case==1:
        count=0
        pol=poly_1.copy()
        init = initial_value_1.copy()
        value=msg.copy()
        value.append('0b'+'0'*(len(poly_1)-1))
        pol.append((len(value)-len(pol)))
        init.append((len(value)-len(initial_value_1)))
        value = value^init
        for i in range(len(value.bin)):
            if int(value.bin[i],2)==0:
                count+=1
            else:
                pol = pol>>count
                break

        while value.bin[:-len(poly_1)+1]!='0'*(len(value)-len(poly_1)+1):
            #print value.bin
            #print pol.bin
            count = 0
            value=value^pol
            pol= poly_1.copy()
            pol.append(len(value)-len(poly_1))
```

```
        for i in range(len(value.bin)):
            if int(value.bin[i],2)==0:
                count+=1
            else:
                pol=pol>>count
                break
        # print 'pol '+pol.bin
    return value.bin[-len(poly_1)+1:]

else:
    count=0
    pol=poly_2.copy()
    init = initial_value_2.copy()
    value=msg.copy()
    value.append('0b'+'0'*(len(poly_2)-1))
    pol.append((len(value)-len(pol)))
    init.append((len(value)-len(initial_value_2)))
    value = value^init
    for i in range(len(value.bin)):
        if int(value.bin[i],2)==0:
            count+=1
        else:
            pol = pol>>count
            break

    while value.bin[:-len(poly_2)+1]!='0'*(len(value)-len(poly_2)+1):
      # print value.bin
       #print pol.bin
        count = 0
        value=value^pol
        pol= poly_2.copy()
        pol.append(len(value)-len(poly_2))

        for i in range(len(value.bin)):
            if int(value.bin[i],2)==0:
                count+=1
            else:
                pol=pol>>count
                break
        # print 'pol '+pol.bin
    return value.bin[-len(poly_2)+1:]
```

Annex B: first_level_class.py documentation screenshots





Class SPI:

This class uses ftdi class to configure the cable to fit the NRF24L01 requirements: CE and CS pins, read and write operations,...

class `first_level_class.` **`SPI`**(*name*, *mode*, *frequency*, *endianness=0*, *cs=8*)

`setupmode`()

Configures when data is captured and propagated.

`startcmd`()

Returns: built init gpio command.
Return type: bytes string.

Initialize SK.

`endcmd`()

Returns: built end gpio command.
Return type: bytes string.

Ends SK.

`write`(*data*)

Parameters: **data** (*str*) – supports bytes' string **only**.

One byte through the TDO cable connection from the ftdi cable.

Note: This function doesn't convert data into a bytes' string because it would complicate the code unnecessarily as some data will be already converted.

`read`()

Annex C: Packet sending - Manual mode code

UPC_nRF24L01.c

```
// Last modified 02/06/2016 Moreno-Rafols

#include<p18f4520.h>
#include<delays.h>
#include"UPC_nRF24L01.h"


//***********************************************************************************
//
// FUNCTION to Initialize SPI Ports and Set SPI Configuration          //
//                                                                     //
//    Clock_Frequency: 0b00 => Fosc/4, 0b01 => Fosc/16, 0b10 => Fosc/64      //
//                                                                     //
//***********************************************************************************
//

void SPI_Start (unsigned char Clock_Frequency)
{
  TRISCbits.TRISC3=0;                 // SCK is an Output, because the PIC18 is the
Master Device and the NRF24l01 is the Slave Device
  TRISCbits.TRISC4=1;                 // SDI (MISO) is an Input
  TRISCbits.TRISC5=0;                 // SDO (MOSI) is an Output

  SSPCON1 = (SSPCON1 & 0xF0) | Clock_Frequency;     // Sets SPI port of the PIC18 as
Master and sets Clock Frequency
                          // 00: Fosc/4, 01: Fosc/16, 10: Fosc/64, 11: TMR2 output/2

  SSPCON1bits.CKP=0;                  // Sets SPI mode:    For the nRF24L01 => Clock
Polarity (CPOL) = 0
  SSPSTATbits.CKE=1;                  // Sets SPI mode:    For the nRF24L01 => Clock
Select Bit, Clock Phase (CPHA) = 1
  SSPSTATbits.SMP=1;                  // Sets SPI mode:    For the nRF24L01 => Sample Bit
(SMP) = 1

  PIE1bits.SSPIE=0;                   // SPI Interruptions configuration:    1: Enable the MSSP
interrupt, 0: Disable the MSSP interrupt
  IPR1bits.SSPIP=0;                   // SPI Interruptions configuration:    1: High Priority, 0:
Low priority
  PIR1bits.SSPIF=0;                   // SPI Interruptions configuration:    1:
```

```
Transmission/Reception Completed, 0: Waiting to transmit/receive

  SSPCON1bits.SSPEN=1;                    // Enable SPI Communication
}


//*************************************************************************//
// FUNCTION to Initialize PIC18 Ports needed for nRF24L01 Communication  //
//*************************************************************************//

void nRF24L01_Ports_Start(void)
{
  ADCON1=0x0F;                      // As PORTA bits 2 to 4 are used as Digital Ports,
ADCON1 can be only 0x0F, 0x0E or 0x0D

  TRISAbits.TRISA2=0;                 // CE is an Output Pin
  TRISAbits.TRISA3=0;                // CSN is an Output Pin
  TRISAbits.TRISA4=1;                // IRQ is an Input Pin

  CSN=1;                        // At program start CSN must be set High (Inactive) => Signal
used for Starting a new SPI Communication
  CE=0;                         // At program start CE must be set Low (Inactive) => Signal
used for turning Standby into RX or TX Mode
}


//*************************************************//
// FUNCTION to Send and Receive Data throught SPI  //
//*************************************************//

unsigned char SPI_Transfer(unsigned char byte_to_send)
{
  SSPBUF = byte_to_send;              // Byte to Send is loaded in the SSP Buffer, when
nRF24L01 is ready it will be exchanged for an Incoming Byte

  while(PIR1bits.SSPIF==0);           // Wait until exchange is done
  PIR1bits.SSPIF=0;                  // Clear SPI Interrupt Flag

  return(SSPBUF);                    // We return the Received Byte
}


/*************************************************************************
*************************************************************************
************************************/
```

```
//*********************************************************************//
// FUNCTIONS to Read and Write nRF24L01 Registers (1 Byte Registers)  //
//*********************************************************************//

// Introduce nRF24L01 Register_Address to Read

unsigned char Read_nRF24L01_Register (unsigned char Register_Address)
{
  unsigned char Register_Content, nRF24L01_Status;

  CSN=0;                                  // ACTIVATE CSN
  nRF24L01_Status=SPI_Transfer(R_REGISTER+Register_Address);      // Send
Instruction Word while Receiving nRF24L01_Status
  Register_Content=SPI_Transfer(0x00);              // Receive Data Byte while sending
dummydata (ignored by the nRF24L01)
  CSN=1;                                  // DEACTIVATE CSN
  return(Register_Content);                       // RETURN Register Content
}

// Introduce nRF24L01 Register_Address and Register_Content to be loaded

void Write_nRF24L01_Register (unsigned char Register_Address, unsigned char
Register_Content)
{
  unsigned char dummydata, nRF24L01_Status;

  CSN=0;                                  // ACTIVATE CSN
  nRF24L01_Status=SPI_Transfer(W_REGISTER+Register_Address);      // Send
Instruction Word while Receiving nRF24L01_Status
  dummydata=SPI_Transfer(Register_Content);          // Send Data Byte while receiving
dummydata (discarded by the PIC18)
  CSN=1;                                  // DEACTIVATE CSN

  while(Read_nRF24L01_Register(Register_Address)!=Register_Content)  // Check if the
Register was well loaded, if it was not, repeat the process
  {
    CSN=0;
    nRF24L01_Status=SPI_Transfer(W_REGISTER+Register_Address);
    dummydata=SPI_Transfer(Register_Content);
    CSN=1;
  }
}

// Funtion to Update and Return nRF24L01 Status
```

```
unsigned char Read_nRF24L01_Status (void)
{
  unsigned char nRF24L01_Status;

  CSN=0;                              // ACTIVATE CSN
  nRF24L01_Status=SPI_Transfer(NOP);          // Send Instruction Word while
Receiving nRF24L01_Status
  CSN=1;                              // DEACTIVATE CSN

  return(nRF24L01_Status);
}

// Introduce Register_Content to be loaded on the STATUS Register

void Write_nRF24L01_Status (unsigned char Register_Content)
{
  unsigned char dummydata, nRF24L01_Status;

  CSN=0;                              // ACTIVATE CSN
  nRF24L01_Status=SPI_Transfer(W_REGISTER+STATUS);      // Send Instruction
Word (Write STATUS Register) while Receiving nRF24L01_Status
  dummydata=SPI_Transfer(Register_Content);       // Send Data Byte while receiving
dummydata (discarded by the PIC18)
  CSN=1;                              // DEACTIVATE CSN
}


//*************************************************************************
**********//
// FUNCTIONS to Read and Write nRF24L01 Address Registers (3, 4 or 5 Byte Registers)  //
//*************************************************************************
**********//

// Introduce TX_RX_Address_Width (from 0b01 to 0b11: from 3 to 5 bytes long) and
Register_Address to Read, also a variable to load the Register_Content (used for
RX_ADDR_P0, RX_ADDR_P1, TX_ADDR registers)

void Read_nRF24L01_Address_Register (unsigned char TX_RX_Address_Width, unsigned
char Register_Address, unsigned char *Register_Content)
{
  unsigned char i, nRF24L01_Status;

  CSN=0;                                  // ACTIVATE CSN
  nRF24L01_Status=SPI_Transfer(R_REGISTER+Register_Address);          //
Send Instruction Word while Receiving nRF24L01_Status
  for(i=0; i<(TX_RX_Address_Width+0b10); i++)                  // SPI
exchange of data has to be done for each byte
```

```
  {
    Register_Content[i]=SPI_Transfer(0x00);                    // Receive Data
Bytes while sending dummydata (ignored by the nRF24L01)
  }
  CSN=1;                                                       // DEACTIVATE CSN
}

// Introduce TX_RX_Address_Width (from 0b01 to 0b11: from 3 to 5 bytes long) and
Register_Address and Register_Content to be loaded (used for RX_ADDR_P0,
RX_ADDR_P1, TX_ADDR registers)

void Write_nRF24L01_Address_Register (unsigned char TX_RX_Address_Width,
unsigned char Register_Address, unsigned char *Register_Content)
{
  unsigned char dummydata, i, nRF24L01_Status;
  unsigned char Process_Finished;
  unsigned char Address_Verification[5];

  Process_Finished=0;                                          // At funtion start the variable
is set low, so process is not finished yet
  while(!Process_Finished)                                     // The process of writing the
Address is repeated until it is confirmed
  {
    CSN=0;                                                     // ACTIVATE CSN
    nRF24L01_Status=SPI_Transfer(W_REGISTER+Register_Address);                    //
Send Instruction Word while Receiving nRF24L01_Status
    for(i=0; i<(TX_RX_Address_Width+0b10); i++)                // SPI
exchange of data has to be done for each byte
    {
      dummydata=SPI_Transfer(Register_Content[i]);             // Send Data Byte
while receiving dummydata (discarded by the PIC18)
    }
    CSN=1;                                                     // DEACTIVATE CSN

    Read_nRF24L01_Address_Register(TX_RX_Address_Width, Register_Address,
Address_Verification);   // The address is read to be sure it was well writen
    Process_Finished=1;
    for(i=0; i<(TX_RX_Address_Width+0b10); i++)                // Every
address byte that was read is compared with the one defined by the user
    {
      if(Register_Content[i]!=Address_Verification[i])
      {
        Process_Finished=0;                                    // When there is a mismatch
between them, the variable is set low in order to repeat process
      }
    }
  }
```

```
}


//*********************************************************************//
// FUNCTIONS to Read and Write nRF24L01 Payloads (1 to 32 Byte FIFO)  //
//*********************************************************************//

// Function to Read Data received in RX FIFO. If Enable_Checksum is Enabled, a Checksum
byte is also received and evaluated. Funtion returns 1 if RX Payload is correct.  (Used in RX
Mode only)

unsigned char Read_nRF24L01_RX_Payload (unsigned char Enable_Checksum, unsigned
char TX_RX_Payload_Width, unsigned char *RX_Payload)
{
  unsigned char nRF24L01_Status, TX_Checksum;                    // Two Checksums
are defined: TX_Checksum is the one received from TX Device
  unsigned char RX_Checksum, Checksum_Conclusion;                 // RX_Checksum
is the one calculated by the RX Device. Both are compared to verify transmission.
  unsigned char i;

  CSN=0;                                    // ACTIVATE CSN
  nRF24L01_Status=SPI_Transfer(R_RX_PAYLOAD);                    // Send
Instruction Word while Receiving nRF24L01_Status
  if(!Enable_Checksum)                      // Operation in case that Chechsum
Byte is NOT used
  {
    for(i=0; i<TX_RX_Payload_Width; i++)                         // For the whole Payload
Width
    {
      RX_Payload[i]=SPI_Transfer(0x00);                          // Receive Data Bytes while
sending dummydata (ignored by the nRF24L01)
    }
    Checksum_Conclusion=1;                  // RX_Payload is assumed to be
correct
  }
  else if(Enable_Checksum)                  // Operation in case that Chechsum
Byte is used
  {
    RX_Checksum=0b00000000;
    for(i=0; i<TX_RX_Payload_Width; i++)                         // For the whole Payload
Width
    {
      RX_Payload[i]=SPI_Transfer(0x00);                          // Receive Data Bytes while
sending dummydata (ignored by the nRF24L01)
      RX_Checksum=RX_Checksum+RX_Payload[i];                     // Update RX
Checksum byte with every Data Byte received
```

```
    }
    TX_Checksum=SPI_Transfer(0x00);                    // Receive Checksum Byte
from TX Device(an extra bit for Receiveing Payload)

    if(RX_Checksum==TX_Checksum)
    {
      Checksum_Conclusion=1;                           // RX_Payload is assumed to be
correct
    }
    else if(RX_Checksum!=TX_Checksum)
    {
      Checksum_Conclusion=0;                           // RX_Payload is assumed NOT to be
correct
    }

  }
  CSN=1;                                   // DEACTIVATE CSN

  return Checksum_Conclusion;
}

// Function to Write Data to send in TX FIFO. If Enable_Checksum is Enabled, a Checksum
byte is also sent (Used in TX Mode only)

void Write_nRF24L01_TX_Payload (unsigned char Enable_Checksum, unsigned char
TX_RX_Payload_Width, unsigned char *TX_Payload)
{
  unsigned char nRF24L01_Status, TX_Checksum;
  unsigned char i, dummydata;

  CSN=0;                                   // ACTIVATE CSN
  nRF24L01_Status=SPI_Transfer(W_TX_PAYLOAD);                    // Send
Instruction Word while Receiving nRF24L01_Status
  if(!Enable_Checksum)                             // Operation in case that Chechsum
Byte is NOT used
  {
    for(i=0; i<TX_RX_Payload_Width; i++)                 // For the whole Payload
Width
    {
      dummydata=SPI_Transfer(TX_Payload[i]);             // Send Data Byte while
receiving dummydata (discarded by the PIC18)
    }
  }
  else if(Enable_Checksum)                         // Operation in case that Chechsum
Byte is used
  {
    TX_Checksum=0b00000000;
```

```
    for(i=0; i<TX_RX_Payload_Width; i++)                    // For the whole Payload
Width
    {
      dummydata=SPI_Transfer(TX_Payload[i]);                // Send Data Byte while
receiving dummydata (discarded by the PIC18)
      TX_Checksum=TX_Checksum+TX_Payload[i];                // Update Checksum
byte with every Data Byte form TX_Payload is sent
    }
    dummydata=SPI_Transfer(TX_Checksum);                    // Send Checksum Byte
Result, an extra bit for Sending Payload
  }
  CSN=1;                                    // DEACTIVATE CSN
}



/*************************************************************************************
*************************************************************************************
************************************/



//************************************************************************************
************//
// FUNCTION to Reset the nRF24L01 Status Register and Flush TX and RX Payload
Memories  //
//************************************************************************************
************//

void Reset_nRF24L01_Status_and_nRF24L01_Payloads (void)
{
  unsigned char nRF24L01_Status;

  CSN=0;                                    // ACTIVATE CSN
  nRF24L01_Status=SPI_Transfer(FLUSH_TX);               // Send Instruction to Flush
TX while receiving nRF24L01 Status
  CSN=1;                                    // DEACTIVATE CSN

  Delay1TCY();                              // Minimum CSN=0 delay

  CSN=0;                                    // ACTIVATE CSN
  nRF24L01_Status=SPI_Transfer(FLUSH_RX);               // Send Instruction to Flush
RX while receiving nRF24L01 Status
  CSN=1;                                    // DEACTIVATE CSN

  Write_nRF24L01_Status(RX_DR);                     // Delete RX_DR flag if it is High
  Write_nRF24L01_Status(TX_DS);                     // Delete TX_DS flag if it is High
```

```
  Write_nRF24L01_Status(MAX_RT);                    // Delete MAX_RT flag if it is High
}


//*************************************************************************************
************************************//
// FUNCTION to Start RX Mode and Turn ON the nRF24L01                                 //
//                                                                                    //
//   TX_RX_Address_Width: 0b01 => 3 bytes, 0b10 => 4 bytes, 0b11 => 5 bytes           //
//                                                                                    //
//   Frequency_Channel: from 0b0000000 to 0b1111111                                   //
//   RF_Data_Rate: 0    => 1Mbps, 1 => 2Mbps                                          //
//   RF_Output_Power: 0b00 => -18dBm, 0b01 => -12dBm, 0b10 => -6dBm, 0b11 =>
0dBm                 //
//   LNA_Gain: 0 => Disabled, 1 => Enabled                                            //
//   CRC_Setup: 0 => 1byte, 1 => 2byte                                                //
//                                                                                    //
//   RX_Pipe0_Address: from 3 to 5 bytes                                              //
//   RX_Pipe1_Address: from 3 to 5 bytes                                              //
//   RX_Pipe2_Address: 1 byte                                                         //
//   RX_Pipe3_Address: 1 byte                                                         //
//   RX_Pipe4_Address: 1 byte                                                         //
//   RX_Pipe5_Address: 1 byte                                                         //
//                                                                                    //
//   Enable_Checksum: 0=> Disable Checksum byte, 1 => Enable 1 byte Checksum for RX
Payload           //
//                                                                                    //
//   If Enable_Checksum = 0                                                           //
//      RX_Pipe0_Payload_Width: from 1 to 32                                          //
//      RX_Pipe1_Payload_Width: from 0 to 32 (0 if Disabled)                          //
//      RX_Pipe2_Payload_Width: from 0 to 32 (0 if Disabled)                          //
//      RX_Pipe3_Payload_Width: from 0 to 32 (0 if Disabled)                          //
//      RX_Pipe4_Payload_Width: from 0 to 32 (0 if Disabled)                          //
//      RX_Pipe5_Payload_Width: from 0 to 32 (0 if Disabled)                          //
//      (Summatory of RX_Pipe(i)_Payload_Width <= 32)                                 //
//                                                                                    //
//   If Enable_Checksum = 1                                                           //
//      RX_Pipe0_Payload_Width: from 1 to 31                                          //
//      RX_Pipe1_Payload_Width: from 0 to 31 (0 if Disabled)                          //
//      RX_Pipe2_Payload_Width: from 0 to 31 (0 if Disabled)                          //
//      RX_Pipe3_Payload_Width: from 0 to 31 (0 if Disabled)                          //
//      RX_Pipe4_Payload_Width: from 0 to 31 (0 if Disabled)                          //
//      RX_Pipe5_Payload_Width: from 0 to 31 (0 if Disabled)                          //
//      (Summatory of RX_Pipe(i)_Payload_Width <= 26)                                 //
//                                                                                    //
//*************************************************************************************
************************************//
```

```
void Start_RX_Mode_nRF24L01 ( unsigned char TX_RX_Address_Width, unsigned char
Frequency_Channel, unsigned char RF_Data_Rate, unsigned char RF_Output_Power,
                unsigned char LNA_Gain, unsigned char CRC_Setup, unsigned
char *RX_Pipe0_Address, unsigned char *RX_Pipe1_Address,
                unsigned char RX_Pipe2_Address, unsigned char RX_Pipe3_Address,
unsigned char RX_Pipe4_Address, unsigned char RX_Pipe5_Address,
                unsigned char Enable_Checksum, unsigned char RX_P0_Payload_Width,
unsigned char RX_P1_Payload_Width, unsigned char RX_P2_Payload_Width,
                unsigned char RX_P3_Payload_Width, unsigned char
RX_P4_Payload_Width, unsigned char RX_P5_Payload_Width )
{
  CE=0;                                    // We rest in Standby I Mode (Low
Consumption) until all the configuration is set

  Reset_nRF24L01_Status_and_nRF24L01_Payloads();          // Reset the
nRF24L01 Status Register and Flush TX and RX Payload Memories

  Write_nRF24L01_Register (EN_AA, 0b00111111);          // Enable Auto
Acknowledgement for all RX Pipes
  Write_nRF24L01_Register (EN_RXADDR, 0b00111111);          // Enable
all RX Pipes
  Write_nRF24L01_Register (SETUP_AW, TX_RX_Address_Width);          //
Set Width for all Addresses: 0b01 => 3 bytes, 0b10 => 4 bytes, 0b11 => 5 bytes
  Write_nRF24L01_Register (RF_CH, Frequency_Channel);          // Set the
frequency channel nRF24L01 operates on
  Write_nRF24L01_Register (RF_SETUP, RF_Data_Rate*0b1000 +
RF_Output_Power*0b10 + LNA_Gain);    // Set nRF24L01 RF Data Rate, Power and LNA
Gain

  Write_nRF24L01_Address_Register (TX_RX_Address_Width, RX_ADDR_P0,
RX_Pipe0_Address);       // Define RX Address for Pipe0 (3, 4 or 5 bytes)
  Write_nRF24L01_Address_Register (TX_RX_Address_Width, RX_ADDR_P1,
RX_Pipe1_Address);       // Define RX Address for Pipe1 (3, 4 or 5 bytes)
  Write_nRF24L01_Register (RX_ADDR_P2, RX_Pipe2_Address);          //
Define RX Address for Pipe2 (1 byte => MSBytes will be equal to RX_ADDR_P1)
  Write_nRF24L01_Register (RX_ADDR_P3, RX_Pipe3_Address);          //
Define RX Address for Pipe3 (1 byte => MSBytes will be equal to RX_ADDR_P1)
  Write_nRF24L01_Register (RX_ADDR_P4, RX_Pipe4_Address);          //
Define RX Address for Pipe4 (1 byte => MSBytes will be equal to RX_ADDR_P1)
  Write_nRF24L01_Register (RX_ADDR_P5, RX_Pipe5_Address);          //
Define RX Address for Pipe5 (1 byte => MSBytes will be equal to RX_ADDR_P1)

  Write_nRF24L01_Register (RX_PW_P0, RX_P0_Payload_Width+Enable_Checksum);
// Define Payload Width for Pipe0 in RX Mode (extra byte if Checksum is Enabled)
  Write_nRF24L01_Register (RX_PW_P1, RX_P1_Payload_Width+Enable_Checksum);
// Define Payload Width for Pipe1 in RX Mode (extra byte if Checksum is Enabled)
```

```
  Write_nRF24L01_Register (RX_PW_P2, RX_P2_Payload_Width+Enable_Checksum);
// Define Payload Width for Pipe2 in RX Mode (extra byte if Checksum is Enabled)
  Write_nRF24L01_Register (RX_PW_P3, RX_P3_Payload_Width+Enable_Checksum);
// Define Payload Width for Pipe3 in RX Mode (extra byte if Checksum is Enabled)
  Write_nRF24L01_Register (RX_PW_P4, RX_P4_Payload_Width+Enable_Checksum);
// Define Payload Width for Pipe4 in RX Mode (extra byte if Checksum is Enabled)
  Write_nRF24L01_Register (RX_PW_P5, RX_P5_Payload_Width+Enable_Checksum);
// Define Payload Width for Pipe5 in RX Mode (extra byte if Checksum is Enabled)

  Write_nRF24L01_Register (CONFIG, EN_CRC + CRC_Setup*0b100 + PWR_UP +
PRIM_RX);          // nRF24L01 Configuration: RX Mode, Power Up and 2_byte CRC
Encoding
  Delay100TCYx(15);                                    // TIMING CONDITION #1:
Delay required between Power Down and Stanby Mode I => 1.5 ms
}


//*************************************************************************
**********************************//
// FUNCTION to Start TX Mode and Turn ON the nRF24L01                      //
//                                                                         //
//   TX_RX_Address_Width: 0b01 => 3 bytes, 0b10 => 4 bytes, 0b11 => 5 bytes
//
//   Frequency_Channel: from 0b0000000 to 0b1111111                        //
//   RF_Data_Rate: 0    => 1Mbps, 1 => 2Mbps                               //
//   RF_Output_Power: 0b00 => -18dBm, 0b01 => -12dBm, 0b10 => -6dBm, 0b11 =>
0dBm                //
//   LNA_Gain: 0 => Disabled, 1 => Enabled                                 //
//   CRC_Setup: 0 => 1byte, 1 => 2byte                                     //
//                                                                         //
//   Auto_Retransmit_Delay: from 0b0000 to 0b1111                          //
//   Max_Auto_Retransmit: 0 => Disable Auto_Retransmit, from 1 to 15 => Number of
Auto_Retransmit Allowed   //
//                                                                         //
//   Enable_Checksum: 0=> Disable Checksum byte, 1 => Enable 1 byte Checksum for TX
Payload          //
//                                                                         //
//   TX_RX_Payload_Width: If Enable_Checksum = 0,  from 1 to 32            //
//              If Enable_Checksum = 1,  from 1 to 31                      //
//                                                                         //
//*************************************************************************
**********************************//

void Start_TX_Mode_nRF24L01 ( unsigned char TX_RX_Address_Width, unsigned char
Frequency_Channel,   unsigned char RF_Data_Rate,        unsigned char
RF_Output_Power,
                    unsigned char LNA_Gain,        unsigned char CRC_Setup,        unsigned
```

```
char Auto_Retransmit_Delay,   unsigned char Max_Auto_Retransmit,
                unsigned char Enable_Checksum,    unsigned char TX_RX_Payload_Width
)
{
  CE=0;                                    // We rest in Standby I Mode (Low
Consumption) until all the configuration is set

  Reset_nRF24L01_Status_and_nRF24L01_Payloads();                    // Reset the
nRF24L01 Status Register and Flush TX and RX Payload Memories

  Write_nRF24L01_Register (EN_AA, 0b00111111);                    // Enable Auto
Acknowledgement for all RX Pipes
  Write_nRF24L01_Register (EN_RXADDR, 0b00111111);                    // Enable
all RX Pipes
  Write_nRF24L01_Register (SETUP_AW, TX_RX_Address_Width);                    //
Set Width for all Addresses: 0b01 => 3 bytes, 0b10 => 4 bytes, 0b11 => 5 bytes
  Write_nRF24L01_Register (SETUP_RETR, Auto_Retransmit_Delay*0b10000 +
Max_Auto_Retransmit);   // Automatic Retransmission Setup: Delay between and Maximum
number
  Write_nRF24L01_Register (RF_CH, Frequency_Channel);                    // Set the
frequency channel nRF24L01 operates on
  Write_nRF24L01_Register (RF_SETUP, RF_Data_Rate*0b1000 +
RF_Output_Power*0b10 + LNA_Gain);   // Set nRF24L01 RF Data Rate, Power and LNA
Gain

  Write_nRF24L01_Register (RX_PW_P0, TX_RX_Payload_Width+Enable_Checksum);
// Define Payload Width for Pipe0 (extra byte if Checksum is Enabled)

  Write_nRF24L01_Register (CONFIG,EN_CRC + CRC_Setup*0b100 + PWR_UP +
0*PRIM_RX);          // nRF24L01 Configuration: TX Mode, Power Up and 2_byte CRC
Encoding

  Write_nRF24L01_Register (DYNPD,0x01);  // 02/06/2016 Moreno-Rafols   Enable
Dynamic Payload in Pipe

  Write_nRF24L01_Register (FEATURE,0x04);   // 02/06/2016 Moreno-Rafols Enable
Dynamic Payload Lenght


  Delay100TCYx(15);                                    // TIMING CONDITION #1:
Delay required between Power Down and Stanby Modes => 1.5 ms
}
```

```
/*************************************************************************
*************************************************************************
```

```
**********************************/



//*****************************************************************************
*****************************************//
// FUNCTION to Check if nRF24L01 Received Data, Obtain RX Data and Identify Pipe (Used
only in RX Mode)          //
//                                                                             //
//   Enable_Checksum: 0=> Disable Checksum byte, 1 => Enable 1 byte Checksum for TX
Payload            //
//                                                                             //
//   Max_Waiting_ds: from 0 to 255 (DeciSeconds permited until Waiting IRQ Error is
considered          //
//                                                                             //
//   TX_RX_Payload_Width: from 1 to 32                                         //
//   RX_Payload: from 1 to 32 bytes                                            //
//                                                                             //
//   RETURNS:   Bit 0 (Checksum_Conlusion):      1 => Checksum was Disabled or Succeed,
0 => Checksum Failed      //
//          Bit 1 to 3 (Origin_Pipe):     Data Pipe Number, from 0b000 to 0b101       //
//          Bit 4 (IRQ_Error):        1 => IRQ Error Ocurred, 0 => IRQ worked correctly
//
//                                                                             //
//*****************************************************************************
*****************************************//

unsigned char Receive_Data_RX_Mode_nRF24L01 (unsigned char Enable_Checksum,
unsigned char Max_Waiting_ds, unsigned char TX_RX_Payload_Width, unsigned char
*RX_Payload)
{
  unsigned char nRF24L01_Status;
  unsigned char Origin_Pipe;
  unsigned char Checksum_Conclusion, IRQ_Error;
  unsigned int i;
  unsigned char j;

  IRQ_Error=0;
  CE=1;                                          // Activate CE to leave Standby I mode
and enter RX Mode (nRF24L01 searchs incoming signal)
  Delay10TCYx(13);                                  // TIMING CONDITION #2:
Delay required between Stanby Modes and RX or TX Mode => 130 us
  Delay10TCYx(1);                                   // TIMING CONDITION #3:
Minimum CE High => 10 us

  i=0; j=0;
  while(IRQ)                                     // Wait until RX_DR is set High
```

```
(Reception of Data), and then IRQ is set Low.
  {                                          // Sometimes the nRF24L01 fails to do this,
so a maximum waiting time is needed
    i++;
    if(i==3333)                                  // i reaches 3333 after 100000us =
100ms
    {                                          // IRQ is analized every 30us
      i=0; j++;
    }
    if(j==Max_Waiting_ds)                        // TIMING CONDITION #5:
When we reach the Max_Waiting_ds defined by user, loop is then broken.
    {                                          // If the device did not receive data before
Max_Waiting_ds, IRQ ERROR is considered.
      IRQ_Error=1;                             // The Result of the TX operation
was: IRQ ERROR
      break;
    }
  }
  CE=0;                                    // Deactivate CE to leave RX Mode (Low
Current Consumtion)
  Delay10TCYx(1);                              // TIMING CONDITION #4:
Delay required between CE edge and CSN low  => 4 us

  nRF24L01_Status=Read_nRF24L01_Status();                  // Update the
nRF24L01 Status to get the Origin_Pipe
  Origin_Pipe=(nRF24L01_Status & RX_P_NO);                 // Data Pipe
Number is obtained from the STATUS Register

Checksum_Conclusion=Read_nRF24L01_RX_Payload(Enable_Checksum,TX_RX_Payload
_Width, RX_Payload);    // After Reading the RX FIFO, all data is deleted from FIFO.
  Write_nRF24L01_Status(RX_DR);                    // Clear the RX_DR bit
from the nRF24L01_Status Register

  return (IRQ_Error*0b10000+Origin_Pipe+Checksum_Conclusion);          //
IRQ_Error, Checksum_Result and Data Pipe Number will be returned: from 0b000 to 0b101
(6 RX Pipes)
}


//*****************************************************************************
*************************************************************//
// FUNCTION to Define TX Address, Load Data to transmit into nRF24L01 and Enable TX
Mode starting transmission  (Used only in TX Mode)    //
//                                                                  //
//   Enable_Checksum: 0=> Disable Checksum byte, 1 => Enable 1 byte Checksum for TX
Payload                    //
//                                                                  //
```

```
//   TX_RX_Address_Width: 0b01 => 3 bytes, 0b10 => 4 bytes, 0b11 => 5 bytes
//
//   TX_Address: from 3 to 5 bytes                                          //
//   TX_RX_Payload_Width: from 1 to 32                                        //
//   TX_Payload: from 1 to 32 bytes                                          //
//                                                                  //
//**************************************************************************
************************************************************//

void Send_Data_TX_Mode_nRF24L01(unsigned char Enable_Checksum, unsigned char
TX_RX_Address_Width, unsigned char *TX_Address, unsigned char
TX_RX_Payload_Width, unsigned char *TX_Payload)
{
  Write_nRF24L01_Address_Register (TX_RX_Address_Width, TX_ADDR, TX_Address);
// Define TX Address (3, 4 or 5 bytes)
  Write_nRF24L01_Address_Register (TX_RX_Address_Width, RX_ADDR_P0,
TX_Address);      // Define RX Address for Pipe0: equal to TX Address to receive Auto
Acknowledgement (3, 4 or 5 bytes)

  Write_nRF24L01_TX_Payload (Enable_Checksum, TX_RX_Payload_Width,
TX_Payload);      // Load Data into nRF24L01 TX FIFO, it will be sent after CE is set High

  CE=1;                                  // Activate CE to leave Standby I mode and
enter RX Mode (nRF24L01 searchs incoming signal)
  Delay10TCYx(13);                                 // TIMING CONDITION #2: Delay
required between Stanby Modes and RX or TX Mode => 130 us
  Delay10TCYx(1);                                  // TIMING CONDITION #3:
Minimum CE High => 10 us
  CE=0;                                  // Deactivate CE to leave RX Mode (Low
Current Consumtion)
  Delay10TCYx(1);                                  // TIMING CONDITION #4: Delay
required between CE edge and CSN low  => 4 us
}


//**************************************************************************
************************************************************//
// FUNCTION to Check if Data was sent (Auto-Acknowledgemente Received) or Error on
IRQ occurred. Also how many retries were needed to succeed  //
//                                                                  //
//   RETURNS:   Bits 1 to 0:   0b01 => Data was correctly sent, 0b10 => Data was not
correctly sent, 0b11 => IRQ ERROR                //
//           Bits 5 to 2:   from 0 to 15 Retransmissions                          //
//                                                                  //
//**************************************************************************
************************************************************//
```

```
unsigned char Check_Data_Sent_TX_Mode_nRF24L01 (void)              // Funtion to
obtain TX Operation Result
{
  unsigned char TX_Operation_Result, TX_Retransmit_Counter;
  unsigned char nRF24L01_Status;
  unsigned int i;
  unsigned char j;

  i=0; j=0;
  while(IRQ)                              // We wait until TX_DS or MAX_RT is set High,
and then IRQ is set Low.
  {                                       // Sometimes the nRF24L01 fails to do this, so a
maximum waiting time is needed => 70ms                                  // Sometimes the
nRF24L01 fails to do this, so a maximum waiting time is needed
    i++;
    if(i==333)                            // i reaches 333 after 10000us = 10ms
    {                                     // IRQ is analized every 30us
      i=0; j++;
    }
    if(j==7)                              // TIMING CONDITION #6: When counter reaches
70ms, it is confirm that nRF24L01 failed
    {
      TX_Operation_Result=0b11;           // The Result of the TX operation was:
IRQ ERROR
      break;
    }
  }

  while(!IRQ)                             // Process only continues if the proper bit,
TX_DS or MAX_RT, was set low.
  {
    nRF24L01_Status=Read_nRF24L01_Status();          // Update the nRF24L01
Status to get cause of the IRQ

    if(nRF24L01_Status & MAX_RT)          // If MAX_RT was set high
    {
      TX_Operation_Result=0b00;           // 0b00 => Data was not correctly sent
to RX Device
      TX_Retransmit_Counter=Read_nRF24L01_Register(OBSERVE_TX) & 0x0F;  //
Obtain the ARC_CNT: Retransmission Counter reset after every new packet transmission
      Write_nRF24L01_Status(MAX_RT);      // We clear the MAX_RT bit from
the nRF24L01_Status Register writing a 1.
    }
    else if(nRF24L01_Status & TX_DS)      // If TX_DS was set high
    {
      TX_Operation_Result=0b01;           // 0b01 => Data was correctly sent to
RX Device
```

```
        TX_Retransmit_Counter=Read_nRF24L01_Register(OBSERVE_TX) & 0x0F;   //
Obtain the ARC_CNT: Retransmission Counter reset after every new packet transmission
        Write_nRF24L01_Status(TX_DS);                    // We clear the TX_DS bit from the
nRF24L01_Status Register writing a 1.
      }
    }

    return (TX_Retransmit_Counter*0b100 + TX_Operation_Result);
}



/*************************************************************************************
*************************************************************************************
************************************/



//************************************************************************************
****************//
// FUNCTION to Turn Off the nRF24L01 by setting low the PWR_UP bit of the CONFIG
Register  //
//************************************************************************************
****************//

void Finish_nRF24L01_Operation (void)
{
    unsigned char previous_config_register;
    previous_config_register=Read_nRF24L01_Register(CONFIG);          // Read    the
previous CONFIG Register to change just the PWR_UP bit
    Write_nRF24L01_Register(CONFIG, previous_config_register & 0b11111101);      // Set
Power Up bit Low
}



//***********************************//
// FUNCTION to Disable the SPI port    //
//***********************************//

void Finish_SPI_Operation (void)
{
    SSPCON1bits.SSPEN=0;                          // Disable the Enable Bit of the MSSP
Control Register
}
```

```
/*****************************************************************************
*****************************************************************************
************************************/
/*****************************************************************************
*****************************************************************************
************************************/
```

nRF24L01_LED_Control.c (1)

```
#include<p18f4520.h>
#include<delays.h>
#include"config.h"
#include"UPC_nRF24L01.h"

  unsigned char Fifo[32];
  unsigned char in;

#define CHECKSUM 0

//#define TRANSMITTER
#define RECEIVER

//#ifdef TRANSMITTER & RECEIVE
/*
 *
 *  C?DIGO DE TEST Y VALIDACI?N DEL nRF24L01
 *  (Comentar o descomentar una de las dos fuciones main() a la hora de compilar o
programar)
 *
 *
 */


#ifdef TRANSMITTER
//////////////////////////////////// PROGRAMA PARA EL DISPOSITIVO TX
////////////////////////////////////////////////

void main(void)
{
  unsigned char direccion_transmision[5] = {0xB2, 0xB2, 0xB3, 0xB4, 0x01};
  unsigned char mensaje[8] = {0b00000001, 0b00000010, 0b00000100, 0b00001000,
0b00010000, 0b00100000, 0b01000000, 0b10000000};
  unsigned char informe;

  ADCON0bits.ADON=0;    // Disable the Analog-to-Digital (A/D) Converter module
  TRISB=0x00;
  TRISD=0xFF;

  while(1)
  {

    LATB=0b00011000;       // Secuencia LEDs de inicializaci?n
    Delay10KTCYx(30);
    LATB=0b00100100;
    Delay10KTCYx(30);
    LATB=0b01000010;
    Delay10KTCYx(30);
```

```
    LATB=0b10000001;
    Delay10KTCYx(30);

    SPI_Start(0b10);
    nRF24L01_Ports_Start();
    Start_TX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, 0b0000, 10, CHECKSUM,
8);
    for(in=0;in<30;in++)
    {
      Fifo[in] = Read_nRF24L01_Register(in);
    }

    Send_Data_TX_Mode_nRF24L01(CHECKSUM, 0b11, direccion_transmision, 8,
mensaje);
    for(in=0;in<30;in++)
    {
      Fifo[in] = Read_nRF24L01_Register(in);
    }
    informe=Check_Data_Sent_TX_Mode_nRF24L01();
    for(in=0;in<30;in++)
    {
      Fifo[in] = Read_nRF24L01_Register(in);
    }

    Finish_nRF24L01_Operation();

    if((informe & 0b11)==0b01)
    {
      LATB=0b11110000; Delay10KTCYx(100);    // Well Sent Message
      LATB=informe>>2;
    }
    else if((informe & 0b11)==0b00)
    {
      LATB=0b00001111; Delay10KTCYx(100);      // Not Well Sent Message
      LATB=informe>>2;
    }
    else if((informe & 0b11)==0b11)
    {
      LATB=0b00111100;              // nRF24L01 Error message
    }

    while(PORTDbits.RD1);
  }

}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#else
```

```
//////////////////////////////////// PROGRAMA PARA EL DISPOSITIVO RX
//////////////////////////////////////////

void main (void)
{
  unsigned char direccion_recepcion_A[5]={0xB2, 0xB2, 0xB3, 0xB4, 0x01};
  unsigned char direccion_recepcion_B[5]={0x11, 0x11, 0x11, 0x11, 0x11};
  unsigned char mensaje_llegada[8];
  unsigned char informe;

  ADCON0bits.ADON=0;    // Disable the Analog-to-Digital (A/D) Converter module
  TRISB=0x00;
  TRISD=0xFF;

  while(1)
  {
    LATB=0b00011000;        // Secuencia LEDs de inicializaci◆n
    Delay10KTCYx(30);
    LATB=0b00100100;
    Delay10KTCYx(30);
    LATB=0b01000010;
    Delay10KTCYx(30);
    LATB=0b10000001;
    Delay10KTCYx(30);

    SPI_Start(0b10);
    nRF24L01_Ports_Start();
    Start_RX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, direccion_recepcion_A,
direccion_recepcion_B, 0x00, 0x00, 0x00, 0x00, CHECKSUM, 8, 8, 0, 0, 0, 0);
    informe=Receive_Data_RX_Mode_nRF24L01(CHECKSUM, 100, 8, mensaje_llegada);
    Finish_nRF24L01_Operation();

    if(informe & 0b00010000)
    {
      LATB=0b00111100;
    }
    else if(!(informe & 0b00010000))
    {
      if(informe & 0x01)
      {
        LATB=0b11111111;
        Delay10KTCYx(100);
        LATB=informe>>1;
        Delay10KTCYx(100);
      }
      else if(!(informe & 0x01))
      {
        LATB=0b01010101;
```

```
            Delay10KTCYx(100);
            LATB=informe>>1;
            Delay10KTCYx(100);

        }
        LATB=mensaje_llegada[0];
        Delay10KTCYx(50);
        LATB=mensaje_llegada[1];
        Delay10KTCYx(50);
        LATB=mensaje_llegada[2];
        Delay10KTCYx(50);
        LATB=mensaje_llegada[3];
        Delay10KTCYx(50);
        LATB=mensaje_llegada[4];
        Delay10KTCYx(50);
        LATB=mensaje_llegada[5];
        Delay10KTCYx(50);
        LATB=mensaje_llegada[6];
        Delay10KTCYx(50);
        LATB=mensaje_llegada[7];
        Delay10KTCYx(50);
    }

    while(PORTDbits.RD1);
  }
}

#endif

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

Annex D: Packet sending – Automatic/Multireceiving mode code

nRF24L01_LED_Control.c (2) and UPC_nRF24L01.c from annex 2

```
#include<p18f4520.h>
#include<delays.h>
#include"config.h"
#include"UPC_nRF24L01.h"

  unsigned char Fifo[32];
  unsigned char in;

#define CHECKSUM 0

#define ADDRESS    0b10 // 4 bytes
       // 0b01 // 3 bytes
       // 0b10 // 4 bytes
       // 0b11 // 5 bytes

#define TRANSMITTER
//#define RECEIVER

//#ifdef TRANSMITTER & RECEIVE
/*
 *
 *   C?DIGO DE TEST Y VALIDACI?N DEL nRF24L01
 *   (Comentar o descomentar una de las dos fuciones main() a la hora de compilar o
programar)
 *
 *
 */


#ifdef TRANSMITTER
////////////////////////////////////// PROGRAMA PARA EL DISPOSITIVO TX
////////////////////////////////////////////////

void main(void)
{
  unsigned char direccion_transmision[5] = {0xB2, 0xB2, 0xB3, 0xB4, 0x01};
  unsigned char mensaje[8] = {0b00000001, 0b00000010, 0b00000100, 0b000001000,
0b00010000, 0b00100000, 0b01000000, 0b10000000};
  unsigned char informe;
```

```
  ADCON0bits.ADON=0;     // Disable the Analog-to-Digital (A/D) Converter module
  TRISB=0x00;
  TRISD=0xFF;

  while(1)
  {

    LATB=0b00011000;        // Secuencia LEDs de inicializaci?n
    Delay10KTCYx(30);
    LATB=0b00100100;
    Delay10KTCYx(30);
    LATB=0b01000010;
    Delay10KTCYx(30);
    LATB=0b10000001;
    Delay10KTCYx(30);

    SPI_Start(0b10);
    nRF24L01_Ports_Start();
    Start_TX_Mode_nRF24L01(ADDRESS, 0b1000000, 0, 0b11, 1, 1, 0b0000, 10,
CHECKSUM, 8);
    for(in=0;in<30;in++)
    {
      Fifo[in] = Read_nRF24L01_Register(in);
    }

    Send_Data_TX_Mode_nRF24L01(CHECKSUM, ADDRESS, direccion_transmision, 8,
mensaje);
    for(in=0;in<30;in++)
    {
      Fifo[in] = Read_nRF24L01_Register(in);
    }
    informe=Check_Data_Sent_TX_Mode_nRF24L01();
    for(in=0;in<30;in++)
    {
      Fifo[in] = Read_nRF24L01_Register(in);
    }

    Finish_nRF24L01_Operation();

    if((informe & 0b11)==0b01)
    {
      LATB=0b11110000; Delay10KTCYx(100);    // Well Sent Message
      LATB=informe>>2;
    }
    else if((informe & 0b11)==0b00)
    {
      LATB=0b00001111; Delay10KTCYx(100);      // Not Well Sent Message
      LATB=informe>>2;
    }
    else if((informe & 0b11)==0b11)
```

```
    {
      LATB=0b00111100;              // nRF24L01 Error message
    }

    while(PORTDbits.RD1);
  }

}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#else


////////////////////////////////////// PROGRAMA PARA EL DISPOSITIVO RX
//////////////////////////////////////////

void main (void)
{
  unsigned char direccion_recepcion_A[5]={0xB2, 0xB2, 0xB3, 0xB4, 0x01};
  unsigned char direccion_recepcion_B[5]={0x11, 0x11, 0x11, 0x11, 0x11};
  unsigned char mensaje_llegada[8];
  unsigned char informe;

  ADCON0bits.ADON=0;    // Disable the Analog-to-Digital (A/D) Converter module
  TRISB=0x00;
  TRISD=0xFF;

  while(1)
  {
    LATB=0b00011000;        // Secuencia LEDs de inicializaci?n
    Delay10KTCYx(30);
    LATB=0b00100100;
    Delay10KTCYx(30);
    LATB=0b01000010;
    Delay10KTCYx(30);
    LATB=0b10000001;
    Delay10KTCYx(30);

    SPI_Start(0b10);
    nRF24L01_Ports_Start();
    Start_RX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, direccion_recepcion_A,
direccion_recepcion_B, 0x00, 0x00, 0x00, 0x00, CHECKSUM, 8, 8, 0, 0, 0, 0);
    informe=Receive_Data_RX_Mode_nRF24L01(CHECKSUM, 100, 8, mensaje_llegada);
    Finish_nRF24L01_Operation();

    if(informe & 0b00010000)
    {
      LATB=0b00111100;
```

```
    }
    else if(!(informe & 0b00010000))
    {
      if(informe & 0x01)
      {
        LATB=0b11111111;
        Delay10KTCYx(100);
        LATB=informe>>1;
        Delay10KTCYx(100);
      }
      else if(!(informe & 0x01))
      {
        LATB=0b01010101;
        Delay10KTCYx(100);
        LATB=informe>>1;
        Delay10KTCYx(100);

      }
      LATB=mensaje_llegada[0];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[1];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[2];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[3];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[4];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[5];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[6];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[7];
      Delay10KTCYx(50);
    }

    while(PORTDbits.RD1);
  }
}
#endif
```

Annex E: File sending code

E.1 File converter

Converts the text file letters into bytes using ascii encoding.

```
def conv(path):
    file = open(path, 'r')
    text = file.read()
    file.close()
    text = text.encode("hex")
    tconv=''
    for i in range(0,len(text),2):
        tconv+='0x'+text[i:i+2]+', '
    fconv = open(path,'w')
    fconv.write(tconv[:-2])
    fconv.close()

conv("C:\Users\Joan\Desktop\TFG\Codigo_Final\Automatic mode.txt")
```

E.2 C files to send the file

nRF24L01_LED_Control.c (3) and UPC_nRF24L01.c from annex 2

```
#include<p18f4520.h>
#include<delays.h>
#include"config.h"
#include"UPC_nRF24L01.h"

  unsigned char Fifo[32];
  unsigned char in;

#define CHECKSUM 0

#define ADDRESS    0b10 // 4 bytes
       // 0b01 // 3 bytes
       // 0b10 // 4 bytes
       // 0b11 // 5 bytes

#define TRANSMITTER
//#define RECEIVER

//#ifdef TRANSMITTER & RECEIVE
/*
```

```
 *
 *  C?DIGO DE TEST Y VALIDACI?N DEL nRF24L01
 *  (Comentar o descomentar una de las dos fuciones main() a la hora de compilar o
programar)
 *
 *
 */

const rom unsigned char Ini[] = {"txt file0108"};
const rom unsigned char End[] = {"end file    "};

const rom unsigned char File[] = {0x4c, 0x6f, 0x73, 0x20, 0x63, 0x61, 0x6e, 0x64, 0x69,
                  0x64, 0x61, 0x74, 0x6f, 0x73, 0x20, 0x73, 0x65, 0x20,
                  0x61, 0x63, 0x75, 0x73, 0x61, 0x6e, 0x20, 0x64, 0x65,
                  0x20, 0x64, 0x65, 0x6c, 0x69, 0x74, 0x6f, 0x73, 0x20,
                  0x67, 0x72, 0x61, 0x76, 0x65, 0x73, 0x20, 0x65, 0x6e,
                  0x20, 0x65, 0x6c, 0x20, 0x62, 0x6c, 0x6f, 0x71, 0x75,
                  0x65, 0x20, 0x64, 0x65, 0x20, 0x6c, 0x61, 0x20, 0x63,
                  0x6f, 0x72, 0x72, 0x75, 0x70, 0x63, 0x69, 0xf3, 0x6e,
                  0x20, 0x79, 0x20, 0x6d, 0x61, 0x6e, 0x74, 0x69, 0x65,
                  0x6e, 0x65, 0x6e, 0x20, 0x73, 0x75, 0x73, 0x20, 0x70,
                  0x6f, 0x73, 0x74, 0x75, 0x72, 0x61, 0x73, 0x20, 0x73,
                  0x6f, 0x62, 0x72, 0x65, 0x20, 0x43, 0x61, 0x74, 0x61
};



#ifdef TRANSMITTER
////////////////////////////////////// PROGRAMA PARA EL DISPOSITIVO TX
//////////////////////////////////////////////////

void main(void)
{
  unsigned char direccion_transmision[5] = {0xB2, 0xB2, 0xB3, 0xB4, 0x01};
  unsigned char mensaje[32];// = {0b00000001, 0b00000010, 0b00000100, 0b000001000,
0b00010000, 0b00100000, 0b01000000, 0b10000000};
  unsigned char informe;
  unsigned char k,i;

  ADCON0bits.ADON=0;    // Disable the Analog-to-Digital (A/D) Converter module
  TRISB=0x00;
  TRISD=0xFF;

  while(1)
  {
```

```
    LATB=0b00011000;        // Secuencia LEDs de inicializaci◆n
    Delay10KTCYx(30);
    LATB=0b00100100;
    Delay10KTCYx(30);
    LATB=0b01000010;
    Delay10KTCYx(30);
    LATB=0b10000001;
    Delay10KTCYx(30);

    SPI_Start(0b10);
    nRF24L01_Ports_Start();
    Start_TX_Mode_nRF24L01(ADDRESS, 0b1000000, 0, 0b11, 1, 1, 0b0000, 10,
CHECKSUM, 12);

    //in = strlenpgm(Ini);
    for(i=0;i<12;i++)
    {
      mensaje[i] = Ini[i];
    }
    Send_Data_TX_Mode_nRF24L01(CHECKSUM, ADDRESS, direccion_transmision, 12,
mensaje);
    informe=Check_Data_Sent_TX_Mode_nRF24L01();
    Finish_nRF24L01_Operation();
    Nop();

//   Send_Data_TX_Mode_nRF24L01(CHECKSUM, ADDRESS, direccion_transmision, 9,
mensaje);
//   informe=Check_Data_Sent_TX_Mode_nRF24L01();

    Delay10KTCYx(250);  //2
    Delay10KTCYx(250);

    k = 0;
    while(k<108)
    {
      SPI_Start(0b10);
      nRF24L01_Ports_Start();
      Start_TX_Mode_nRF24L01(ADDRESS, 0b1000000, 0, 0b11, 1, 1, 0b0000, 10,
CHECKSUM, 12);
      for(i=0;i<12;i++)
      {
        mensaje[i] = File[k];
        k = k + 1;
      }
      Send_Data_TX_Mode_nRF24L01(CHECKSUM, ADDRESS, direccion_transmision,
12, mensaje);
      informe=Check_Data_Sent_TX_Mode_nRF24L01();
```

```
    Finish_nRF24L01_Operation();
    Nop();
    Delay10KTCYx(250);
    Delay10KTCYx(250);
  }

  for(i=0;i<12;i++)
  {
    mensaje[i] = End[i];
  }
  SPI_Start(0b10);
  nRF24L01_Ports_Start();
  Send_Data_TX_Mode_nRF24L01(CHECKSUM, ADDRESS, direccion_transmision, 9,
mensaje);
  informe=Check_Data_Sent_TX_Mode_nRF24L01();
  Finish_nRF24L01_Operation();
    while(1);


  for(in=0;in<30;in++)
  {
    Fifo[in] = Read_nRF24L01_Register(in);
  }

  Send_Data_TX_Mode_nRF24L01(CHECKSUM, ADDRESS, direccion_transmision, 8,
mensaje);
  for(in=0;in<30;in++)
  {
    Fifo[in] = Read_nRF24L01_Register(in);
  }
  informe=Check_Data_Sent_TX_Mode_nRF24L01();
  for(in=0;in<30;in++)
  {
    Fifo[in] = Read_nRF24L01_Register(in);
  }

  Finish_nRF24L01_Operation();

  if((informe & 0b11)==0b01)
  {
    LATB=0b11110000; Delay10KTCYx(100);    // Well Sent Message
    LATB=informe>>2;
  }
  else if((informe & 0b11)==0b00)
  {
    LATB=0b00001111; Delay10KTCYx(100);      // Not Well Sent Message
    LATB=informe>>2;
```

```
    }
    else if((informe & 0b11)==0b11)
    {
      LATB=0b00111100;              // nRF24L01 Error message
    }

    while(PORTDbits.RD1);
  }

}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#else

////////////////////////////////////// PROGRAMA PARA EL DISPOSITIVO RX
//////////////////////////////////////////////

void main (void)
{
  unsigned char direccion_recepcion_A[5]={0xB2, 0xB2, 0xB3, 0xB4, 0x01};
  unsigned char direccion_recepcion_B[5]={0x11, 0x11, 0x11, 0x11, 0x11};
  unsigned char mensaje_llegada[8];
  unsigned char informe;

  ADCON0bits.ADON=0;    // Disable the Analog-to-Digital (A/D) Converter module
  TRISB=0x00;
  TRISD=0xFF;

  while(1)
  {
    LATB=0b00011000;       // Secuencia LEDs de inicializaci?n
    Delay10KTCYx(30);
    LATB=0b00100100;
    Delay10KTCYx(30);
    LATB=0b01000010;
    Delay10KTCYx(30);
    LATB=0b10000001;
    Delay10KTCYx(30);

    SPI_Start(0b10);
    nRF24L01_Ports_Start();
    Start_RX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, direccion_recepcion_A,
direccion_recepcion_B, 0x00, 0x00, 0x00, 0x00, CHECKSUM, 8, 8, 0, 0, 0, 0);
    informe=Receive_Data_RX_Mode_nRF24L01(CHECKSUM, 100, 8, mensaje_llegada);
    Finish_nRF24L01_Operation();
```

```
    if(informe & 0b00010000)
    {
      LATB=0b00111100;
    }
    else if(!(informe & 0b00010000))
    {
      if(informe & 0x01)
      {
        LATB=0b11111111;
        Delay10KTCYx(100);
        LATB=informe>>1;
        Delay10KTCYx(100);
      }
      else if(!(informe & 0x01))
      {
        LATB=0b01010101;
        Delay10KTCYx(100);
        LATB=informe>>1;
        Delay10KTCYx(100);

      }
      LATB=mensaje_llegada[0];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[1];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[2];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[3];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[4];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[5];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[6];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[7];
      Delay10KTCYx(50);
    }

    while(PORTDbits.RD1);
  }
}

#endif
```

Annex F: Keyboard sending code

nRF24L01_LED_Control.c (4) and UPC_nRF24L01.c from annex 2

```
// RF + PS2 keyboard

#include<p18f4520.h>
#include<delays.h>
#include"config.h"
#include"UPC_nRF24L01.h"

  unsigned char Fifo[32];
  unsigned char in;

#define CHECKSUM 0

#define ADDRESS    0b10 // 4 bytes
       // 0b01 // 3 bytes
       // 0b10 // 4 bytes
       // 0b11 // 5 bytes

#define TRANSMITTER
//#define RECEIVER

//#ifdef TRANSMITTER & RECEIVE
/*
 *
 *   C?DIGO DE TEST Y VALIDACI?N DEL nRF24L01
 *   (Comentar o descomentar una de las dos fuciones main() a la hora de compilar o
programar)
 *
 *
 */

#define SET_PS2_DA()   LATBbits.LATB1 = 1
#define IN_PS2_DA()      TRISBbits.TRISB1 = 1
#define GET_PS2_DA()   PORTBbits.RB1

#define SET_PS2_CL()   LATBbits.LATB0 = 1
#define IN_PS2_CL()      TRISBbits.TRISB0 = 1


volatile  unsigned char ps2_status;
volatile  unsigned char ps2_data;
volatile  unsigned char ps2_parity;

unsigned char in = 0, out = 0;

#pragma udata section = 0x100  // Indicamos al compilador que queremos que la
```

```
unsigned char queue_keys[256]; // cola circular est? situada a partir de la direcci?n
0x100
#pragma udata

void Init_kb(void);
void ps2Init(void);
unsigned char ps2GetChar(unsigned char ps2char);
void InterruptHandlerHigh();



#ifdef TRANSMITTER
//////////////////////////////////// PROGRAMA PARA EL DISPOSITIVO TX
////////////////////////////////////////////////

void main(void)
{
  unsigned char direccion_transmision[5] = {0xB2, 0xB2, 0xB3, 0xB4, 0x01};
  unsigned char mensaje[32];// = {0b00000001, 0b00000010, 0b00000100, 0b000001000,
0b00010000, 0b00100000, 0b01000000, 0b10000000};
  unsigned char informe;
  unsigned char k,i;
  unsigned char aux;

// ADCON0bits.ADON=0;     // Disable the Analog-to-Digital (A/D) Converter module
// TRISB=0x00;
// TRISD=0xFF;


  Init_kb();
  ps2Init();
  while(1)
  {

    if(in!=out)
    {
      if(queue_keys[out]== 0xF0)
      {
        INTCONbits.GIEH = 0;
        in--;
        Delay10KTCYx(1);
        INTCONbits.INT0IF = 0;
        INTCONbits.GIEH = 1;
        Nop();

      }
      else
      {
        mensaje[0] = ps2GetChar(queue_keys[out++]);
        SPI_Start(0b10);
```

```
        nRF24L01_Ports_Start();
        Start_TX_Mode_nRF24L01(ADDRESS, 0b1000000, 0, 0b11, 1, 1, 0b0000, 10,
CHECKSUM, 1);
        Send_Data_TX_Mode_nRF24L01(CHECKSUM, ADDRESS, direccion_transmision,
1, mensaje);
        informe=Check_Data_Sent_TX_Mode_nRF24L01();
        Finish_nRF24L01_Operation();
        Nop();
      }
    }

  }



}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#else



////////////////////////////////////// PROGRAMA PARA EL DISPOSITIVO RX
////////////////////////////////////////////

void main (void)
{
  unsigned char direccion_recepcion_A[5]={0xB2, 0xB2, 0xB3, 0xB4, 0x01};
  unsigned char direccion_recepcion_B[5]={0x11, 0x11, 0x11, 0x11, 0x11};
  unsigned char mensaje_llegada[8];
  unsigned char informe;

  ADCON0bits.ADON=0;    // Disable the Analog-to-Digital (A/D) Converter module
  TRISB=0x00;
  TRISD=0xFF;

  while(1)
  {
    LATB=0b00011000;        // Secuencia LEDs de inicializaci?n
    Delay10KTCYx(30);
    LATB=0b00100100;
    Delay10KTCYx(30);
    LATB=0b01000010;
    Delay10KTCYx(30);
    LATB=0b10000001;
    Delay10KTCYx(30);

    SPI_Start(0b10);
    nRF24L01_Ports_Start();
```

```
    Start_RX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, direccion_recepcion_A,
direccion_recepcion_B, 0x00, 0x00, 0x00, 0x00, CHECKSUM, 8, 8, 0, 0, 0, 0);
    informe=Receive_Data_RX_Mode_nRF24L01(CHECKSUM, 100, 8, mensaje_llegada);
    Finish_nRF24L01_Operation();

    if(informe & 0b00010000)
    {
      LATB=0b00111100;
    }
    else if(!(informe & 0b00010000))
    {
      if(informe & 0x01)
      {
        LATB=0b11111111;
        Delay10KTCYx(100);
        LATB=informe>>1;
        Delay10KTCYx(100);
      }
      else if(!(informe & 0x01))
      {
        LATB=0b01010101;
        Delay10KTCYx(100);
        LATB=informe>>1;
        Delay10KTCYx(100);

      }
      LATB=mensaje_llegada[0];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[1];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[2];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[3];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[4];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[5];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[6];
      Delay10KTCYx(50);
      LATB=mensaje_llegada[7];
      Delay10KTCYx(50);
    }

    while(PORTDbits.RD1);
  }
}

#endif
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
h

void Init_kb(void)
{
  ADCON1=0X07;
  TRISB=0xFF;
  INTCON2bits.INTEDG0 = 0;   // External interrupt INT0 is falling edge
  INTCONbits.INT0IE = 1;       // Enable external interrupt INT0;
  INTCONbits.INT0IF = 0;       // Delete interrupt flag
  RCONbits.IPEN = 1;          // Enable priority levels
  INTCONbits.GIEH = 1;      // Enable interrupts
}

void ps2Init(void)
{
  ps2_status=0;
  ps2_data=0;
  ps2_parity=0;


  IN_PS2_CL();
  SET_PS2_CL();
  IN_PS2_DA();
  SET_PS2_DA();
}



unsigned char ps2GetChar(unsigned char ps2char)
{
  unsigned char aux;
  switch(ps2char)
  {
    case 0x1C: aux = ('a'); break;
    case 0x32: aux = ('b'); break;
    case 0x21: aux = ('c'); break;
    case 0x23: aux = ('d'); break;
    case 0x24: aux = ('e'); break;
    case 0x2B: aux = ('f'); break;
    case 0x34: aux = ('g'); break;
    case 0x33: aux = ('h'); break;
    case 0x43: aux = ('i'); break;
    case 0x3B: aux = ('j'); break;
    case 0x42: aux = ('k'); break;
    case 0x4B: aux = ('l'); break;
    case 0x3A: aux = ('m'); break;
    case 0x31: aux = ('n'); break;
    case 0x44: aux = ('o'); break;
    case 0x4D: aux = ('p'); break;
```

```
    case 0x15: aux = ('q'); break;
    case 0x2D: aux = ('r'); break;
    case 0x1B: aux = ('s'); break;
    case 0x2C: aux = ('t'); break;
    case 0x3C: aux = ('u'); break;
    case 0x2A: aux = ('v'); break;
    case 0x1D: aux = ('w'); break;
    case 0x22: aux = ('x'); break;
    case 0x35: aux = ('y'); break;
    case 0x1A: aux = ('z'); break;
    case 0x45: aux = ('0'); break;
    case 0x16: aux = ('1'); break;
    case 0x1E: aux = ('2'); break;
    case 0x26: aux = ('3'); break;
    case 0x25: aux = ('4'); break;
    case 0x2E: aux = ('5'); break;
    case 0x36: aux = ('6'); break;
    case 0x3D: aux = ('7'); break;
    case 0x3E: aux = ('8'); break;
    case 0x46: aux = ('9'); break;
    case 0x0E: aux = ('`'); break;
    case 0x4E: aux = ('-'); break;
    case 0x55: aux = ('='); break;
    case 0x5D: aux = ('\\'); break;
    case 0x29: aux = (' '); break;
    case 0x54: aux = ('['); break;
    case 0x5B: aux = (']'); break;
    case 0x4C: aux = (';'); break;
    case 0x52: aux = ('\''); break;
    case 0x41: aux = (','); break;
    case 0x49: aux = ('.'); break;
    case 0x4A: aux = ('/'); break;
    case 0x71: aux = ('.'); break;
    case 0x70: aux = ('0'); break;
    case 0x69: aux = ('1'); break;
    case 0x72: aux = ('2'); break;
    case 0x7A: aux = ('3'); break;
    case 0x6B: aux = ('4'); break;
    case 0x73: aux = ('5'); break;
    case 0x74: aux = ('6'); break;
    case 0x6C: aux = ('7'); break;
    case 0x75: aux = ('8'); break;
    case 0x7D: aux = ('9'); break;
    default: aux = 0; break;
  }
  return(aux);
}

//-----------------------------------------------------------------------
#pragma code InterruptVectorHigh = 0x08
```

```
void InterruptVectorHigh(void)
{
  _asm
    goto InterruptHandlerHigh //jump to interrupt routine (ISR)
  _endasm
}
//-----------------------------------------------------------------------
// High priority interrupt routine (Interrupt Service Routine ISR)
#pragma code
#pragma interrupt InterruptHandlerHigh
void InterruptHandlerHigh()
{
  if(INTCONbits.INT0IF)
  {
    ps2_status++;

    switch(ps2_status)
    {
      case 1:      if(GET_PS2_DA()) ps2_status=0;
            break;

      case 10:   if(GET_PS2_DA()) ps2_parity++;
            break;

      case 11:   if((ps2_parity & 0x01))
             {
               queue_keys[in++] = ps2_data;
               ps2_status=0;
               ps2_parity=0;
             }
            break;

      default:   if(ps2_status<10 && ps2_status>0)
              {
              ps2_data=ps2_data>>1;
             if(GET_PS2_DA()) {ps2_data |= 0x80; ps2_parity++;}
             else ps2_data &= 0x7F;
              }
            else
            {
               ps2_status=0;
               ps2_parity=0;
              }
            break;
    }

    INTCONbits.INT0IF = 0;
  }
}
```

Annex G: CRC full calculation

0100110110110011101101000000011100000000100000000

0100000111000000000000000000000000000000000000000

0000110001110011101101000000011100000000100000000

0000100000111000000000000000000000000000000000000

0000010001001011101101000000011100000000100000000

0000010000011100000000000000000000000000000000000

0000000001010111011010000000111000000001000000000

0000000001000001110000000000000000000000000000000

0000000000010110011101000000011100000000100000000

0000000000010000011100000000000000000000000000000

0000000000000110000001000000011100000000100000000

0000000000000010000011100000000000000000000000000

0000000000000010000011000000000111000000010000000

0000000000000010000011100000000000000000000000000

0000000000000000000010110000001110000000010000000

0000000000000000000010000011100000000000000000000

0000000000000000000011001110111000000010000000

0000000000000000000010000011100000000000000000000

0000000000000000000010011010110000000010000000

0000000000000000000010000011100000000000000000000

0000000000000000000011001010000000010000000

0000000000000000000010000011100000000000000000

0000000000000000000010010011000000100000000

0000000000000000000010000011100000000000000000

0000000000000000000010000100001000000000000

0000000000000000000010000011100000000000000

0000000000000000000000000000111100100000000

0000000000000000000000000000010000011100000000

00000000000000000000000000000000000111000110000000

00000000000000000000000000000000000010000011100000

00000000000000000000000000000000000001100000100000

00000000000000000000000000000000000001000001110000

00000000000000000000000000000000000001000001010000

00000000000000000000000000000000000001000001110000

00000000000000000000000000000000000000001101000 <- CRC