Serak Gebremedhin
Github root directory: https://github.com/Ber-geb/solid-octo-tribble.git

Date Submitted: **12/11/19**

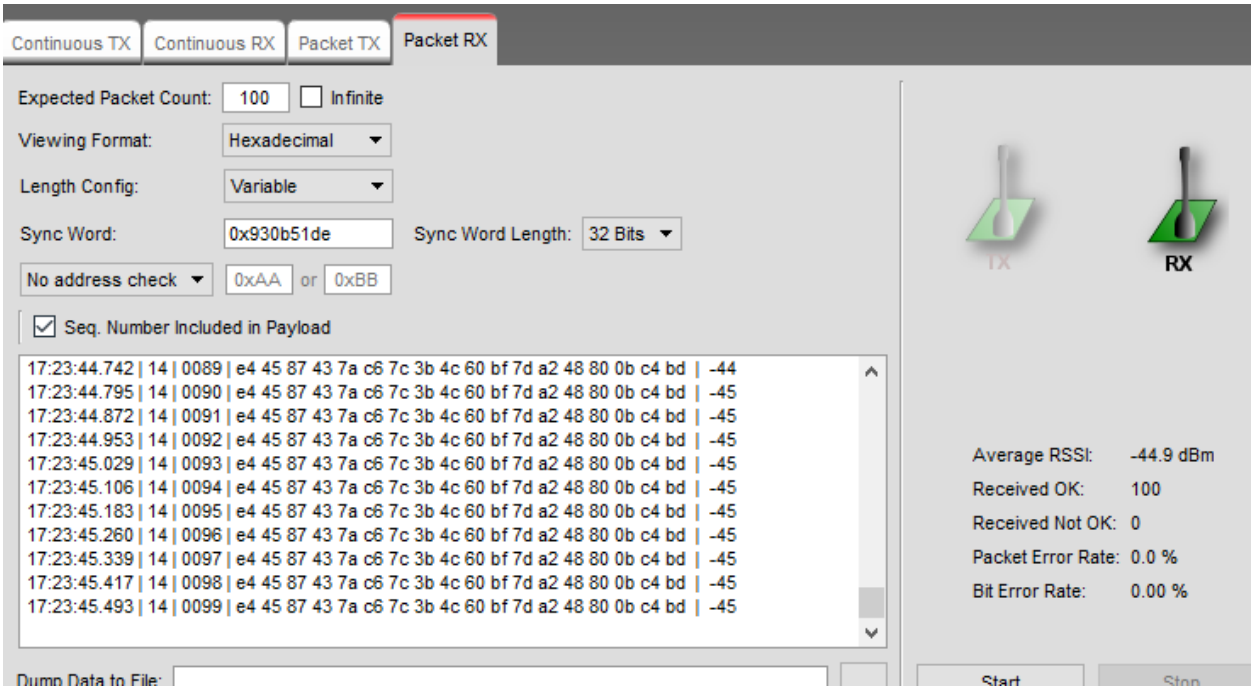
Task 01 (Rx):

Youtube Link:

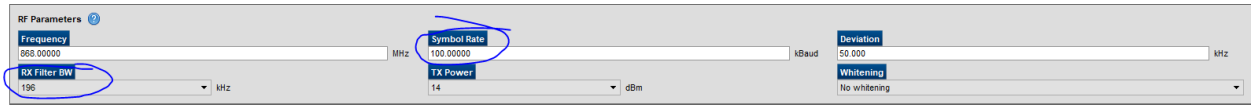
Task 2 & 3: https://youtu.be/yt8pltB5Qmc
Task 4: https://youtu.be/gnarXjTdR68
Task 7: https://youtu.be/LTb_cUq8pSs

Modified Schematic (if applicable):

Task 1:



Task 3:



Modified Code:

```
/*
 * Copyright (c) 2019, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

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

/***** Includes *****/
/* Standard C Libraries */
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

/*xdctools header files*/
#include <xdc/std.h>
#include <xdc/cfg/global.h>
#include <xdc/runtime/Assert.h>

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

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

/* Driverlib Header files */
#include <ti/devices/DeviceFamily.h>
#include DeviceFamily_constructPath(driverlib/rf_prop_mailbox.h)

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

/* Application Header files */
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

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

#include <ti/drivers/UART.h>


/***** Defines *****/

/* Packet RX Configuration */
#define DATA_ENTRY_HEADER_SIZE 8  /* Constant header size of a Generic Data Entry */
#define MAX_LENGTH             30 /* Max length byte the radio will accept */
#define NUM_DATA_ENTRIES       2  /* NOTE: Only two data entries supported at the
moment */
#define NUM_APPENDED_BYTES     2  /* The Data Entries data field will contain:
                                   * 1 Header byte (RF_cmdPropRx.rxConf.bIncludeHdr =
0x1)
                                   * Max 30 payload bytes
                                   * 1 status byte (RF_cmdPropRx.rxConf.bAppendStatus
= 0x1) */
#define TX_TASK_STACK_SIZE   1024
#define PAYLOAD_LENGTH         30


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

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

static RF_Object rfObject;
static RF_Handle rfHandle;

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

static uint8_t packet[PAYLOAD_LENGTH];
static PIN_Handle pinHandle;

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

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

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

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

/*
 * Application LED pin configuration table:
 *   - All LEDs board LEDs are off.
 */

PIN_Config pinTable[] =
{
 Board_PIN_LED1 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL |
 PIN_DRVSTR_MAX,
#if defined __CC1352R1_LAUNCHXL_BOARD_H__
 Board_DIO30_RFSW | PIN_GPIO_OUTPUT_EN | PIN_GPIO_HIGH | PIN_PUSHPULL |
 PIN_DRVSTR_MAX,
#endif
#ifdef POWER_MEASUREMENT
#if !defined(__CC1352R1_LAUNCHXL_BOARD_H__) &&
 !defined(__CC26X2R1_LAUNCHXL_BOARD_H__)
 CC1350_LAUNCHXL_DIO30_SWPWR | PIN_GPIO_OUTPUT_EN | PIN_GPIO_HIGH |
 PIN_PUSHPULL | PIN_DRVSTR_MAX,
#endif
#endif
 PIN_TERMINATE
};




/*
PIN_Config pinTable[] =
{
    Board_PIN_LED2 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL |
PIN_DRVSTR_MAX,
       PIN_TERMINATE
};
*/
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
/***** Function definitions *****/
void TxTask_init(PIN_Handle inPinHandle)
{
    pinHandle = inPinHandle;
    Task_Params_init(&txTaskParams);
    txTaskParams.stackSize = TX_TASK_STACK_SIZE;
//    txTaskParams.priority = TX_TASK_PRIORITY;
    txTaskParams.stack = &txTaskStack;
    txTaskParams.arg0 = (UInt)1000000;

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


static void txTaskFunction(UArg arg0, UArg arg1)
{
#ifdef POWER_MEASUREMENT
    /* Shutdown external flash */
    Board_shutDownExtFlash();
#if !defined(__CC1352R1_LAUNCHXL_BOARD_H__) &&
    !defined(__CC26X2R1_LAUNCHXL_BOARD_H__)
         /* Route out PA active pin to Board_DIO30_SWPWR */
         PINCC26XX_setMux(ledPinHandle, Board_DIO30_SWPWR,
                          PINCC26XX_MUX_RFC_GPO1);
#endif
#endif
    /* Init UART */
    char input;
    const char startPrompt[] = "Start typing\r\n";
    UART_Handle uart;
    UART_Params uartParams;
    UART_init();
    /* Create a UART with data processing off. */
    UART_Params_init(&uartParams);
    uartParams.writeDataMode = UART_DATA_BINARY;
    uartParams.readDataMode = UART_DATA_BINARY;
    uartParams.readReturnMode = UART_RETURN_FULL;
    uartParams.readEcho = UART_ECHO_OFF;
    uartParams.baudRate = 115200;
    uart = UART_open(Board_UART0, &uartParams);
    if (uart == NULL) {
        /* UART_open() failed */
        while (1);
    }
    /* Write to the UART before starting RX */
    UART_write(uart, startPrompt, sizeof(startPrompt));
    /* Init RF */
    RF_Params rfParams;
    RF_Params_init(&rfParams);
    RF_cmdPropTx.pktLen = PAYLOAD_LENGTH;
    RF_cmdPropTx.pPkt = packet;
    RF_cmdPropTx.startTrigger.triggerType = TRIG_NOW;
    /* Request access to the radio */
    rfHandle = RF_open(&rfObject, &RF_prop,
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
                        (RF_RadioSetup*)&RF_cmdPropRadioDivSetup, &rfParams);
    /* Set the frequency */
    RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
    /* Get current time */
    while(1)
    {
        uint8_t i = 0;
        do
        {
            UART_read(uart, &input, 1);
            UART_write(uart, &input, 1);
            packet[i++] = input;
        }
        while (input != '\r');
        /*skip CR */
        RF_cmdPropTx.pktLen = i-1;
        /* Send packet */
        RF_EventMask terminationReason = RF_runCmd(rfHandle,
                                                   (RF_Op*)&RF_cmdPropTx,
                                                   RF_PriorityNormal, NULL,
                                                   0);


#ifndef POWER_MEASUREMENT
        PIN_setOutputValue(pinHandle,
                           Board_PIN_LED1,!PIN_getOutputValue(Board_PIN_LED1));
#endif
    }
}



void *mainThread(void *arg0)
{
    RF_Params rfParams;
    RF_Params_init(&rfParams);

    /* Open LED pins */
    ledPinHandle = PIN_open(&ledPinState, pinTable);
    if (ledPinHandle == NULL)
    {
        while(1);
    }

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

    /* Modify CMD_PROP_RX command for application needs */
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

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

    /* Request access to the radio */
#if defined(DeviceFamily_CC26X0R2)
    rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioSetup,
&rfParams);
#else
    rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
&rfParams);
#endif// DeviceFamily_CC26X0R2

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

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

    switch(terminationReason)
    {
        case RF_EventLastCmdDone:
            // A stand-alone radio operation command or the last radio
            // operation command in a chain finished.
            break;
        case RF_EventCmdCancelled:
            // Command cancelled before it was started; it can be caused
            // by RF_cancelCmd() or RF_flushCmd().
            break;
        case RF_EventCmdAborted:
            // Abrupt command termination caused by RF_cancelCmd() or
            // RF_flushCmd().
            break;
        case RF_EventCmdStopped:
            // Graceful command termination caused by RF_cancelCmd() or
            // RF_flushCmd().
            break;
        default:
            // Uncaught error event
            while(1);
    }

    uint32_t cmdStatus = ((volatile RF_Op*)&RF_cmdPropRx)->status;
    switch(cmdStatus)
    {
        case PROP_DONE_OK:
```

```
            // Packet received with CRC OK
            break;
        case PROP_DONE_RXERR:
            // Packet received with CRC error
            break;
        case PROP_DONE_RXTIMEOUT:
            // Observed end trigger while in sync search
            break;
        case PROP_DONE_BREAK:
            // Observed end trigger while receiving packet when the command is
            // configured with endType set to 1
            break;
        case PROP_DONE_ENDED:
            // Received packet after having observed the end trigger; if the
            // command is configured with endType set to 0, the end trigger
            // will not terminate an ongoing reception
            break;
        case PROP_DONE_STOPPED:
            // received CMD_STOP after command started and, if sync found,
            // packet is received
            break;
        case PROP_DONE_ABORT:
            // Received CMD_ABORT after command started
            break;
        case PROP_ERROR_RXBUF:
            // No RX buffer large enough for the received data available at
            // the start of a packet
            break;
        case PROP_ERROR_RXFULL:
            // Out of RX buffer space during reception in a partial read
            break;
        case PROP_ERROR_PAR:
            // Observed illegal parameter
            break;
        case PROP_ERROR_NO_SETUP:
            // Command sent without setting up the radio in a supported
            // mode using CMD_PROP_RADIO_SETUP or CMD_RADIO_SETUP
            break;
        case PROP_ERROR_NO_FS:
            // Command sent without the synthesizer being programmed
            break;
        case PROP_ERROR_RXOVF:
            // RX overflow observed during operation
            break;
        default:
            // Uncaught error event - these could come from the
            // pool of states defined in rf_mailbox.h
            while(1);
    }

    while(1);
}

void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e)
{
```

```
    if (e & RF_EventRxEntryDone)
    {
        /* Toggle pin to indicate RX */
        PIN_setOutputValue(ledPinHandle, Board_PIN_LED2,
                           !PIN_getOutputValue(Board_PIN_LED2));

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

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

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

        RFQueue_nextEntry();
    }
}
```

Task 02(Tx):

Modified Code:

```
// Insert code here

/*
 * Copyright (c) 2019, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * *  Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * *  Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
```

```
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/***** Includes *****/
/* Standard C Libraries */
#include <stdlib.h>
#include <unistd.h>

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

/* Driverlib Header files */
#include DeviceFamily_constructPath(driverlib/rf_prop_mailbox.h)

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

/***** Defines *****/

/* Do power measurement */
//#define POWER_MEASUREMENT

/* Packet TX Configuration */
#define PAYLOAD_LENGTH      30
#ifdef POWER_MEASUREMENT
#define PACKET_INTERVAL     5  /* For power measurement set packet interval to 5s */
#else
#define PACKET_INTERVAL     500000  /* Set packet interval to 500000us or 500ms */
#endif

/***** Prototypes *****/

/***** Variable declarations *****/
static RF_Object rfObject;
static RF_Handle rfHandle;

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

static uint8_t packet[PAYLOAD_LENGTH];
static uint16_t seqNumber;

/*
 * Application LED pin configuration table:
 *   - All LEDs board LEDs are off.
 */
PIN_Config pinTable[] =
{
```

```
    Board_PIN_LED1 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL |
PIN_DRVSTR_MAX,
#ifdef POWER_MEASUREMENT
#if defined(Board_CC1350_LAUNCHXL)
    Board_DIO30_SWPWR | PIN_GPIO_OUTPUT_EN | PIN_GPIO_HIGH | PIN_PUSHPULL |
PIN_DRVSTR_MAX,
#endif
#endif
    PIN_TERMINATE
};

/***** Function definitions *****/

void *mainThread(void *arg0)
{
    RF_Params rfParams;
    RF_Params_init(&rfParams);

    /* Open LED pins */
    ledPinHandle = PIN_open(&ledPinState, pinTable);
    if (ledPinHandle == NULL)
    {
        while(1);
    }

#ifdef POWER_MEASUREMENT
#if defined(Board_CC1350_LAUNCHXL)
    /* Route out PA active pin to Board_DIO30_SWPWR */
    PINCC26XX_setMux(ledPinHandle, Board_DIO30_SWPWR, PINCC26XX_MUX_RFC_GPO1);
#endif
#endif

    RF_cmdPropTx.pktLen = PAYLOAD_LENGTH;
    RF_cmdPropTx.pPkt = packet;
    RF_cmdPropTx.startTrigger.triggerType = TRIG_NOW;

    /* Request access to the radio */
#if defined(DeviceFamily_CC26X0R2)
    rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioSetup,
&rfParams);
#else
    rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
&rfParams);
#endif// DeviceFamily_CC26X0R2

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

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

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

/* Send packet */
RF_EventMask terminationReason = RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTx,
                                           RF_PriorityNormal, NULL, 0);

switch(terminationReason)
{
    case RF_EventLastCmdDone:
        // A stand-alone radio operation command or the last radio
        // operation command in a chain finished.
        break;
    case RF_EventCmdCancelled:
        // Command cancelled before it was started; it can be caused
    // by RF_cancelCmd() or RF_flushCmd().
        break;
    case RF_EventCmdAborted:
        // Abrupt command termination caused by RF_cancelCmd() or
        // RF_flushCmd().
        break;
    case RF_EventCmdStopped:
        // Graceful command termination caused by RF_cancelCmd() or
        // RF_flushCmd().
        break;
    default:
        // Uncaught error event
        while(1);
}

uint32_t cmdStatus = ((volatile RF_Op*)&RF_cmdPropTx)->status;
switch(cmdStatus)
{
    case PROP_DONE_OK:
        // Packet transmitted successfully
        break;
    case PROP_DONE_STOPPED:
        // received CMD_STOP while transmitting packet and finished
        // transmitting packet
        break;
    case PROP_DONE_ABORT:
        // Received CMD_ABORT while transmitting packet
        break;
    case PROP_ERROR_PAR:
        // Observed illegal parameter
        break;
    case PROP_ERROR_NO_SETUP:
        // Command sent without setting up the radio in a supported
        // mode using CMD_PROP_RADIO_SETUP or CMD_RADIO_SETUP
        break;
    case PROP_ERROR_NO_FS:
        // Command sent without the synthesizer being programmed
        break;
    case PROP_ERROR_TXUNF:
```

```
                // TX underflow observed during operation
                break;
            default:
                // Uncaught error event - these could come from the
                // pool of states defined in rf_mailbox.h
                while(1);
        }

#ifndef POWER_MEASUREMENT
        PIN_setOutputValue(ledPinHandle,
Board_PIN_LED1,!PIN_getOutputValue(Board_PIN_LED1));
#endif
        /* Power down the radio */
        RF_yield(rfHandle);

#ifdef POWER_MEASUREMENT
        /* Sleep for PACKET_INTERVAL s */
        sleep(PACKET_INTERVAL);
#else
        /* Sleep for PACKET_INTERVAL us */
        usleep(PACKET_INTERVAL);
#endif

    }
}
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

Serak Gebremedhin
Github root directory: https://github.com/Ber-geb/solid-octo-tribble.git

Task 03:

Youtube Link:

Modified Schematic (if applicable):

Modified Code:

```
// Insert code here
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.