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

**Task 01 (Rx):**

Youtube Link:

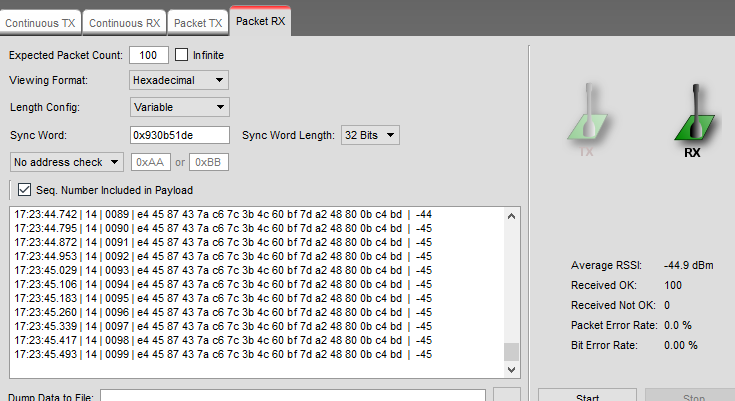
Task 2 & 3: <https://youtu.be/yt8pltB5Qmc>

Task 4: <https://youtu.be/gnarXjTdR68>

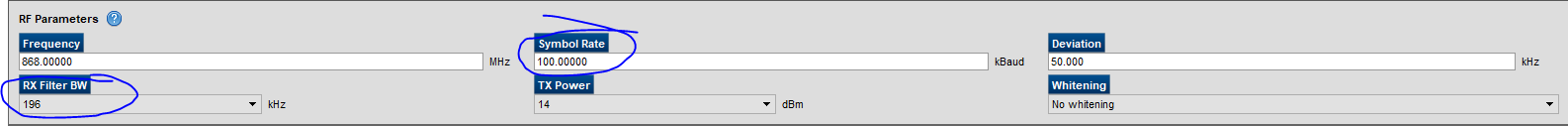
Task 7: <https://youtu.be/LTb_cUq8pSs>

**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

\* 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 \*/

**#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

};

\*/

/\*\*\*\*\* 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,

(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 \*/

/\* 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**

}

}

**------------------------------------------------------------------------------------**

**Task 03:**

Youtube Link:

**Modified Schematic (if applicable):**

**Modified Code:**

**// Insert code here**

**------------------------------------------------------------------------------------**