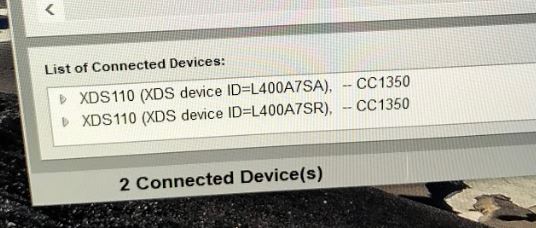
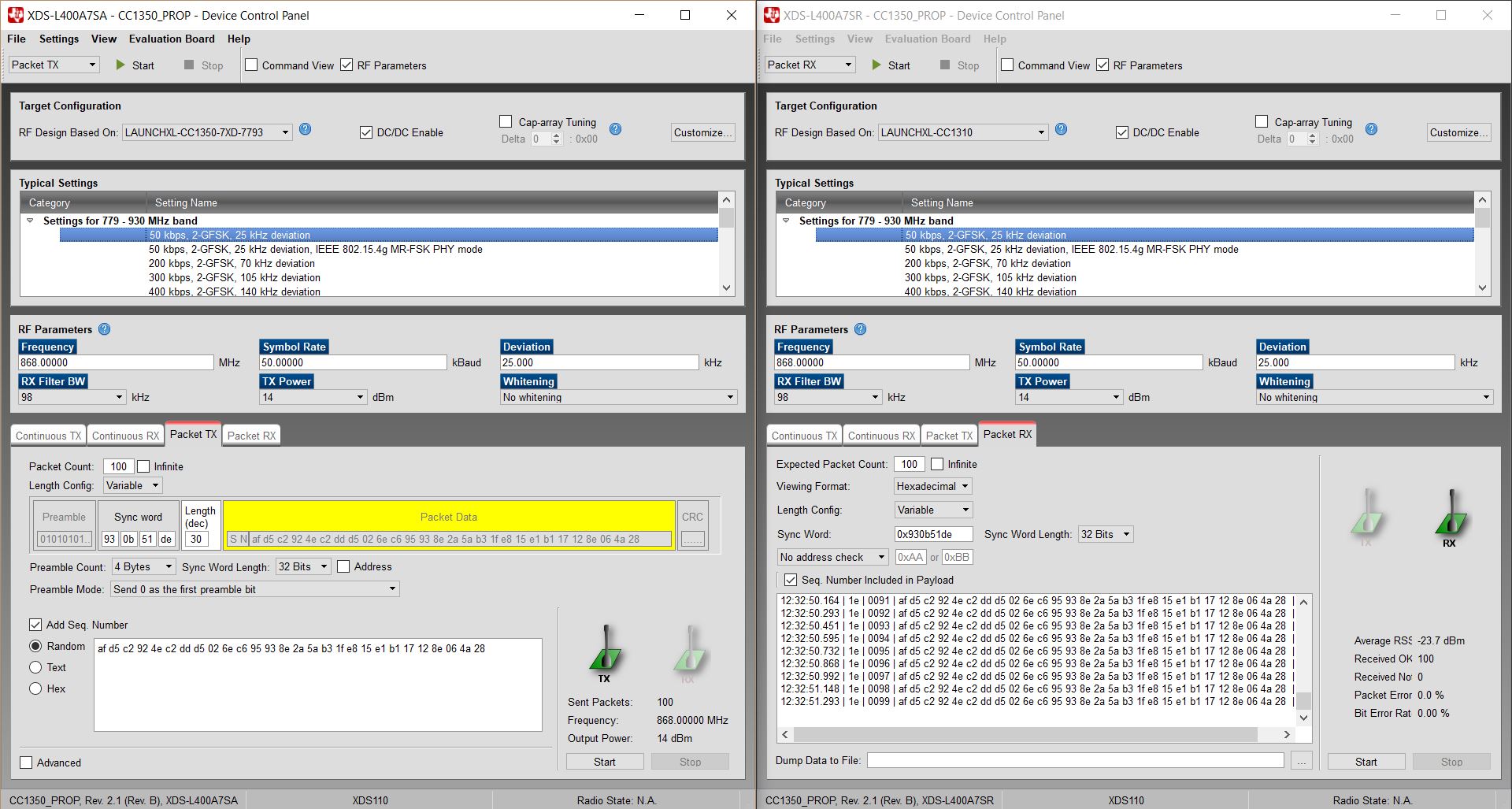
**Date Submitted: 11/19/2018**

**Task 00:** For this task we just had to label our launchpads. The first device listed in the image below is our Transmit while the other is our receive.

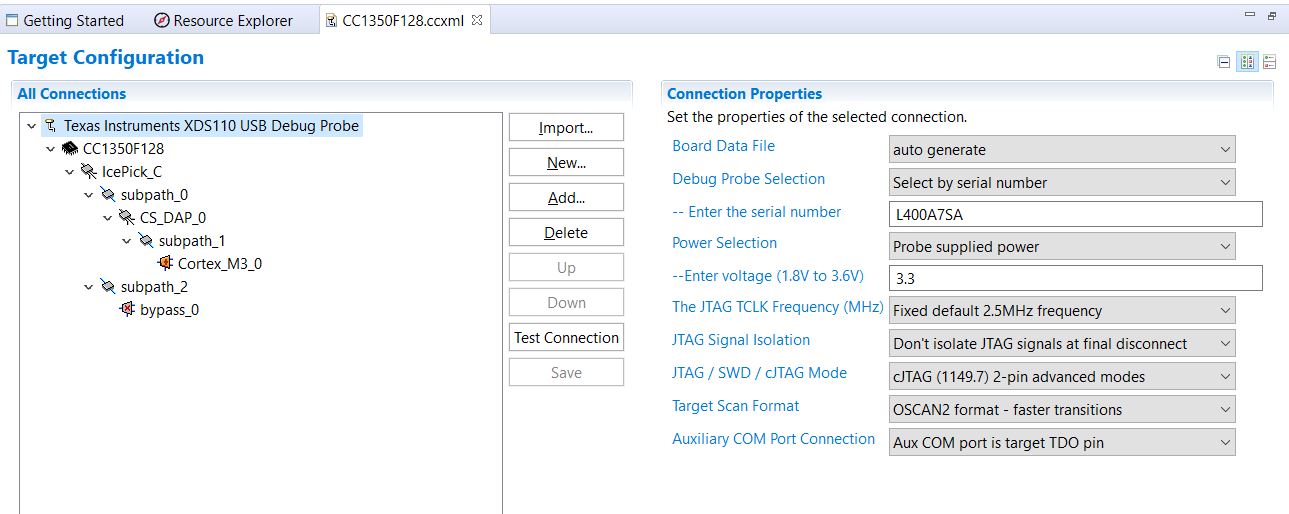


**Task 01:** The goal of this task is to transmit 100 packets of data over to our receive launchpad which was done successfully as shown by the video and image below.

****

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

**Task 02:** For this task we had to import the rfPacketTx example project found in our simplelink sdk cc13x folder. The code below was the imported code that was responsible for transmitting packets of data over to the receive launchpad.The image below shows the modification to the rfPacketTX’s target configuration, so it knows that the launchpad with the serial number L400A7SA will act as our transmitter.

****

Youtube Link: <https://www.youtube.com/watch?v=aF38S6YWQdk>

**Modified Schematic (if applicable):**

**Modified Code:**

**// Insert code here**

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

**#if** defined Board\_CC1352R1\_LAUNCHXL

Board\_DIO30\_RFSW | PIN\_GPIO\_OUTPUT\_EN | PIN\_GPIO\_HIGH | PIN\_PUSHPULL | PIN\_DRVSTR\_MAX,

**#endif**

**#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 \*/

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);

**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);

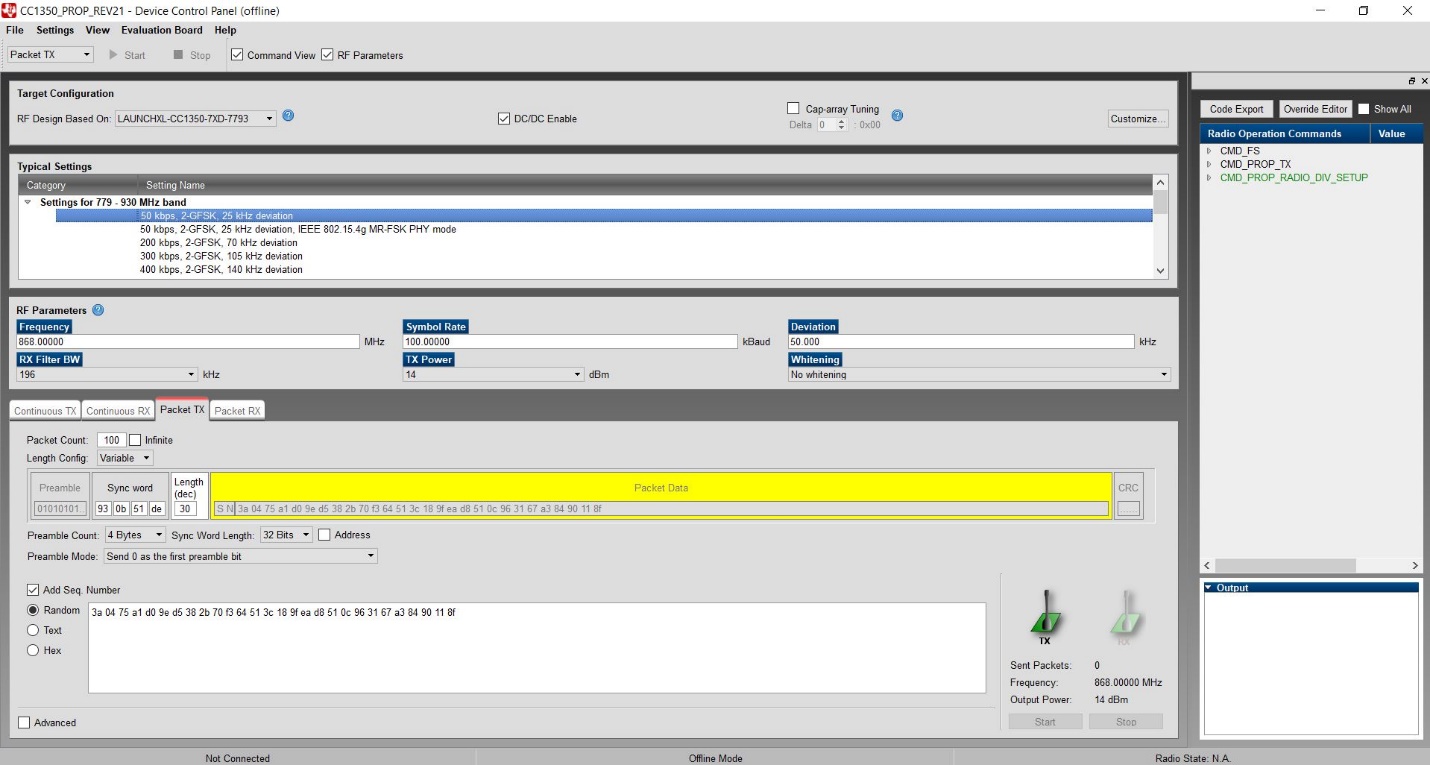
**#endif**

}

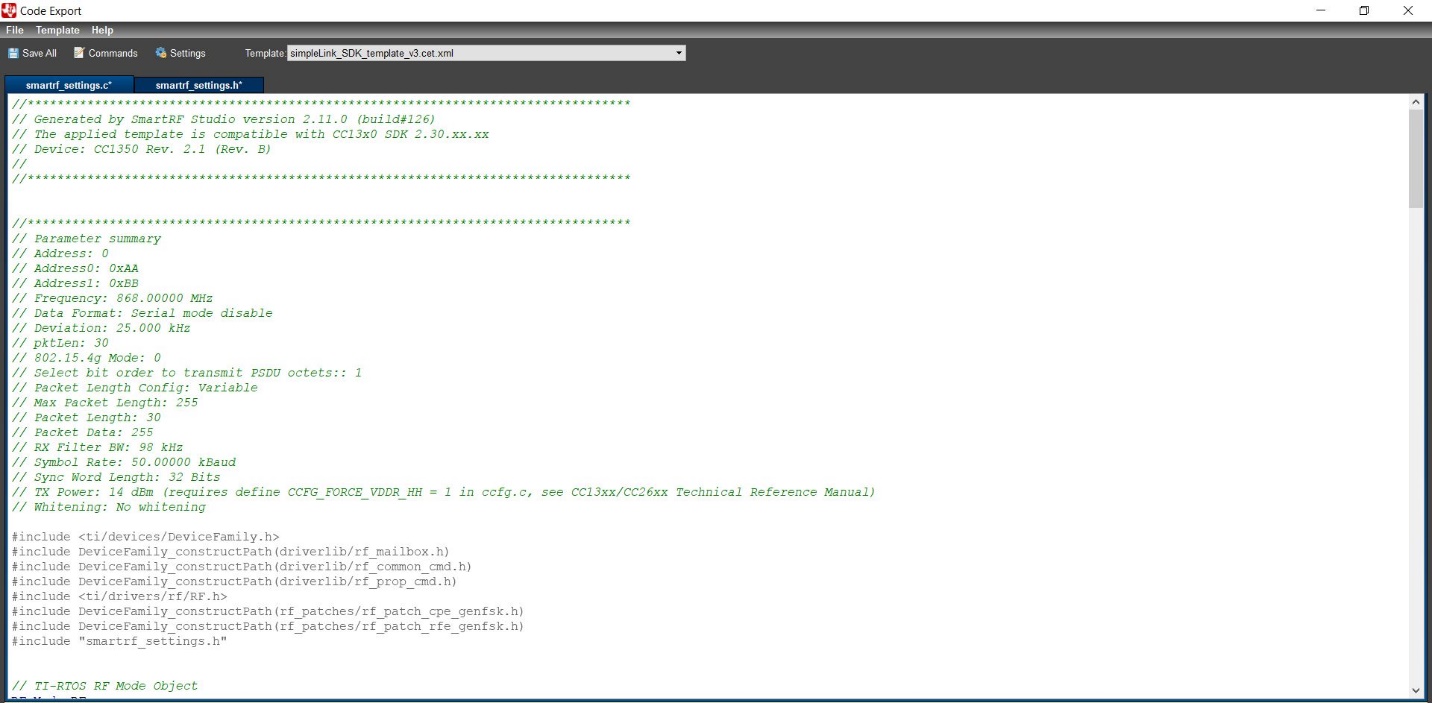
}

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

**Task 03:** The goal of this task is to export RF settings from the SmartRF studio. The image below shows the settings we used in the RF parameters for the packet Tx.



The image below shows the new set of smartrf\_settings files after we exported the code. These were then saved to the project files.

****

Youtube Link: <https://www.youtube.com/watch?v=luJDzc6Q1js>

**Modified Schematic (if applicable): N/A**

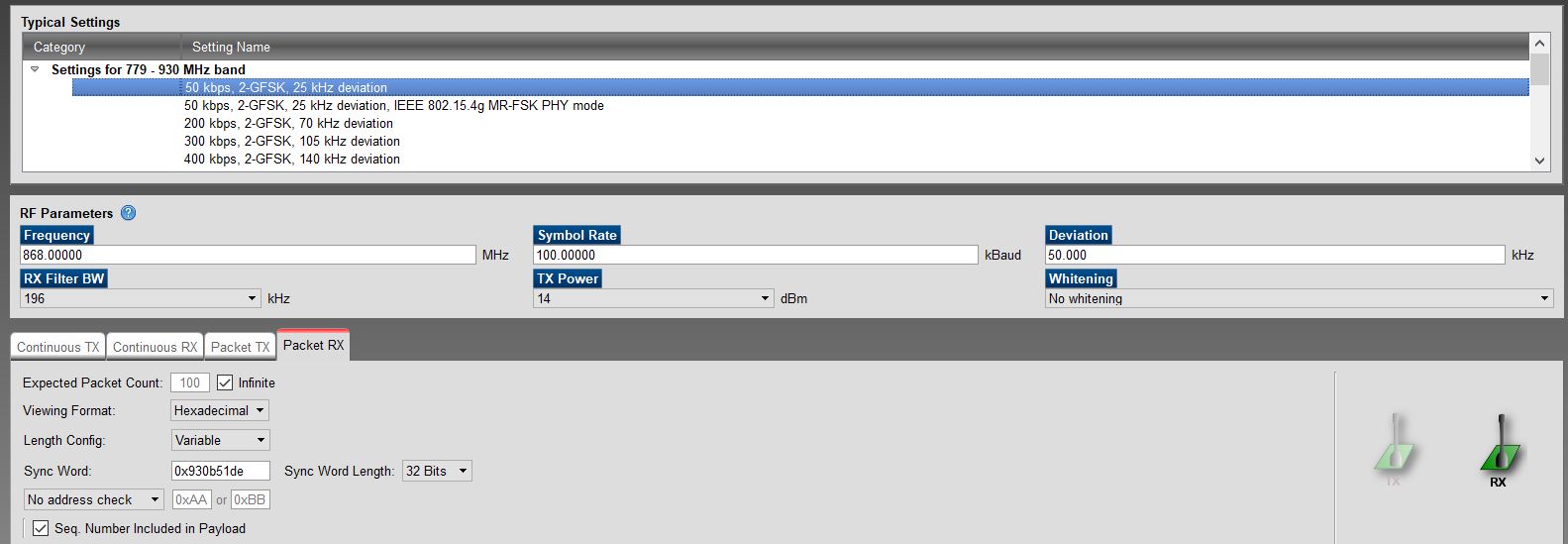
**Modified Code:**

**// Insert code here**

**No Modifications to the code were made, we only had to use the SmartRF Studio.**

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

**Task 04:** For this task, all we had to do was create another instance of SmartRF studio with our RX launchpad. The image below shows the settings we used to configure the Packet RX.

****

Youtube Link: <https://youtu.be/Mo-PQ4IHJ0U>

**Modified Schematic (if applicable): N/A**

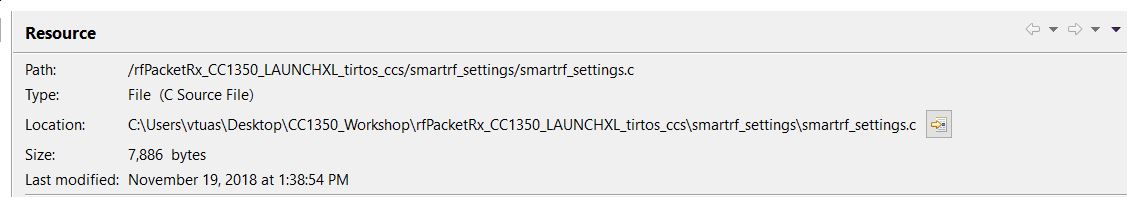
**Modified Code:**

**// Insert code here**

**No Modifications to the code were made, we only had to use the SmartRF Studio.**

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

**Task 05:** In this task we had to import another project which contained the rfPacketRx code and flash our RX launchpad with it. The image below shows the new path for the new smartrf\_settings files.

****

Youtube Link: <https://www.youtube.com/watch?v=aF38S6YWQdk>

**Modified Schematic (if applicable):**

**Modified Code:**

**// Insert code here**

/\*\*\*\*\* Includes \*\*\*\*\*/

/\* Standard C Libraries \*/

**#include** <stdlib.h>

/\* TI Drivers \*/

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

**#include** <ti/drivers/PIN.h>

/\* Driverlib Header files \*/

**#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"

/\*\*\*\*\* 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) \*/

/\*\*\*\*\* Prototypes \*\*\*\*\*/

**static** **void** **callback**(RF\_Handle h, RF\_CmdHandle ch, RF\_EventMask e);

/\*\*\*\*\* Variable declarations \*\*\*\*\*/

**static** RF\_Object rfObject;

**static** RF\_Handle rfHandle;

/\* Pin driver handle \*/

**static** PIN\_Handle ledPinHandle;

**static** PIN\_State ledPinState;

/\* 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\_LED2 | PIN\_GPIO\_OUTPUT\_EN | PIN\_GPIO\_LOW | PIN\_PUSHPULL | PIN\_DRVSTR\_MAX,

**#if** defined Board\_CC1352R1\_LAUNCHXL

Board\_DIO30\_RFSW | PIN\_GPIO\_OUTPUT\_EN | PIN\_GPIO\_HIGH | PIN\_PUSHPULL | PIN\_DRVSTR\_MAX,

**#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);

}

**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 \*/

rfHandle = **RF\_open**(&rfObject, &RF\_prop,

(RF\_RadioSetup\*)&RF\_cmdPropRadioDivSetup, &rfParams);

/\* Set the frequency \*/

**RF\_postCmd**(rfHandle, (RF\_Op\*)&RF\_cmdFs, *RF\_PriorityNormal*, NULL, 0);

/\* Enter RX mode and stay forever in RX \*/

RF\_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 06:** The goal of this task is to just let the two launchpads communicate with each other. This is demonstrated in the video linked below.

Youtube Link: <https://www.youtube.com/watch?v=aF38S6YWQdk>

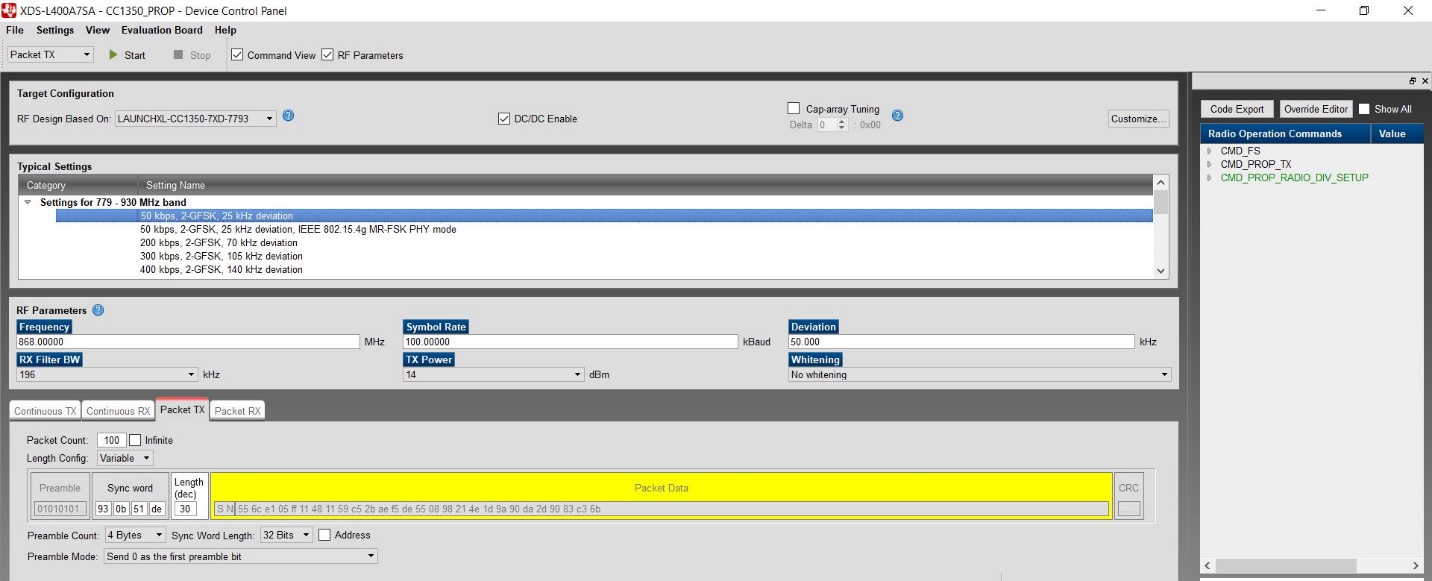
**Modified Schematic (if applicable):**

**Modified Code:**

**No Modifications to the code were made, we only had to change the settings in SmartRF Studio.**

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

**Task 07:** For this task, we had to transmit packets of data using the SmartRF studio and receive those packets of data with the firmware RX. The image below shows the settings used for RF parameters and the Packet TX tab. One of the launchpads will have no blinking LED because it is not used in this task since the SmartRF studio will be the one transmitting while the one launchpad receives.

****

Youtube Link: <https://youtu.be/9pmSrILy_Ms>

**Modified Schematic (if applicable): N/A**

**Modified Code:**

**No Modifications to the code were made, we only had to change the settings in SmartRF Studio.**

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