**Date Submitted: 11/20/2018**

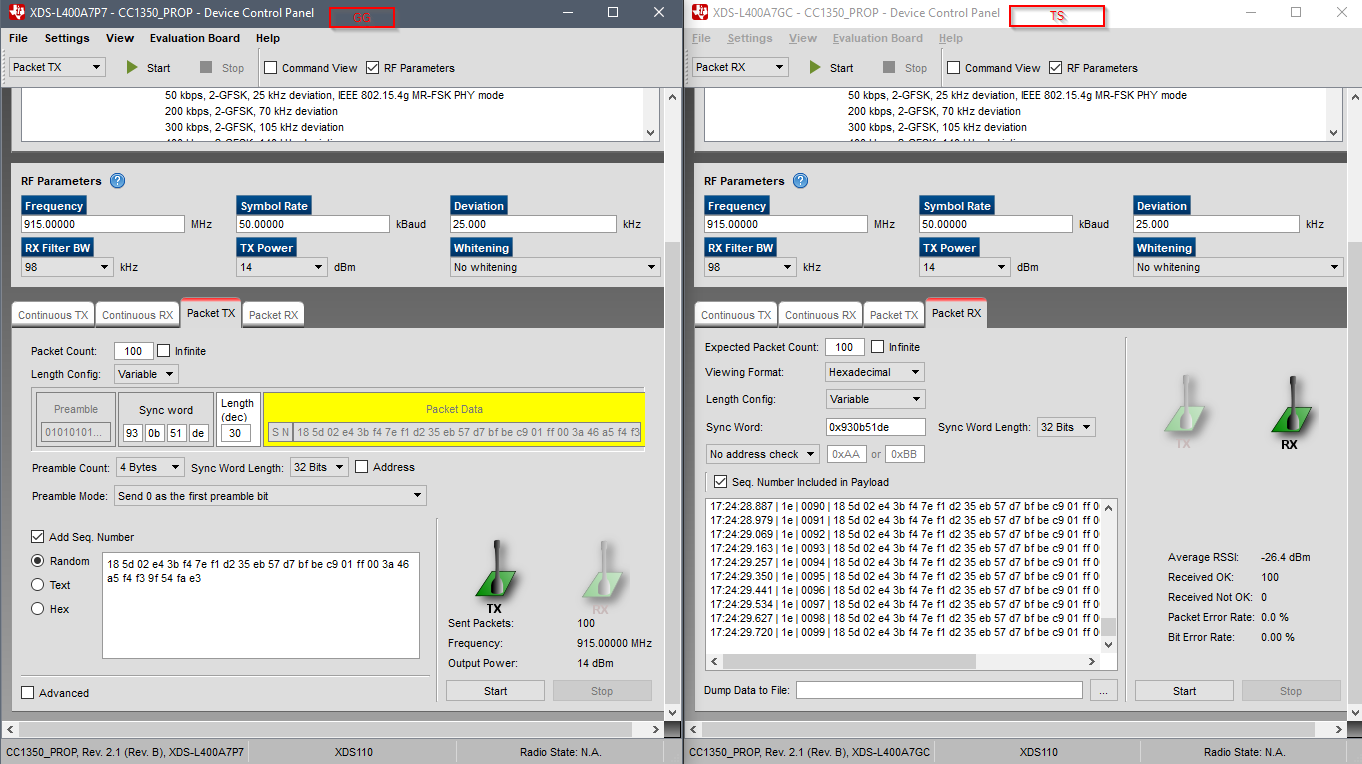
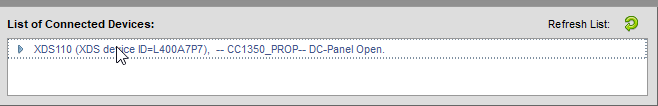
Guillermo Gálvez

Github root directory:(https://github.com/galveg1/VMs\_House-of-Fun-Or-Pain.git)

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

**Task 1: Tx**

Youtube Link: N/A



*/\*\*\*\*\* 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 2: Tx**

Youtube: <https://youtu.be/nM4mXKl9i_w>

**The PDf states it should be the red LED blinking but the green is what was blinking after running the package.**

**Modified code:**

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

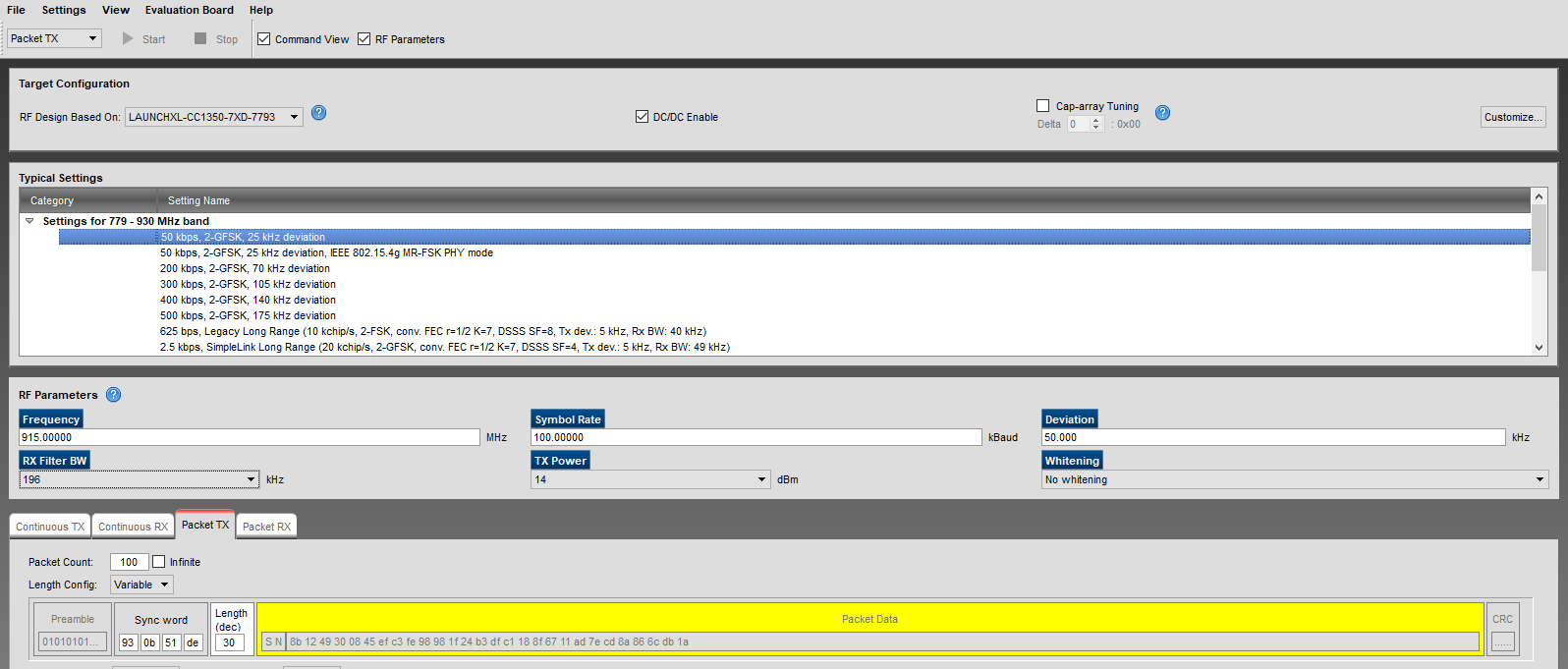
#endif

}

}

**Task 3: Exporting and using RF config**

Youtube:N/A

****

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Generated by SmartRF Studio version 2.11.0 (build#126)

// The applied template is compatible with CC13x0 SDK 2.30.xx.xx

// Device: CC1350 Rev. 2.1

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

EXPORTED:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Parameter summary

// Address: 0

// Address0: 0xAA

// Address1: 0xBB

// Frequency: 915.00000 MHz

// Data Format: Serial mode disable

// Deviation: 50.000 kHz

// pktLen: 30

// 802.15.4g Mode: 0

// Select bit order to transmit PSDU octets:: 1

// Packet Length Config: Variable

// Max Packet Length: 255

// Packet Length: 20

// Packet Data: 255

// RX Filter BW: 196 kHz

// Symbol Rate: 100.00000 kBaud

// Sync Word Length: 32 Bits

// TX Power: 14 dBm (requires define CCFG\_FORCE\_VDDR\_HH = 1 in ccfg.c, see CC13xx/CC26xx Technical Reference Manual)

// Whitening: No whitening

#include <ti/devices/DeviceFamily.h>

#include DeviceFamily\_constructPath(driverlib/rf\_mailbox.h)

#include DeviceFamily\_constructPath(driverlib/rf\_common\_cmd.h)

#include DeviceFamily\_constructPath(driverlib/rf\_prop\_cmd.h)

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

#include DeviceFamily\_constructPath(rf\_patches/rf\_patch\_cpe\_genfsk.h)

#include DeviceFamily\_constructPath(rf\_patches/rf\_patch\_rfe\_genfsk.h)

#include "smartrf\_settings.h"

// TI-RTOS RF Mode Object

RF\_Mode RF\_prop =

{

.rfMode = RF\_MODE\_PROPRIETARY\_SUB\_1,

.cpePatchFxn = &rf\_patch\_cpe\_genfsk,

.mcePatchFxn = 0,

.rfePatchFxn = &rf\_patch\_rfe\_genfsk,

};

// Overrides for CMD\_PROP\_RADIO\_DIV\_SETUP

uint32\_t pOverrides[] =

{

// override\_use\_patch\_prop\_genfsk.xml

// PHY: Use MCE ROM bank 4, RFE RAM patch

MCE\_RFE\_OVERRIDE(0,4,0,1,0,0),

// override\_synth\_prop\_863\_930\_div5.xml

// Synth: Set recommended RTRIM to 7

HW\_REG\_OVERRIDE(0x4038,0x0037),

// Synth: Set Fref to 4 MHz

(uint32\_t)0x000684A3,

// Synth: Configure fine calibration setting

HW\_REG\_OVERRIDE(0x4020,0x7F00),

// Synth: Configure fine calibration setting

HW\_REG\_OVERRIDE(0x4064,0x0040),

// Synth: Configure fine calibration setting

(uint32\_t)0xB1070503,

// Synth: Configure fine calibration setting

(uint32\_t)0x05330523,

// Synth: Set loop bandwidth after lock to 20 kHz

(uint32\_t)0x0A480583,

// Synth: Set loop bandwidth after lock to 20 kHz

(uint32\_t)0x7AB80603,

// Synth: Configure VCO LDO (in ADI1, set VCOLDOCFG=0x9F to use voltage input reference)

ADI\_REG\_OVERRIDE(1,4,0x9F),

// Synth: Configure synth LDO (in ADI1, set SLDOCTL0.COMP\_CAP=1)

ADI\_HALFREG\_OVERRIDE(1,7,0x4,0x4),

// Synth: Use 24 MHz XOSC as synth clock, enable extra PLL filtering

(uint32\_t)0x02010403,

// Synth: Configure extra PLL filtering

(uint32\_t)0x00108463,

// Synth: Increase synth programming timeout (0x04B0 RAT ticks = 300 us)

(uint32\_t)0x04B00243,

// override\_phy\_rx\_aaf\_bw\_0xd.xml

// Rx: Set anti-aliasing filter bandwidth to 0xD (in ADI0, set IFAMPCTL3[7:4]=0xD)

ADI\_HALFREG\_OVERRIDE(0,61,0xF,0xD),

// override\_phy\_gfsk\_rx.xml

// Rx: Set LNA bias current trim offset to 3

(uint32\_t)0x00038883,

// Rx: Freeze RSSI on sync found event

HW\_REG\_OVERRIDE(0x6084,0x35F1),

// override\_phy\_gfsk\_pa\_ramp\_agc\_reflevel\_0x1a.xml

// Tx: Configure PA ramping setting (0x41). Rx: Set AGC reference level to 0x1A.

HW\_REG\_OVERRIDE(0x6088,0x411A),

// Tx: Configure PA ramping setting

HW\_REG\_OVERRIDE(0x608C,0x8213),

// override\_phy\_rx\_rssi\_offset\_5db.xml

// Rx: Set RSSI offset to adjust reported RSSI by +5 dB (default: 0), trimmed for external bias and differential configuration

(uint32\_t)0x00FB88A3,

// TX power override

// Tx: Set PA trim to max (in ADI0, set PACTL0=0xF8)

ADI\_REG\_OVERRIDE(0,12,0xF8),

(uint32\_t)0xFFFFFFFF

};

// CMD\_PROP\_RADIO\_DIV\_SETUP

// Proprietary Mode Radio Setup Command for All Frequency Bands

rfc\_CMD\_PROP\_RADIO\_DIV\_SETUP\_t RF\_cmdPropRadioDivSetup =

{

.commandNo = 0x3807,

.status = 0x0000,

.pNextOp = 0, // INSERT APPLICABLE POINTER: (uint8\_t\*)&xxx

.startTime = 0x00000000,

.startTrigger.triggerType = 0x0,

.startTrigger.bEnaCmd = 0x0,

.startTrigger.triggerNo = 0x0,

.startTrigger.pastTrig = 0x0,

.condition.rule = 0x1,

.condition.nSkip = 0x0,

.modulation.modType = 0x1,

.modulation.deviation = 0xC8,

.symbolRate.preScale = 0xF,

.symbolRate.rateWord = 0x10000,

.symbolRate.decimMode = 0x0,

.rxBw = 0x27,

.preamConf.nPreamBytes = 0x4,

.preamConf.preamMode = 0x0,

.formatConf.nSwBits = 0x20,

.formatConf.bBitReversal = 0x0,

.formatConf.bMsbFirst = 0x1,

.formatConf.fecMode = 0x0,

.formatConf.whitenMode = 0x0,

.config.frontEndMode = 0x0,

.config.biasMode = 0x1,

.config.analogCfgMode = 0x0,

.config.bNoFsPowerUp = 0x0,

.txPower = 0xAB3F,

.pRegOverride = pOverrides,

.centerFreq = 0x0393,

.intFreq = 0x8000,

.loDivider = 0x05

};

// CMD\_FS

// Frequency Synthesizer Programming Command

rfc\_CMD\_FS\_t RF\_cmdFs =

{

.commandNo = 0x0803,

.status = 0x0000,

.pNextOp = 0, // INSERT APPLICABLE POINTER: (uint8\_t\*)&xxx

.startTime = 0x00000000,

.startTrigger.triggerType = 0x0,

.startTrigger.bEnaCmd = 0x0,

.startTrigger.triggerNo = 0x0,

.startTrigger.pastTrig = 0x0,

.condition.rule = 0x1,

.condition.nSkip = 0x0,

.frequency = 0x0393,

.fractFreq = 0x0000,

.synthConf.bTxMode = 0x0,

.synthConf.refFreq = 0x0,

.\_\_dummy0 = 0x00,

.\_\_dummy1 = 0x00,

.\_\_dummy2 = 0x00,

.\_\_dummy3 = 0x0000

};

// CMD\_PROP\_TX

// Proprietary Mode Transmit Command

rfc\_CMD\_PROP\_TX\_t RF\_cmdPropTx =

{

.commandNo = 0x3801,

.status = 0x0000,

.pNextOp = 0, // INSERT APPLICABLE POINTER: (uint8\_t\*)&xxx

.startTime = 0x00000000,

.startTrigger.triggerType = 0x0,

.startTrigger.bEnaCmd = 0x0,

.startTrigger.triggerNo = 0x0,

.startTrigger.pastTrig = 0x0,

.condition.rule = 0x1,

.condition.nSkip = 0x0,

.pktConf.bFsOff = 0x0,

.pktConf.bUseCrc = 0x1,

.pktConf.bVarLen = 0x1,

.pktLen = 0x14, // SET APPLICATION PAYLOAD LENGTH

.syncWord = 0x930B51DE,

.pPkt = 0 // INSERT APPLICABLE POINTER: (uint8\_t\*)&xxx

};

// CMD\_PROP\_RX

// Proprietary Mode Receive Command

rfc\_CMD\_PROP\_RX\_t RF\_cmdPropRx =

{

.commandNo = 0x3802,

.status = 0x0000,

.pNextOp = 0, // INSERT APPLICABLE POINTER: (uint8\_t\*)&xxx

.startTime = 0x00000000,

.startTrigger.triggerType = 0x0,

.startTrigger.bEnaCmd = 0x0,

.startTrigger.triggerNo = 0x0,

.startTrigger.pastTrig = 0x0,

.condition.rule = 0x1,

.condition.nSkip = 0x0,

.pktConf.bFsOff = 0x0,

.pktConf.bRepeatOk = 0x0,

.pktConf.bRepeatNok = 0x0,

.pktConf.bUseCrc = 0x1,

.pktConf.bVarLen = 0x1,

.pktConf.bChkAddress = 0x0,

.pktConf.endType = 0x0,

.pktConf.filterOp = 0x0,

.rxConf.bAutoFlushIgnored = 0x0,

.rxConf.bAutoFlushCrcErr = 0x0,

.rxConf.bIncludeHdr = 0x1,

.rxConf.bIncludeCrc = 0x0,

.rxConf.bAppendRssi = 0x0,

.rxConf.bAppendTimestamp = 0x0,

.rxConf.bAppendStatus = 0x1,

.syncWord = 0x930B51DE,

.maxPktLen = 0xFF, // MAKE SURE DATA ENTRY IS LARGE ENOUGH

.address0 = 0xAA,

.address1 = 0xBB,

.endTrigger.triggerType = 0x1,

.endTrigger.bEnaCmd = 0x0,

.endTrigger.triggerNo = 0x0,

.endTrigger.pastTrig = 0x0,

.endTime = 0x00000000,

.pQueue = 0, // INSERT APPLICABLE POINTER: (dataQueue\_t\*)&xxx

.pOutput = 0 // INSERT APPLICABLE POINTER: (uint8\_t\*)&xxx

};

**Task 4:** Firmware TX → SmartRF Studio RX

: <https://www.youtube.com/watch?v=M3qBRHW998k>

Second unit T. Stewart

TS 1350 Tx -> GG Rx 1350

Repeat of 3&4

