Application Note

CC13xx IQ Samples

ABSTRACT

This application report describes an IQ dump patch for the CC13xx SimpleLink™ Sub-1 GHz ultra-low power wireless microcontroller (MCU).

Project collateral and source code mentioned in this document can be downloaded from the following link: http://www.ti.com/lit/zip/swra571. The smartrf_settings.c file is for the CC1310, but similar files can be made for the other CC13x0 and CC13x2 devices, following the steps explained in Section 2.1.1 and Section 2.1.2.

Table of Contents

1 Introduction 1
2 IQ Dump Patch 2
2.1 Recommended Operating Limits 2
3 Building a Software Example 3
4 Testing the Patch Using the Built-In Test Pattern 6
5 References 7
6 Revision History 7

List of Figures

Figure 4-1. Built-In Test Pattern Stored as I and Q Samples 7

List of Tables

Table 2-1. Format of IQ Samples Stored in RAM 2
Table 2-2. Overrides and Mode of Operation 2
Table 2-3. API Modifications 3

Trademarks

1 Introduction

CC13xx SimpleLink™ Sub-1 GHz ultralow power wireless microcontroller (MCU) is centered around an ARM® Cortex®-M3 (CC13x0) or ARM Cortex-M4F (CC13x2) series processor that handles the application and an autonomous RF Core that handles all the low-level radio control and processing needed to transfer digital bits over the air. Normally the customers use the CM3/CM4F to implement their application/high level protocols on top of the physical layer, but it is also possible to use it to implement other novel or legacy physical layer modulation schemes. In order to do so, the CM3/CM4F requires access to the raw IQ samples in RX mode. With the default genfsk/prop PHY, IQ samples are not available outside RF Core and a dedicated patch is needed to automatically copy IQ samples to a partial read RX entry. This application note describes how to get access to these IQ samples using the IQ dump patch.

2 IQ Dump Patch

The IQ Dump patch (rf_patch_mce_iqdump.h) can run in two different modes; IQFifoBlind and IQFifoSync. IQFifoBlind mode starts copying IQ samples immediately while IQFifoSync mode starts copying IQ samples after a sync word has been detected. The mode of operation is selected by the MCE_RFE override (see Table 2-2). For both modes, IQ samples are copied through the RF Core's internal FIFO to one or more partial read RX entries in the system RAM. The application simply waits for an RX_ENTRY_DONE interrupt saying that a partial read entry is full.

The IQ sample rate is fixed to 4 times oversampling and the IQ sample size is 12 bits. This means that each IQ pair will occupy 3 bytes in RAM in the format shown in Table 2-1. The format is signed meaning that MSB is the sign bit (two's complement format).

Table 2-1. Format of IQ Samples Stored in RAM

Byte	Bit Definition							
0	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0
1	Q_3	Q_2	Q_1	Q_0	I_{11}	I_{10}	I_9	I_8
2	Q_{11}	Q_{10}	Q_9	Q_8	Q_7	Q_6	Q_5	Q_4

The patch has a built-in test pattern where the IQ samples are replaced with two counter values. The I-sample is replaced with an increasing counter value and the Q-sample is replaced with a decreasing counter value. The test pattern is enabled by the following register override:

```
HW_REG_OVERRIDE(0x52B4, 0x070D) // CC13x0
HW_REG_OVERRIDE(0x5328, 0x070D) // CC13x2
```

2.1 Recommended Operating Limits

When using the IQ Dump patch in RX the data rate is limited upwards to 12.5 kbps for CC13x0 and 25 kbps for CC13x2. In TX, the patch can be used within the same operating limits as the genfsk/prop PHY. The 50-kbps settings from SmartRF™ Studio [1] should be used as a starting point. The associated zip file contains a smartrf_settings.c file (for CC1310) that has the complete override list and API settings to be used with the patch.

2.1.1 Register Overrides

The MCE_RFE override [1]needs to be modified when running the IQ Dump patch. Table 2-2 shows how this should be done. In addition, there is one other override necessary to add when running the patch.

Table 2-2. Overrides and Mode of Operation

Override	Description

MCE_RFE_OVERRIDE(1,0,2,1,0,0) // CC13x0
MCE_RFE_OVERRIDE(1,0,2,0,4,0) // CC13x2
	Setting the mode of operation to IQFifoBlind

MCE_RFE_OVERRIDE(1,0,3,1,0,0) // CC13x0
MCE_RFE_OVERRIDE(1,0,3,0,4,0) // CC13x2
	Setting the mode of operation to IQFifoSync

(uint32_t)0x001082C3	Set to avoid internal FIFO overflow

HW_REG_OVERRIDE(0x52B4, 0x070D) (optional) // CC13x0
HW_REG_OVERRIDE(0x5328, 0x070D) (optional) // CC13x2
	Enable built-in test pattern. Should only be included when testing the patch. For more details, see Section 3 (optional).

[1] Only one MCE_RFE can be present in the override list.

In addition you need to include the patch and update the TI_RTOS RF Mode Object:

CC13x0:

```
#include DeviceFamily_constructPath(rf_patches/rf_patch_cpe_genfsk.h)
#include DeviceFamily_constructPath(rf_patches/rf_patch_mce_iqdump.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 = &rf_patch_mce_iqdump,
    .rfePatchFxn = &rf_patch_rfe_genfsk,
};
```

CC13x2:

```
#include DeviceFamily_constructPath(rf_patches/rf_patch_cpe_prop.h)
#include DeviceFamily_constructPath(rf_patches/rf_patch_mce_iqdump.h)
#include "smartrf_settings.h"
// TI-RTOS RF Mode Object
RF_Mode RF_prop =
{
    .rfMode = RF_MODE_AUTO,
    .cpePatchFxn = &rf_patch_cpe_prop,
    .mcePatchFxn = &rf_patch_mce_iqdump,
    .rfePatchFxn = 0,
};
```

2.1.2 API Configuration

When using the patch some changes have to be done to the API exported from SmartRF Studio. formatConf.bMsbFirst in CMD_PROP_RADIO_DIV_SETUP must be set to 0 to allow for LSB to be transmitted first and maxPktLen in CMD_PROP_RX must be set to 0 for unlimited packet length. The RX bandwidth should be set to 39/38.9 kHz (CC13x0/CC13x2), and a good starting point for the deviation is to set it to half the data rate (see Table 2-3).

Table 2-3. API Modifications

API Field	Value	Comment
RF_cmdPropRadioDivSetup.formatConf.bMsbFirst	0	Least significant bit transmitted first
RF_cmdPropRx.maxPktLen	0	Unlimited length
RF_cmdPropRadioDivSetup.modulation.deviation	0x19	6.25 kHz
RF_cmdPropRadioDivSetup.rxBw	0x20 0x4D	39 kHz (CC13x0) 38.9 kHz (CC13x2)
RF_cmdPropRadioDivSetup.symbolRate.rateWord	0x2000	12.5 kbps

3 Building a Software Example

To test the RF performance of the patch, see the *rfPacketRX* example available when downloading [2] or [3].

The smartrf_settings.c file must be replaced with the one from the zip file that can be downloaded from the following link: http://www.ti.com/lit/zip/swra571. The following modifications must be done to rfPacketRX.c to be able to test the patch:

1. Define how many IQ sample pairs you want.

   ```
   #define NUMBER_OF_SAMPLE_PAIRS 300
   ```

   Setting `NUMBER_OF_SAMPLE_PAIRS` to 300 means that each data entry used must have room for 300 x 3 bytes.

2. Configure two partial read buffers for the received data. Make sure that the buffers are 4 byte aligned.

```
#define PARTIAL_RX_ENTRY_HEADER_SIZE 12

#if defined(__TI_COMPILER_VERSION__)
#pragma DATA_ALIGN (rxDataEntryBuf1, 4);
static uint8_t rxDataEntryBuf1[PARTIAL_RX_ENTRY_HEADER_SIZE +
                               (NUMBER_OF_SAMPLE_PAIRS * 3)];
#pragma DATA_ALIGN (rxDataEntryBuf2, 4);
static uint8_t rxDataEntryBuf2[PARTIAL_RX_ENTRY_HEADER_SIZE +
                               (NUMBER_OF_SAMPLE_PAIRS * 3)];
#endif
rfc_dataEntryPartial_t* partialReadEntry1 = (rfc_dataEntryPartial_t*)&rxDataEntryBuf1;
rfc_dataEntryPartial_t* partialReadEntry2 = (rfc_dataEntryPartial_t*)&rxDataEntryBuf2;
rfc_dataEntryPartial_t* currentReadEntry = (rfc_dataEntryPartial_t*)&rxDataEntryBuf1;

void *mainThread(void *arg0)
{
    RF_Params rfParams;
    RF_Params_init(&rfParams);
    partialReadEntry1->length = (NUMBER_OF_SAMPLE_PAIRS * 3) + 4;
    partialReadEntry1->config.type = DATA_ENTRY_TYPE_PARTIAL;
    partialReadEntry1->status = DATA_ENTRY_PENDING;

    partialReadEntry2->length = (NUMBER_OF_SAMPLE_PAIRS * 3) + 4;
    partialReadEntry2->config.type = DATA_ENTRY_TYPE_PARTIAL;
    partialReadEntry2->status = DATA_ENTRY_PENDING;

    partialReadEntry1->pNextEntry = (uint8_t*)partialReadEntry2;
    partialReadEntry2->pNextEntry = (uint8_t*)partialReadEntry1;

    dataQueue.pCurrEntry = (uint8_t*)partialReadEntry1;
    dataQueue.pLastEntry = NULL;
```

3. Remove *RFQueue_defineQueue* and the modifications of *RF_cmdPropRX*, except for the *RF_cmdPropRx.pQueue*.

```
// 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);
//}

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;
```

4. Implement handling of the IQ samples in the callback. In the callback the samples should simply be read from the data entries to make the data entries available for new samples. The processing of the IQ samples should be done outside the callback. It is not the scope of this application report to show how this can be done. The code below simply shows how to get access to the samples and how to handle the queue.

```
void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e)
{
    if (e & RF_EventRxEntryDone)
    {
        // Toggle pin to indicate RX
        PIN_setOutputValue(pinHandle,
                           Board_PIN_LED2,!PIN_getOutputValue(Board_PIN_LED2));
        // Get a pointer to the first IQ sample byte
        packetDataPointer = &currentReadEntry->rxData;
        //--------------------------------------------------------------------------
        // Implement code for handling the IQ data
        //    .
        //    .
        //    .
        //    .
        //--------------------------------------------------------------------------
        currentReadEntry->status = DATA_ENTRY_PENDING;
        currentReadEntry = (rfc_dataEntryPartial_t*)currentReadEntry->pNextEntry;
    }
}
```

4 Testing the Patch Using the Built-In Test Pattern

To test that the data entries are set up correctly and that the patch is working you can enable the built-in test pattern (see Table 2-2) and declare two arrays (iSamples and qSamples) that can hold the “received” I and Q samples.

```
#define NUMBER_OF_BUFFERS 5
static uint16_t iSamples[NUMBER_OF_SAMPLE_PAIRS*NUMBER_OF_BUFFERS];
static uint16_t qSamples[NUMBER_OF_SAMPLE_PAIRS*NUMBER_OF_BUFFERS];
```

For test purposes, set *NUMBER_OF_SAMPLE_PAIRS* to a low number[2] to easier be able to go through the array to see that everything is OK.

```
#define NUMBER_OF_SAMPLE_PAIRS 8
```

In the callback, where code for handling the samples should be implemented, the following code was added:

```
static uint16_t index = 0;
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 a pointer to the first IQ sample byte
        packetDataPointer = &currentReadEntry->rxData;
        //------------------------------------------------------------------------
        // Implement code for handling the IQ data
        // In this example, I and Q data are simply copied into two separate array
        {
            uint16_t i;
            // IQ Sample Handling
            for (i = index; i < (NUMBER_OF_SAMPLE_PAIRS + index); i++)
            {
                iSamples[i] = (((*(packetDataPointer + 1)) << 8) |
                               (*packetDataPointer)) & 0x0FFF;
                qSamples[i] = (((*(packetDataPointer + 2)) << 8) |
                               (*(packetDataPointer + 1))) >> 4;
                packetDataPointer += 3;
            }
        }
        index += NUMBER_OF_SAMPLE_PAIRS;
        if (index == (NUMBER_OF_SAMPLE_PAIRS*NUMBER_OF_BUFFERS))
        {
            index = 0;
        }
        //------------------------------------------------------------------------
        currentReadEntry->status = DATA_ENTRY_PENDING;
        currentReadEntry = (rfc_dataEntryPartial_t*)currentReadEntry->pNextEntry;
    }
}
```

[2] In this example, *NUMBER_OF_SAMPLE_PAIRS* cannot be set lower than 8, as this will make the data entry overflow (*RF_cmdPropRx.status* = *PROP_ERROR_RXOVF*)

Figure 4-1 shows the five buffers with eight IQ sample pairs in each stored in an iSamples and qSamples array, each holding 40 samples (*NUMBER_OF_BUFFERS* · *NUMBER_OF_SAMPLE_PAIRS*).

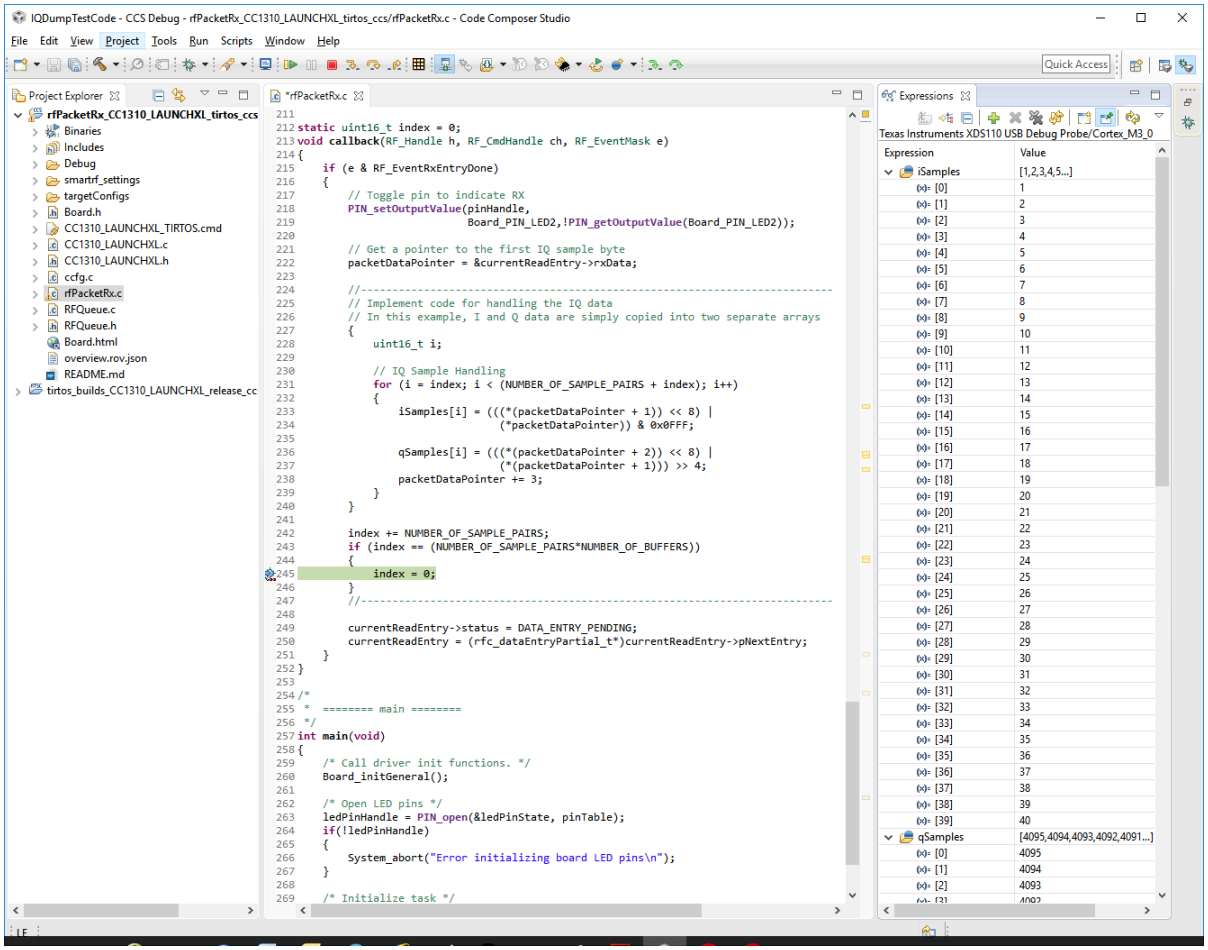

Figure 4-1. Built-In Test Pattern Stored as I and Q Samples

5 References

1. *SmartRF Studio 7*
2. *SimpleLink™ Sub-1 GHz CC13x0 Software Development Kit*
3. *SimpleLink™ CC13x2 and CC26x2 Software Sevelopment Kit*

6 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision A (April 2019) to Revision B (August 2020)	Page
• Updated the numbering format for tables, figures, and cross-references throughout the document	1
• Updated Table 2-2 *Overrides and Mode of Operation* with MCE_RFE_OVERRIDE for CC13x2	2

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265