参考資料

Application Note

SysConfig または SmartRF™ Studio からインポート / エクスポートされた TX および RX コマンドを変更する方法

Siri Johnsrud

概要

このドキュメントでは、simplelink_cc13xx_cc26xx_sdk_x_xx_xx_xx [1] にある rfPacketRx および rfPacketTx の例を、標準、そして高度な TX および RX コマンド (1 バイト長および 2 バイト長) の両方で動作させるために必要な変更点、ならびに 802.15.4g フォーマットを使用してパケットの送受信を行うためのコードの修正方法について説明します。

1 バイト長を使用する場合、ペイロードは 255 バイトに制限されます。2 バイト長を使用する場合、ペイロードは 4093 バイトに制限されます。802.15.4g フォーマットを使用する場合、長さフィールドは 11 ビット長で、ペイロードは 2045 バイトに制限されます。

4093 バイトを超えるパケットを送受信するには、無制限のパケット長を使用する必要があります。これについては、アプリの注記では説明されていません。

目次

商標

1 概要

SmartRF™ Studio (2) と SysConfig (3) のいずれも、特定デバイスで利用可能なすべての特性評価済み PHY のレジスタ設定をエクスポート/インポートできます。独自の PHY グループの場合、標準の TX および RX コマンド (CMD_PROP_TX および CMD_PROP_RX) または高度なコマンド (CMD_PROP_TX_ADV および CMD_PROP_RX_ADV) を使用して、設定値をエクスポート / インポートします。Basic RX および TX SLA (Simplelink Academy) では、SysConfig から設定をインポートしたり、SmartRF Studio から設定をエクスポートしたりする方法が示されています。両方のコマンドセットがサポートされているパケット フォーマットに関して非常に柔軟であっても、デフォルトのコードエクスポート / インポートでは 2 つの異なるパケット フォーマットのみが実装されています。図 1-1 では、2 つの異なる場合に設定されたパケットフ ォーマットが示されています。

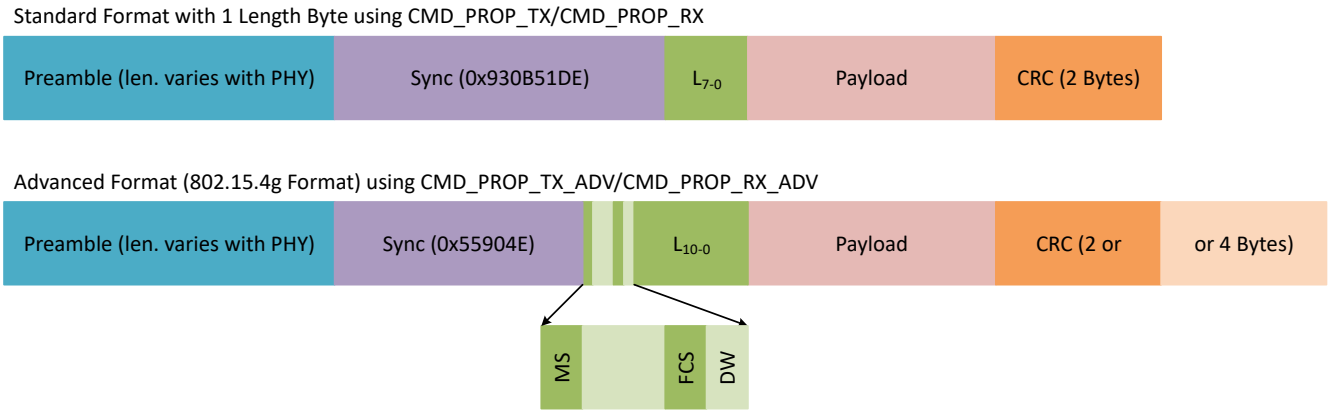

図 1-1. 標準パケット フォーマットと高度なパケット フォーマット

標準パケット フォーマットは、高度なコマンドを使用して実装することも可能であり、1 バイト長の代わりに 2 バイト長のフィールドを使用するように簡単に変更できます。また、デフォルトで高度なパケット フォーマットを使用する PHY についても、標準パケット フォーマット (1 バイトおよび 2 バイト長フィールドの両方) のパケットを送受信するように変更することができます。表 1-1 に、このアプリの注記で説明するさまざまな組み合わせを示します。

表 1-1. コマンドタイプとパケット フォーマットの組み合わせ

コマンドタイプインポート / エクスポート	パケット フォーマット	使用するコマンドタイプ	モード	最大ペイロード長	セクション
標準 CMD_PROP_TX CMD_PROP_RX	標準 (1 バイト長)	標準	TX	255	2.1.1
		標準	RX	255	2.1.2
		高度	TX	255	2.1.3
		高度	RX	255	2.1.4
	標準 (2 バイト長)	高度	TX	4093	2.2.1
		高度	RX	4093	2.2.2
高度な CMD_PROP_TX_ADV CMD_PROP_RX_ADV	高度 (802.15.4g)	高度	TX	2043 または 2045[1]	3.1.1
		高度	RX	2047[2]	3.1.2
	標準 (1 バイト長)	高度	TX	255	3.2.1
		高度	RX	255	3.2.2
	標準 (2 バイト長)	高度	TX	4093	3.3.1
		高度	RX	4093	3.3.2

[1] 最大ペイロード長は FCS タイプによって異なります (2 バイトまたは 4 バイトの CRC)
[2] RX 側の最大長には、CRC バイト数 (2 または 4) が含まれます

2 標準コマンドを使用してエクスポートされた PHY 設定

標準の TX および RX コマンドを使用する PHY 設定をエクスポート / インポートする場合、コマンドは標準パケット フォーマット (1 バイト長) をネイティブでサポートしているため、これらの PHY で実行するために rfPacketTx および rfPacketRx のサンプルを変更する必要はありません。以下のセクション (2.1 - 2.2) では、両方のコマンドタイプ (標準および詳細) を持つ標準パケット フォーマット (1 バイト長) を実装する方法と、標準パケット フォーマットを 1 バイトの代わりに 2 バイト長に変更する方法が示されています。

2.1 標準パケットフォーマット (1 バイト長)

3 バイト (0x01、0x02、0x03) のペイロードを送信すると仮定します。

無線では、パケットは図 2-1 に示すように見えます。

図 2-1. 標準パケットフォーマット (1 バイト長)

2.1.1 CMD_PROP_TX および標準パケット フォーマット TX (1 バイト長)

rfPacketTx の例は、このフォーマットをサポートするように記述されており、ペイロードの長さと内容以外に、サンプルコードを変更する必要はありません。ただし、次のコードスニペットでは、読みやすさを向上させるためにいくつかの変更が実装されています。次のコードは、rfPacketTx.c を変更して 0.5 秒ごとに必要なペイロードを送信する方法を示しています。

```
1:  //---------------------------------------------------------------------------------------------
2:  // Transmit Standard Packet Format with CMD_PROP_TX (1 Length Byte)
3:  //---------------------------------------------------------------------------------------------
4:
5:  // Defines
6:  #define PAYLOAD_LENGTH  3 // Max 255 bytes
7:
8:  uint8_t packet[PAYLOAD_LENGTH];
9:
10: static RF_Object rfObject;
11: static RF_Handle rfHandle;
12:
13: void *mainThread(void *arg0)
14: {
15:     RF_Params rfParams;
16:     RF_Params_init(&rfParams);
17:
18:     RF_cmdPropTx.pktLen = PAYLOAD_LENGTH; // Application specific settings
19:     RF_cmdPropTx.pPkt   = packet;
20:
21:     rfHandle = RF_open(&rfObject, &RF_prop,
22:                        (RF_RadioSetup*)&RF_cmdPropRadioDivSetup, &rfParams);
23:     RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
24:
25:     while(1)
26:     {
27:         //-----------------------------------------------------------------------------------
28:         // Could be placed outside the while(1) since the packet does not change
29:         for (uint8_t i = 0; i < PAYLOAD_LENGTH; i++)
30:         {
31:             packet[i] = i + 1;
32:         }
33:         //-----------------------------------------------------------------------------------
34:         RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTx, RF_PriorityNormal, NULL, 0);
35:         RF_yield(rfHandle);
36:         usleep(500000);
37:     }
38: }
```

標準 TX コマンドを使用する場合、最大ペイロード長は 255 であり、バイト長さ (PAYLOAD_LENGTH) はコマンドの .pktLen フィールドに書き込む必要があります。

2.1.2 CMD_PROP_RX および標準パケット フォーマット (1 バイト長) を使用した RX

CMD_PROP_RX を使用して、図 1-1 に示す標準パケット フォーマットのパケットを受信できるようにするには、以下のコードを使用できます (ここでは、読みやすさを向上させ、デバッグを容易にするために、元の rfPacketRx と比較してコードがわずかに変更されています)。

```
1:  //---------------------------------------------------------------------------------------------
2:  // Receive Standard Packet Format with CMD_PROP_RX (1 Length Byte)
3:  //---------------------------------------------------------------------------------------------
4:
5:  // Defines
6:  #define DATA_ENTRY_HEADER_SIZE  8   // Constant header size of a Generic Data Entry
7:  #define NUM_DATA_ENTRIES        2   // NOTE: Only two data entries supported
8:  #define CRC                     2   // 2 if .rxConf.bIncludeCrc = 0x1, 0 otherwise
9:  #define RSSI                    1   // 1 if .rxConf.bAppendRssi = 0x1, 0 otherwise
10: #define TIMESTAMP               4   // 4 if .rxConf.bAppendTimestamp = 0x1, 0 otherwise
11: #define STATUS                  1   // 1 if .rxConf.bAppendStatus = 0x1, 0 otherwise
12: #define LENGTH_FIELD            1   // RF_cmdPropRx.rxConf.bIncludeHdr = 0x1
13: #define MAX_LENGTH              255 // Max length the radio will accept
14: #define NUM_APPENDED_BYTES      LENGTH_FIELD + CRC + RSSI + TIMESTAMP + STATUS
15:
16: uint8_t packet[MAX_LENGTH + NUM_APPENDED_BYTES - LENGTH_FIELD]; // Length stored in
17: uint8_t packetLength;                                          // packetLength
18:
19: static void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e);
20:
21: static RF_Object rfObject;
22: static RF_Handle rfHandle;
23:
24: static uint8_t rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES, MAX_LENGTH,
25:                                    NUM_APPENDED_BYTES)]__attribute__((aligned(4)));
26: static dataQueue_t dataQueue;
27: static rfc_dataEntryGeneral_t* currentDataEntry;
28: static uint8_t* packetDataPointer;
29: rfc_propRxOutput_t rxStatistics;
30: uint16_t crc16;
31: int8_t rssi;
32: uint32_t timestamp;
33: uint8_t status;
34:
35: void *mainThread(void *arg0)
36: {
37:     RF_Params rfParams;
38:     RF_Params_init(&rfParams);
39:
40:     if(RFQueue_defineQueue(&dataQueue, rxDataEntryBuffer, sizeof(rxDataEntryBuffer),
41:                            NUM_DATA_ENTRIES, MAX_LENGTH + NUM_APPENDED_BYTES))
42:     {
43:         while(1);
44:     }
45:
46:     RF_cmdPropRx.pktConf.bRepeatOk = 0x1; // Application specific settings
47:     RF_cmdPropRx.pktConf.bRepeatNok = 0x1;
48:     RF_cmdPropRx.rxConf.bAutoFlushIgnored = 0x1;
49:     RF_cmdPropRx.rxConf.bAutoFlushCrcErr = 0x1;
50:     RF_cmdPropRx.maxPktLen = MAX_LENGTH;
51:     RF_cmdPropRx.pQueue = &dataQueue;
52:
53:     RF_cmdPropRx.rxConf.bIncludeCrc = 0x1; // Optional bytes to append
54:     RF_cmdPropRx.rxConf.bAppendRssi = 0x1;
55:     RF_cmdPropRx.rxConf.bAppendTimestamp = 0x1;
56:     RF_cmdPropRx.rxConf.bAppendStatus = 0x1;
57:
58:     RF_cmdPropRx.pOutput = (uint8_t*)&rxStatistics; // Optional (for debug)
59:
60:     rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
61:                        &rfParams);
62:     RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
63:     RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropRx, RF_PriorityNormal,
64:               &callback, RF_EventRxEntryDone);
65:     while(1);
66: }
67:
```

```
68: //---------------------------------------------------------------------------------------------
69: // Callback for Receiving Standard Packet Format with CMD_PROP_RX (1 Length Byte)
70: //---------------------------------------------------------------------------------------------
71: void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e)
72: {
73:     if(e & RF_EventRxEntryDone)
74:     {
75:         currentDataEntry = RFQueue_getDataEntry();
76:
77:         packetLength = *(uint8_t*)(&currentDataEntry->data);
78:         packetDataPointer = (uint8_t*)(&currentDataEntry->data + LENGTH_FIELD);
79:
80:         memcpy(packet, packetDataPointer, (packetLength + NUM_APPENDED_BYTES - LENGTH_FIELD));
81:
82:         crc16 = ((uint16_t)(packet[packetLength + 0] << 8) +
83:                  (uint16_t)(packet[packetLength + 1] << 0));
84:
85:         rssi = packet[packetLength + 2];
86:
87:         timestamp = ((uint32_t)(packet[packetLength + 3] <<  0) +
88:                      (uint32_t)(packet[packetLength + 4] <<  8) +
89:                      (uint32_t)(packet[packetLength + 5] << 16) +
90:                      (uint32_t)(packet[packetLength + 6] << 24));
91:
92:         status = packet[packetLength + 7];
93:
94:         RFQueue_nextEntry();
95:     }
96: }
```

このコード例では、長さフィルタリングは実装されておらず、付加可能なすべてのオプションバイトが付加されています。受信されるパケットが、図 2-1 に示されたパケットのみの場合、MAX_LENGTH を 255 から 3 に変更できます。これにより、バイト長が 3 より大きいすべてのパケットは破棄されます (rxConf.bAutoFlushIgnored = 1 以降)。

2.1.3 CMD_PROP_TX_ADV および標準パケット フォーマット (1 バイト長) を使用した TX

場合によっては、パケット形式が標準であっても、高度なコマンドを使用してパケットを送信することができます。これは、例えばスニフモードのアプリケーションなどにおいて、標準の TX コマンドでサポートされているものよりも長いプリアンブルを最終的に送信したい場合などが考えられます。

SysConfig (3) と SmartRF Studio (2) のいずれも、デフォルトで PHY に使用されているコマンド以外のコマンドのインポートとエクスポートをサポートしています。図 2-2 では、SysConfig を使用してプロジェクトにインポートする追加コマンドを選択する方法が示されています。また図 2-3 では、SmartRF Studio を使用したコードのエクスポートによる方法が示されています。

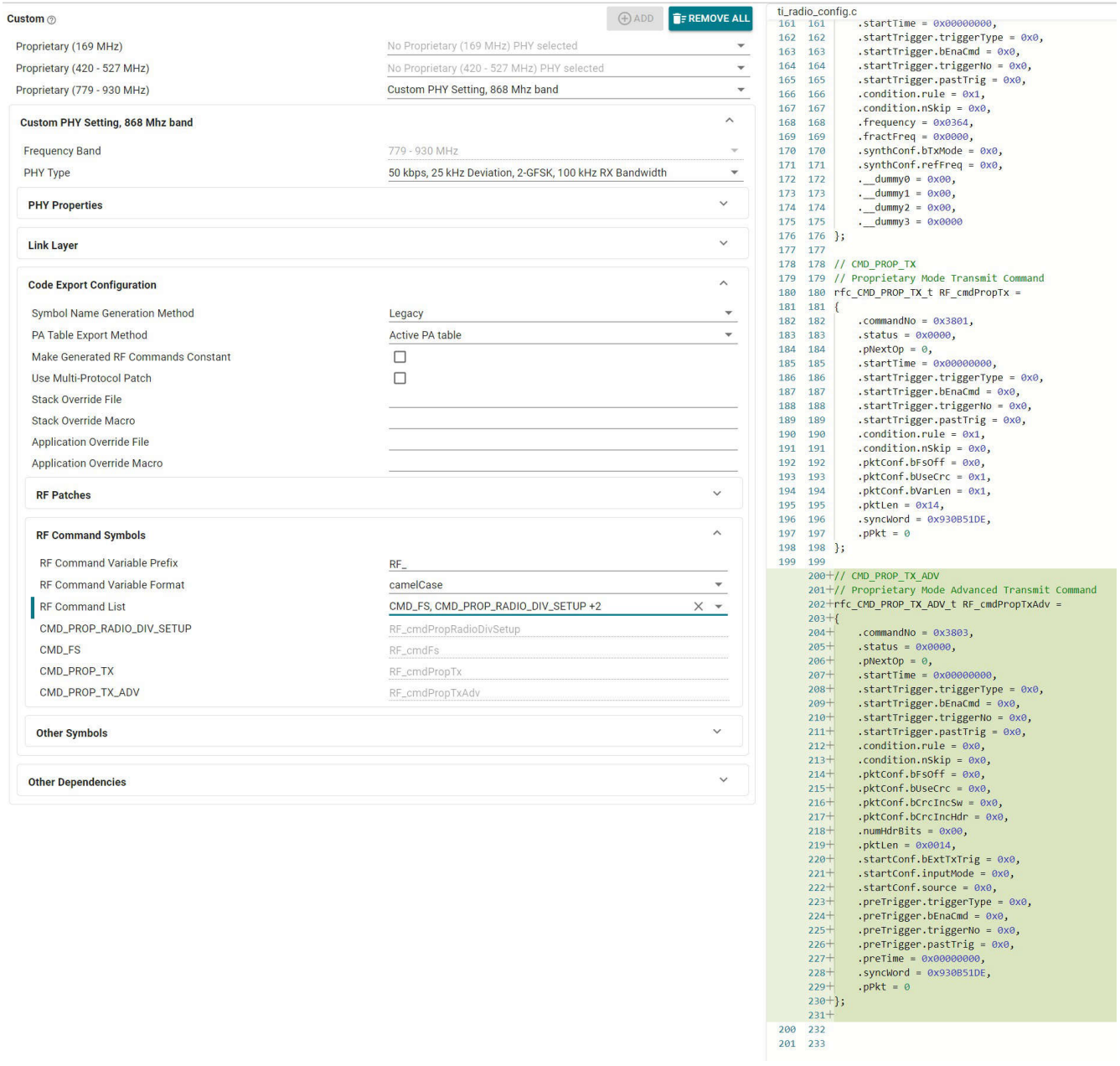

図 2-2. SysConfig の標準コマンドに加えて、高度な TX コマンドをインポートする

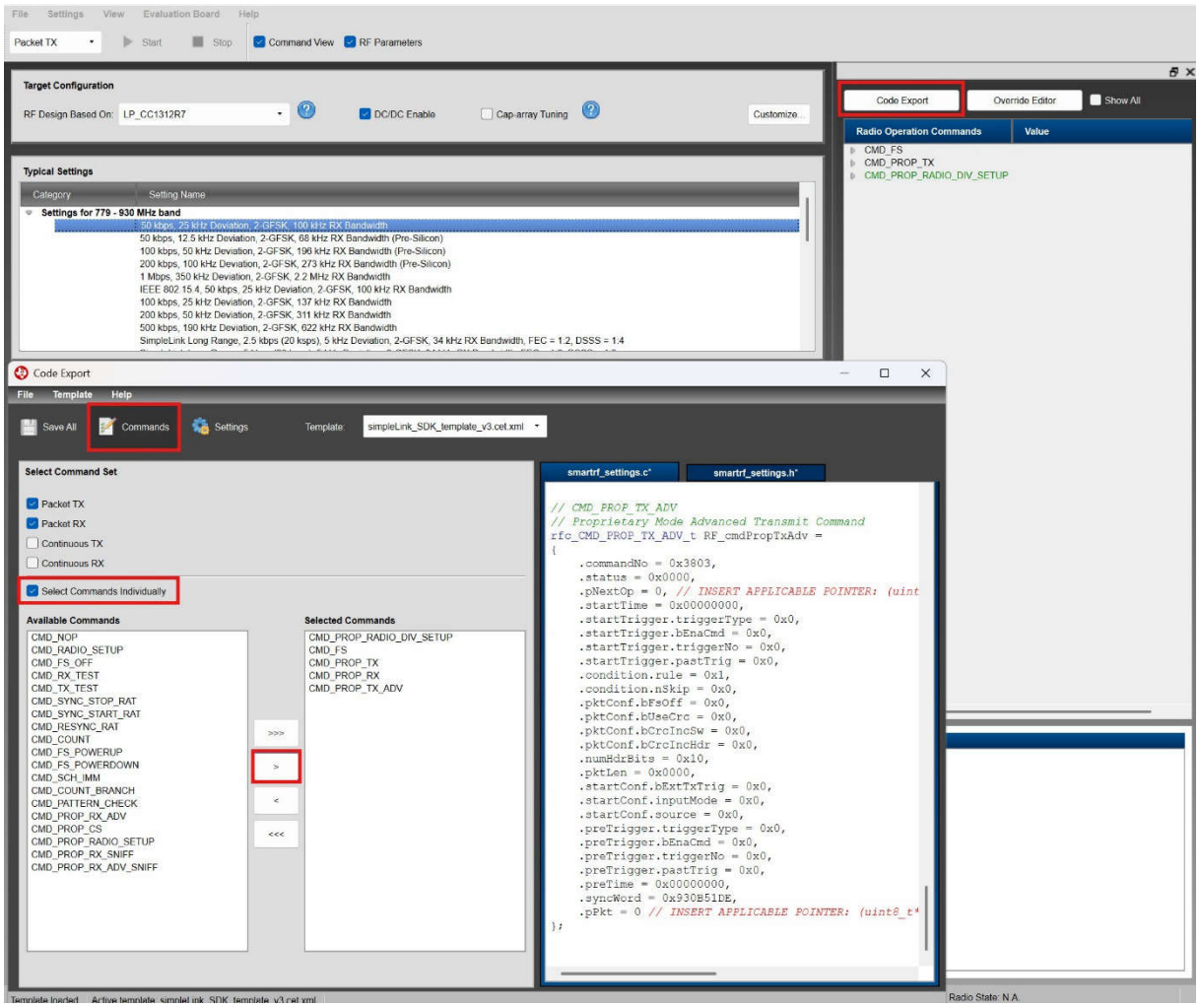

図 2-3. SmartRF Studio の標準コマンドに加えて高度な TX コマンドをエクスポートする

SmartRF Studio と SysConfig では、このような追加コマンドを追加する際の設定が完全に一致するわけではないため、このコード例では両方のケースに対応するように記述されています。高度な TX コマンドを使用する場合、バイト長はペイロードとともに手動でパケットに書き込む必要があります。また、コマンドの .pktLen フィールドには、長さフィールドとペイロード長の両方を含める必要があります。レイアウト例を以下に示します。

```
1:  //---------------------------------------------------------------------------------------------
2:  // Transmit Standard Packet Format with CMD_PROP_TX_ADV (1 Length Byte)
3:  //---------------------------------------------------------------------------------------------
4:
5:  // Defines
6:  #define PAYLOAD_LENGTH 3 // Max 255 bytes
7:  #define LENGTH_FIELD   1
8:
9:  uint8_t packet[LENGTH_FIELD + PAYLOAD_LENGTH];
10:
11: static RF_Object rfObject;
12: static RF_Handle rfHandle;
13:
14: void *mainThread(void *arg0)
15: {
16:     RF_Params rfParams;
17:     RF_Params_init(&rfParams);
18:
19:     RF_cmdPropTxAdv.numHdrBits = 0x0; // Settings that must change to support
20:                                       // the standard packet format
21:
22:     RF_cmdPropTxAdv.pktLen = LENGTH_FIELD + PAYLOAD_LENGTH; // Application specific settings
23:     RF_cmdPropTxAdv.pPkt = packet;
24:
25:     // Settings to modify if going from a PHY that uses the standard TX command
26:     // to use the advanced TX command
27:     RF_cmdPropTxAdv.condition.rule = 0x1;
28:     RF_cmdPropTxAdv.pktConf.bUseCrc = 0x1;
29:
30:     rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
31:                        &rfParams);
31:     RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
32:
33:     while(1)
34:     {
35:         //-----------------------------------------------------------------------------------
36:         // Could be placed outside the while(1) since the packet does not change
37:         packet[0] = PAYLOAD_LENGTH;
38:
39:         for (uint16_t i = 1; i < (LENGTH_FIELD + PAYLOAD_LENGTH); i++)
40:         {
41:             packet[i] = i;
42:         }
43:         //-----------------------------------------------------------------------------------
44:         RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTxAdv, RF_PriorityNormal, NULL, 0);
45:         RF_yield(rfHandle);
46:         usleep(500000);
47:     }
48: }
```

2.1.4 CMD_PROP_RX_ADV および標準パケット フォーマット (1 バイト長) を使用した RX

標準の RX コマンドの代わりに高度な RX コマンドを使用して標準パケット フォーマットのパケットを受信する場合も、セクション 2.1.3 の高度な TX コマンドで説明されている方法でコマンドを追加できます。以下に、高度な RX コマンドを使用して標準パケットを受信する方法を示すコード例を示します。また、これにより、MAX_LENGTH を 255 から 3 に変更して、パケット長のフィルタリングを有効にすることもできます。

```
1:   //---------------------------------------------------------------------------------------
2:   // Receive Standard Packet Format with CMD_PROP_RX_ADV (1 Length Byte)
3:   //---------------------------------------------------------------------------------------
4:   // Defines
5:   #define DATA_ENTRY_HEADER_SIZE  8   // Constant header size of a Generic Data Entry
6:   #define NUM_DATA_ENTRIES        2   // NOTE: Only two data entries supported
7:   #define CRC                     2   // 2 if .rxConf.bIncludeCrc = 0x1, 0 otherwise
8:   #define RSSI                    1   // 1 if .rxConf.bAppendRssi = 0x1, 0 otherwise
9:   #define TIMESTAMP               4   // 4 if .rxConf.bAppendTimestamp = 0x1, 0 otherwise
10:  #define STATUS                  1   // 1 if .rxConf.bAppendStatus = 0x1, 0 otherwise
11:  #define LENGTH_FIELD            1   // RF_cmdPropRx.rxConf.bIncludeHdr = 0x1
12:  #define MAX_LENGTH              255 // Max length the radio will accept
13:  #define NUM_APPENDED_BYTES      LENGTH_FIELD + CRC + RSSI + TIMESTAMP + STATUS
14:
15:  uint8_t packet[MAX_LENGTH + NUM_APPENDED_BYTES - LENGTH_FIELD]; // Length stored in
16:  uint8_t packetLength;                                          // packetLength
17:  static void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e);
18:
19:  static RF_Object rfObject;
20:  static RF_Handle rfHandle;
21:
22:  static uint8_t rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES, MAX_LENGTH,
23:                                   NUM_APPENDED_BYTES)]__attribute__((aligned(4)));
24:  static dataQueue_t dataQueue;
25:  static rfc_dataEntryGeneral_t* currentDataEntry;
26:  static uint8_t* packetDataPointer;
27:  rfc_propRxOutput_t rxStatistics;
28:  uint16_t crc16;
29:  int8_t rssi;
30:  uint32_t timestamp;
31:  uint8_t status;
32:
33:  void *mainThread(void *arg0)
34:  {
35:      RF_Params rfParams;
36:      RF_Params_init(&rfParams);
37:
38:      if(RFQueue_defineQueue(&dataQueue, rxDataEntryBuffer, sizeof(rxDataEntryBuffer),
39:                             NUM_DATA_ENTRIES, MAX_LENGTH + NUM_APPENDED_BYTES))
40:      {
41:          while(1);
42:      }
43:
44:      RF_cmdPropRxAdv.pktConf.bCrcIncHdr = 0x1; // Settings that must change to support
45:      RF_cmdPropRxAdv.hdrConf.numHdrBits = 0x8; // the standard packet format
46:      RF_cmdPropRxAdv.hdrConf.numLenBits = 0x8;
47:
48:      RF_cmdPropRxAdv.pktConf.bRepeatOk = 0x1; // Application specific settings
49:      RF_cmdPropRxAdv.pktConf.bRepeatNok = 0x1;
50:      RF_cmdPropRxAdv.rxConf.bAutoFlushIgnored = 0x1;
51:      RF_cmdPropRxAdv.rxConf.bAutoFlushCrcErr = 0x1;
52:      RF_cmdPropRxAdv.maxPktLen = MAX_LENGTH;
53:      RF_cmdPropRxAdv.pQueue = &dataQueue;
54:
55:      // Settings to modify if going from a PHY that uses the standard RX command
56:      // to use the advanced RX command
57:      RF_cmdPropRxAdv.condition.rule = 0x1;
58:      RF_cmdPropRxAdv.pktConf.bUseCrc = 0x1;
59:      RF_cmdPropRxAdv.rxConf.bIncludeHdr = 0x1;
60:      RF_cmdPropRxAdv.endTrigger.triggerType = 0x1;
61:
62:      RF_cmdPropRxAdv.rxConf.bIncludeCrc = 0x1; // Optional bytes to append
63:      RF_cmdPropRxAdv.rxConf.bAppendRssi = 0x1;
64:      RF_cmdPropRxAdv.rxConf.bAppendTimestamp = 0x1;
65:      RF_cmdPropRxAdv.rxConf.bAppendStatus = 0x1;
66:
67:      RF_cmdPropRxAdv.pOutput = (uint8_t*)&rxStatistics; // Optional (for debug)
68:
69:      rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
70:                         &rfParams);
71:      RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
72:      RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropRxAdv, RF_PriorityNormal,
73:                &callback, RF_EventRxEntryDone);
74:      while(1);
75:  }
76:
```

```
77:  //---------------------------------------------------------------------------------------------
78:  // Callback for Receiving Standard Packet Format with CMD_PROP_RX_ADV (1 Length Byte)
79:  //---------------------------------------------------------------------------------------------
80:  void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e)
81:  {
82:      if(e & RF_EventRxEntryDone)
83:      {
84:          currentDataEntry = RFQueue_getDataEntry();
85:
86:         packetLength = *(uint8_t*)(&currentDataEntry->data);
87:         packetDataPointer = (uint8_t*)(&currentDataEntry->data + LENGTH_FIELD);
88:
89:         memcpy(packet, packetDataPointer, (packetLength + NUM_APPENDED_BYTES - LENGTH_FIELD));
90:
91:         crc16 = ((uint16_t)(packet[packetLength + 0] << 8) +
92:                  (uint16_t)(packet[packetLength + 1] << 0));
93:
94:         rssi = packet[packetLength + 2];
95:
96:         timestamp = ((uint32_t)(packet[packetLength + 3] << 0 ) +
97:                      (uint32_t)(packet[packetLength + 4] << 8 ) +
98:                      (uint32_t)(packet[packetLength + 5] << 16) +
99:                      (uint32_t)(packet[packetLength + 6] << 24));
100:
101:        status = packet[packetLength + 7];
102:
103:        RFQueue_nextEntry();
104:    }
105: }
```

2.2 標準パケット フォーマット (2 バイト長)

標準コマンドから高度なコマンドへに変更を希望するもう一つの理由は、255 バイトを超えるパケットの送受信を行うため、長さフィールドに 1 バイト以上が必要となる場合が考えられます。

2 バイト長のフィールドの場合、前述の例で使用されたパケットは、図 2-4 に示すようになります。

図 2-4. 標準パケット フォーマット (2 バイト長)

2.2.1 CMD_PROP_TX_ADV および標準パケット フォーマット (2 バイト長) を使用した TX

セクション 2.1.3 の例に示されている設定から変更する必要はありません。

必要な変更は次のとおりです。

- LENGTH_FIELD を 1 から 2 に変更します
- 両方のバイト長をパケットに追加します

必要な変更点は 7 行目と 38 から 48 行目に記載されています。

```
1:  //---------------------------------------------------------------------------------------------
2:  // Transmit Standard Packet Format with CMD_PROP_TX_ADV (2 Length Bytes)
3:  //---------------------------------------------------------------------------------------------
4:
5:  // Defines
6:  #define PAYLOAD_LENGTH 3 // Max 4093 bytes
7:  #define LENGTH_FIELD   2 // Changed from 1
8:
9:  uint8_t packet[LENGTH_FIELD + PAYLOAD_LENGTH];
10:
11: static RF_Object rfObject;
12: static RF_Handle rfHandle;
13:
14: void *mainThread(void *arg0)
15: {
16:     RF_Params rfParams;
17:     RF_Params_init(&rfParams);
18:
19:     RF_cmdPropTxAdv.numHdrBits = 0x0; // Settings that must change to support
20:                                       // the standard packet format
21:
22:     RF_cmdPropTxAdv.pktLen = LENGTH_FIELD + PAYLOAD_LENGTH; // Application specific
23:     RF_cmdPropTxAdv.pPkt = packet;                          // settings
24:
25:     // Settings to modify if going from a PHY that uses the standard TX command
26:     // to use the advanced TX command
27:     RF_cmdPropTxAdv.condition.rule = 0x1;
28:     RF_cmdPropTxAdv.pktConf.bUseCrc = 0x1;
29:
30:     rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
31:                        &rfParams);
32:     RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
33:
34:     while(1)
35:     {
36:         //-----------------------------------------------------------------------------------
37:         // Could be placed outside the while(1) since the packet does not change
38: #ifdef SLR_MODE
39:         packet[0] = (uint8_t)(PAYLOAD_LENGTH);
40:         packet[1] = (uint8_t)(PAYLOAD_LENGTH >> 8);
41: #else
42:         packet[0] = (uint8_t)(PAYLOAD_LENGTH >> 8);
43:         packet[1] = (uint8_t)(PAYLOAD_LENGTH);
44: #endif
45:         for (uint16_t i = 2; i < (LENGTH_FIELD + PAYLOAD_LENGTH); i++)
46:         {
47:             packet[i] = i - 1;
48:         }
49:         //-----------------------------------------------------------------------------------
50:         RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTxAdv, RF_PriorityNormal, NULL, 0);
51:         RF_yield(rfHandle);
52:         usleep(500000);
53:     }
54: }
```

SimpleLink Long Range (SLR) PHY またはレガシー Long Range PHY (CC13x0 のみ) のいずれかを使用する場合、他の独自 PHY を使用する場合と比較して、2 つのバイト長を逆の順序で書き込む必要があることに注意してください。

2.2.2 CMD_PROP_RX_ADV および標準パケット フォーマット (2 バイト長) を使用した RX

2 つのバイト長の標準パケット フォーマットを受信するには、セクション 2.1.4 の例に示されているものと比較して設定を変更する必要があります。また、コールバックのアプリケーションコードとともに、一部の定義と変数を変更する必要があります。11 および 12 行目、15 および 16 行目、46 および 47 行目、86 および 89 行目に必要な変更点について記載されています。

```
1:   //---------------------------------------------------------------------------------------------
2:   // Receive Standard Packet Format with CMD_PROP_RX_ADV (2 Length Bytes)
3:   //---------------------------------------------------------------------------------------------
4:   // Defines
5:   #define DATA_ENTRY_HEADER_SIZE  8    // Constant header size of a Generic Data Entry
6:   #define NUM_DATA_ENTRIES        2    // NOTE: Only two data entries supported
7:   #define CRC                     2    // 2 if .rxConf.bIncludeCrc = 0x1, 0 otherwise
8:   #define RSSI                    1    // 1 if .rxConf.bAppendRssi = 0x1, 0 otherwise
9:   #define TIMESTAMP               4    // 4 if .rxConf.bAppendTimestamp = 0x1, 0 otherwise
10:  #define STATUS                  1    // 1 if .rxConf.bAppendStatus = 0x1, 0 otherwise
11:  #define LENGTH_FIELD            2    // Changed from 1
12:  #define MAX_LENGTH              4093 // Changed from 255
13:  #define NUM_APPENDED_BYTES      LENGTH_FIELD + CRC + RSSI + TIMESTAMP + STATUS
14:
15:  uint8_t packet[MAX_LENGTH + NUM_APPENDED_BYTES - LENGTH_FIELD]; // Length stored in
16:  uint16_t packetLength; // Changed from uint8_t                    packetLength
17:
18:  static void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e);
19:  static RF_Object rfObject;
20:  static RF_Handle rfHandle;
21:
22:  static uint8_t rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES, MAX_LENGTH,
23:                                   NUM_APPENDED_BYTES)]__attribute__((aligned(4)));
24:  static dataQueue_t dataQueue;
25:  static rfc_dataEntryGeneral_t* currentDataEntry;
26:  static uint8_t* packetDataPointer;
27:  rfc_propRxOutput_t rxStatistics;
28:  uint16_t crc16;
29:  int8_t rssi;
30:  uint32_t timestamp;
31:  uint8_t status;
32:
33:  void *mainThread(void *arg0)
34:  {
35:      RF_Params rfParams;
36:      RF_Params_init(&rfParams);
37:
38:      if(RFQueue_defineQueue(&dataQueue, rxDataEntryBuffer, sizeof(rxDataEntryBuffer),
39:                             NUM_DATA_ENTRIES, MAX_LENGTH + NUM_APPENDED_BYTES))
40:      {
41:          while(1);
42:      }
43:
44:      RF_cmdPropRxAdv.pktConf.bCrcIncHdr = 0x1;  // Settings that must change
45:                                                 // to support the standard format
46:      RF_cmdPropRxAdv.hdrConf.numHdrBits = 0x10; // Changed from 0x8
47:      RF_cmdPropRxAdv.hdrConf.numLenBits = 0x10; // Changed from 0x8
48:
49:      RF_cmdPropRxAdv.pktConf.bRepeatOk = 0x1;   // Application specific settings
50:      RF_cmdPropRxAdv.pktConf.bRepeatNok = 0x1;
51:      RF_cmdPropRxAdv.rxConf.bAutoFlushIgnored = 0x1;
52:      RF_cmdPropRxAdv.rxConf.bAutoFlushCrcErr = 0x1;
53:      RF_cmdPropRxAdv.maxPktLen = MAX_LENGTH;
54:      RF_cmdPropRxAdv.pQueue = &dataQueue;
55:
56:      // Settings to modify if going from a PHY that uses the standard RX command
57:     // to use the advanced RX command
58:      RF_cmdPropRxAdv.condition.rule = 0x1;
59:      RF_cmdPropRxAdv.pktConf.bUseCrc = 0x1;
60:      RF_cmdPropRxAdv.rxConf.bIncludeHdr = 0x1;
61:      RF_cmdPropRxAdv.endTrigger.triggerType = 0x1;
62:
63:      RF_cmdPropRxAdv.rxConf.bIncludeCrc = 0x1; // Optional bytes to append
64:      RF_cmdPropRxAdv.rxConf.bAppendRssi = 0x1;
65:      RF_cmdPropRxAdv.rxConf.bAppendTimestamp = 0x1;
66:      RF_cmdPropRxAdv.rxConf.bAppendStatus = 0x1;
67:      RF_cmdPropRxAdv.pOutput = (uint8_t*)&rxStatistics; // Optional (for debug)
68:
69:      rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
70:                         &rfParams);
71:      RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
72:      RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropRxAdv, RF_PriorityNormal,
73:                &callback, RF_EventRxEntryDone);
74:      while(1);
75:  }
76:
```

```
77:  //---------------------------------------------------------------------------------------------
79:  // Callback for Receiving Standard Packet Format with CMD_PROP_RX_ADV (2 Length Bytes)
79:  //---------------------------------------------------------------------------------------------
80:  void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e)
81:  {
82:      if(e & RF_EventRxEntryDone)
83:      {
84:          currentDataEntry = RFQueue_getDataEntry();
85:
86:          packetLength = ((uint16_t)(*(uint8_t*)(&currentDataEntry->data + 1) << 8) |
87:                          (uint16_t)(*(uint8_t*)(&currentDataEntry->data + 0) << 0));
88:          // Changed from
89:          // packetLength = *(uint8_t*)(&currentDataEntry->data);
90:
91:          packetDataPointer = (uint8_t*)(&currentDataEntry->data + LENGTH_FIELD);
92:
93:          memcpy(packet, packetDataPointer,
94:                 (packetLength + NUM_APPENDED_BYTES - LENGTH_FIELD));
95:
96:          crc16 = ((uint16_t)(packet[packetLength + 0] << 8) +
97:                   (uint16_t)(packet[packetLength + 1] << 0));
98:
99:          rssi = packet[packetLength + 2];
100:
101:         timestamp = ((uint32_t)(packet[packetLength + 3] << 0 ) +
102:                      (uint32_t)(packet[packetLength + 4] << 8 ) +
103:                      (uint32_t)(packet[packetLength + 5] << 16) +
104:                      (uint32_t)(packet[packetLength + 6] << 24));
105:
106:         status = packet[packetLength + 7];
107:
108:         RFQueue_nextEntry();
109:     }
110: }
```

3 高度なコマンドでエクスポートされた TX および RX 設定

デフォルトでは、CMD_PROP_TX_ADV または CMD_PROP_RX_ADV コマンドを使用してエクスポート / インポートされるすべての PHY は、IEEE 802.15.4g で指定されたパケット フォーマットを使用します。このフォーマットは、図 3-1 に示されています。

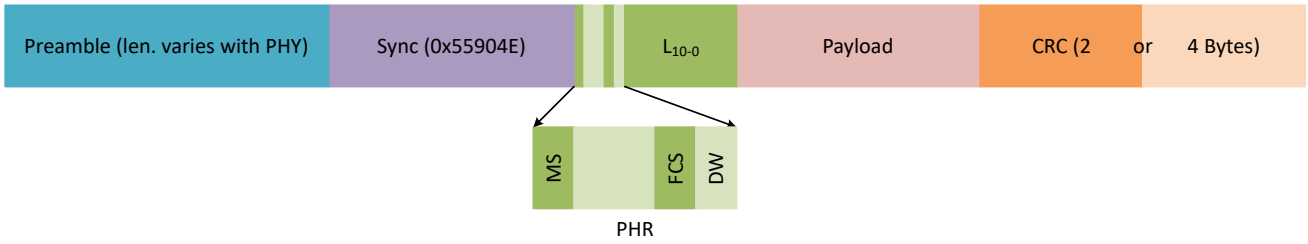

図 3-1. 高度なパケット フォーマット

同期ワード (0x55904E) の後に、以下のフィールドを含む 2 バイト長のヘッダー (PHR) が続きます。

PHR[15] MS: モードスイッチ (常に 0 に設定)

PHR[14:13] 予約済み (特に指定なし)

PHR[12] FCS：FCS タイプ (0：4 バイト CRC、1：2 バイト CRC)

PHR[11] DW：データホワイトニング (0：ディスエーブル、1：イネーブル)

PHR[10:0] の長さ

長さフィールドには CRC バイト (PHR の FCS フィールドに応じて 2 または 4 バイト) が含まれている必要があります。つまり、前の例で使用した 3 バイト長のペイロードを送信する場合、ヘッダーの長さは 5 または 7 になります。たとえば、4 バイト長の CRC (FCS = 0) とホワイトニング (DW = 1) を使用する場合、ヘッダーは 0x0807 になります。

3.1 高度なパケット フォーマット

次の 2 つのセクションでは、rfPacketTx および rfPacketRX の例を変更して、高度なパケット フォーマットを使用してパケットを送受信する方法について説明します。TX の例では、4 バイト長の CRC にホワイトニングを加えた値を使用すると仮定します。

3.1.1 CMD_PROP_TX_ADV および高度なパケット フォーマットを使用した TX

ヘッダーは最下位バイトから順にパケットに書き込まれるため、CMD_PROP_TX_ADV コマンドにより送信するパケットは、0x07、0x08、0x01、0x02、0x03 となります。

長さフィールドは 11 ビットであるため、2 バイト長の CRC を使用する場合のペイロードバイトの最大数は 2045、CRC が 4 バイト長の場合は 2043 です。次に、高度なパケット フォーマットを使用してパケットを送信するために使用できるコードを示します。

```
1:  //---------------------------------------------------------------------------------------------
2:  // Transmit Advanced Packet Format with CMD_PROP_TX_ADV
3:  //---------------------------------------------------------------------------------------------
4:
5:  // Defines
6:  #define PAYLOAD_LENGTH 3 // Max 2045 or 2043, depending on FSC type
7:  #define HEADER_FIELD   2
8:
9:  // #define _802_15_4_G_HEADER   0x00 // MS = 0, FCS = 0 (4 bytes CRC), DW = 0 (Whitening Off)
10: #define _802_15_4_G_HEADER      0x08 // MS = 0, FCS = 0 (4 bytes CRC), DW = 1 (Whitening On)
11: // #define _802_15_4_G_HEADER   0x10 // MS = 0, FCS = 1 (2 bytes CRC), DW = 0 (Whitening Off)
12: // #define _802_15_4_G_HEADER   0x18 // MS = 0, FCS = 1 (2 bytes CRC), DW = 1 (Whitening On)
13:
14: uint8_t packet[HEADER_FIELD + PAYLOAD_LENGTH];
15: uint16_t header;
16:
17: static RF_Object rfObject;
18: static RF_Handle rfHandle;
19:
20: void *mainThread(void *arg0)
21: {
22:     RF_Params rfParams;
23:     RF_Params_init(&rfParams);
24:
25:     RF_cmdPropTxAdv.startTrigger.triggerType = 0x0; // Application specific settings
26:     RF_cmdPropTxAdv.startTrigger.pastTrig = 0x0;
27:     RF_cmdPropTxAdv.pktLen = HEADER_FIELD + PAYLOAD_LENGTH;
28:     RF_cmdPropTxAdv.preTrigger.triggerType = 0x0;
29:     RF_cmdPropTxAdv.preTrigger.pastTrig = 0x0;
30:     RF_cmdPropTxAdv.pPkt = packet;
31:
32:     rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
33:                        &rfParams);
34:     RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
35:
36:     while(1)
37:     {
38:         //-----------------------------------------------------------------------------------
39:         // Could be placed outside the while(1) since the packet does not change
40:         if((_802_15_4_G_HEADER == 0x00) || (_802_15_4_G_HEADER == 0x08))
41:         {
42:             header = (_802_15_4_G_HEADER << 8) + ((4 + PAYLOAD_LENGTH) & 0x07FF);
43:         }
44:         else
45:         {
46:             header = (_802_15_4_G_HEADER << 8) + ((2 + PAYLOAD_LENGTH) & 0x07FF);
47:         }
48:         packet[0] = (uint8_t)(header);
49:         packet[1] = (uint8_t)(header >> 8);
50:
51:         for (uint16_t i = 2; i < (HEADER_FIELD + PAYLOAD_LENGTH); i++)
52:         {
53:             packet[i] = i - 1;
54:         }
55:         //-----------------------------------------------------------------------------------
56:         RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTxAdv, RF_PriorityNormal, NULL, 0);
57:         RF_yield(rfHandle);
58:         usleep(500000);
```

```
59:     }
60: }
```

3.1.2 CMD_PROP_RX_ADV および高度なパケット フォーマットを使用した RX

以下に、高度なパケット フォーマットを使用してパケットを受信するために使用できるコードを示します。このコードは、デフォルトで高度なコマンドをエクスポート / インポートする PHY 設定にのみ使用できます。

```
1:   //*****************************************************************************************************
2:   // Receive Advanced Packet Format with CMD_PROP_RX_ADV
3:   //*****************************************************************************************************
4:
5:   // Defines
6:   #define DATA_ENTRY_HEADER_SIZE  8 // Constant header size of a Generic Data Entry
7:   #define NUM_DATA_ENTRIES        2 // NOTE: Only two data entries supported
8:   #define CRC                     4 // 4/2 (based on FCS) if .rxConf.bIncludeCrc = 0x1, else 0
9:   #define RSSI                    1 // 1 if .rxConf.bAppendRssi = 0x1, 0 otherwise
10:  #define TIMESTAMP               4 // 4 if .rxConf.bAppendTimestamp = 0x1, 0 otherwise
11:  #define STATUS                  1 // 1 if .rxConf.bAppendStatus = 0x1, 0 otherwise
12:  #define HEADER_FIELD            2 // RF_cmdPropRx.rxConf.bIncludeHdr = 0x1
13:  #define MAX_LENGTH              2047 // Max length the radio will accept (incl. CRC bytes)
14:  #define NUM_APPENDED_BYTES      HEADER_FIELD + RSSI + TIMESTAMP + STATUS
15:
16:  uint8_t packet[MAX_LENGTH + NUM_APPENDED_BYTES - HEADER_FIELD]; // Header/Length stored in
17:  uint16_t packetLength;                                         // packetLength variable
18:
19:  static void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e);
20:
21:  static RF_Object rfObject;
22:  static RF_Handle rfHandle;
23:
24:  static uint8_t rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES,
25:                                   MAX_LENGTH, NUM_APPENDED_BYTES)]__attribute__((aligned(4)));
26:  static dataQueue_t dataQueue;
27:  static rfc_dataEntryGeneral_t* currentDataEntry;
28:  static uint8_t* packetDataPointer;
29:  rfc_propRxOutput_t rxStatistics;
30:  uint16_t crc16;
31:  uint32_t crc32;
32:  int8_t rssi;
33:  uint32_t timestamp;
34:  uint8_t status;
35:
36:  void *mainThread(void *arg0)
37:  {
38:      RF_Params rfParams;
39:      RF_Params_init(&rfParams);
40:
41:      if(RFQueue_defineQueue(&dataQueue, rxDataEntryBuffer, sizeof(rxDataEntryBuffer),
42:                             NUM_DATA_ENTRIES, MAX_LENGTH + NUM_APPENDED_BYTES))
43:      {
44:          while(1);
45:      }
46:
47:      RF_cmdPropRxAdv.pktConf.bRepeatOk = 0x1; // Application specific settings
48:      RF_cmdPropRxAdv.pktConf.bRepeatNok = 0x1;
49:      RF_cmdPropRxAdv.rxConf.bAutoFlushCrcErr = 0x1;
50:      RF_cmdPropRxAdv.maxPktLen = MAX_LENGTH;
51:      RF_cmdPropRxAdv.pQueue = &dataQueue;
52:
53:      RF_cmdPropRxAdv.pOutput = (uint8_t*)&rxStatistics; // Optional (for debug)
54:
55:      rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
56:                         &rfParams);
57:      RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
58:      RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropRxAdv, RF_PriorityNormal,
59:                &callback, RF_EventRxEntryDone);
60:      while(1);
61:  }
62:
```

```
63:  //-------------------------------------------------------------------------------------------
64:  // Callback for Receiving Advanced Packet Format with CMD_PROP_RX_ADV
65:  //-------------------------------------------------------------------------------------------
66:
67:  void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e)
68:  {
69:      if(e & RF_EventRxEntryDone)
70:      {
71:          currentDataEntry = RFQueue_getDataEntry();
72:
73:          uint16_t header = ((uint16_t)(*(uint8_t*)(&currentDataEntry->data + 1) << 8) |
74:                             (uint16_t)(*(uint8_t*)(&currentDataEntry->data + 0) << 0));
75:
76:          bool fcs = (bool)(0x1000 & header);
77:
78:          packetLength = 0x07FF & header;
79:
80:          packetDataPointer = (uint8_t*)(&currentDataEntry->data + HEADER_FIELD);
81:
82:          memcpy(packet, packetDataPointer,
83:                 (packetLength + NUM_APPENDED_BYTES - HEADER_FIELD));
84:
85:          // The FCS field in the header tell us if the CRC is 2 or 4 bytes long
86:          if(fcs) // FCS = 1 -> 2 bytes CRC
87:          {
88:              crc16 = ((uint16_t)(packet[packetLength - 2] <<  8) +
89:              (uint16_t)(packet[packetLength - 1] <<  0));
90:          }
91:          else // FCS = 0 -> 4 bytes CRC
92:          {
93:              crc32 = ((uint32_t)(packet[packetLength - 4] << 24) +
94:                       (uint32_t)(packet[packetLength - 3] << 16) +
95:                       (uint32_t)(packet[packetLength - 2] <<  8) +
96:                       (uint32_t)(packet[packetLength - 1] <<  0));
97:          }
98:
99:          rssi = packet[packetLength + 0];
100:
101:         timestamp = ((uint32_t)(packet[packetLength + 1] << 0 ) +
102:                      (uint32_t)(packet[packetLength + 2] << 8 ) +
103:                      (uint32_t)(packet[packetLength + 3] << 16) +
104:                      (uint32_t)(packet[packetLength + 4] << 24));
105:
106:         status = packet[packetLength + 5];
107:
108:         RFQueue_nextEntry();
109:     }
110: }
```

このフォーマットでサポートされている最大パケット長は 2047 バイトです (CRC (2 バイトまたは 4 バイト) を含む)。

IEEE 802.15.4g では、ペイロードのフォーマット (PSDU データ) は指定されていません。PSDU はビットのストリームとしてのみ記述され、内部にあるものは上位の MAC 仕様によって決定されます。

セクション 3.1.1 およびセクション 3.1.2 のコード例では、ペイロードは MSB ファーストで送信されます。SmartRF Studio の仕様 (デフォルトでペイロード LSB ファーストで伝送) に準拠させるには、(RX および TX の両方で) 以下のコードを使用できます。

```
1:  //-------------------------------------------------------------------------------------------
2:  // Code for Converting Payload from MSB First to LSB First
3:  //-------------------------------------------------------------------------------------------
4:
5:  uint8_t lsbFirst(uint8_t b)
6:  {
7:      b = (b & 0xF0) >> 4 | (b & 0x0F) << 4;
8:      b = (b & 0xCC) >> 2 | (b & 0x33) << 2;
9:      b = (b & 0xAA) >> 1 | (b & 0x55) << 1;
10:     return b;
11: }
```

3.2 標準パケットフォーマット (1 バイト長)

SmartRF Studio および SysConfig が高度なパケット フォーマット用に設定をエクスポート / インポートする場合でも、ユーザーは標準のパケット フォーマットを使用することもできます。セクション 3.2.1 および 3.2.2 では、1 バイト長でこの機能をサポートする方法についての TX および RX コードが示されています。

RX と TX の両方について、セットアップコマンド (CMS_PROP_RADIO_DIV_SETUP) も変更する必要があることに注意してください。このコマンドは 32 ビット長の同期ワード用に設定する必要があり、ホワイトニングをオフにする必要があります (行 29-30 (TX 例) と行 60-61 (RX 例))。

3.2.1 CMD_PROP_TX_ADV および標準パケット フォーマット (1 バイト長) を使用した TX

```
1:  //*****************************************************************************************************
2:  // Transmit Standard Packet Format (1 Length Byte) with PHYs using CMD_PROP_TX_ADV by Default
3:  //*****************************************************************************************************
4:
5:  // Defines
6:  #define PAYLOAD_LENGTH 3 // Max 255 bytes
7:  #define LENGTH_FIELD   1
8:
9:  uint8_t packet[LENGTH_FIELD + PAYLOAD_LENGTH];
10:
11: static RF_Object rfObject;
12: static RF_Handle rfHandle;
13:
14: void *mainThread(void *arg0)
15: {
16:     RF_Params rfParams;
17:     RF_Params_init(&rfParams);
18:
19:     RF_cmdPropTxAdv.startTrigger.triggerType = 0x0; // Application specific settings
20:     RF_cmdPropTxAdv.startTrigger.pastTrig = 0x0;
21:     RF_cmdPropTxAdv.numHdrBits = 0x0;
22:     RF_cmdPropTxAdv.pktLen = LENGTH_FIELD + PAYLOAD_LENGTH;
23:     RF_cmdPropTxAdv.preTrigger.triggerType = 0x0;
24:     RF_cmdPropTxAdv.preTrigger.pastTrig = 0x0;
25:     RF_cmdPropTxAdv.syncWord = 0x930B51DE;
26:     RF_cmdPropTxAdv.pPkt = packet;
27:
28:     // Necessary changes to the Setup command to support the standard packet format
29:     RF_cmdPropRadioDivSetup.formatConf.nSwBits = 0x20; // 32 bits sync word
30:     RF_cmdPropRadioDivSetup.formatConf.whitenMode = 0x0; // No whitening
31:
32:     rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
33:                        &rfParams);
34:     RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
35:
36:     while(1)
37:     {
38:         //-------------------------------------------------------------------------------------
39:         // Could be placed outside the while(1) since the packet does not change
40:         packet[0] = PAYLOAD_LENGTH;
41:
42:         for (uint16_t i = 1; i < (LENGTH_FIELD + PAYLOAD_LENGTH); i++)
43:         {
44:             packet[i] = i;
45:         }
46:         //-------------------------------------------------------------------------------------
47:         RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTxAdv, RF_PriorityNormal, NULL, 0);
48:         RF_yield(rfHandle);
49:         usleep(500000);
50:     }
51: }
```

3.2.2 CMD_PROP_RX_ADV および標準パケット フォーマット (1 バイト長) を使用した RX

```
1:   //---------------------------------------------------------------------------------------------
2:   // Receive Standard Packet Format (1 Length Byte) with PHYs using CMD_PROP_RX_ADV by Default
3:   //---------------------------------------------------------------------------------------------
4:
5:   // Defines
6:   #define DATA_ENTRY_HEADER_SIZE  8   // Constant header size of a Generic Data Entry
7:   #define NUM_DATA_ENTRIES        2   // NOTE: Only two data entries supported
8:   #define CRC                     2   // 2 if .rxConf.bIncludeCrc = 0x1, 0 otherwise
9:   #define RSSI                    1   // 1 if .rxConf.bAppendRssi = 0x1, 0 otherwise
10:  #define TIMESTAMP               4   // 4 if .rxConf.bAppendTimestamp = 0x1, 0 otherwise
11:  #define STATUS                  1   // 1 if .rxConf.bAppendStatus = 0x1, 0 otherwise
12:  #define LENGTH_FIELD            1   // RF_cmdPropRx.rxConf.bIncludeHdr = 0x1
13:  #define MAX_LENGTH              255 // Max length the radio will accept
14:  #define NUM_APPENDED_BYTES      LENGTH_FIELD + CRC + RSSI + TIMESTAMP + STATUS
15:
16:  uint8_t packet[MAX_LENGTH + NUM_APPENDED_BYTES - LENGTH_FIELD]; // Length stored in
17:  uint8_t packetLength;                                          // packetLength
18:
19:  static void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e);
20:
21:  static RF_Object rfObject;
22:  static RF_Handle rfHandle;
23:
24:  static uint8_t rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES, MAX_LENGTH,
25:                                   NUM_APPENDED_BYTES)]__attribute__((aligned(4)));
26:  static dataQueue_t dataQueue;
27:  static rfc_dataEntryGeneral_t* currentDataEntry;
28:  static uint8_t* packetDataPointer;
29:  rfc_propRxOutput_t rxStatistics;
30:  uint16_t crc16;
31:  int8_t rssi;
32:  uint32_t timestamp;
33:  uint8_t status;
34:
35:  void *mainThread(void *arg0)
36:  {
37:      RF_Params rfParams;
38:      RF_Params_init(&rfParams);
39:
40:      if(RFQueue_defineQueue(&dataQueue, rxDataEntryBuffer, sizeof(rxDataEntryBuffer),
41:                             NUM_DATA_ENTRIES, MAX_LENGTH + NUM_APPENDED_BYTES))
42:      {
43:          while(1);
44:      }
45:
46:      RF_cmdPropRxAdv.pktConf.bRepeatOk = 0x1; // Application specific settings
47:      RF_cmdPropRxAdv.pktConf.bRepeatNok = 0x1;
48:      RF_cmdPropRxAdv.pktConf.bCrcIncHdr = 0x1;
49:      RF_cmdPropRxAdv.rxConf.bAutoFlushCrcErr = 0x1;
50:      RF_cmdPropRxAdv.syncWord0 = 0x930B51DE;
51:      RF_cmdPropRxAdv.maxPktLen = MAX_LENGTH;
52:      RF_cmdPropRxAdv.hdrConf.numHdrBits = 0x8;
53:      RF_cmdPropRxAdv.hdrConf.numLenBits = 0x8;
54:      RF_cmdPropRxAdv.lenOffset = 0x0;
55:      RF_cmdPropRxAdv.pQueue = &dataQueue;
56:
57:      RF_cmdPropRxAdv.pOutput = (uint8_t*)&rxStatistics; // Opt. (for debug)
58:
59:      // Necessary changes to the Setup command to support the standard packet format
60:      RF_cmdPropRadioDivSetup.formatConf.nSwBits = 0x20; // 32 bits sync word
61:      RF_cmdPropRadioDivSetup.formatConf.whitenMode = 0x0; // No whitening
62:
63:      rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
64:                         &rfParams);
65:      RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
66:      RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropRxAdv, RF_PriorityNormal,
67:                &callback, RF_EventRxEntryDone);
68:      while(1);
69:  }
70:
```

```
71:  //---------------------------------------------------------------------------------------------
72:  // Callback for Receiving Standard Packet Format (1 Length Byte) with PHYs using
73:  // CMD_PROP_RX_ADV by Default
74:  //---------------------------------------------------------------------------------------------
75:
76:  void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e)
77:  {
78:      if(e & RF_EventRxEntryDone)
79:      {
80:          currentDataEntry = RFQueue_getDataEntry();
81:
82:          packetLength = *(uint8_t*)(&currentDataEntry->data);
83:          packetDataPointer = (uint8_t*)(&currentDataEntry->data + LENGTH_FIELD);
84:
85:          memcpy(packet, packetDataPointer,
86:                    (packetLength + NUM_APPENDED_BYTES - LENGTH_FIELD));
87:
88:          crc16 = ((uint16_t)(packet[packetLength + 0] << 8) +
89:                   (uint16_t)(packet[packetLength + 1] << 0));
90:
91:          rssi = packet[packetLength + 2];
92:
93:          timestamp = ((uint32_t)(packet[packetLength + 3] << 0 ) +
94:                       (uint32_t)(packet[packetLength + 4] << 8 ) +
95:                       (uint32_t)(packet[packetLength + 5] << 16) +
96:                       (uint32_t)(packet[packetLength + 6] << 24));
97:
98:          status = packet[packetLength + 7];
99:
100:         RFQueue_nextEntry();
101:     }
102: }
```

3.3 標準パケット フォーマット (2 バイト長)

1 バイト長から 2 バイト長 (高度なフォーマットでエクスポートされる PHY の標準パケット フォーマット) に変更する際に必要な変更は、セクション 3.3.1 および 3.3.2 にあるサンプルコードにおいて、以下の行に記載されています。

- デフォルトで CMD_PROP_TX_ADV を使用して、PHY で標準パケット フォーマット (2 バイト長) を送信します
  - 6 行目および 7 行目
  - 40 から 50 行目
- デフォルトで CMD_PROP_RX_ADV を使用する PHY での標準パケット フォーマット (2 バイト長) を受信します
  - 12 行目および 13 行目
  - 18 行目
  - 53 行目および 54 行目
- デフォルトで CMD_PROP_RX_ADV を使用した PHY で標準パケット フォーマット (2 バイト長) を受信するためのコールバック
  - 83 から 86 行目

3.3.1 CMD_PROP_TX_ADV および標準パケット フォーマット (2 バイト長) を使用した TX

```
1:  //---------------------------------------------------------------------------------------------
2:  // Transmit Standard Packet Format (2 Length Bytes) with PHYs using CMD_PROP_TX_ADV by Default
3:  //---------------------------------------------------------------------------------------------
4:
5:  // Defines
6:  #define PAYLOAD_LENGTH 3 // Max 4093 bytes
7:  #define LENGTH_FIELD   2 // Changed from 1
8:
9:  uint8_t packet[LENGTH_FIELD + PAYLOAD_LENGTH];
10:
11: static RF_Object rfObject;
12: static RF_Handle rfHandle;
13:
14: void *mainThread(void *arg0)
15: {
16:     RF_Params rfParams;
17:     RF_Params_init(&rfParams);
18:
19:     RF_cmdPropTxAdv.startTrigger.triggerType = 0x0; // Application specific settings
20:     RF_cmdPropTxAdv.startTrigger.pastTrig = 0x0;
21:     RF_cmdPropTxAdv.numHdrBits = 0x0;
22:     RF_cmdPropTxAdv.pktLen = LENGTH_FIELD + PAYLOAD_LENGTH;
23:     RF_cmdPropTxAdv.preTrigger.triggerType = 0x0;
24:     RF_cmdPropTxAdv.preTrigger.pastTrig = 0x0;
25:     RF_cmdPropTxAdv.syncWord = 0x930B51DE;
26:     RF_cmdPropTxAdv.pPkt = packet;
27:
28:     // Necessary changes to the Setup command to support the standard packet format
29:     RF_cmdPropRadioDivSetup.formatConf.nSwBits = 0x20; // 32 bits sync word
30:     RF_cmdPropRadioDivSetup.formatConf.whitenMode = 0x0;  // No whitening
31:
32:     rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
33:                        &rfParams);
34:     RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
35:
36:     while(1)
37:     {
38:         //---------------------------------------------------------------------------------
39:         // Could be placed outside the while(1) since the packet does not change
40: #ifdef SLR_MODE
41:         packet[0] = (uint8_t)(PAYLOAD_LENGTH);
42:         packet[1] = (uint8_t)(PAYLOAD_LENGTH >> 8);
43: #else
44:         packet[0] = (uint8_t)(PAYLOAD_LENGTH >> 8);
45:         packet[1] = (uint8_t)(PAYLOAD_LENGTH);
46: #endif
47:         for (uint16_t i = 2; i < (LENGTH_FIELD + PAYLOAD_LENGTH); i++)
48:         {
49:             packet[i] = i - 1;
50:         }
51:         //---------------------------------------------------------------------------------
52:         RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTxAdv, RF_PriorityNormal, NULL, 0);
53:         RF_yield(rfHandle);
54:         usleep(500000);
55:     }
56: }
```

3.3.2 CMD_PROP_RX_ADV および標準パケット フォーマット (2 バイト長) を使用した RX

```
1:   //---------------------------------------------------------------------------------------------
2:   // Receive Standard Packet Format (2 Length Bytes) with PHYs using CMD_PROP_RX_ADV by Default
3:   //---------------------------------------------------------------------------------------------
4:
5:   // Defines
6:   #define DATA_ENTRY_HEADER_SIZE  8    // Constant header size of a Generic Data Entry
7:   #define NUM_DATA_ENTRIES        2    // NOTE: Only two data entries supported
8:   #define CRC                     2    // 2 if .rxConf.bIncludeCrc = 0x1, 0 otherwise
9:   #define RSSI                    1    // 1 if .rxConf.bAppendRssi = 0x1, 0 otherwise
10:  #define TIMESTAMP               4    // 4 if .rxConf.bAppendTimestamp = 0x1, 0 otherwise
11:  #define STATUS                  1    // 1 if .rxConf.bAppendStatus = 0x1, 0 otherwise
12:  #define LENGTH_FIELD            2    // Changed from 1
13:  #define MAX_LENGTH              4093 // Changed from 255
14:  #define NUM_APPENDED_BYTES      LENGTH_FIELD + CRC + RSSI + TIMESTAMP + STATUS
15:
16:  uint8_t packet[MAX_LENGTH + NUM_APPENDED_BYTES - LENGTH_FIELD]; // Length stored in
17:                                                                  // packetLength
18:  uint16_t packetLength; // Changed from uint8_t
19:
20:  static void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e);
21:
22:  static RF_Object rfObject;
23:  static RF_Handle rfHandle;
24:
25:  static uint8_t rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES, MAX_LENGTH,
26:                                   NUM_APPENDED_BYTES)]__attribute__((aligned(4)));
27:  static dataQueue_t dataQueue;
28:  static rfc_dataEntryGeneral_t* currentDataEntry;
29:  static uint8_t* packetDataPointer;
30:  rfc_propRxOutput_t rxStatistics;
31:  uint16_t crc16;
32:  int8_t rssi;
33:  uint32_t timestamp;
34:  uint8_t status;
35:
36:  void *mainThread(void *arg0)
37:  {
38:      RF_Params rfParams;
39:      RF_Params_init(&rfParams);
40:
41:      if(RFQueue_defineQueue(&dataQueue, rxDataEntryBuffer, sizeof(rxDataEntryBuffer),
42:                             NUM_DATA_ENTRIES, MAX_LENGTH + NUM_APPENDED_BYTES))
43:      {
44:          while(1);
45:      }
46:
47:      RF_cmdPropRxAdv.pktConf.bRepeatOk = 0x1; // Application specific settings
48:      RF_cmdPropRxAdv.pktConf.bRepeatNok = 0x1;
49:      RF_cmdPropRxAdv.pktConf.bCrcIncHdr = 0x1;
50:      RF_cmdPropRxAdv.rxConf.bAutoFlushCrcErr = 0x1;
51:      RF_cmdPropRxAdv.syncWord0 = 0x930B51DE;
52:      RF_cmdPropRxAdv.maxPktLen = MAX_LENGTH;
53:      RF_cmdPropRxAdv.hdrConf.numHdrBits = 0x10; // Changed from 0x8
54:      RF_cmdPropRxAdv.hdrConf.numLenBits = 0x10; // Changed from 0x8
55:      RF_cmdPropRxAdv.lenOffset = 0x0;
56:      RF_cmdPropRxAdv.pQueue = &dataQueue;
57:
58:      RF_cmdPropRxAdv.pOutput = (uint8_t*)&rxStatistics; // Optional (for debugging)
59:
60:      // Necessary changes to the Setup command to support the standard packet format
61:      RF_cmdPropRadioDivSetup.formatConf.nSwBits = 0x20; // 32 bits sync word
62:      RF_cmdPropRadioDivSetup.formatConf.whitenMode = 0x0; // No whitening
63:
64:      rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
65:                         &rfParams);
66:      RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
67:      RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropRxAdv, RF_PriorityNormal,
68:                &callback, RF_EventRxEntryDone);
69:      while(1);
70:  }
71:
```

```
72:  //---------------------------------------------------------------------------------------------
73:  // Callback for Receiving Standard Packet Format (2 Length Bytes) with PHYs
74:  // using CMD_PROP_RX_ADV by Default
75:  //---------------------------------------------------------------------------------------------
76:
77:  void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e)
78:  {
79:      if(e & RF_EventRxEntryDone)
80:      {
81:          currentDataEntry = RFQueue_getDataEntry();
82:
83:          packetLength = ((uint16_t)(*(uint8_t*)(&currentDataEntry->data + 1) << 8) |
84:                          (uint16_t)(*(uint8_t*)(&currentDataEntry->data + 0) << 0));
85:          // Changed from
86:          // packetLength = *(uint8_t*)(&currentDataEntry->data);
87:
88:          packetDataPointer = (uint8_t*)(&currentDataEntry->data + LENGTH_FIELD);
89:
90:          memcpy(packet, packetDataPointer,
91:                 (packetLength + NUM_APPENDED_BYTES - LENGTH_FIELD));
92:
93:          crc16 = ((uint16_t)(packet[packetLength + 0] << 8) +
94:                   (uint16_t)(packet[packetLength + 1] << 0));
95:
96:          rssi = packet[packetLength + 2];
97:
98:          timestamp = ((uint32_t)(packet[packetLength + 3] << 0 ) +
99:                       (uint32_t)(packet[packetLength + 4] << 8 ) +
100:                      (uint32_t)(packet[packetLength + 5] << 16) +
101:                      (uint32_t)(packet[packetLength + 6] << 24));
102:
103:         status = packet[packetLength + 7];
104:
105:         RFQueue_nextEntry();
106:     }
107: }
```

4 参考資料

1. テキサス インスツルメンツ、SIMPLELINK-LOWPOWER-F2-SDK、ウェブページ。
2. テキサス インスツルメンツ、『SmartRF™ Studio』、ウェブページ。
3. テキサス インスツルメンツ、SysConfig、ウェブページ。

重要なお知らせと免責事項

TI は、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、TI 製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した TI 製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている TI 製品を使用するアプリケーションの開発の目的でのみ、TI はその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。TI や第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、 TI の販売条件 、 TI の総合的な品質ガイドライン 、 ti.com または TI 製品などに関連して提供される他の適用条件に従い提供されます。TI がこれらのリソースを提供することは、適用される TI の保証または他の保証の放棄の拡大や変更を意味するものではありません。 TI がカスタム、またはカスタマー仕様として明示的に指定していない限り、TI の製品は標準的なカタログに掲載される汎用機器です。

お客様がいかなる追加条項または代替条項を提案する場合も、TI はそれらに異議を唱え、拒否します。

最終更新日：2025 年 10 月