AN5687

Application note

Long-packet operation with STM32CubeWL

Introduction

This application note describes how to send and receive long packets (greater than 255 bytes) with the STM32CubeWL MCU package that runs on the STM32WL Series microcontrollers.

To support longer packets, some algorithms have been implemented in the software radio driver.

This document details the receive and transmit data buffer operations, the packet length limitation, and how to overcome this limitation. It also describes the algorithms to implement virtually limitless packet length by software.

This application note explains how to integrate the new long-packet APIs into an application such as PER (packet error rate).

1 General information

The STM32CubeWL runs on STM32WL Series microcontrollers based on the Arm® Cortex®-M processor.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Table 1. **Terms and acronyms**

Acronym	Definition
CCITT	International Telegraph and Telephone Consultative Committee
GFSK	Gaussian frequency shift keying
PER	Packet error rate
Rx	Reception
Tx	Transmission

Reference documents

[1] STM32WLEx reference manual (RM0461) or STM32WL5x reference manual (RM0453)

2 Radio buffer limitation and beyond

Before digging into long-packet operation, a description of the limitation is important to understand how to unlock this limitation.

2.1 Radio data buffer description

The data interface between the CPU and the sub-GHz radio is done through a sub-GHz radio data buffer to store Tx data bytes to be sent, or Rx data bytes being received (see the document [1] for more details).

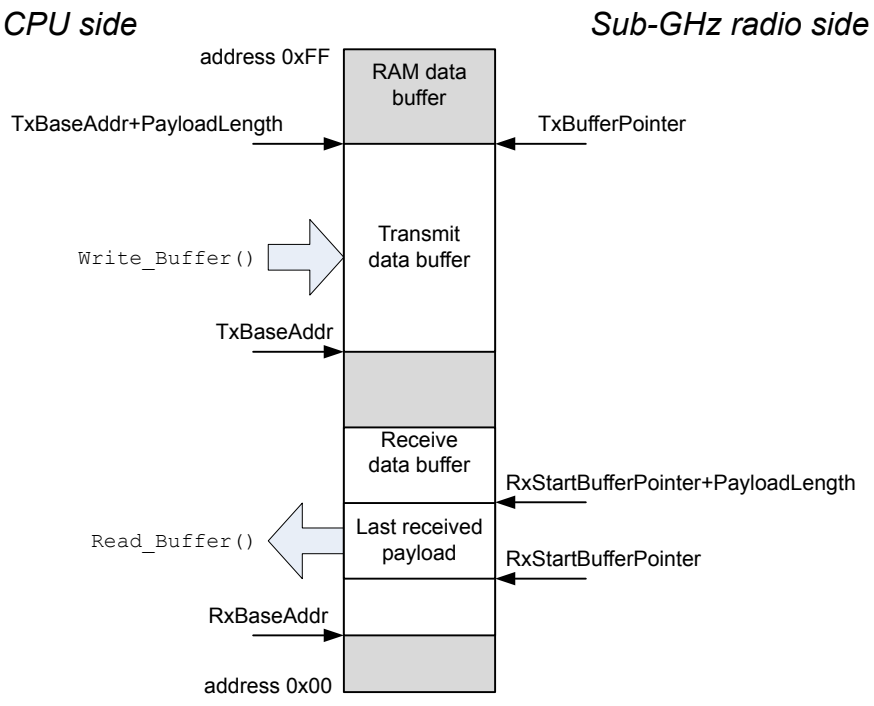
Radio data buffer in transmit

TxBaseAddr determines the transmit-buffer offset in the sub-GHz radio buffer. For each transmission, when the sub-GHz radio enters Tx mode, TxBufferPointer is automatically initialized to TxBaseAddr. Each time a payload byte is transmitted, TxBufferPointer is incremented until TxBufferPointer reaches TxBaseAddr+PayloadLength, meaning all bytes are transmitted. At that point, a TxDone interrupt is issued.

Radio data buffer in reception

RxBaseAddr determines the receive-buffer offset in the sub-GHz radio buffer. For each reception, when the sub-GHz radio enters Rx mode, RxstartBufferPointer is automatically initialized to RxBaseAddr. Each time a payload byte is received, RxstartBufferPointer is incremented until RxstartBufferPointer reaches RxBaseAddr+PayloadLength, meaning all bytes are received. At that point, a RxDone interrupt is issued.

Figure 1. Radio buffer



Note:

- *Usually, the TxBaseAddr and RxBaseAddr are set to 0 at the start of the transmission or reception, respectively.*
- *TxBufferPointer is circular by nature, meaning it wraps around to 0 after 255. For example, if TxBaseAddr is set to 254 and PayloadLength set to 20, TxBufferPointer is initialized to 254 and sends byte 254, 255, 0 …17. The transmission stops when TxBufferPointer equal to modulo (254 + PayloadLength - 1, 256).*

2.2 Unlock packet-length limitation

As explained above:

- The transmission stops when TxBufferPointer reaches TxBaseAddr+PayloadLength.
- The reception stops when RxstartBufferPointer reaches RxBaseAddr+PayloadLength.

On STM32WL revision Z, PayloadLength is fixed (latched in the radio) after a `SetTx()` command. Therefore a long-packet operation cannot be implemented on revision Z. On STM32WL revision Y, PayloadLength can be updated during packet transmission. PayloadLength can be updated during packet reception on both STM32WL Y and Z revisions.

In order to send or receive longer packets, the PayloadLength is dynamically adjusted during one packet handling until the end of the long packet.

Note:

- *PayloadLength is accessed via the SUBGHZ_RTXPLDLEN register.*
- *TxBufferPointer is accessed via the SUBGHZ_TX_ADR_PTR register.*
- *RxstartBufferPointer is accessed via the SUBGHZ_RX_ADR_PTR register.*

3 Long-packet operations

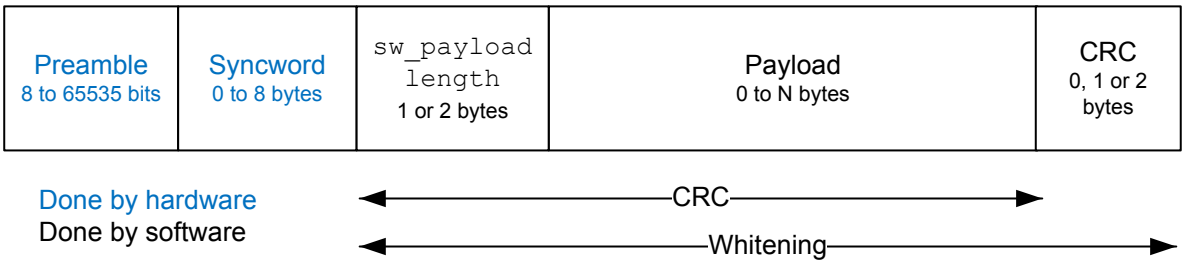
3.1 Packet coding

Since the STM32WL sub-GHz radio natively supports a 255-byte maximum payload length, the payload length field is 8-bit long (see the document [1] for more details). For longer packet operations, a 16-bit payload length field is required (`sw_payloadlength`). Fixed length generic packet format is used with Address, CRC, and whitening disabled.

The software driver builds a packet as shown in the figure below.

- The first 2 bytes (optionally 1 byte if payload length is shorter than 256 bytes) are used to specify the payload size. `sw_payloadlength` contains the payload number of bytes excluding CRC.
- The CRC is calculated over `sw_payloadlength`, `Payload`, and is appended after the payload.
- Data whitening is applied on `sw_payloadlength`, `Payload`, and `CRC` (both IBM and CCITT whitening algorithms are supported by the software algorithm).

Figure 2. Long packet with fixed-length generic packet



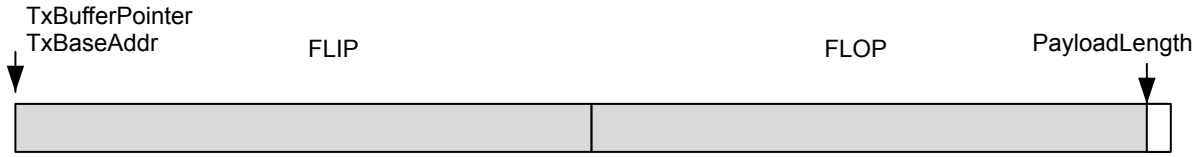
3.2 Transmit long-packet description

Important:
Transmit long-packet operation is available only on STM32WL revision Y. The revision can be populated by connecting the STM32CubeProgammer tool. In the bottom right-hand-side, the Revision ID is displayed. If Rev Z appears, the Tx long-packet mode is not supported.

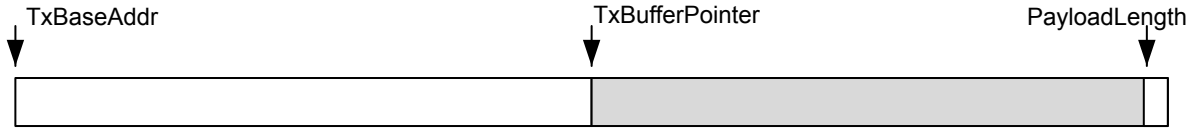
To decrease the real time constraints, the double-buffer technique (flip/flop) is used on the STM32WL sub-GHz radio buffer. While FLOP is sent to the air by the radio, the CPU fills FLIP. Conversely, while FLIP is sent to the air by the radio, the CPU fills FLOP, as shown in the figure below. A timer is used to determine the FLIP/FLOP boundary. The timer timeout is calculated using the transmit bitrate.

Figure 3. Tx dual-buffer mechanism

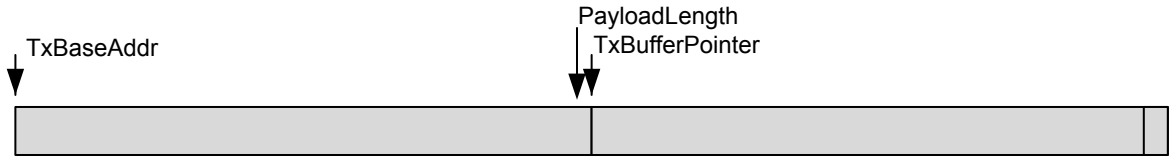
1. Fill the first 255-byte payload in the buffer.



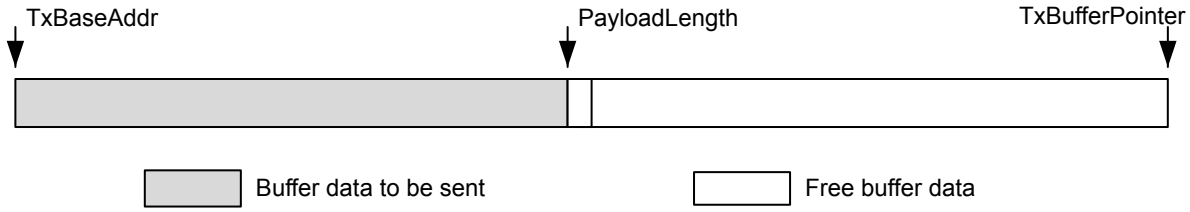
2. After 128 bytes (timer IRQ), the first 128 bytes (FLIP) are sent to the air and The FLIP is free. TxBufferPointer has advanced. There is still a 128-byte **margin** to fill FLIP while the last 128 bytes (FLOP) are being sent to the air.



3. Free buffer bytes are being filled while FLOP is being sent, and PayloadLength is updated. New timer is programmed.



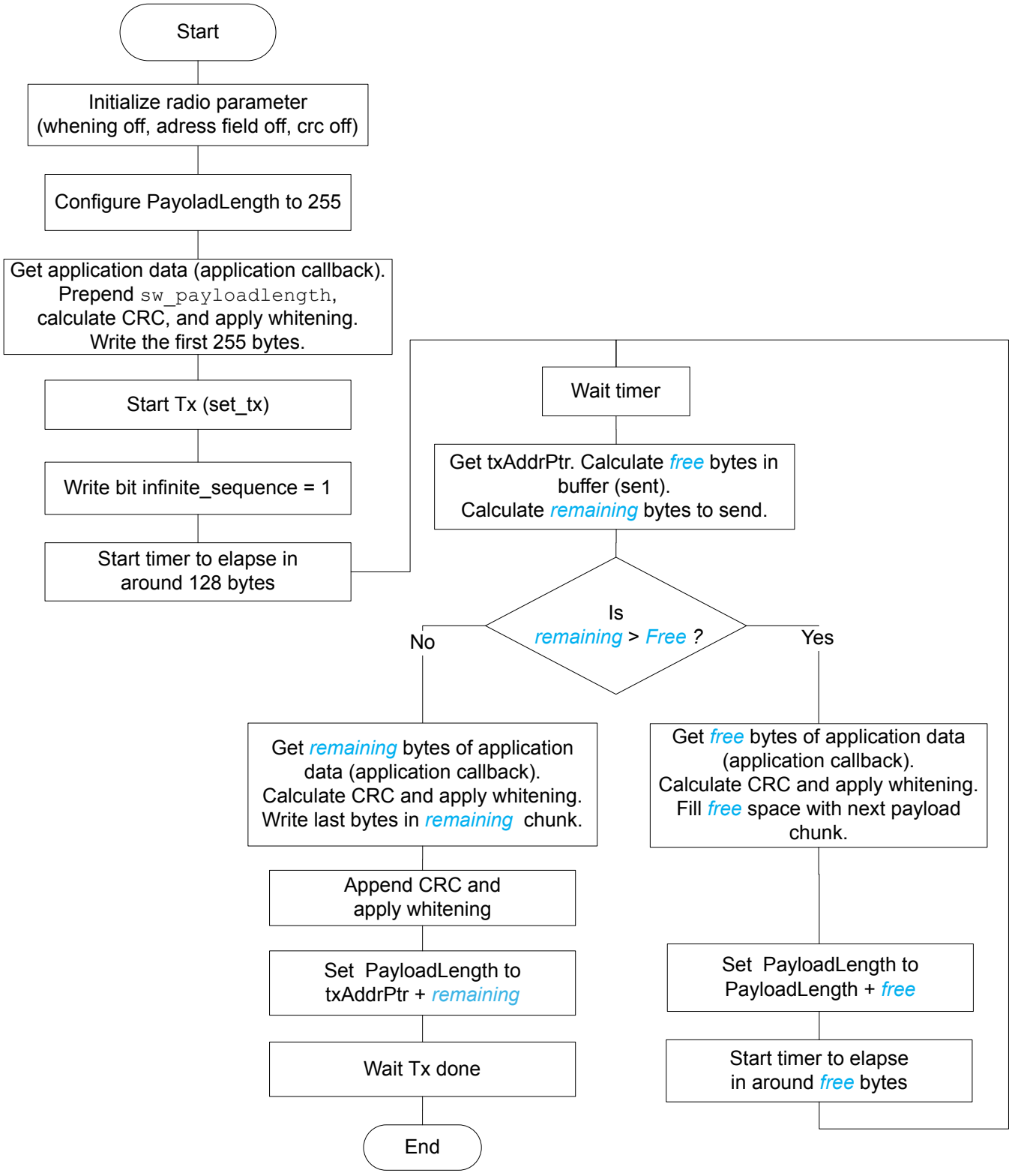
4. FLOP has been sent to the air and FLIP is being sent. FLOP is then free and can be filled with new data.



Note: At 50 Kbit/s, 128 bytes last 20 ms. At 300 Kbit/s, 128 bytes lasts 3.4 ms.

The figure below explains the Tx long-packet flow chart.

Figure 4. Tx long-packet flow chart

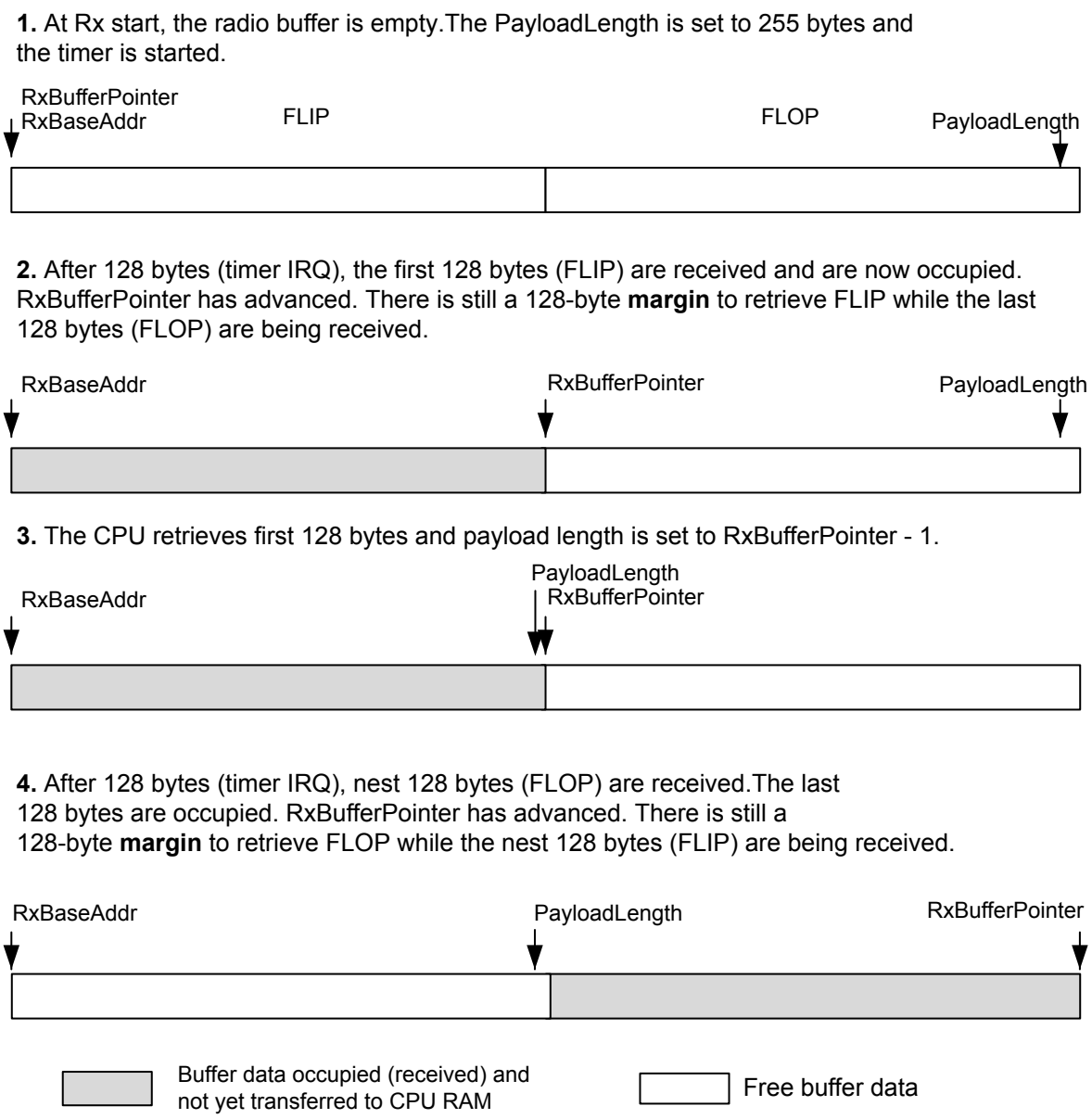


3.3 Receive long-packet description

Note: Receive long-packet operation is available on all STM32WL revisions.

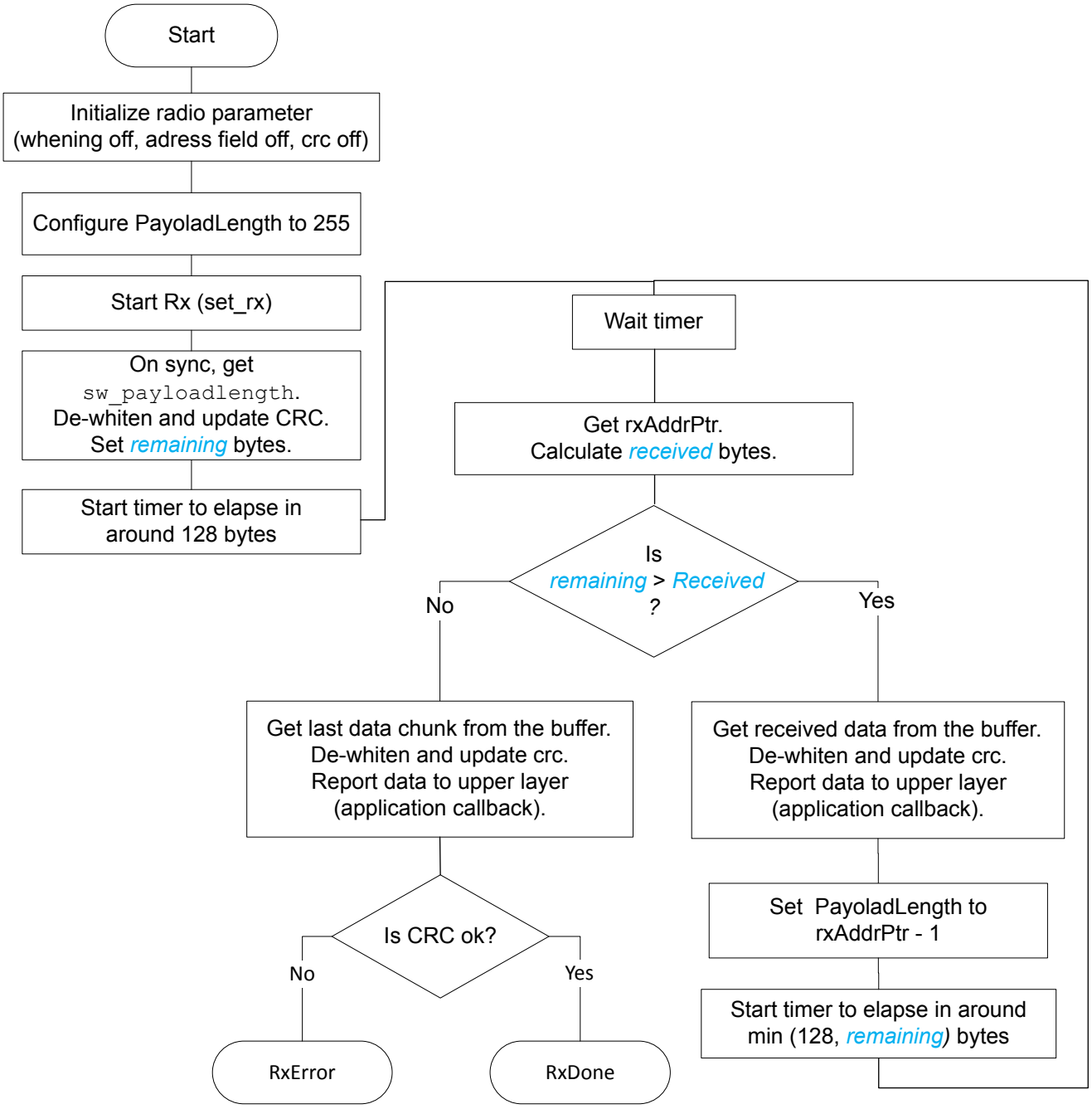
Like in the transmit long-packet operation, the double-buffer technique is used. While the radio receives data into FLOP, the CPU saves FLIP into the CPU RAM. Conversely, while the radio receives data into FLIP, the CPU saves FLOP into the CPU RAM, as shown in the figure below. A timer is used to determine the FLIP/FLOP boundary. The timer timeout is calculated using the reception bitrate.

Figure 5. Rx dual-buffer mechanism



The figure below describes the Rx long-packet flow chart.

Figure 6. Rx long-packet flow chart



4 Application integration

The algorithms described above are implemented in `radio_fw.c`. This section describes the APIs and how to integrate them.

Note: Only the GFSK modulation can support long packet as GFSK is not much sensitive to timing drift issue.

4.1 Transmit and receive APIs

The APIs detailed in the table below have been added to `radio.h` to manage long-packet operation.

Table 2. Send/receive additional APIs

Function	Description
`int32_t RFW_TransmitLongPacket( uint16_t payload_size, uint32_t timeout, void (*TxLongPacketGetNextChunkCb) (uint8_t** buffer, uint8_t buffer_size) );`	Starts transmitting long packets (packet length may be on 1 byte depending on the configuration). `*TxLongPacketGetNextChunkCb` callback is used to request application new chunk of data to be sent. `RFW_TransmitLongPacket` is accessible though `radio.h` interface via `Radio.TransmitLongPacket`.
`int32_t RFW_ReceiveLongPacket( uint8_t boosted_mode, uint32_t timeout, void (*RxLongStorePacketChunkCb) (uint8_t* buffer, uint8_t chunk_size) );`	Starts receiving long packets (that maybe shorts). `*RxLongStorePacketChunkCb` callback is used to request application to store chunk of received data. `RFW_ ReceiveLongPacket` is accessible though `radio.h` interface via `Radio. ReceiveLongPacket`.

Note: The legacy `Radio.SetRxConfig` and `Radio.SetTxConfig` APIs cannot be used with long-packet APIs. The radio configuration must be done through `Radio.RadioSetRxGenericConfig` and `Radio.RadioSetTxGenericConfig`.

4.2 Radio initialization

As for the legacy radio APIs, the radio initialization and frequency configuration are done before any transmission or reception can be done, with the code below.

```
RadioEvents.TxDone = OnTxDone;
RadioEvents.RxDone = OnRxDone;
RadioEvents.TxTimeout = OnTxTimeout;
RadioEvents.RxTimeout = OnRxTimeout;
RadioEvents.RxError = OnRxError;
Radio.Init(&RadioEvents);
/* Radio Set frequency */
Radio.SetChannel(RF_FREQUENCY);
```

4.3 Rx long-packet examples

Example of radio reception configuration

`LengthMode` is extended to support a 2-byte payload length for long packets.

```
  RxConfig.fsk.ModulationShaping = RADIO_FSK_MOD_SHAPING_G_BT_05;
  RxConfig.fsk.Bandwidth = FSK_BANDWIDTH;
  RxConfig.fsk.BitRate = FSK_DATARATE; /*BitRate*/
  RxConfig.fsk.PreambleLen = 4; /*in Byte*/
  RxConfig.fsk.SyncWordLength = sizeof(syncword); /*in Byte*/
  RxConfig.fsk.PreambleMinDetect = RADIO_FSK_PREAMBLE_DETECTOR_08_BITS;
  RxConfig.fsk.SyncWord = syncword; /*SyncWord Buffer*/
  RxConfig.fsk.whiteSeed = 0x01FF ; /*WhiteningSeed*/
  RxConfig.fsk.LengthMode  = RADIO_FSK_PACKET_2BYTES_LENGTH; /* 2 bytes long payload length
field */
  RxConfig.fsk.CrcLength = RADIO_FSK_CRC_2_BYTES_IBM;        /* 2 Bytes CRC length with IBM
*/
  RxConfig.fsk.CrcPolynomial = 0x8005;
  RxConfig.fsk.CrcSeed = 0xFFFF;
  RxConfig.fsk.Whitening = RADIO_FSK_DC_IBM_WHITENING; /*IBM payload whitening */
  RxConfig.fsk.MaxPayloadLength = MAX_APP_BUFFER_SIZE;
  RxConfig.fsk.StopTimerOnPreambleDetect = 0;
  RxConfig.fsk.AddrComp = RADIO_FSK_ADDRESSCOMP_FILT_OFF;

  if (0UL != Radio.RadioSetRxGenericConfig(GENERIC_FSK, &RxConfig, RX_CONTINUOUS_ON, 0))
  {
    while (1);
  }
```

Example of radio reception start

```
data_offset = 0;
(void) Radio.ReceiveLongPacket(0, RX_TIMEOUT_VALUE, RxLongPacketChunk);
```

Example of Rx callback

```
#define MAX_APP_BUFFER_SIZE 1000
uint8_t data_buffer[MAX_APP_BUFFER_SIZE] UTIL_MEM_ALIGN(4);
uint16_t data_offset = 0;

void RxLongPacketChunk(uint8_t *buffer, uint8_t chunk_size)
{
  uint8_t *rxdata = &data_buffer[data_offset];
  uint8_t *rxbuffer = buffer;

  if (data_offset + chunk_size > MAX_APP_BUFFER_SIZE)
  {
    return;
  }
  for (int i = 0; i < chunk_size; i++)
  {
    *rxdata++ = *rxbuffer++;
  }
  data_offset += chunk_size;
}
```

4.4 Tx long-packet examples

Example of radio transmission configuration

`HeaderType` is extended to support a 2-byte payload length for long packets.

```
TxConfig.fsk.ModulationShaping = RADIO_FSK_MOD_SHAPING_G_BT_05;
TxConfig.fsk.Bandwidth = FSK_BANDWIDTH;
TxConfig.fsk.FrequencyDeviation = FSK_DEVIATION;
TxConfig.fsk.BitRate = FSK_BIT_RATE; /*BitRate*/
TxConfig.fsk.PreambleLen = 4;    /*in Byte        */
TxConfig.fsk.SyncWordLength = sizeof(syncword); /*in Byte        */
TxConfig.fsk.SyncWord = syncword; /*SyncWord Buffer*/
TxConfig.fsk.whiteSeed =  0x01FF ; /*WhiteningSeed  */
TxConfig.fsk.HeaderType = RADIO_FSK_PACKET_2BYTES_LENGTH; /* 2 Bytes playload length field*/
TxConfig.fsk.CrcLength = RADIO_FSK_CRC_2_BYTES_IBM;       /* 2 Bytes CRC length with IBM */
TxConfig.fsk.CrcPolynomial = 0x8005;
TxConfig.fsk.CrcSeed = 0xFFFF;
TxConfig.fsk.Whitening = RADIO_FSK_DC_IBM_WHITENING; /*IBM payload whitening */

if (0UL !=Radio.RadioSetTxGenericConfig(GENERIC_FSK,&TxConfig, TX_OUTPUT_POWER,
TX_TIMEOUT_VALUE))
{
  while(1);
}
```

Example of radio transmission start

```
data_offset=0;
if (0UL !=Radio.TransmitLongPacket( sw_payloadlength, TX_TIMEOUT_VALUE,
TxLongPacketGetNextChunk))
{
  while(1);
}
```

Example of Tx callback

Tx callback is used by the radio layer to request the application to feed new data chunk in real time.

```
#define DATA_BUFFER_SIZE 1000
uint8_t data_buffer[DATA_BUFFER_SIZE] UTIL_MEM_ALIGN(4); /*App buffer*/
uint16_t data_offset=0;                                  /*App buffer index to be
initialized buffer any tx start*/


/*Radio layer requests a new chunk of size chunk_size to send
  It is up to the application to maintain data_offset (index)*/

void TxLongPacketGetNextChunk(uint8_t** buffer, uint8_t chunk_size)
{
  *buffer=&data_buffer[data_offset];
  data_offset+=chunk_size;
}
```

5 Packet error rate application

The PER (packet error rate) application requires two STM32WL devices: one configured in transmit mode and the other configured in receive mode.

The PER application implements the APIs shown previously. The PER application is located in the STM32CubeWL folder

`Firmware\Projects\NUCLEO-WL55JC\Applications\SubGHz_Phy\SubGHz_Phy_Per`.

By default, this application is configured in short-packet mode. In order to configure this application in long-packet mode, the following actions are needed:

- In `subghz_phy_app.h`, define `PAYLOAD_LEN` to a length greater than 255 bytes.
- In `subghz_phy_app.c`, define `APP_LONG_PACKET` to 1 and make sure the die version is revision Y.
- In `subghz_phy_app.c`:
  - Define `TEST_MODE` to `RADIO_TX` for the transmitter board, compile and load.
  - Define `TEST_MODE` to `RADIO_RX` for the receiving board, compile and load.

The `radio_conf.h` file is located in `SubGHz_Phy_Per\SubGHz_Phy\Target\`.

`RFW_ENABLE` and `RFW_LONGPACKET_ENABLE` switches must be defined to 1.

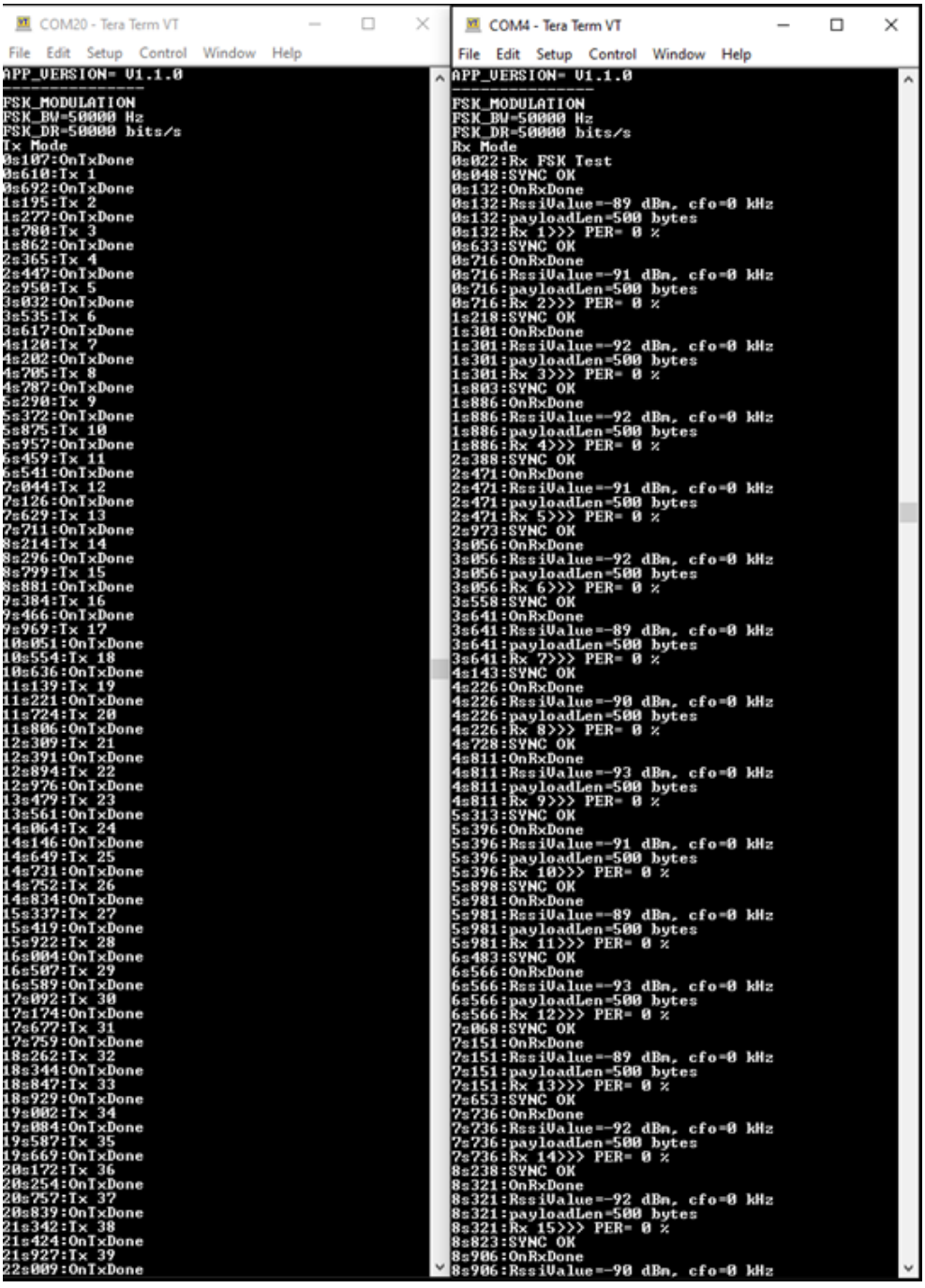
`RFW_ENABLE` enables the custom packet type like IBM whitening. `RFW_LONGPACKET_ENABLE` enables the long-packet mode.

```
/**
  * @brief enables the RFW module
  * @note disabled by default
  */
#define RFW_ENABLE 1

/**
  * @brief enables the RFW long packet feature
  * @note disabled by default
  */
#define RFW_LONGPACKET_ENABLE 1
```

The figure below shows a terminal output between two boards.

Figure 7. Terminal output



Revision history

Table 3. **Document revision history**

Date	Version	Changes
21-Jul-2021	1	Initial release.

Contents

1 General information 2
2 Radio buffer limitation and beyond 3
2.1 Radio data buffer description 3
2.2 Unlock packet-length limitation 4
3 Long-packet operations 5
3.1 Packet coding 5
3.2 Transmit long-packet description 6
3.3 Receive long-packet description 8
4 Application integration 10
4.1 Transmit and receive APIs 10
4.2 Radio initialization 10
4.3 Rx long-packet examples 11
4.4 Tx long-packet examples 12
5 Packet error rate application 13
Revision history 15
List of tables 17
List of figures 18

List of tables

Table 1. Terms and acronyms 2
Table 2. Send/receive additional APIs 10
Table 3. Document revision history 15

List of figures

Figure 1. Radio buffer . . . 3
Figure 2. Long packet with fixed-length generic packet . . . 5
Figure 3. Tx dual-buffer mechanism . . . 6
Figure 4. Tx long-packet flow chart . . . 7
Figure 5. Rx dual-buffer mechanism . . . 8
Figure 6. Rx long-packet flow chart . . . 9
Figure 7. Terminal output . . . 14