Chipcon Products
from Texas Instruments

CC1100/CC1150DK & CC2500/CC2550DK
Development Kit

Examples and Libraries
User Manual

Rev. 1.3

Table of contents

1. Introduction

This User Manual covers the software examples and libraries used with the CC2500/CC2550DK Development Kit and CC1100/CC1150DK Development Kit.

2. Definitions

SmartRF®04DK	A collective term used for all development kits for the SmartRF®04 platform, i.e. CC2500/CC2550DK and CC1100/CC1150 DK
USB MCU	The Silicon Labs C8051F320 MCU used to provide a USB interface on the SmartRF®04EB
Factory firmware	The firmware that is programmed into the USB MCU from the factory. This firmware supports SmartRF® Studio operation as well as a stand-alone PER tester.
PER	Packet Error Rate. Measures the percentage of packets that contain errors or are lost.

3. General Notes about the Software

Both the examples and libraries are written for the Keil C51 C complier for the 8051 platform. You may have to modify the source code somewhat if you intend to compile the code using another 8051 C compiler. The examples are supplied in both source code and .hex file form. There are several .hex files supplied for each example. For every frequency band (315, 433, 868 etc.) there are two .hex files; one for stand-alone use (requires that you have access to Silicon Labs' EC2 programming tool) and one for use with the bootloader. Even if you do not have access to an 8051 C compiler, you can program the *filename*_bootloader.hex files into the USB MCU using SmartRF® Studio (as long as you have not overwritten the bootloader that come programmed into the EB from the factory).

4. Running the Examples

It is easy to run the examples Chipcon provide for this platform. SmartRF® Studio can be used to load different .hex files into the USB MCU. Note that the examples have to be linked with the bootloader libraries to work (all examples provided from Chipcon have been linked with these libraries).

Connect the Evaluation Board to a PC using the USB interface. Start SmartRF® Studio and select the SmartRF® 04 tab. Select the Evaluation Board (do not choose the "Calculation Windows") and click on the "Load USB Firmware" button. You are then presented with a file selection dialog box where you can select the file to download (*filename*_bootloader.hex).

If you have overwritten the bootloader, you must program the bootloader into the USB MCU using Silicon Labs' EC2 serial programmer before you can program the examples by the aforementioned method. The bootloader .hex file and programming software that uses the EC2 are included in the SmartRF® Studio installation. These files are installed into the SmartRF® Studio folder, but the installer does not generate desktop shortcut or start menu shortcuts for this program.

If you have loaded one of the examples and then attempt to run SmartRF® Studio, SmartRF® Studio will detect that the factory firmware has been overwritten and prompt you that it will attempt to write the factory firmware to the USB MCU.

5. MCU Examples

The examples should be installed into the C:\Keil\C51\Examples\Chipcon\srf04 directory.

5.1 Audio

This example runs a loop-back test from the audio input to the audio output. The USB MCU samples the incoming audio using the built-in ADC, and sends it right back out again using the PWM functionality of the MCU. Copying from the ADC to the PWM is done in an interrupt routine triggered by the timer 1 interrupt.

To test the example, connect a headset (the headset should have separate mini-jacks for microphone and headphones) to the microphone input and the headphone output on the Evaulation Board. You should now be able to hear your own voice in the headphones. Make sure that the volume control is not turned all the way down.

Note that the audio quality of this example can be improved by performing processing of the raw data. Sampling at a higher rate and then performing averaging to implement digital filtering would help in reducing noise caused by aliasing.

5.2 Joystick

This example demonstrates reading the joystick and writing to the LCD. This program runs in an infinite loop reading the status of the joystick and reporting this status on the LCD. To save pins, the joystick position is coded as an analogue value on the Evaluation Board, and is read using the ADC of the USB MCU. The ebGetJoystickPosition() function uses the ADC to read the status and decodes this to a direction.

To test the example, simply move the joystick and see the status change on the LCD display. The LCD display will also indicate if you have pressed down the integrated joystick button.

5.3 Potmeter

This example demonstrates reading the potmeter position using the ADC and reports the value on the LCD display.

The program runs in an infinite loop waiting for the push button (S1) on the EB to be pushed. When this happens, it reads the potmeter postion and sends this information to the LCD display.

To test this example, press S1 and the current potmeter value will be displayed on the LCD. Turn the potmeter knob and press S1 again.

5.4 Spi

This example demonstrates writing and reading CCxx00/CCxx50 registers and communication with a PC using the RS-232 interface.

The program first displays a menu, and then runs in an infinite loop waiting for input from the RS-232 port. Depending on the option selected by the user, the program will either read or write CCxx00/CCxx50 registers and report the results back via RS-232.

To test this example, connect the Evaluation Board to a PC using a male-to-female one-to-one RS-232 cable. Start HyperTerminal or another terminal program, setting it up to 115200 baud, 8 data bits, 1 stop bit and no hardware handshaking. In HyperTerminal, make sure to choose “Connect” or press a key on the PC keyboard to connect. The program will display a menu on the screen, and you can access registers by pressing the appropriate character. Make sure you have inserted an EM before running this example.

5.5 Timer01

This example demonstrates use of timer 0/1 and the LEDs on the Evaluation Board.

The program runs in an infinite loop reading the status of the potmeter, writing this value into a global variable. An interrupt routine is triggered by Timer 1. This routine reads the global variable that contains the status of the potmeter, and updates the timing of Timer 1. The LEDs are controlled by the interrupt routine so that they each show one bit of a 4-bit counter that is incremented every time the interrupt routine is executed.

To test this example, simply turn the potmeter. This will adjust the speed at which the LEDs blink.

5.6 Timer23

This example demonstrates use of timer 2/3 and the LEDs on the Evaluation Board.

The program runs in an infinite loop after setting up an interrupt routine triggered by timer 2. Timer 2 is configured to function as two 8-bit timers with different periods. The interrupt routine is triggered by the two timers. Every 10000'th time the timer overflows, a LED is toggled. One 8-bit timer triggers the green LED, the other triggers the red LED.

When the program is run, the red LED will blink at a different rate from the green LED.

6. Radio Examples

The examples should be installed into the C:\Keil\C51\Examples\Chipcon\srf04\CCxx00 directory.

6.1 Link

This program demonstrates how to set up a simple RF link between two units.

By moving the joystick right or left, the user can set up one unit as transmitter (left) and one unit as receiver (right). After selecting correct mode, the joystick button should be pushed. Now the transmitter will send one packet every time the S1 button is being pushed. The number of packets transmitted and received is displayed on the LCD display on the TX and RX unit, respectively. On the receiver, the CRC is checked before the display is updated (i.e. received packets containing bit errors are not counted).

6.2 Link1

This example demonstrates how to set up a simple link between two CCxx00EMs run from SmartRF04EB. Packet transmission and packet reception is implemented by polling the chip status byte every time a timer interrupt occurs (every 200 us). On the transmitter, this is done to see if there are more available/free bytes in the TX FIFO in case the TX FIFO needs to be re-filled. On the receiver it is done to see of more bytes have been received. This method is useful in cases where the packet size is greater than the FIFO size. The joystick is used to navigate through a menu, setting different parameters.

Parameter	Settings
Packet Length	10, 30, 50, ……., 230, 250
Number of Packets	100, 200, 300, ……, 900, 1000
Whitening	Enabled, Disabled
Radio Mode	Rx, Tx

The following steps must be done to start the link test:

Rx Unit:

- Enable/disable Whitening
- Set radio mode to RX.
- Move joystick down until the message "Press S1 to start" is showed on the LCD display
- Press S1

The LCD display will show number of packets received with CRC OK.

Tx Unit:

- Set packet length and number of packets to transmit
- Enable/disable Whitening (set to the same as on the RX unit)
- Set radio mode to TX
- Press S1 to Start

The LCD will show number of packets transmitted. After all the packets have been transmitted, S1 can be pressed to run the test once more or the joystick can be used to change packet length and number of packets before running a new test.

The main loop is implemented as a state machine and the state diagram is showed in Figure 1.

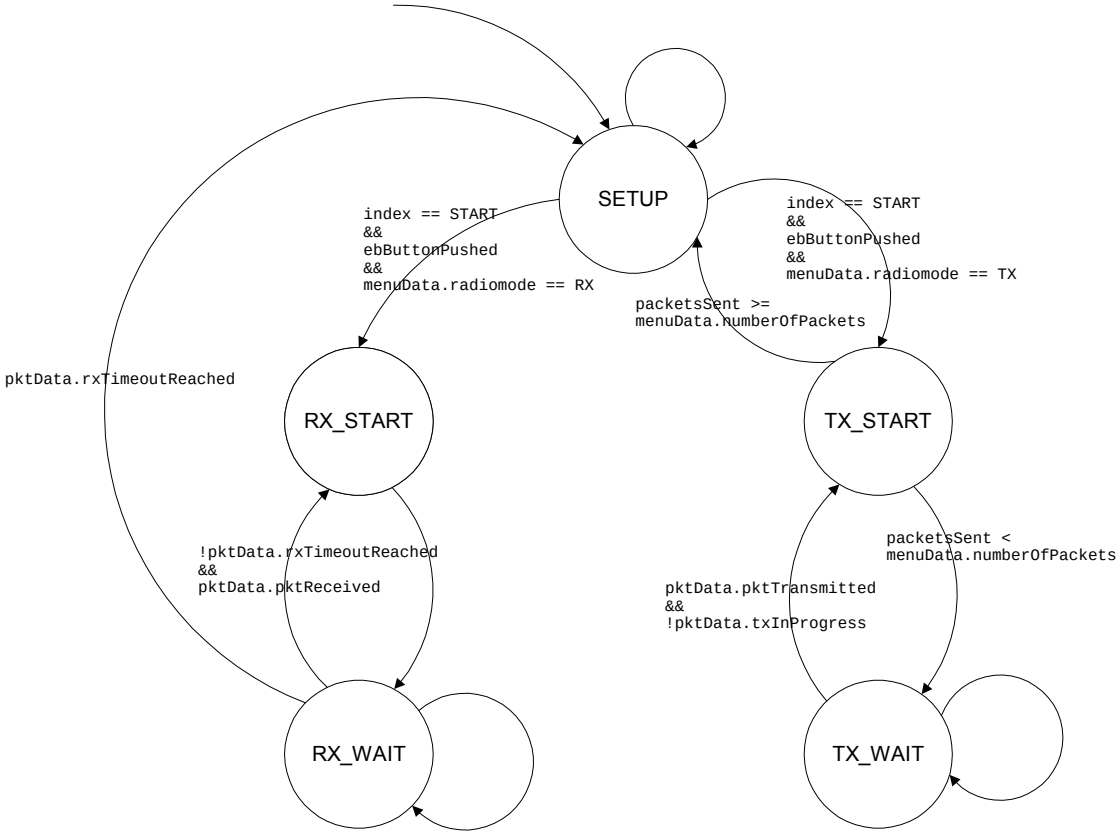


Figure 1. Main loop state diagram (Link1)

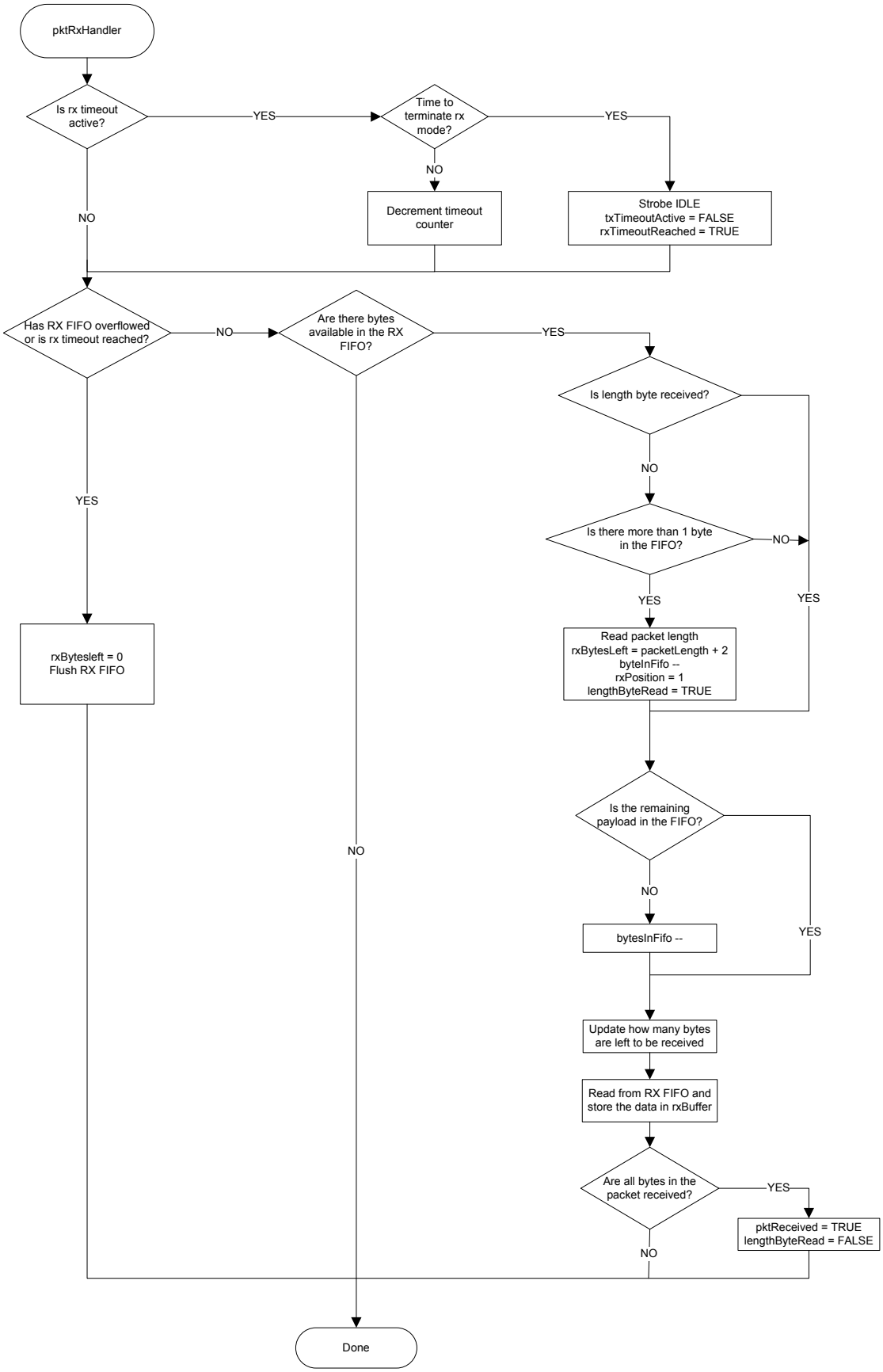


Figure 2. Flow chart for pktRxHandler (Link1)

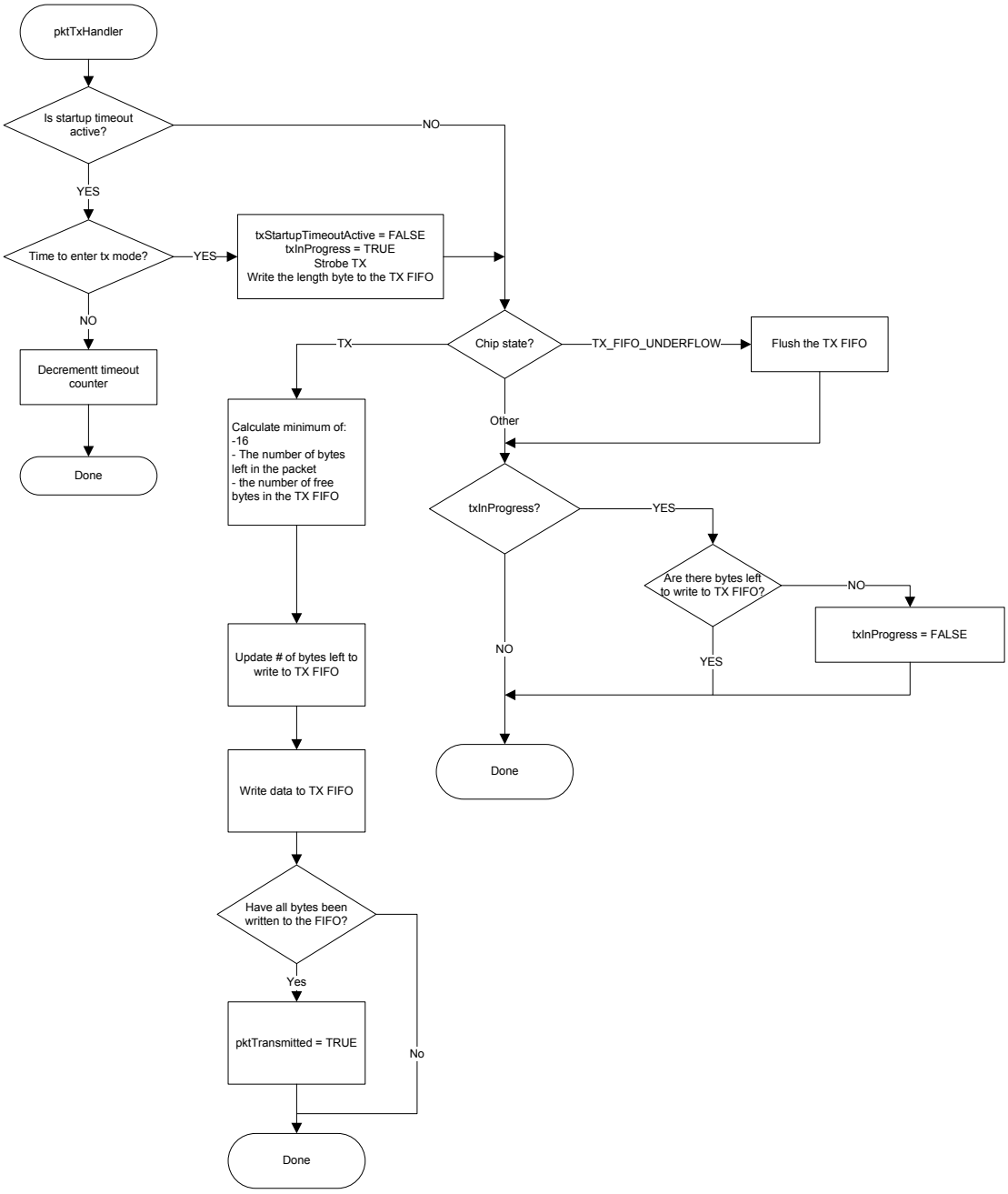


Figure 3. Flow chart for pktTxHandler (Link1)

6.3 SerialLink

This program demonstrates how to set up a simple RF link between two units using serial synchronous mode.

By moving the joystick right or left, the user can set up one unit as transmitter (left) and one unit as receiver (right). After selecting correct mode, the joystick button should be pushed. Now the transmitter will send one packet every time the S1 button is being pushed. The number of packets transmitted and received is displayed on the LCD display on the TX and RX unit, respectively. On the receiver, the CRC is checked before the display is updated (i.e. received packets containing bit errors are not counted).

6.4 Link2

This program demonstrates how it is possible to transmit and receive packets that are longer than the size of the FIFO (64 bytes) without doing any SPI polling of the status registers (see the CC1100/CC1150 and the CC2500/CC2550 Errata Notes). Packet transmission and packet reception is implemented using two external interrupts. The joystick is used to navigate through a menu, setting different parameters.

Parameter	Settings
Packet Length	10, 30, 50, ……., 230, 250
Number of Packets	100, 200, 300, ……, 900, 1000
Radio Mode	Rx, Tx

The following steps must be done to start the link test:

Rx Unit:

- Set radio mode to RX.
- Move joystick down until the message "Press S1 to start" is showed on the LCD display
- Press S1

The LCD display will show number of packets received with CRC OK.

Tx Unit:

- Set packet length and number of packets to transmit
- Set radio mode to TX
- Press S1 to Start

The LCD will show number of packets transmitted. After all the packets have been transmitted, S1 can be pressed to run the test once more or the joystick can be used to change packet length and number of packets before running a new test.

The main loop is implemented as a state machine and the state diagram is showed in Figure 4.

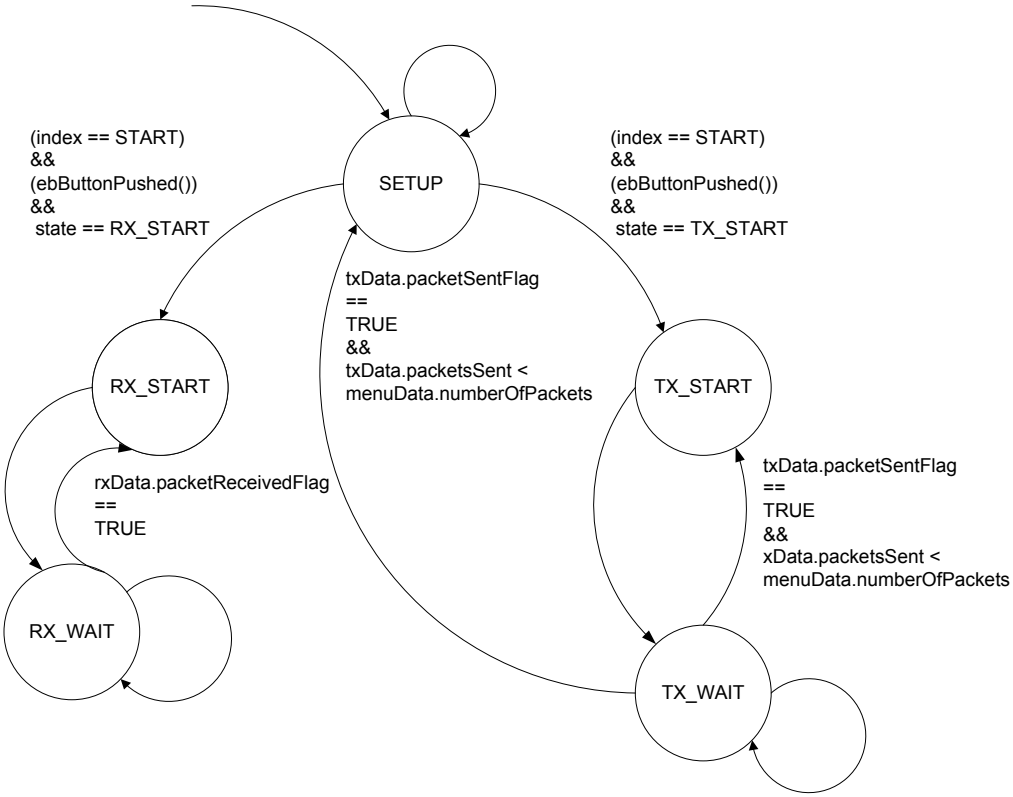


Figure 4. Main loop state diagram (Link2)

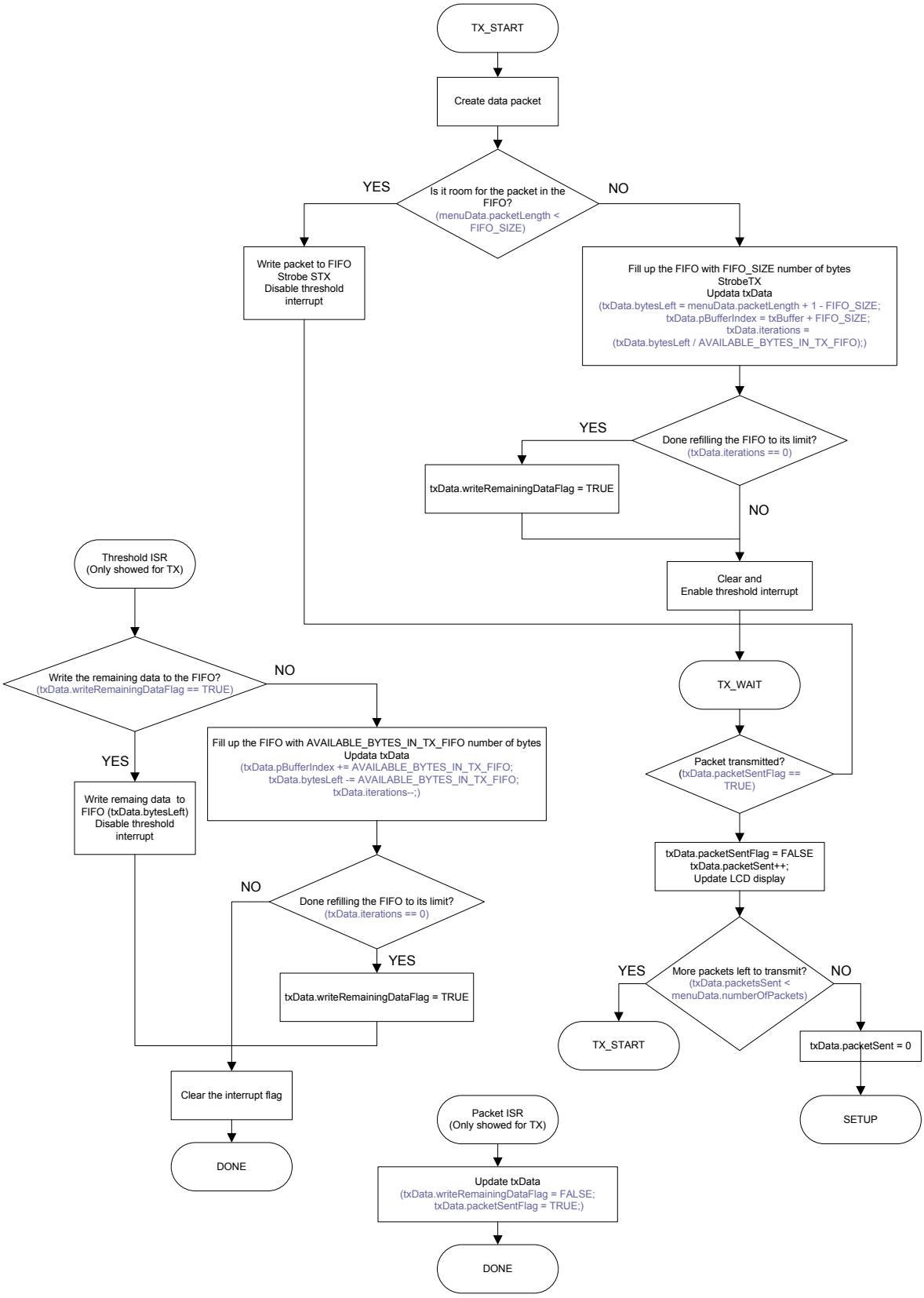


Figure 5. Flowchart for TX (Link2)

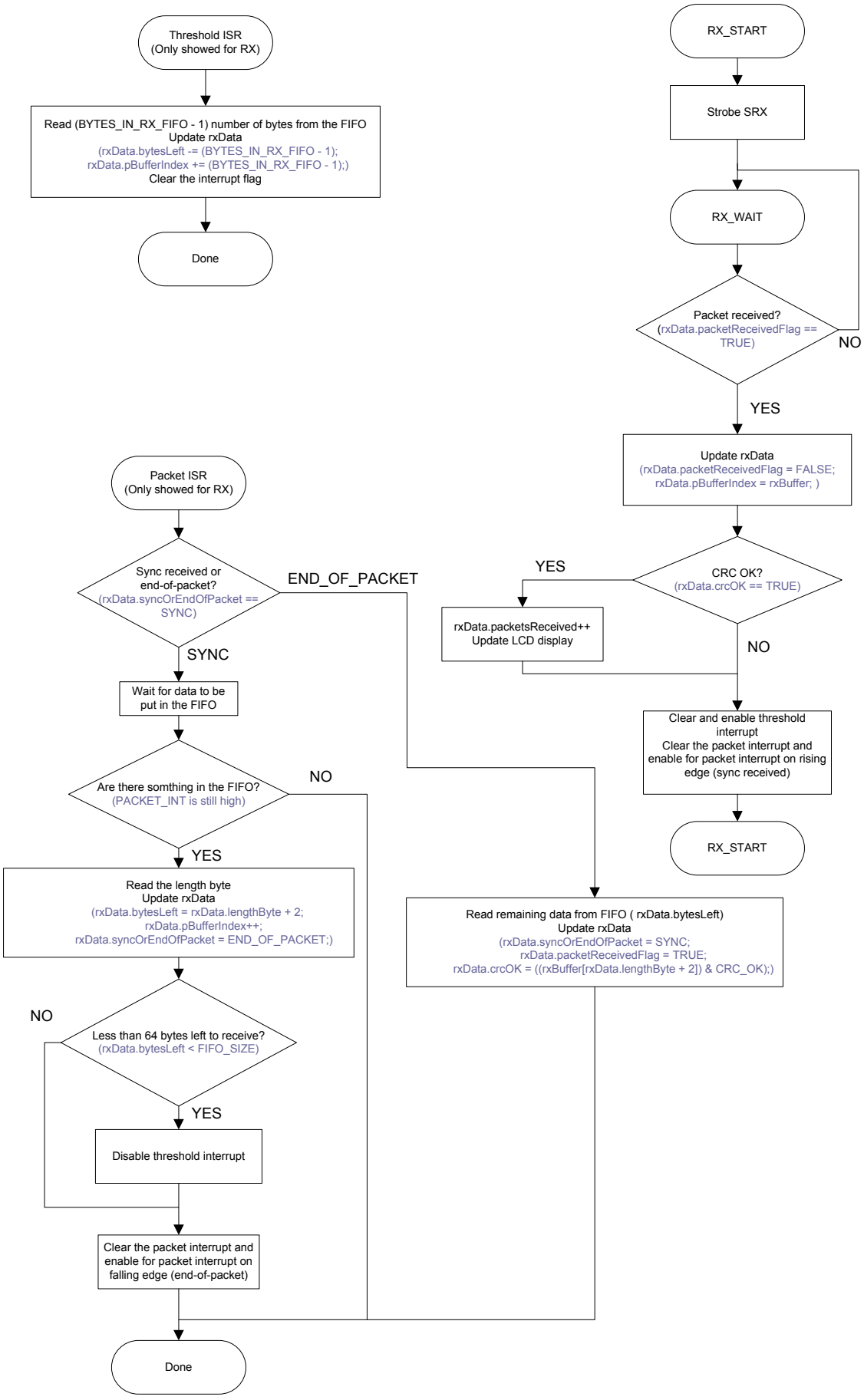


Figure 6. Flowchart for RX (Link2)

Example to demonstrate the program flow:

Both GDO0 and GDO2 are connected to inputs pins on the MCU configured to generate external interrupts. The interrupt related to the GDO2 pin is referred to as the threshold interrupt, while the other interrupt is referred to as the packet interrupt. In TX, the MCU is configured to give an interrupt on falling edges for both interrupts, while in RX, there will be interrupt on both rising and falling edge of GDO0 and on rising edge on GDO2.

Threshold interrupt

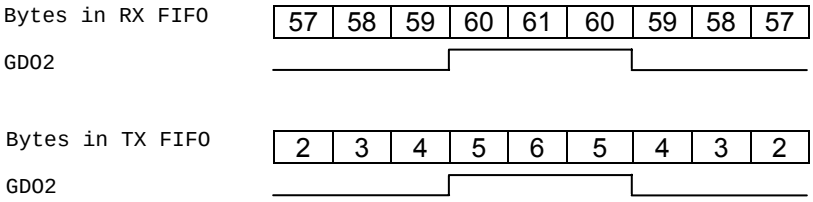
FIFO_THR = 14: 5 bytes in the TX FIFO and 60 bytes in the RX FIFO.

RX mode

IOCFG2 = 0x00: Associated to the RX FIFO: Asserts when RX FIFO is filled above RXFIFO_THR. De-asserts when RX FIFO is drained below RXFIFO_THR. In RX mode there will be an interrupt on the rising edge (BYTES_IN_RX_FIFO = 60)

TX mode

IOCFG2 = 0x02: Associated to the TX FIFO: Asserts when the TX FIFO is filled above TXFIFO_THR. De-asserts when the TX FIFO is below TXFIFO_THR. In TX mode there will be an interrupt on the falling edge (AVAILABLE_BYTES_IN_TX_FIFO = 60)



Packet interrupt

IOCFG0 = 0x06: Asserts when sync word has been sent / received, and de-asserts at the end of the packet. In RX, the pin will de-assert when the optional address check fails or the RX FIFO overflows. In TX the pin will de-assert if the TX FIFO underflows.

Assume a packet with packet length 150 (menuData.packetLength = 150). In this case, 151 bytes should be written to the TX FIFO.

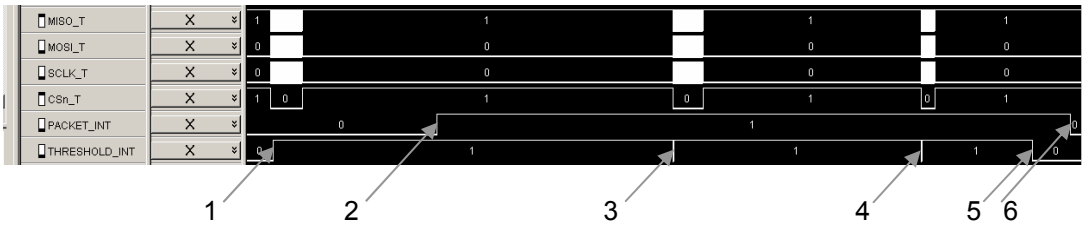


Figure 7. Writing to TX FIFO (Link2)

1: Start by writing 64 bytes to the TX FIFO and strobe STX. After writing byte number 5 to the TX FIFO, GDO2 is asserted (No interrupt on rising edge).

txData.bytesLeft = menuData.packetLength + 1 - FIFO_SIZE = 87 bytes left to write
txData.iterations = (txData.bytesLeft / AVAILABLE_BYTES_IN_TX_FIFO)
= 87 / 60 = 1 (number of times one can fill the TX FIFO all the way up)
txData.writeRemainingDataFlag = FALSE

2: Sync word has been transmitted (No interrupt on rising edge)

3: Threshold interrupt on falling edge. Write 60 bytes to the TX FIFO.

txData.bytesLeft -= AVAILABLE_BYTES_IN_TX_FIFO = 87 – 60
= 27 bytes left to write
txData.iterations = 0, which means that one should not write 60 bytes to the TX FIFO on the next threshold interrupt (only 27 bytes are left to be written)
txData.writeRemainingDataFlag = TRUE;

4: Threshold interrupt on falling edge. Write remaining 27 bytes to the TX FIFO and disable threshold interrupt.

5: No interrupt since interrupt has been disabled.

6: Packet interrupt on falling edge indicating that the packet has been sent.

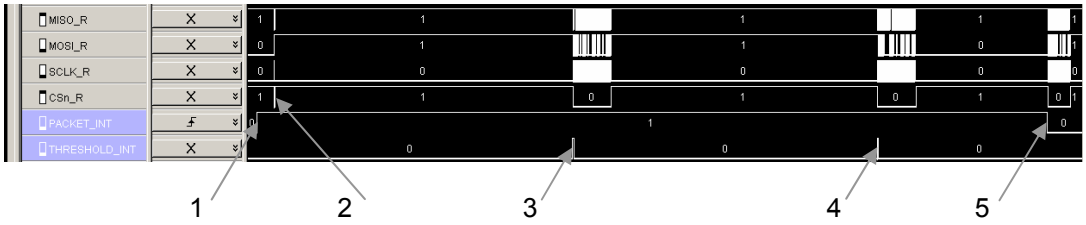


Figure 8. Reading from RX FIFO (Link2)

1: Packet interrupt on rising edge (sync received). Wait for 2 bytes to be put in the RX FIFO.

2: Read the length byte.
rxData.bytesLeft = rxData.lengthByte + 2 status bytes = 150 + 2 = 152 bytes left to read
Enable for interrupt on falling edge (packet received).

3: Threshold interrupt on rising edge (60 or more bytes in the RX FIFO).
Read 59 bytes from the RX FIFO (the RX FIFO should not be emptied)
rxData.bytesLeft -= (BYTES_IN_RX_FIFO - 1) = 152 – 59 = 93 bytes left to read

4: Threshold interrupt on rising edge (60 or more bytes in the RX FIFO).
Read 59 bytes from the RX FIFO (the RX FIFO should not be emptied)
rxData.bytesLeft -= (BYTES_IN_RX_FIFO - 1) = 93 – 59 = 34 bytes left to read

5: Packet interrupt on falling edge (packet received).
Read the remaining bytes from the RX FIFO.

6.5 InfiniteLink

This program demonstrates how it is possible to transmit and receive packets that are longer than 256 bytes. The example does not use any SPI polling of the status registers (see the CC1100/CC1150 and the CC2500/CC2550 Errata Notes). Packet transmission and packet reception is implemented using two external interrupts. The joystick is used to navigate through a menu, setting different parameters.

Parameter	Settings
Packet Length	270, 290, ……., 430, 450
Number of Packets	100, 200, 300, ……, 900, 1000
Radio Mode	Rx, Tx

The following steps must be done to start the link test:

Rx Unit:

- Set radio mode to RX.
- Move joystick down until the message "Press S1 to start" is showed on the LCD display
- Press S1

The LCD display will show number of packets received with CRC OK.

Tx Unit:

- Set packet length and number of packets to transmit
- Set radio mode to TX
- Press S1 to Start

The LCD will show number of packets transmitted. After all the packets have been transmitted, S1 can be pressed to run the test once more or the joystick can be used to change packet length and number of packets before running a new test.

The main loop is implemented as a state machine and the state diagram is showed in Figure 9.

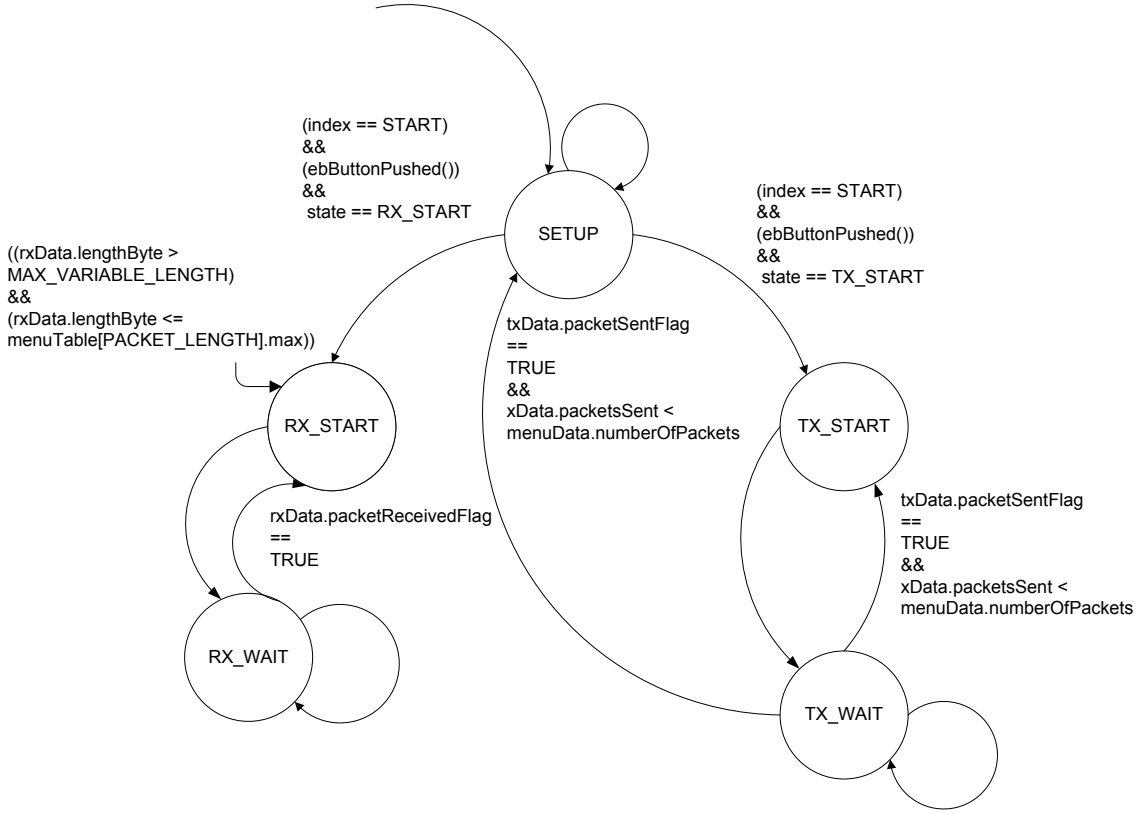


Figure 9. Main loop state diagram (InfiniteLink)

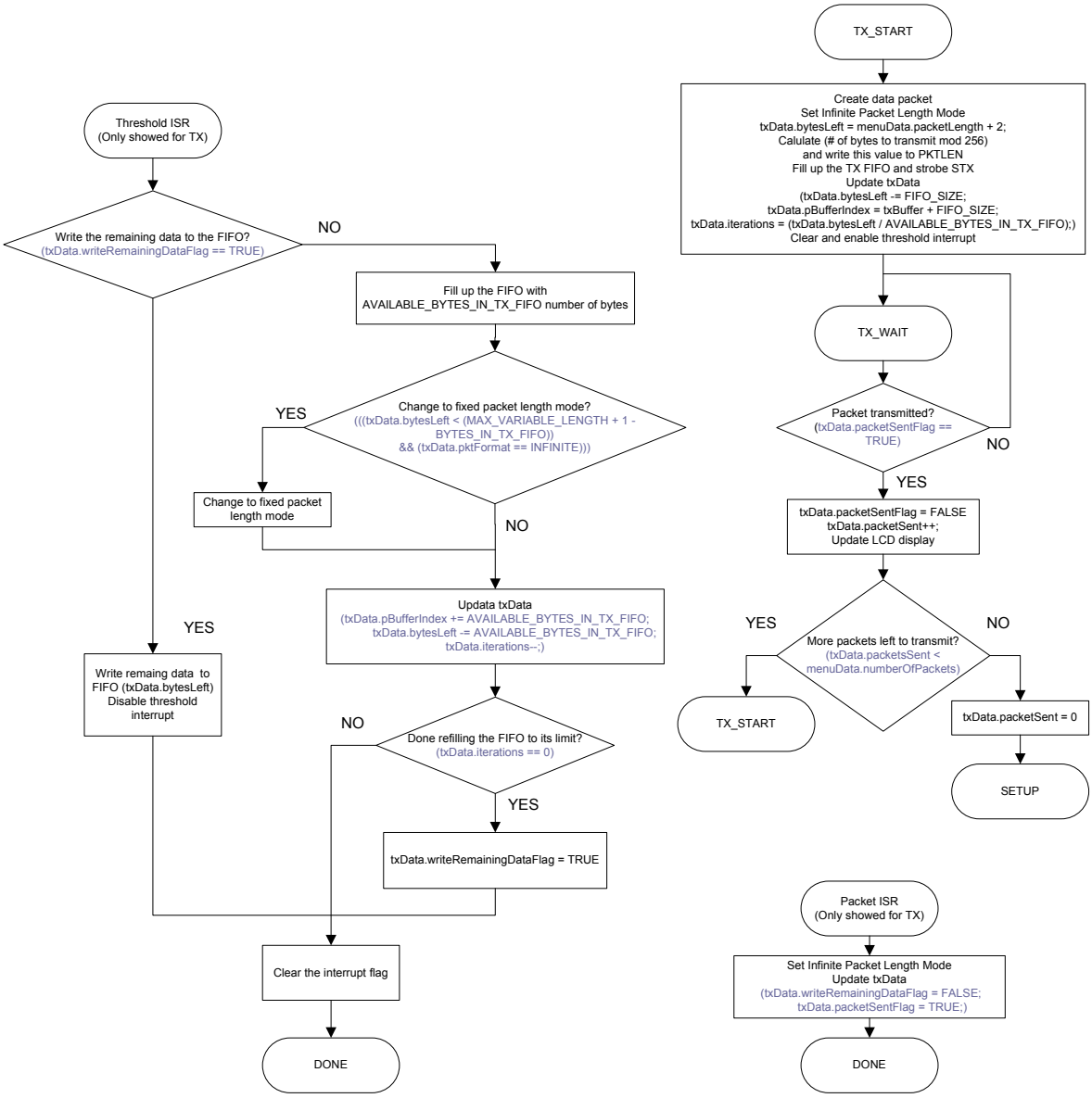


Figure 10. Flowchart for TX (InfiniteLink)

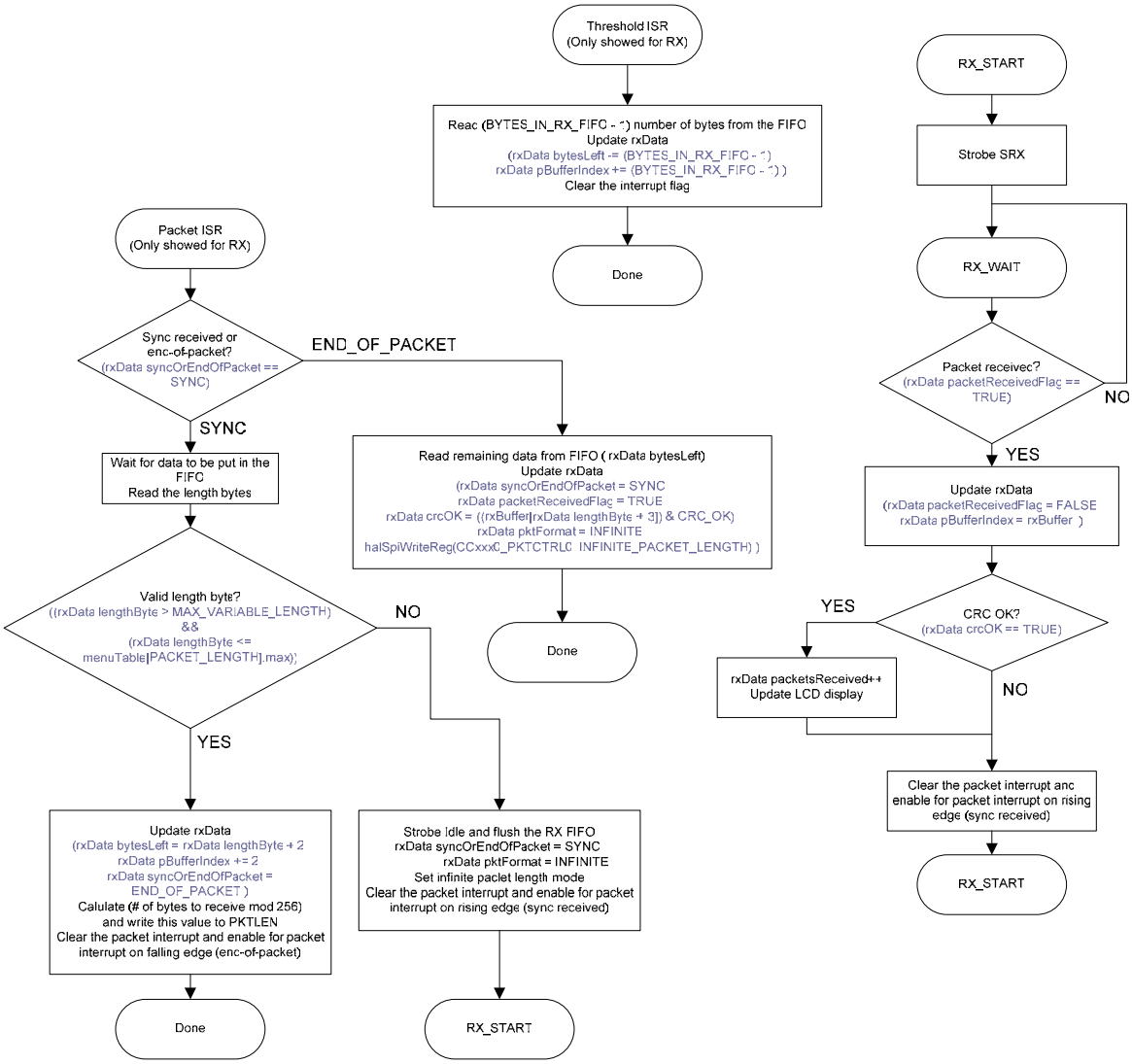


Figure 11. Flowchart for RX (InfiniteLink)

Example to demonstrate the program flow:

Please see the Link2 example above to understand how the packet interrupt (IOCFG0 = 0x06) and the threshold interrupt (IOCFG2 = 0x00 (RX) and IOCFG2 = 0x02 (TX)) are used when writing to the TX FIFO or reading from the RX FIFO. The threshold is the same as in that example.

Assume a packet with packet length 450 (menuData.packetLength = 450). In this case, 452 bytes should be written to the TX FIFO (2 bytes are needed for the length).

Set `PKTCTRL0.LENGTH_CONFIG` = 2 (10).

Pre-program the `PKTLEN` register to mod(452,256) = 196.

Transmit at least 197 bytes (less than 256 bytes left to transmit)

Set `PKTCTRL0.LENGTH_CONFIG` = 0 (00).

The transmission ends when the packet counter reaches 196. A total of 452 bytes are transmitted.

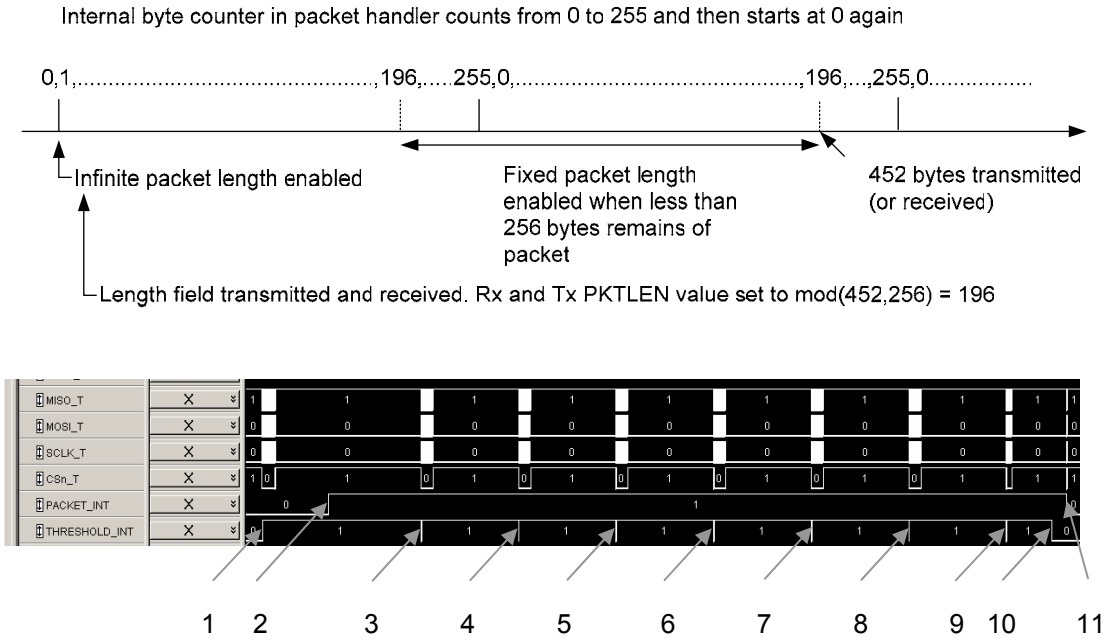


Figure 12. Writing to TX FIFO (InfiniteLink)

1: LENGTH_CONFIG = 2 (Infinite packet length)
txData.bytesLeft = menuData.packetLength + 2 length bytes = 450 + 2 = 452
fixedPacketLength = txData.bytesLeft % (MAX_VARIABLE_LENGTH + 1)
= mod(452, 256) = 196
Start by writing 64 bytes to the TX FIFO and strobe STX. After writing byte number 5 to the TX FIFO, GDO2 is asserted (No interrupt on rising edge).

txData.bytesLeft -= FIFO_SIZE = 452 – 64 = 388
txData.iterations = (txData.bytesLeft / AVAILABLE_BYTES_IN_TX_FIFO)
= 388 / 60 = 6 (number of times one can fill the TX FIFO all the way up)
Set PKTLEN = mod(452, 256) = 196

2: Sync word has been transmitted (No interrupt on rising edge)

3: LENGTH_CONFIG = 2 (Infinite packet length)
Threshold interrupt on falling edge. Write 60 bytes to the TX FIFO.

txData.bytesLeft -= AVAILABLE_BYTES_IN_TX_FIFO = 388 – 60
= 328 bytes left to write
txData.iterations = 5

Check if there is less than 256 bytes left to transmit and if Infinite packet length mode is set:
(bytes left to transmit is the bytes left to write to the TX FIFO (txData.bytesLeft) +
the bytes that are in the TX FIFO when this interrupt occurs (BYTES_IN_TX_FIFO))

((txData.bytesLeft < (MAX_VARIABLE_LENGTH + 1 - BYTES_IN_TX_FIFO))
&&
(txData.pktFormat == INFINITE)) ?

((328 < (255 + 1 - 4))
&&
(txData.pktFormat == INFINITE)) ? NO

4: LENGTH_CONFIG = 2 (Infinite packet length)

Threshold interrupt on falling edge. Write 60 bytes to the TX FIFO.

txData.bytesLeft -= AVAILABLE_BYTES_IN_TX_FIFO = 328 – 60
= 268 bytes left to write
txData.iterations = 4

Check if there is less than 256 bytes left to transmit and if Infinite packet length mode is set:

((268 < (255 + 1 - 4))
&&
(txData.pktFormat == INFINITE)) ? NO

5: LENGTH_CONFIG = 2 (Infinite packet length)

Threshold interrupt on falling edge. Write 60 bytes to the TX FIFO.

txData.bytesLeft -= AVAILABLE_BYTES_IN_TX_FIFO = 268 – 60
= 208 bytes left to write
txData.iterations = 3

Check if there is less than 256 bytes left to transmit and if Infinite packet length mode is set:

((208 < (255 + 1 - 4))
&&
(txData.pktFormat == INFINITE)) ? YES → LENGTH_CONFIG = 0 (Fixed packet length)

6: LENGTH_CONFIG = 0 (Fixed packet length)

Threshold interrupt on falling edge. Write 60 bytes to the TX FIFO.

txData.bytesLeft -= AVAILABLE_BYTES_IN_TX_FIFO = 208 – 60
= 148 bytes left to write
txData.iterations = 2

Check if there is less than 256 bytes left to transmit and if Infinite packet length mode is set:

((148 < (255 + 1 - 4))
&&
(txData.pktFormat == INFINITE)) ? NO

7,8: 60 bytes are written to the TX FIFO each time
txData.bytesLeft = 28 bytes left to write
txData.iterations = 0, which means that one should not write 60 bytes to the TX FIFO on the next threshold interrupt (only 28 bytes are left to be written)
txData.writeRemainingDataFlag = TRUE;

9: Threshold interrupt on falling edge. Write remaining 28 bytes to the TX FIFO and disable threshold interrupt.

10: No interrupt since interrupt has been disabled.

11: Packet interrupt on falling edge indicating that the packet has been sent.

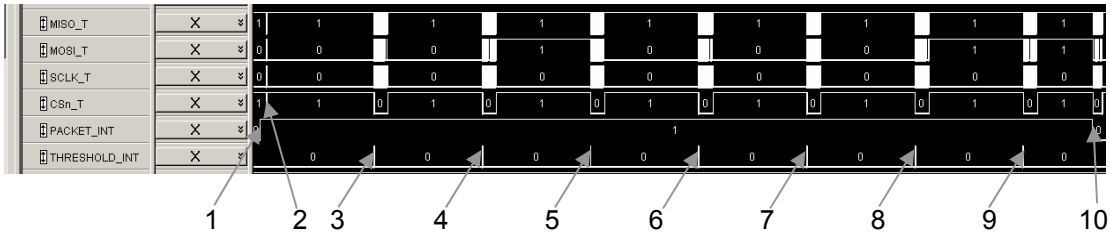


Figure 13. Reading from RX FIFO (InfiniteLink)

1: Packet interrupt on rising edge (sync received). Wait for 3 bytes to be put in the RX FIFO.

2: Read the length bytes (2 bytes).
rxData.bytesLeft = rxData.lengthByte + 2 status bytes = 450 + 2 = 452 bytes left to read

fixedPacketLength = txData.bytesLeft % (MAX_VARIABLE_LENGTH + 1)
= mod(452, 256) = 196

Set PKTLEN = mod(452, 256) = 196

Enable for interrupt on falling edge (packet received).

3: LENGTH_CONFIG = 2 (Infinite packet length)
Threshold interrupt on rising edge (60 or more bytes in the RX FIFO).

Check if there is less than 256 bytes left to receive and if Infinite packet length mode is set:
(bytes left to receive is the bytes left to read from the RX FIFO (txData.bytesLeft) -
the bytes that are in the RX FIFO when this interrupt occurs (BYTES_IN_RX_FIFO))

(((rxData.bytesLeft - BYTES_IN_RX_FIFO) < (MAX_VARIABLE_LENGTH + 1))
&&
(rxData.pktFormat == INFINITE)) ?

(((452 - 60) < (255 + 1))
&&
(rxData.pktFormat == INFINITE)) ? NO

Read 59 bytes from the RX FIFO (the RX FIFO should not be emptied)
rxData.bytesLeft -= (BYTES_IN_RX_FIFO - 1) = 452 – 59 = 393 bytes left to read

4: LENGTH_CONFIG = 2 (Infinite packet length)
Threshold interrupt on rising edge (60 or more bytes in the RX FIFO).

Check if there is less than 256 bytes left to receive and if Infinite packet length mode is set:

(((393 - 60) < (255 + 1))
&&
(rxData.pktFormat == INFINITE)) ? NO

Read 59 bytes from the RX FIFO (the RX FIFO should not be emptied)
rxData.bytesLeft -= (BYTES_IN_RX_FIFO - 1) = 393 – 59 = 334 bytes left to read

5: LENGTH_CONFIG = 2 (Infinite packet length)
Threshold interrupt on rising edge (60 or more bytes in the RX FIFO).

Check if there is less than 256 bytes left to receive and if Infinite packet length mode is set:

(((334 - 60) < (255 + 1))
&&
(rxData.pktFormat == INFINITE)) ? NO

Read 59 bytes from the RX FIFO (the RX FIFO should not be emptied)
rxData.bytesLeft -= (BYTES_IN_RX_FIFO - 1) = 334 – 59 = 275 bytes left to read

6: LENGTH_CONFIG = 2 (Infinite packet length)
Threshold interrupt on rising edge (60 or more bytes in the RX FIFO).

Check if there is less than 256 bytes left to receive and if Infinite packet length mode is set:

(((275 - 60) < (255 + 1))
&&
(rxData.pktFormat == INFINITE)) ? YES → LENGTH_CONFIG = 0 (Fixed packet length)

Read 59 bytes from the RX FIFO (the RX FIFO should not be emptied)
rxData.bytesLeft -= (BYTES_IN_RX_FIFO - 1) = 275 – 59 = 216 bytes left to read

7: LENGTH_CONFIG = 0 (Fixed packet length)
Threshold interrupt on rising edge (60 or more bytes in the RX FIFO).

Check if there is less than 256 bytes left to receive and if Infinite packet length mode is set:

(((216 - 60) < (255 + 1))
&&
(rxData.pktFormat == INFINITE)) ? NO

Read 59 bytes from the RX FIFO (the RX FIFO should not be emptied)
rxData.bytesLeft -= (BYTES_IN_RX_FIFO - 1) = 216 – 59 = 157 bytes left to read

8,9: 59 bytes are read from the RX FIFO on each interrupt.
rxData.bytesLeft = 39

10: Packet interrupt on falling edge (packet received). Read the remaining bytes from the RX FIFO.

7. Libraries

Chipcon supplies several libraries to make it as easy as possible to develop custom software on the SmartRF®04DK platform. The libraries are divided into 2 main groups: the files concerning the Evaluation Board (EB), and the Hardware Abstraction Library (HAL).

The EB files consists of header files and functions that enable you to easily access the circuitry on the SmartRF® 04EB board. This includes reading the joystick direction, writing to the LCD display and so on. It also contains register definitions for both the USB MCU and the CCxx00/CCxx50 radio.

The HAL consists of header files and functions to access the different peripherals of the USB MCU. It also contains functions to access registers on the CCxx00/CCxx50 and functions for transmitting and receiving packets.

7.1 Library Structure

The libraries are structured in the file structure shown in Figure 14.

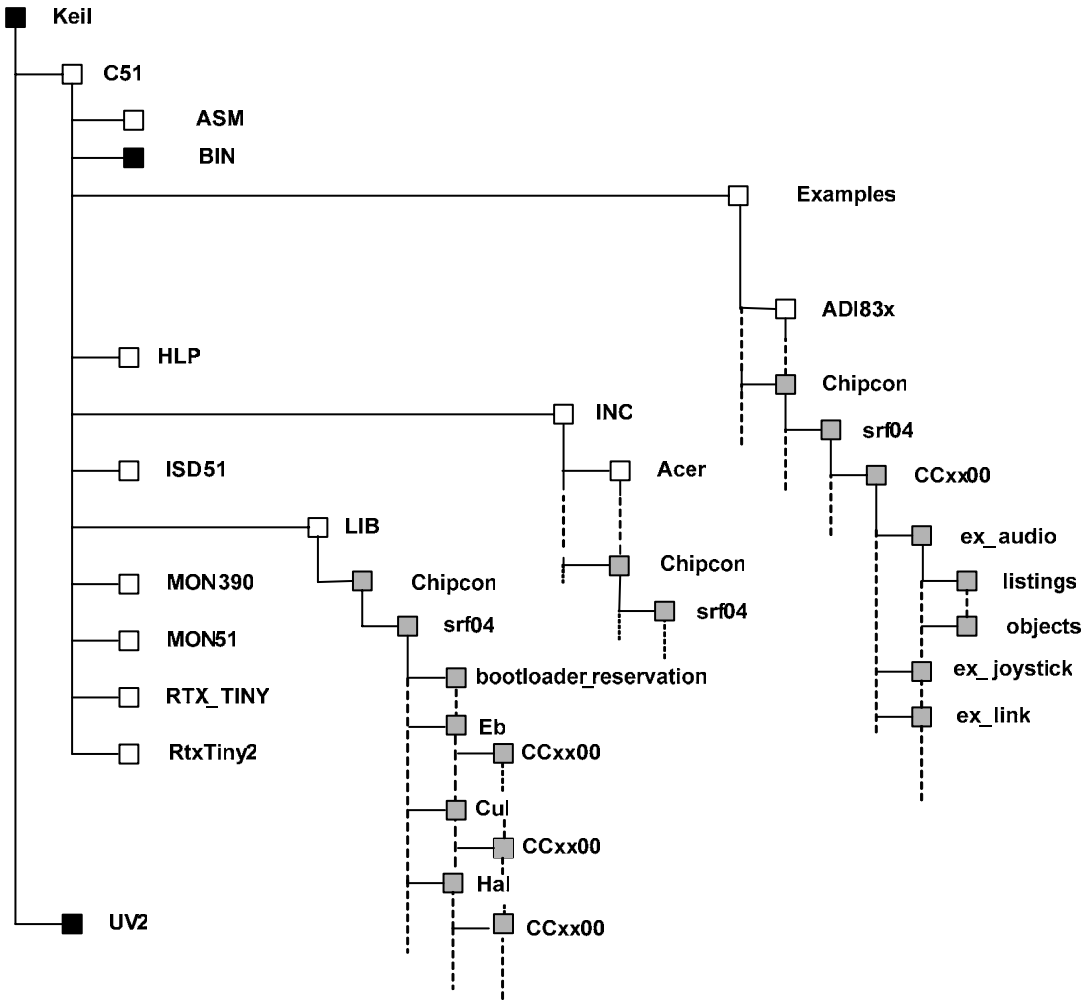


Figure 14. Library file structure

In the ex_audio folder, the audio.c and audio.Uv2 files are found, together with the audio_bootloader.Uv2 and the STARTUP.A51. The audio.hex and audio_bootloader.hex files are found in the objects folder under ex_audio. audio.hex is the .hex file that is created when building the audio.Uv2 project and is a stand-alone application. The audio_bootloader.hex file is the hex file that should be used if SmartRF® Studio is used to download a .hex file into the USB MCU. The table below shows what the other folders in the library contain.

Folder	File
..\NC\Chipcon\srf04	common.h
	ebsrf04.h
	halsrf04.h
	regssrf04.h
	culsrf04.h
	app_descriptor.h
	app_descriptor.a51
	bl_structs.h
..\LIB\Chipcon\srf04	ebsrf04.LIB
	halsrf04.LIB
	culsrf04.LIB
	ebsrf04_bootloader.LIB
	halsrf04_bootloader.LIB
	culsrf04_bootloader.LIB
..\LIB\Chipcon\srf04\Eb\CCxx00	AdcInit.c
	ButtonPushed.c
	GetJoystickPosition.c
	JoyPushed.c
	Lcd.c
	ReadPotentiometer.c
..\LIB\Chipcon\srf04\Hal\CCxx00	RfReceivePacket.c
	RfSendPacket.c
	RfWriteRfSettings.c
	SetupTimer01.c
	SetupTimer23.c
	SpiReadBurstReg.c
	SpiReadReg.c
	SpiReadStatus.c
	SpiStrobe.c
	SpiWriteReg.c
	SpiWriteBurstReg.c
	UartSetup.c
	Wait.c
	RfReceivePacketSerial.c
	RfSendPacketSerial.c
	RfReceivePacketLockDetect.c
	RfSendPacketLockDetect.c
..\LIB\Chipcon\srf04\bootloader_reservation	Bootloader_reservations.c
..\LIB\Chipcon\srf04\Cul\CCxx00	CalcCRC.c
	SyncSearch.c

Table 1. Contents of library directories

7.2 EB Library Reference

Table 3 is showing all the functions and macros found in the EB library. For more details on how to use these functions/macros, please see the ebsrf04.h file, found in the ..\INC\Chipcon\srf04 folder.

Functions	Description
BOOL ebButtonPushed(void)	This function detects if the S1 button is being pushed.
BOOL ebJoyPushed(void)	This function detects if the joystick button is being pushed.
void ebAdcInit(UINT8 adcInput)	Function used to initialize the ADC.
UINT8 ebGetJoystickPosition(void)	This function will read the ADC to determine the current joystick position.
UINT8 ebReadPotentiometer(void)	This function reads the potmeter located at the SmartRF04EB using the ADC. The function only reads the 8 MSBs from the ADC.
void ebLcdInit(void)	Function used to initialize the LCD display.
ebLcdUpdate (UINT8 *pLine1, UINT8 *pLine2)	This function takes two ASCII strings (max 16 characters each) and outputs them on the LCD display.
Macros	**Description**
IO_PORT_INIT()	Macro to set up the USB MCU crossbar and I/O ports to communicate with the SmartRF®04EB peripherals
SET_GLED(x)	Macros to turn the 4 LEDs on the SmartRF®04EB on and off
SET_RLED(x)	
SET_YLED(x)	
SET_BLED(x)	
BUTTON_PUSHED()	Macros used to check if the push button (S1) or joystick button is pushed.
JOY_PUSHED()	
RS_232_FORCE_ON()	Macro for turning on/off the RS-232 on-board power supply
RS_232_FORCE_OFF()	
HARDWARE_FLOW_CONTROL_ENABLE()	Enable Hardware Flow Control. CTS is set as an output. It is no longer possible to use the joystick push button
HARDWARE_FLOW_CONTROL_DISABLE()	Disable Hardware Flow Control. CTS is set as an input. It is now possible to use the joystick push button (it shares the same pin as CTS)
UART_CTS_FLOW_ENABLE()	Set/Clear CTS (Clear to Send)
UART_CTS_FLOW_DISABLE()	

Table 2. EB functions and macros

7.3 HAL Library Reference

Table 3 is showing all the functions and macros found in the HAL library. For more details on how to use these functions/macros, please see the halsrf04.h file, found in the ..\INC\Chipcon\srf04 folder.

Functions	Description
void halUartSetup (UINT16 baudRate, UINT8 options)	Function which implements all the initialization necessary to establish a simple serial link.
void halSpiStrobe(BYTE strobe)	Function for writing a strobe command to the CCxx00/CCxx50.
BYTE halSpiReadStatus(BYTE addr)	Function for reading a CCxx00/CCxx50 status register.
void halSpiWriteReg (BYTE addr, BYTE value)	Function for writing to a single CCxx00/CCxx50 register
BYTE halSpiReadReg(BYTE addr)	Function for reading a single CCxx00/CCxx50

	register.
void halSpiWriteBurstReg (BYTE addr, BYTE *buffer, BYTE count)	Function for writing to multiple CCxx00/CCxx50 register, using SPI burst access.
void halSpiReadBurstReg (BYTE addr, BYTE *buffer, BYTE count)	Function for reading multiple CCxx00/CCxx50 register, using SPI burst access
void halSetupTimer01 (UINT8 timer01, UINT8 clkSource, UINT8 mode, BOOL timerInt)	Function for initializing timer 0 or timer 1. This function only supports mode 0, 1, and 2.
void halSetupTimer23 (UINT8 timerOption, clkSourceH, UINT8 clkSourceL, UINT8 mode, BOOL timerInt)	Function for initializing timer 2 or timer 3. This function only supports mode 0 and mode 1.
void RfWriteRfSettings (RF_SETTINGS *pRfSettings)	This function is used to configure the CCxx00/CCxx50 based on a given RF setting
void halRfSendPacket (BYTE *txBuffer, UINT8 size)	This function can be used to transmit a packet with packet length up to 63 bytes. The function implements polling of GDO0.
BOOL halRfReceivePacket (BYTE *rxBuffer, UINT8 *length)	This function can be used to receive a packet of variable packet length (first byte in the packet must be the length byte). The packet length should not exceed the RX FIFO size. The function implements polling of GDO0.
void halWait(UINT16 timeout)	Runs an idle loop for [timeout] microseconds.
void halRfSendPacketSerial (BYTE *txBuffer, UINT8 size, UINT8 startOfPayload, BOOL crcEnable)	This function can be used to send a packet using synchronous serial mode. Length byte and CRC is optional. 4 sync bytes must be used
BOOL halRfReceivePacketSerial (BYTE *rxBuffer, UINT8 sync3, UINT8 sync2, UINT8 sync1, UINT8 sync0, UINT8 fixedLength, BOOL crcEnable)	This function can be used to receive a packet using synchronous serial mode. Length byte and CRC is optional. 4 sync bytes must be used
Macros	**Description**
ENABLE_GLOBAL_INT(on)	Macros used to enable/disable global interrupts.
INT_ENABLE(inum, on)	Macro used together with the INUM_* constants defined in regssrf04.h to enable or disable certain interrupts.
INT_PRIORITY(inum, p)	Macro used together with the INUM_* constants defined in regsrf04.h to set the priority of certain interrupts.
INT_GETFLAG(inum)	Macro used together with the INUM_* constants defined in regsrf04.h to read the interrupt flags.
INT_SETFLAG(inum, f)	Macro used together with the INUM_* constants defined in regsrf04.h to set or clear certain interrupt flags.
SETUP_GDO0_INT(trigger, polarity)	This macro is setting up the GDO0 interrupt from CCxx00. The interrupt is on P0.6 and is assign to external interrupt0. The macro enables external interrupt0.
SETUP_GDO2_INT(trigger, polarity)	This macro is setting up the GDO2 interrupt from CCxx00. The interrupt is on P0.7 and is assigned to external interrupt1. The macro enables external interrupt1.
UART_TX_ENABLE()	Macros which are helpful when transmitting and receiving data over the serial interface.
UART_RX_ENABLE()	
UART_TX_WAIT()	
UART_RX_WAIT()	
UART_TX(x)	

UART_RX(x)	
UART_WAIT_AND_SEND(x)	
UART_WAIT_AND_RECEIVE(x)	
SPI_ENABLE()	Macros used to enable/disable the SPI
SPI_DISABLE()	
SPI_INIT(freq)	Enble SPI (4-wire Single Master Mode, data centered on first edge of SCK period. SCK is low in the Idle State)
SPI_WAIT()	Macro used for communication data polling and wait on the SPI bus.
RESET_CCxxx0()	Macro to reset the CCxxx0 and wait for it to be ready.
POWER_UP_RESET_CCxxx0()	Macro to reset the CCxxx0 after power_on and wait for it to be ready.
TIMER0_RUN(x)	Macros for stopping and starting the timers.
TIMER1_RUN(x)	
TIMER2_RUN(x)	
TIMER3_RUN(x)	
SET_RELOAD_VALUE_TIMER0 (period_us, clock_kHz)	Macros used to calculate the reload value and update the reload registers.
SET_RELOAD_VALUE_TIMER1 (period_us, clock_kHz)	
SET_RELOAD_VALUE_TIMER2_8BIT (periodH_us, periodL_us, clock_kHzH, clock_kHzL)	Macros used to calculate the reload value and update the reload registers.
SET_RELOAD_VALUE_TIMER3_8BIT (periodH_us, periodL_us, clock_kHzH, clock_kHzL)	
SET_RELOAD_VALUE_TIMER2_16BIT (period_us, clock_kHz)	Macros used to calculate the reload value and update the reload registers.
SET_RELOAD_VALUE_TIMER3_16BIT (period_us, clock_kHz)	
ADC_ENABLE()	Macros used to enable/disable the ADC.
ADC_DISABLE()	
ADC_SAMPLE()	This macro clears the ADC0 Conversion Complete Interrupt Flag, initiates ADC0 conversion and waits for the conversion to complete
CLOCK_INIT()	This section contains a macro for initializing the internal oscillator, the system clock and the 4x Clock Multiplier
CLOCK_INIT()	Select the Internal Oscillator as Multiplier input source and disable the watchdog timer SYSCLK = 4X Clock Multiplier / 2

Table 3. HAL functions and macros

7.4 CUL Library Reference

Table 4 is showing all the functions and macros found in the CUL library. For more details on how to use these functions/macros, please see the culsrf04.h file, found in the ..\INC\Chipcon\srf04 folder.

Functions	Description
UINT16 culCalcCRC (BYTE crcData, UINT16 crcReg)	A CRC-16/CCITT implementation.
void culSyncSearch (UINT8 sync3, UINT8 sync2, UINT8 sync1, UINT8 sync0)	Function for searching for a 4 bytes sync word.

Table 4. CUL functions and macros

8. Document history

Revision	Date	Description/Changes
1.3	2007-01-12	Cosmetic changes. Removed WOR examples as they are not up-to-date with AN047. Removed the FEC option in the Link1 example as this option has been removed from the code example.
1.2	2006-05-02	Added more examples
1.1	2005-11-09	Added more examples
1.0	2005-02-11	Initial release.