Chipsmall
深圳市津利帝科技有限公司
Geelead Technology Limited
Chipsmall
采购部

Si4355/Si4455 Programming Guide

1. Introduction

This document provides an overview of how to configure and control the following EZRadio® chips:

- Si4455 transceiver
- Si4355 receiver

The following code examples are covered in this programming guide:

- How to set up a continuous wave (CW) transmission.
- How to set up a pseudo random (PN9) transmission.
- How to transmit in TX direct mode.
- How to receive in RX direct mode (for BER measurement).
- How to transmit a simple packet in Packet Handler mode.
- How to receive a simple packet in Packet Handler mode.
- How to implement bidirectional variable length packet based communication.

2. Hardware Options

The source code is provided for two different hardware platforms:

- RFStick
- Wireless Motherboard + RF Pico Board

A separate Silicon Labs IDE workspace is provided for each example on the two platforms.

2.1. The RFStick Platform

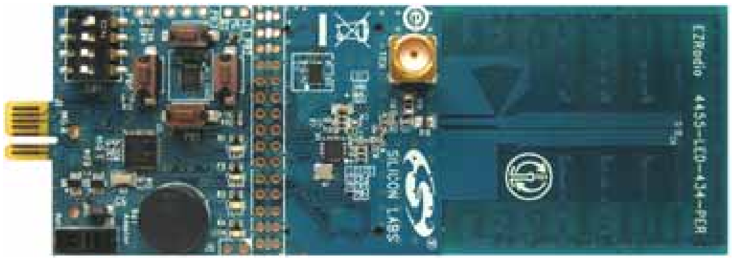

Figure 1. RFStick

The RFStick is a basic demo system for the evaluation of the EZRadio chips. The board has two main parts, the MCU part and the radio part. The MCU part of the board contains a Silicon Labs C8051F930 MCU and basic human interface devices (four push-buttons, four LEDs, four switches and a buzzer). The radio part contains the EZRadio chip, the matching circuit, and the antenna. The RF output is selectable via a 0 Ω resistor between a PCB antenna and an optional (unpopulated) 50 Ω SMA output connector. The MCU is connected to the EZRadio chip via an SPI bus and some other GPIOs (see Table 1). The RF section of the board can be broken off along a perforation between the two rows of J3 and installed in the user's own hardware as a radio module by utilizing the remaining row of J3.

Table 1 contains the signal connections between the EZRadio chip and the MCU:

Table 1. Connections between the EZRadio Chip and the MCU

Si4355, Si4455			RFStick		C80C51F930
Pin Number	**Pin Name**	**Pin Function**	**Connections across J3**	**Signal Name**	**Pin Name**
EP, 1, 6, 9	GND	Ground	3–4	GND	GND
7, 8	VDD	Supply Voltage input	1–2	VDD	VDD
12	NIRQ	Interrupt output, active low	19–20	NIRQ	P1.4
2	SDN	Shutdown input, active high	5–6	SDN	P1.5
16	NSEL	SPI select input	11–12	NSEL	P1.3
13	SCLK	SPI clock input	17–18	SCLK	P1.0
15	SDI	SPI data input	13–14	MOSI	P1.2
14	SDO	SPI data output	15–16	MISO	P1.1
10	GPIO_0	General Purpose I/O	23 x 24	GPIO_0/PB1	P0.0
11	GPIO_1	General Purpose I/O	21 x 22	GPIO_1/PB2	P0.1
19	GPIO_2	General Purpose I/O	9 x 10	GPIO_2/PB3	P0.2
20	GPIO_3	General Purpose I/O	7 x 8	GPIO_3/PB4	P0.3

The four GPIO signals' primary function is push button input to the MCU (PB1–PB4), so these signals are not connected to the EZRadio chip by default (represented by x in Table 1). The user can connect them by soldering in jumpers across the appropriate pins of J3.

2.1.1. Setting up and Connecting the RFStick to a PC

The power source of the board can be selected with the power-supply selector switch (S6). If S6 is in the Adapter position, supply voltage is provided by a Toolstick Base Adapter that is connected to the J1 PCB edge connector. If S6 is in the Battery position, the supply voltage is provided by two AAA batteries in the battery holder on the bottom side of the board. Current consumption of the RF part (RFVDD) can be measured on J6. Since J6 is shorted by a PCB track on the bottom side of the board, the user must cut the track if this feature is used.

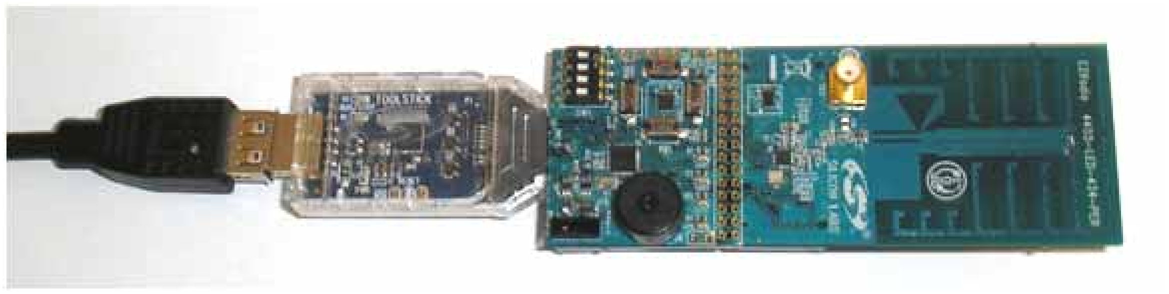

Figure 2. How to Connect the RFStick to the PC

Steps for connecting to a PC:

- Select the desired power source with S6 power selector switch.
- Connect the J1 connector of the RFStick to the Toolstick Base Adapter.
- Connect the Toolstick Base Adapter to the USB port of the PC.
- Wait for Windows to install the driver of the Toolstick Base Adapter, if necessary.

The RFStick is available in three different frequency band versions from Silicon Labs as part of several EZRadio kits (i.e., 434, 868, and 915 MHz).

2.2. The Wireless Motherboard Platform

The Wireless Motherboard (WMB) platform is a demo and development platform for the EZRadio and EZRadioPRO radio ICs. It consists of a Wireless Motherboard and interchangeable MCU Pico Boards and RF Pico Boards.

Figure 3. Wireless Motherboard Platform

Table 2. . Kits that Contain the Wireless Motherboard Platform

Part Number	Kit Name
EZR-LCDK2W-434	EZRadio Two Way Link Development Kit 434 MHz
EZR-LCDK2W-868	EZRadio Two Way Link Development Kit 868 MHz
EZR-LCDK2W-915	EZRadio Two Way Link Development Kit 915 MHz
4012-LCDK1W-434	Si4012 EZRadio One Way Link Development Kit 434 MHz
4012-LCDK1W-915	Si4012 EZRadio One Way Link Development Kit 915 MHz

2.2.1. The Wireless Motherboard

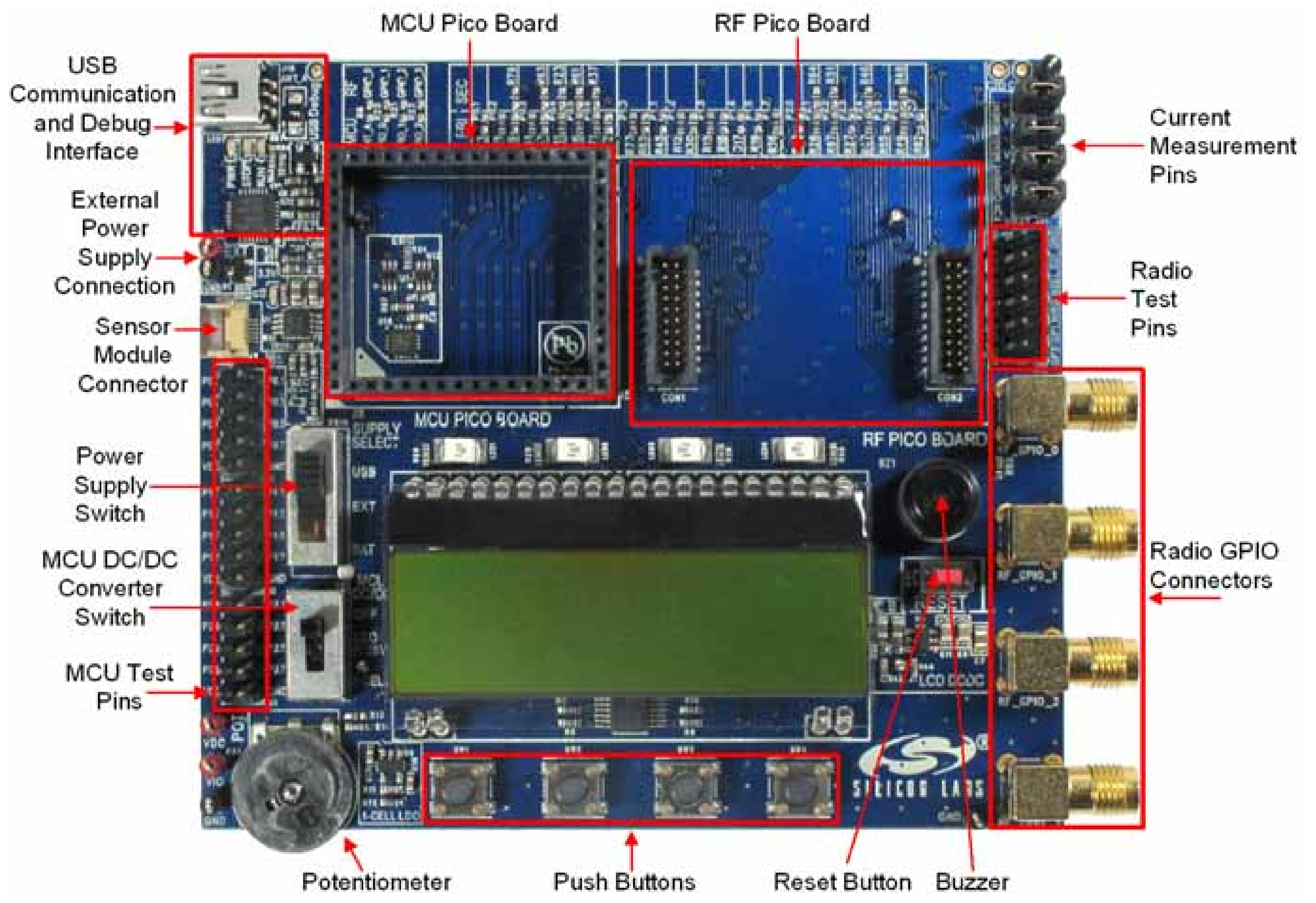

Figure 4. Wireless Motherboard

The wireless motherboard contains four pushbuttons, four LEDs, and a buzzer as simple user interfaces. A graphical LCD displays menu items for range testing purposes and a potentiometer demonstrates analog capabilities of the MCU. A switch supports the power options of the MCU's built-in dc/dc converter. Using the current measurement jumpers, current consumption can be measured separately either for the MCU, the radio, or the peripherals. The motherboard contains test pins for all I/O pins of the MCU and for all digital pins of the radio. In addition, there are SMA connectors for the GPIOs of the radio for test equipment connection. A USB communication interface as well as a built-in Silicon Labs USB-to-C2 debug adapter are integrated onto the board so that the wireless motherboard (WMB) can be directly connected via USB to the PC for downloading and debugging code on the MCU.

The WMB also contains an interface connection to sensor modules. The RF pico boards can be connected to the WMB through a connector pair.

2.2.2. Power Scheme

The power source of the platform can be selected with the power supply selector switch “SUPPLY SELECT” on the WMB board. If this switch is in the ”USB” position, supply voltage is provided by the PC that is connected to the ”J16” mini USB connector. If this switch is in the ”BAT” position, the supply voltage is provided by two AA batteries in the battery holder on the bottom side of the board. If the ”SUPPLY SELECT” switch is in the ”EXT” position, supply voltage is provided by an external power source through the ”TP7” and “TP9” points.

Using the ”MCU dc/dc” switch, the internal dc/dc converter of the C88051F930 MCU on the MCU pico board can be activated if the connected pico board supports this function. If the switch is in ”OFF” position, the MCU's dc/dc converter is inactive and the supply voltage is only determined by the state of the “SUPPLY SELECT” switch.

Positioning the switch to either ”LDO (1.25 V)” or ”1 CELL” position will turn on the MCU's dc/dc converter by connecting 1.25–1.5 V supply voltage to the VBAT pin and removing external power from the VDC pin. The MCU will provide 1.9 V in default setting on its VDC pin to all the other connected loads. Since this current is limited, it may be necessary to disconnect or disable some loading part of the board. For further details, see the MCU data sheet and the board schematic. The board schematic can be found in the EZRadioPRO Development Kit User's Guide. A complete CAD design pack of the board is also available at www.silabs.com.

2.2.3. RF Pico Board

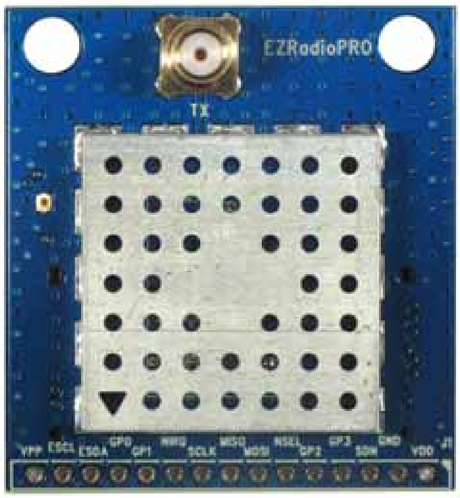

Figure 5. RF Pico Board

The RF Pico Board is a radio module that contains an EZRadio IC, matching network, and pcb antenna. The RF output is a 50 Ω SMA output connector. The boards also have a factory loaded board identification memory (EBID) that contains data that describes the board properties. Via the unified RF pico connector pair on the bottom side of the board, any RF pico board can be connected to the WMB.

2.2.4. Setting up and Connecting the WMB to the PC

Steps for connecting the platform to the PC:

1. Connect an RF Pico Board to the WMB board through the CON1 and CON2 connectors.
2. Insert a UPPI-930-RF MCU pico board in the connectors J5, J6, J7, J8 on the WMB. The dotted corner of the C8051F930 MCU has to point to the triangle symbol on the WMB.
3. Connect an antenna to the SMA connector on the RF Pico Board.
4. Select the desired power source with the SUPPLY SELECT switch.
5. Ensure that all the CURRENT MEASUREMENT jumpers are in place.
6. Connect the WMB board to a USB port of the PC.
7. Wait for Windows to install the driver of the debug interface if necessary.

3. Software Tools

Two software tools are provided by Silicon Labs to help EZRadio software development, the Wireless Development Suite (WDS) and the Silicon Labs Integrated Development Environment (IDE), both available atwww.silabs.com.

3.1. Wireless Development Suite (WDS)

The recommended starting point for Si4355/4455 development is the WDS. It can be downloaded from www.silabs.com.and can be installed on a PC. After connecting one of the hardware platforms described in this document to the PC, WDS is able to identify the connected board by reading the EBID memories.

The Radio Configuration Application GUI is part of the WDS program. This setup interface provides an easy path to quickly selecting and loading the desired configuration for the Si4355/4455 device. After the desired configuration is selected, the EZConfig setup automatically creates the configuration data that can be used to configure the EZRadio chip. The program then gives the option to directly configure the EZRadio chip of the connected hardware, to modify a selected example code with the configuration and download it to the connected hardware, or to launch Silicon Labs IDE with the new configuration data preloaded into the selected example project. For more information on WDS and EZConfig usage, refer to the application notes “AN796: Wireless Development Suite General Description” and “AN797: WDS User’s Guide for EZRadio Devices”, available at www.silabs.com.

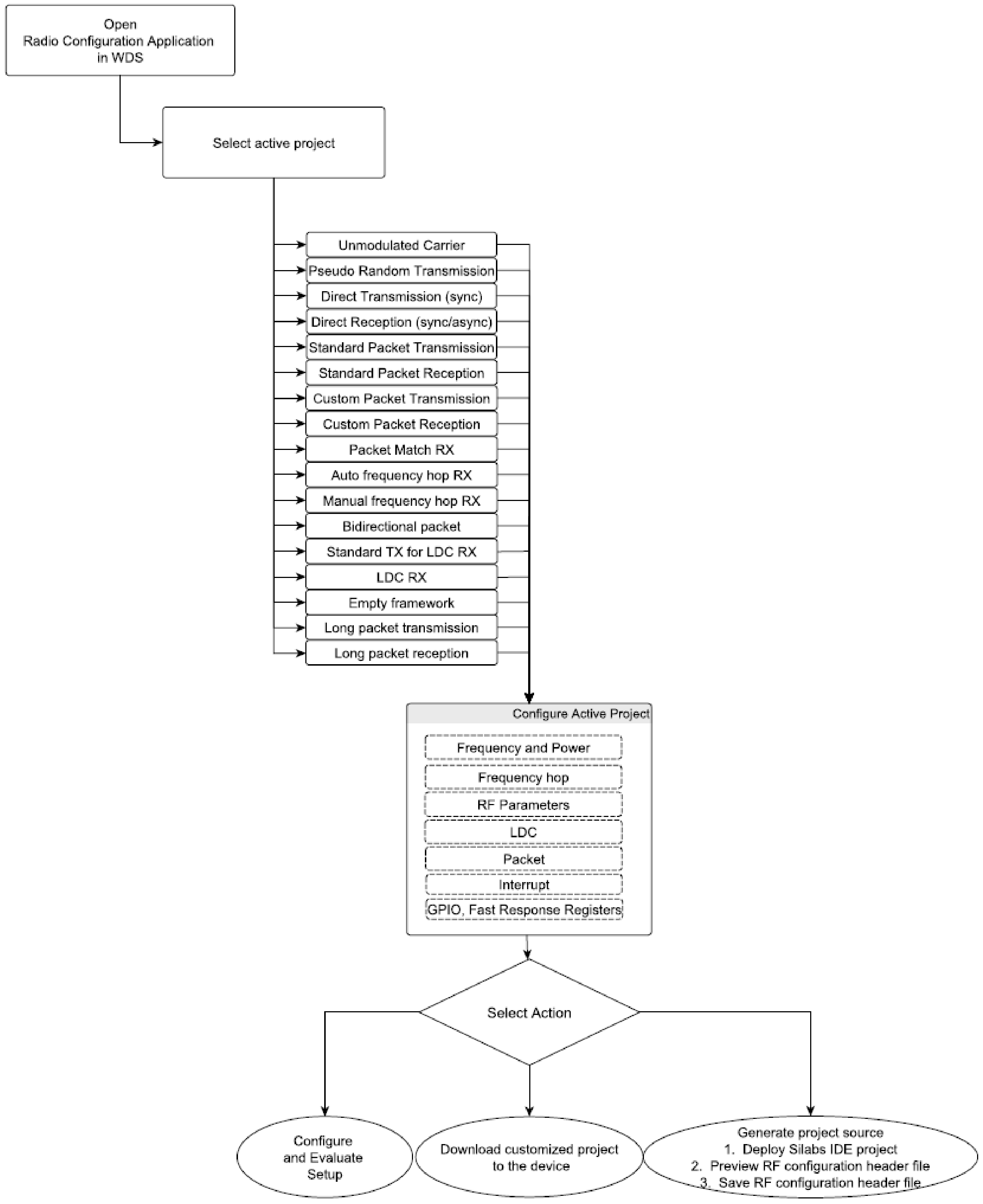
Open
Radio Configuration Application
in WDS
Select active project
Unmodulated Carrier
Pseudo Random Transmission
Direct Transmission (sync)
Direct Reception (sync/async)
Standard Packet Transmission
Standard Packet Reception
Custom Packet Transmission
Custom Packet Reception
Packet Match RX
Auto frequency hop RX
Manual frequency hop RX
Bidirectional packet
Standard TX for LDC RX
LDC RX
Empty framework
Long packet transmission
Long packet reception
Configure Active Project
Frequency and Power
Frequency hop
RF Parameters
LDC
Packet
Interrupt
GPIO, Fast Response Registers
Select Action
Configure and Evaluate Setup
Download customized project to the device
Generate project source
1. Deploy Silabs IDE project
2. Preview RF configuration header file
3. Save RF configuration header file

Figure 6. Device Configuration Options

3.2. Silicon Labs IDE

The Silicon Laboratories Integrated Development Environment (IDE) is a standard tool for program development for any Silicon Labs 8-bit MCUs, including the C8051F930 that is used on the hardware platforms described in this document. The Silicon Laboratories IDE integrates a project manager, a source-code editor, source-level debugger, and an in-system flash programmer. The IDE interfaces to third party development tool chains to provide system designers a complete embedded software development environment. The Keil Demonstration Toolset includes a compiler, linker, and assembler and easily integrates into the IDE.

3.2.1. Downloading and Running the Example Codes

1. Connect the hardware platform to the PC according to the description of the used platform.
2. Start Silicon Labs IDE (IDE 4.40 or higher required) on your computer.
3. Select **Project→Open Project...** to open a previously saved project.
4. Before connecting to the target device, several connection options may need to be set. Open the **Connection Options** window by selecting **Options→Connection Options...** in the IDE menu.
5. Select USB Debug Adapter in the “Serial Adapter” section.
6. If more than one adapter is connected, choose the appropriate serial number from the drop-down list.
7. Check the “Power target after disconnect” if the target board is currently being powered by the USB Debug Adapter. The board will remain powered after a software disconnect by the IDE.
8. Next, the correct “Debug Interface” must be selected. Check the C2 Debug Interface.
9. Once all the selections are made, click the **OK** button to close the window.
10. Click the **Connect** button in the toolbar or select **Debug→Connect** from the menu to connect to the C8051F930 MCU of the platform.
11. Erase the flash of the C8051F930 MCU in the Debug→Download object code→ Erase all code space menu item.
12. Download the desired example HEX file either by hitting the Download code (Alt+D) toolbar button or from the Debug →Download object code menu item.
13. Hit the Disconnect toolbar button or invoke the Debug →Disconnect menu item to release the device from halt and to let it run.

3.3. ToolStick Terminal

The ToolStick Terminal program provides the standard terminal interface to the target microcontroller's UART. However, instead of requiring the usual RS-232 and COM port connection, ToolStick Terminal uses the USB interface of the ToolStick Base Adapter to provide the same functionality. The firmware on the target microcontroller does not need to be customized to use the UART and communicate with ToolStick Terminal. The firmware on the microcontroller should write to the UART as it would in any standard application and all of the translation is handled by the ToolStick Base Adapter.

The ToolStick Base Adapter is integrated on the WMB and is also part of the RFStick platform as a separate device.

The ToolStick Terminal program is part of the Silicon Labs IDE and is also available as a separate application. Both can be installed as part of the Silicon Labs 8-bit Microcontroller Studio from:
http://www.silabs.com/products/mcu/Pages/8-bit-microcontroller-software.aspx

The IDE and its built in ToolStick Terminal can communicate with the target MCU simultaneously on the C2 interface and on the UART respectively.

To use the ToolStick Terminal in the IDE (above v4.60.00), follow these steps:

1. Open the Silabs IDE from the Start→ Programs→ Silicon Laboratories menu.
2. Go to the Options→Connection Options menu and select the desired ToolStick Base Adapter from the drop down list.
3. Click on the Connect button to connect the IDE to the target MCU via the C2 interface.
4. From the Tools menu, start the Toolstick Terminal. In the top left-hand corner of the Terminal application, go to the ToolStick→Settings menu and set the communication parameters. Now the ToolStick Terminal is ready for use. In the "Receive Data" window, text indicating the received characters will appear.

In addition to the standard two UART pins (TX and RX), there are two GPIO/UART handshaking pins on the ToolStick Base Adapter. On both the WMB and RFStick platforms GPIO0 is used for the internal purpose of the WDS to select between the C2 interface of the target MCU and the EBID MCU. GPIO1 is not connected. Although the separate ToolStick Terminal application provides the functionality to control these GPIOs, default settings for GPIO0 should not be changed.

4. Using the Si4355/Si4455 Radios

Si4355 and Si4455 are easy-to-use radio chips that combine plug-and-play simplicity with the flexibility needed to handle a wide variety of applications. This chapter describes how to operate the Si4355/Si4455 radios.

4.1. Radio Hardware Interface

The Si4355/Si4455 radios have several pins to interface to a host MCU. Four pins, SDI, SDO, SCLK, NSEL, are used to control the radio over the SPI bus. The user has access to an Application Programming Interface (API) via the SPI bus, which is described in section "4.2. Application Programming Interface". The Shutdown (SDN) pin is used to completely disable the radio (when the pin is pulled high) and put the device into the lowest power consumption state. After the SDN pin is pulled low, the radio wakes up and performs a Power On Reset. It takes about 1 ms until the chip is ready to receive commands on the SPI bus (GPIO1 pin goes high when the radio is ready for receiving SPI commands). When SDN is high and the radio is in shutdown state, the GPIOs are set to drive an output low level. The radio has an interrupt output pin, NIRQ, which can be used to promptly notify the host MCU of various events. The NIRQ pin is active low, and goes back to high if the pending interrupt flag was cleared by reading the appropriate Interrupt Pending registers.

Table 3. Serial Peripheral Interface Signals

Radio Pin Signal	Description
SCLK	Serial Clock Output From Master
SDI	Master Output, Slave Input
SDO	Master Input, Slave Output
NSEL	Slave Select, Active Low

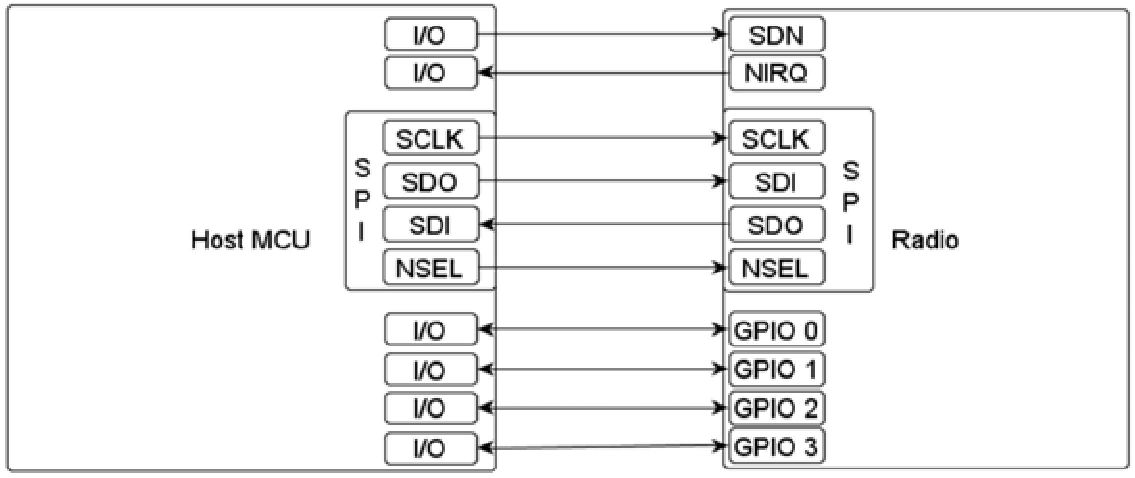

Figure 7. Connections between the Radio Chip and the Host Microcontroller

4.2. Application Programming Interface

The programming interface allows the user to do the following:

- Send commands to the radio.
- Read status information.
- Set and get radio parameters.
- Handle the Transmit and Receive FIFOs.

The API commands are listed in the following table:

Table 4. List of the Radio API Commands

Command ID	Radio API Command	Description
0x00	NOP	No operation command
0x02	POWER_UP	Powerup device and mode selection. Modes include operational function
0x01	PART_INFO	Reports basic information about the device
0x10	FUNC_INFO	Returns the function revision information of the device
0x11	SET_PROPERTY	Sets the value of a property
0x12	GET_PROPERTY	Retrieves a property's value
0x13	GPIO_PIN_CFG	Configures the GPIO pins
0x15	FIFO_INFO	Provides access to transmit and receive FIFO counts and reset
0x19	EZCONFIG_CHECK	Validates the EZConfig array was written correctly
0x20	GET_INT_STATUS	Returns the interrupt status byte
0x31	START_TX	Switches to TX state and starts packet transmission (Si4455only)
0x32	START_RX	Switches to RX state
0x33	REQUEST_DEVICE_STATE	Request current device state
0x34	CHANGE_STATE	Update state machine entries
0x44	READ_CMD_BUFF	Returns Clear to Send (CTS) value and the result of the previous command
0x50	FRR_A_READ	Reads the fast response registers (FRR) starting with FRR_A.
0x51	FRR_B_READ	Reads the fast response registers (FRR) starting with FRR_B.
0x53	FRR_C_READ	Reads the fast response registers (FRR) starting with FRR_C.
0x57	FRR_D_READ	Reads the fast response registers (FRR) starting with FRR_D.
0x66	EZCONFIG_SETUP	Configures device using EZConfig array
0x66	WRITE_TX_FIFO	Writes TX data buffer (max. 64 bytes, Si4455 only)
0x77	READ_RX_FIFO	Reads RX data buffer (max. 64 bytes)

The API properties are listed in Table 5.

Table 5. List of the Radio API Properties

Property Group	Number	Name	Description	Default
0x01	0x00	INT_CTL_ENABLE	Interrupt enable property	0x04
0x01	0x01	INT_CTL_PH_ENABLE	Packet handler interrupt enable property	0x00
0x01	0x02	INT_CTL_MODEM_ENABLE	Modem interrupt enable property	0x00
0x01	0x03	INT_CTL_CHIP_ENABLE	Chip interrupt enable property	0x04
0x02	0x00	FRR_CTL_A_MODE	Fast Response Register A Configuration	0x01
0x02	0x01	FRR_CTL_B_MODE	Fast Response Register B Configuration	0x02
0x02	0x02	FRR_CTL_C_MODE	Fast Response Register C Configuration	0x09
0x02	0x03	FRR_CTL_D_MODE	Fast Response Register D Configuration	0x00
0x22	0x01	PA_PWR_LVL	PA Level Configuration	0x7F
0x24	0x03	EZCONFIG_XO_TUNE	Configure crystal oscillator frequency tuning bank	0x40
0x40	0x00	FREQ_CONTROL_INTE	Frac-N PLL integer number	0x3C
0x40	0x01	FREQ_CONTROL_FRAC_2	Byte 2 of Frac-N PLL fraction number	0x08
0x40	0x02	FREQ_CONTROL_FRAC_1	Byte 1 of Frac-N PLL fraction number	0x08
0x40	0x03	FREQ_CONTROL_FRAC_0	Byte 0 of Frac-N PLL fraction number	0x08
0x40	0x04	FREQ_CONTROL_CHANNEL_-STEP_SIZE_1	Byte 1 of channel step size	0x00
0x40	0x05	FREQ_CONTROL_CHANNEL_-STEP_SIZE_0	Byte 0 of channel step size	0x00

The following sections describe the SPI transactions of sending commands and getting information from the chip.

4.2.1. Sending Commands to a Radio

The behavior of the radio can be changed by sending API commands to the radio (e.g., changing the power states, start packet transmission, etc.). The radio can be configured through several "properties". The properties represent radio configuration settings, such as interrupt settings, modem parameters, packet handler settings, etc., and can be set and read via API commands. For most of the commands, the host MCU does not expect any response from the radio chip. Other commands are used to read back a property from the chip, such as checking the interrupt status flags, reading the transmit/receive FIFOs.

After the radio receives a command, it processes the request. During this time, the radio is not capable of receiving a new command. The host MCU must identify when the next command can be sent. The Clear to Send (CTS) signal shows the actual status of the command buffer of the radio. It can be monitored over the SPI or on GPIOs, or the chip can generate an interrupt if it is ready to receive the next command. These three options are detailed below.

4.2.2. Checking that the Radio is Ready to Receive Commands

4.2.2.1. Software Polling Method

To ensure the radio is ready to receive the next command, the host MCU must pull down the NSEL pin to monitor the status of CTS over the SPI port. The 0x44 command ID has to be sent, and eight clock pulses have to be generated, on the SCLK pin. During the additional eight clock cycles, the radio clocks out the CTS as a byte on the SDO pin. When completed, the NSEL should be pulled back to high. If the CTS byte is 0xFF, then the radio processed the last command successfully and is ready to receive the next command; in any other case, the CTS read procedure has to be repeated from the beginning as long as the CTS byte is not 0xFF.

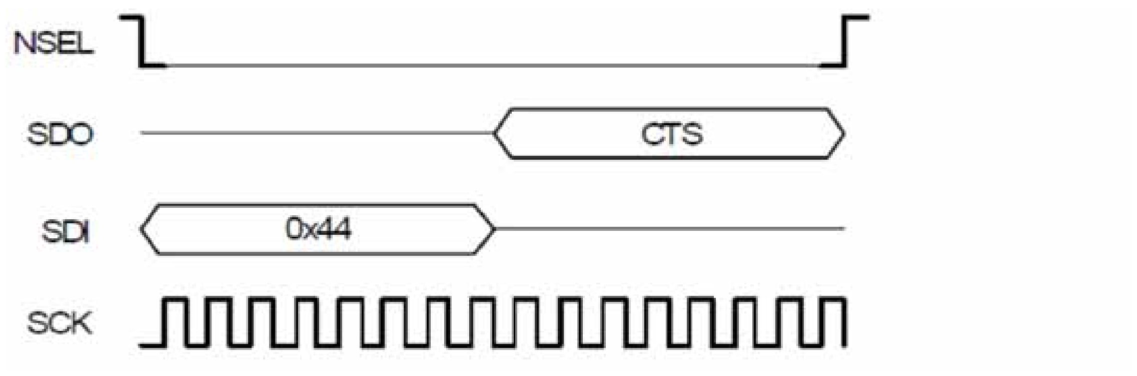

Figure 8. Polling the Radio Availability

4.2.2.2. GPIO Checking Method

Any GPIO can be configured for monitoring the CTS. GPIOs can be configured to go either high or low when the chip completes the command. The function of the GPIOs can be changed by the *GPIO_PIN_CFG* command. By default, GPIO1 is set as “High when command completed, low otherwise” after Power On Reset. Therefore, this pin can be used for monitoring the CTS right after Power On Reset and to identify when the chip is ready to boot up.

4.2.2.3. NIRQ Interrupt Checking Method

The radio asserts the CHIP_READY interrupt flag if a command is completed. The interrupt flag can be monitored by either the GET_CHIP_STATUS or the GET_INT_STATUS command. Apart from monitoring the interrupt flags, the radio may pull down the NIRQ pin if this feature is enabled. If a new command is sent while the CTS is asserted, then the radio ignores the new command. The Si446x can generate an interrupt to communicate this error to the MCU by the CMD_ERROR interrupt flag in the CHIP_STATUS group. The interrupt flag has to be read (by issuing a GET_CHIP_STATUS or GET_INTERRUPT_STATUS command) to clear the pending interrupt and release the NIRQ pin. No other action is needed to reset the command buffer of the radio; however, after a CMD_ERROR, the host MCU should repeat the new command after the radio has processed the previous one.

All the commands that are sent to the radio have the same structure. After pulling down the NSEL pin of the radio, the command ID should be sent first. The commands may have up to 15 input parameters.

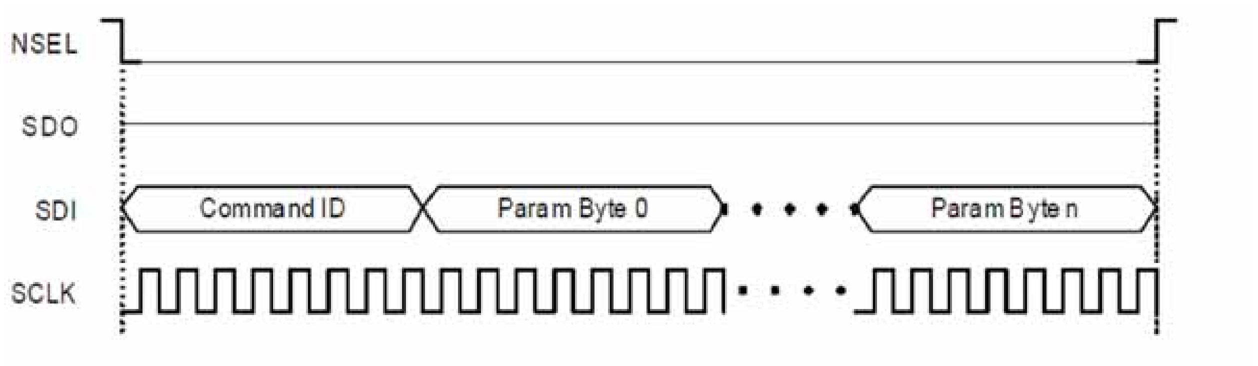

Figure 9. Host MCU Sends Command to Radio

4.2.3. Getting a Command Response from Radio

Reading from the radio requires several steps to be followed. The host MCU should send a command with the address it requests to read. The radio holds the CTS while it retrieves the requested information. Once the CTS is set (0xFF), the host MCU can read the answer from the radio.

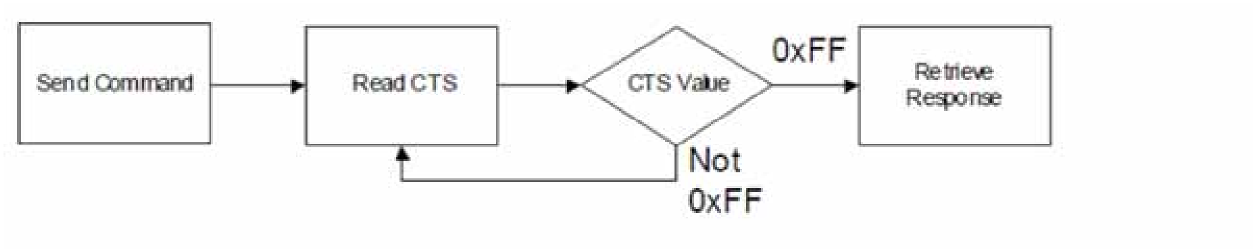

Figure 10. Read Procedure

If the CTS is polled on the GPIOs, or the radio is configured to provide interrupt if the answer is available, then the response can be read out from the radio with the following SPI transaction:

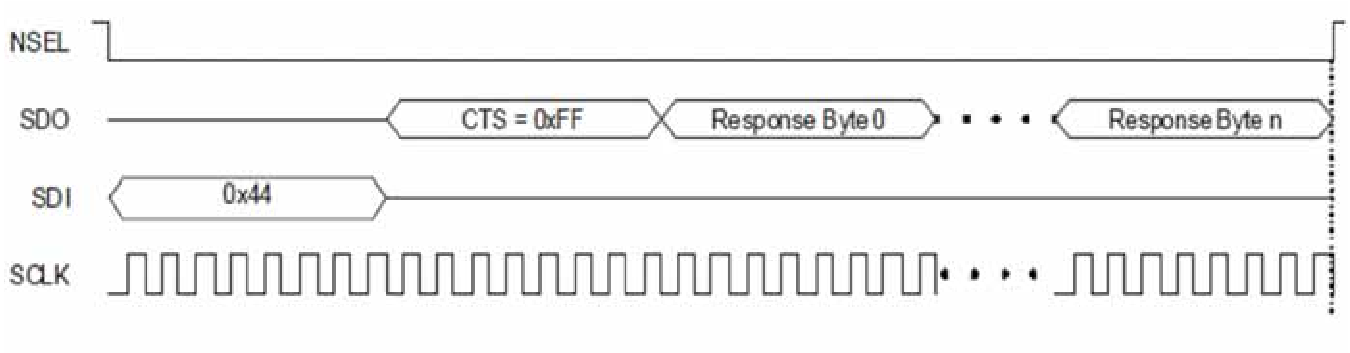

Figure 11. Read the Response from Radio

If the CTS is polled over the SPI bus, first the host MCU should pull the NSEL pin low. This action should be followed by sending out the 0x44 Read command ID and providing an additional eight clock pulses on the SCLK pin. The radio will provide the CTS byte on its SDO pin during the additional clock pulses. If the CTS byte is 0x00, then the response is not yet ready and the host MCU should pull up the NSEL pin and repeat the procedure from the beginning as long as the CTS byte is not 0xFF. If CTS is 0xFF, then the host MCU should keep the NSEL pin low and provide clock cycles on the SCLK pin, as many as the data to be read out requires. The radio will clock out the requested data on its SDO pin during the additional clock pulses.

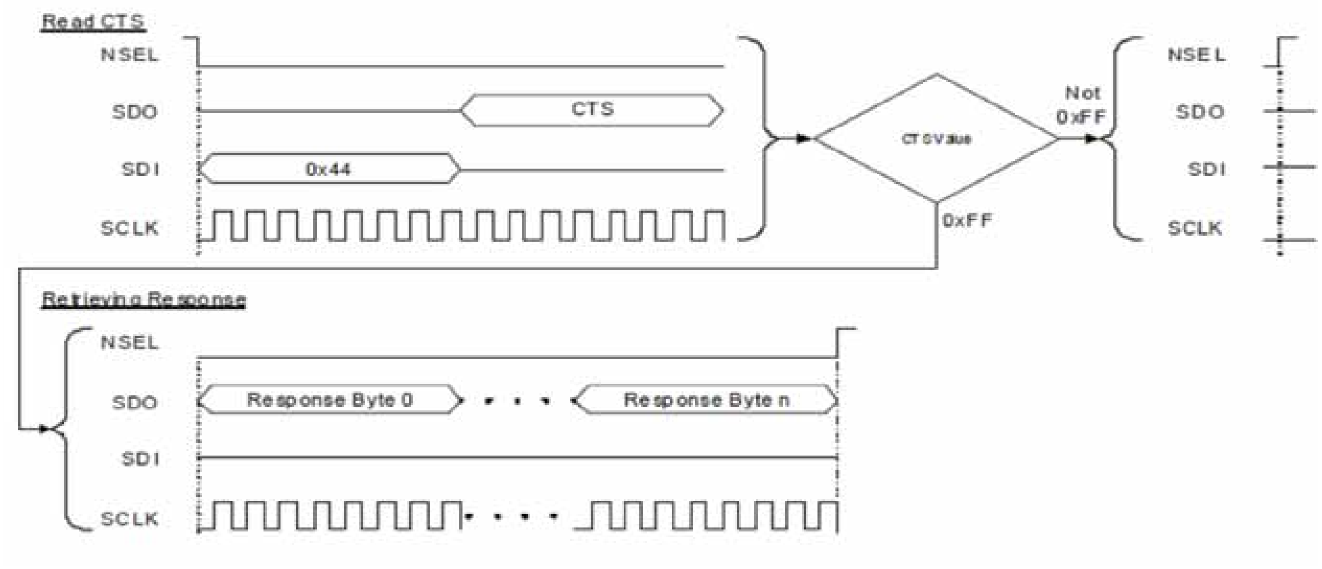

Figure 12. Monitor CTS and Read the Response on the SPI Bus

Reading the response from the radio can be interrupted earlier. For example, if the host MCU asked for five bytes of response, it may read fewer bytes in one SPI transaction. As long as a new command is not sent, the radio keeps the response for the last request in the command buffer. The host MCU can re-read the response in a new SPI transaction. In such a case, the response is always provided from the first byte.

Notes:

- Up to 16 bytes of response can be read from the radio in one SPI transaction. If more bytes are read, the radio will provide the same 16 bytes of response in a circular manner.
- If the command has N bytes of response, but the host MCU provides less than N bytes of clock pulses during the read sequence, it causes no issue for the radio. The response buffer is reset if a new command is issued.
- If the command has N bytes of response, but, during the read sequence, the host MCU provides more than N bytes of clock pulses, the radio will provide unpredictable bytes after the first N bytes. The host MCU does not need to reset the SPI interface; it happens automatically if NSEL is pulled low before the next command is sent.

4.2.4. Using Fast Response Registers

There are several types of status information that can be read out from the radio faster. The FRR_CTL_x_MODE (where x can be A, B, C or D) properties define what status information is assigned to a given fast response register (FRR). The actual value of the registers can be read by pulling down the NSEL pin, issuing the proper command ID, and providing an additional eight clock pulses on the SCLK pin. During these clock pulses, the radio provides the value of the addressed FRR. The NSEL pin has to be pulled high after finishing the register read.

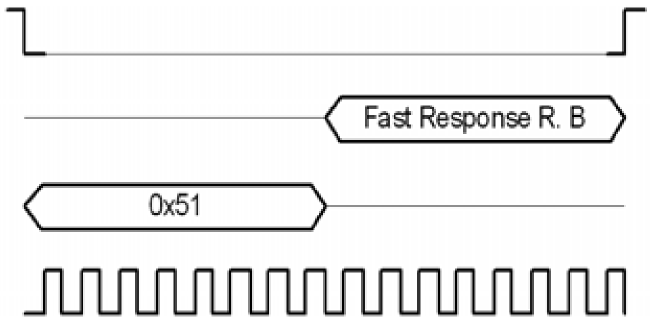

Figure 13. Reading a Single Fast Response Register

It is also possible to read out multiple FRRs in a single SPI transaction. The NSEL pin has to be pulled low, and one of the FRRs has to be addressed with the proper command ID. Providing an additional 8 x N clock cycles will clock out an additional N number of FRRs. After the fourth byte is read, the radio will provide the value of the registers in a circular manner. The reading stops by pulling the NSEL pin high.

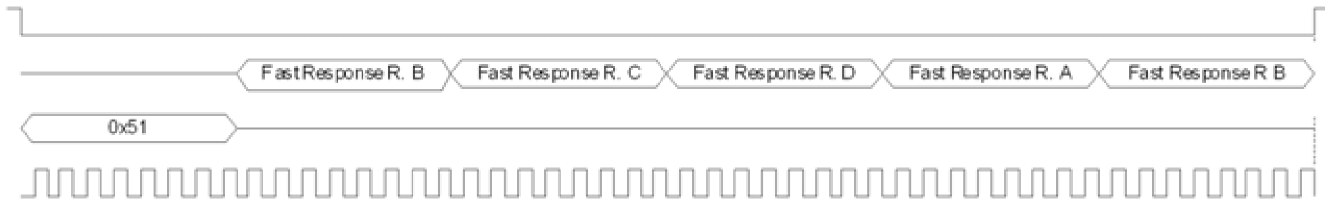

Figure 14. Reading More Fast Response Registers in a Single SPI Transaction

Note: If the pending interrupt status register is read through the FRR, the NIRQ pin does not go back to high. The pending interrupt registers have to be read by a Get response to a command sequence in order to release the NIRQ pin.

4.2.5. Write and Read the FIFOs

There are two 64-byte FIFOs for RX and TX data in the Si4x55.

To fill data into the transmit FIFO, the host MCU should pull the NSEL pin low and send the 0x66 Transmit FIFO Write command ID followed by the bytes to be filled into the FIFO. Finally, the host MCU should pull the NSEL pin high. Up to 64 bytes can be filled into the FIFO during one SPI transaction.

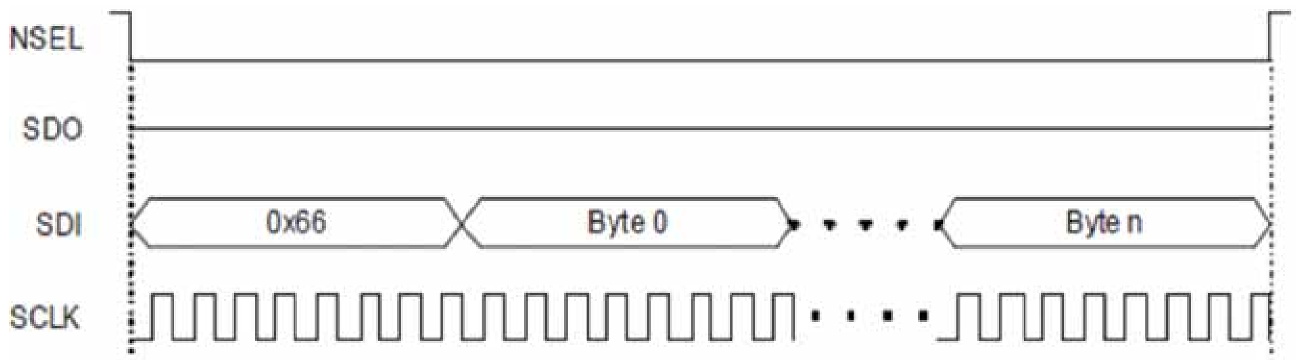

Figure 15. Transmit FIFO Write

If the host MCU needs to read the receive FIFO, it has to pull the NSEL pin low and send the 0x77 Receive FIFO Read command ID. The MCU should provide as many clock pulses on the SCLK pin as necessary for the radio to clock out the requested amount of bytes from the FIFO on the SDO pin. Finally, the host MCU should pull up the NSEL pin.

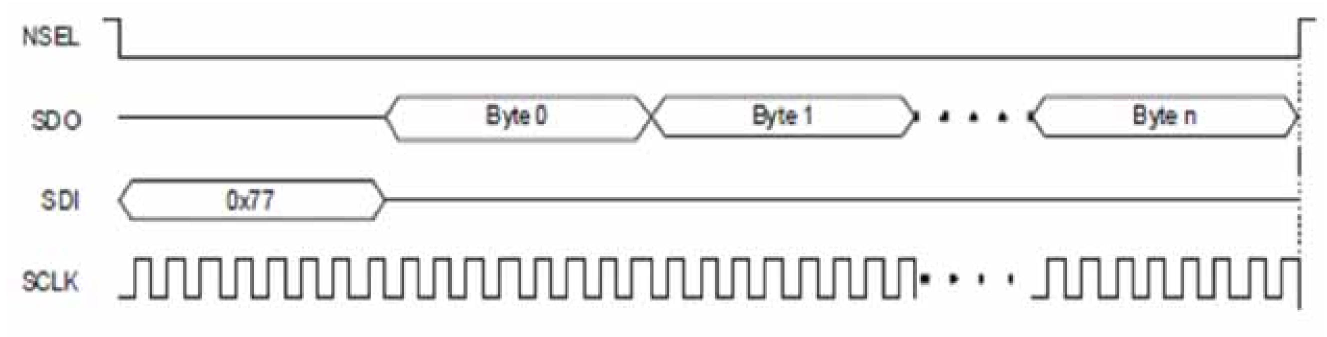

Figure 16. Receive FIFO Read

If more than 64 bytes are written into the Transmit FIFO, then a FIFO overflow occurs. If more bytes are read from the Receive FIFO than it holds, then FIFO underflow occurs. In either of these cases, the FIFO_UNDERFLOW_OVERFLOW_ERROR interrupt flag will be set. The radio can also generate an interrupt on the NIRQ pin if this flag is enabled. The interrupt flag has to be read, by issuing a GET_CHIP_STATUS or GET_INTERRUPT_STATUS command, to clear the pending interrupt and release the NIRQ pin.

4.3. State Transitions of the EZRadio Devices

Ready state is designed to give a fast transition time to TX or RX state with reasonable current consumption. In this mode the crystal oscillator remains enabled reducing the time required to switch to TX or RX mode by eliminating the crystal start-up time. An automatic sequencer will put the chip into RX or TX from any state. It is not necessary to manually step through the states. Although it is not shown in the diagram, any of the lower power states can be returned to automatically after RX or TX.

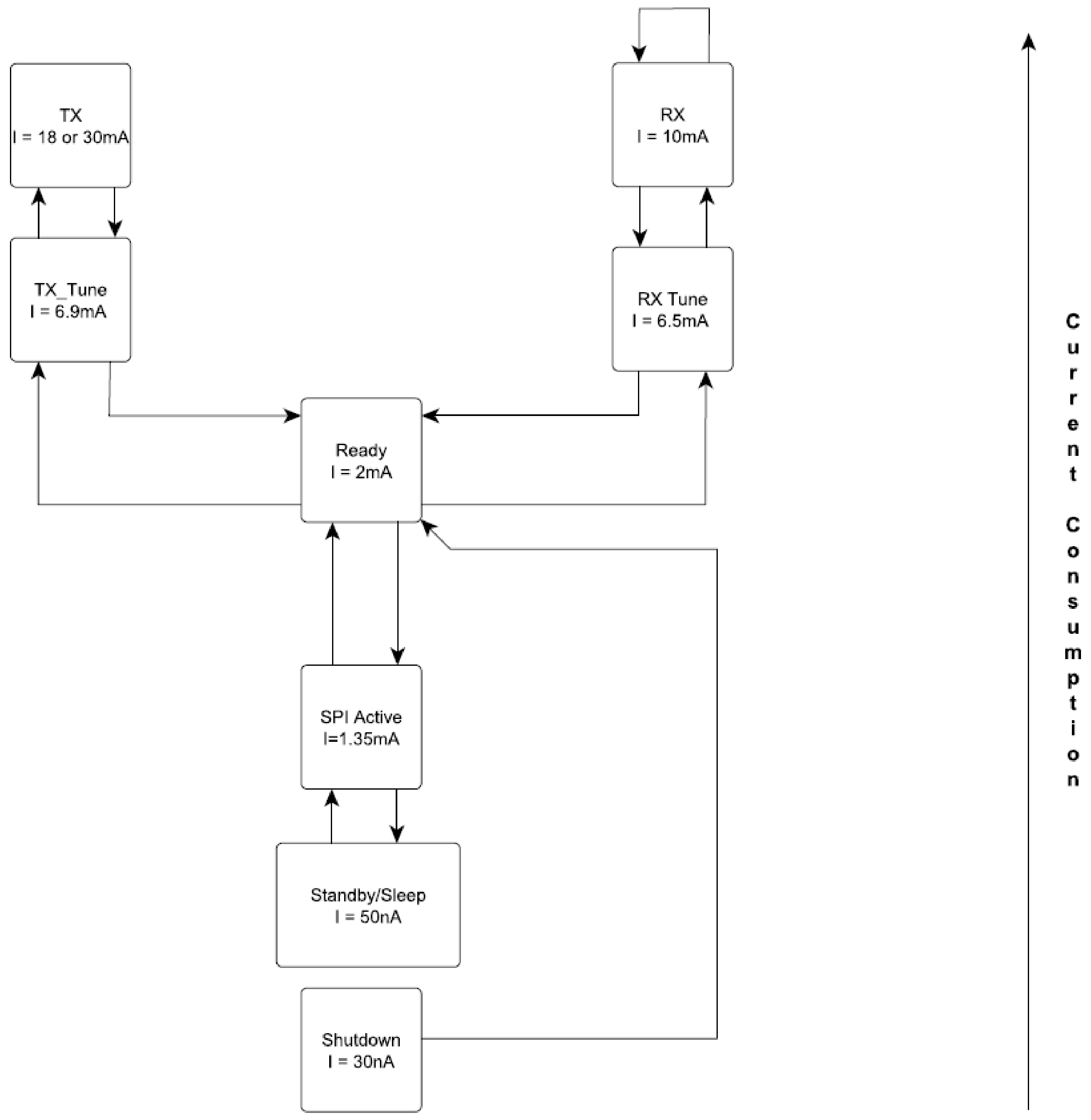

Figure 17. Operational States and Current Consumption

Table 6. Switching Times between Radio States

State/Mode	Response Time to	
	TX	RX
Shutdown	15 ms	15 ms
Sleep	440 µs	440 µs
SPI Active	340 µs	340 µs
Ready	126 µs	122 µs
TX Tune	58 µs	N/A
RX Tune	N/A[1]	74 µs
TX	N/A[1]	138 µs
RX	130 µs	75 µs

Notes:

1. This state change is not possible in the RF chip.
2. While the chip is in sleep state, the NSEL pin has to stay in high state. If the host processor is not able to provide this during sleep, a pullup resistor can be necessary on the NSEL pin.

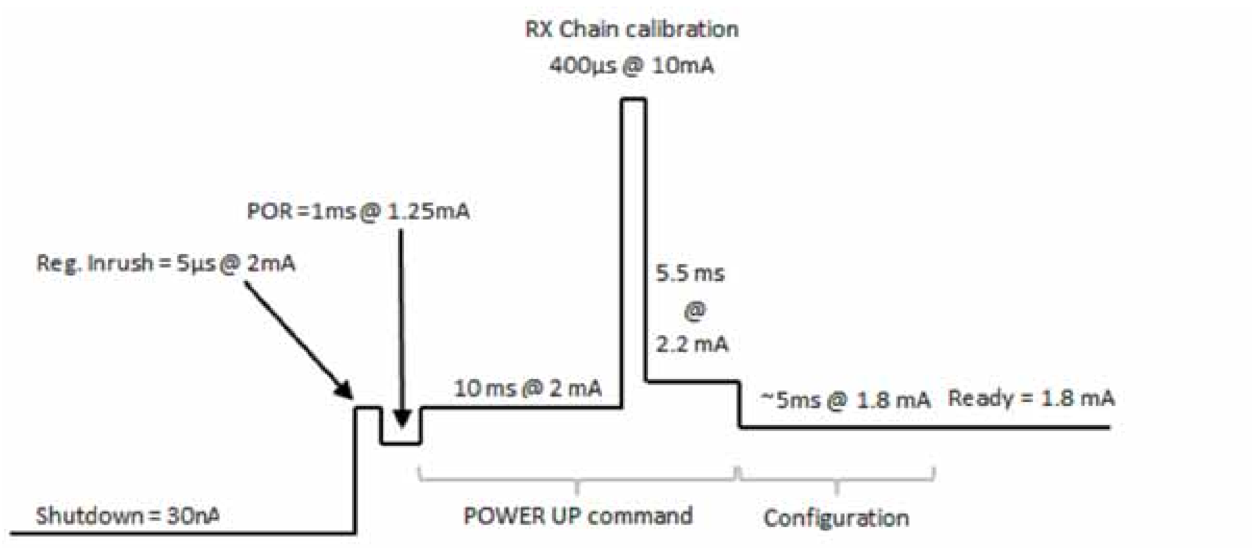

Figure 18. Supply Current versus Time Diagram from Shutdown to RF initialized Ready State

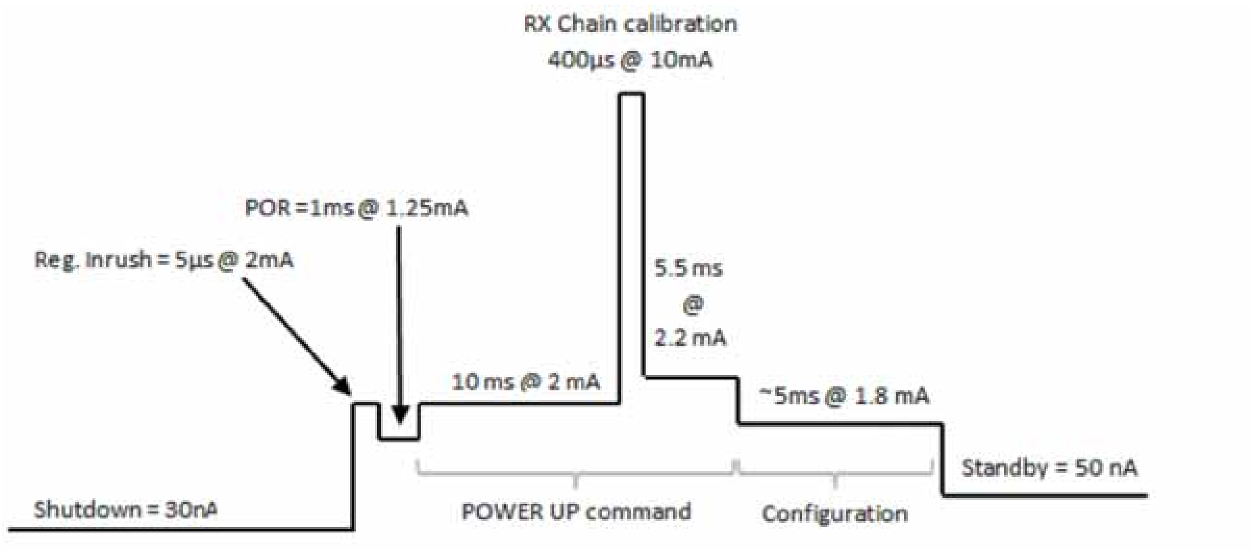

Figure 19. Supply Current versus Time Diagram from Shutdown to Standby State

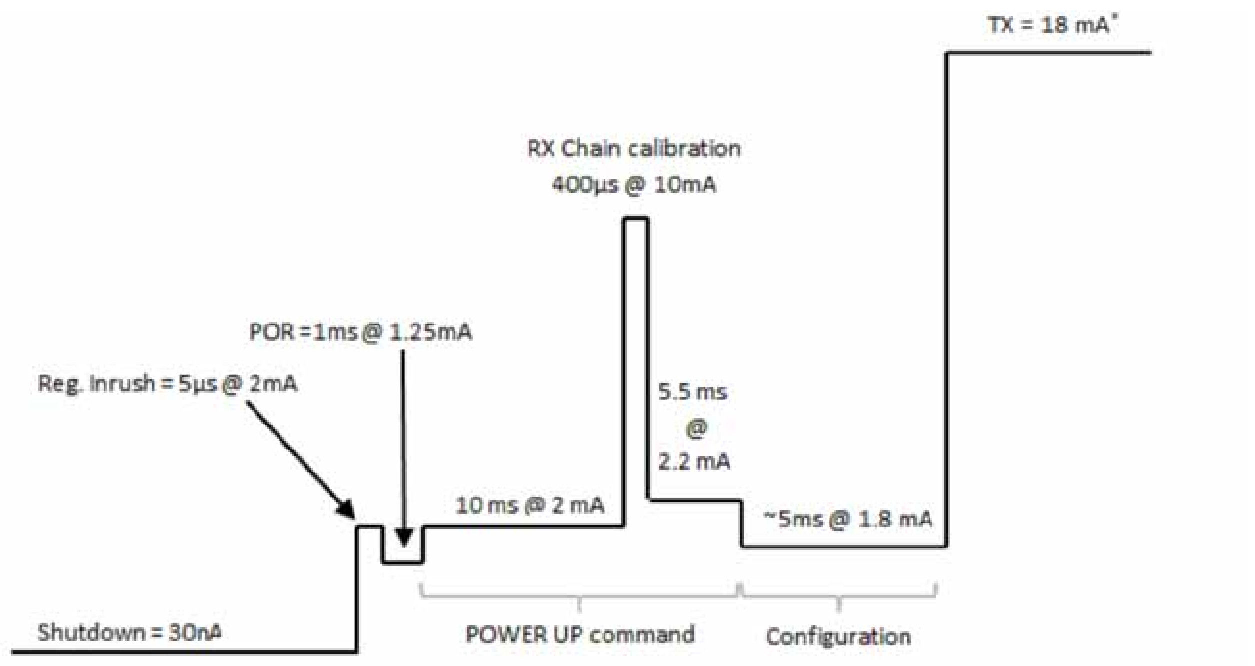

Figure 20. Supply Current versus Time Diagram from Shutdown to TX State

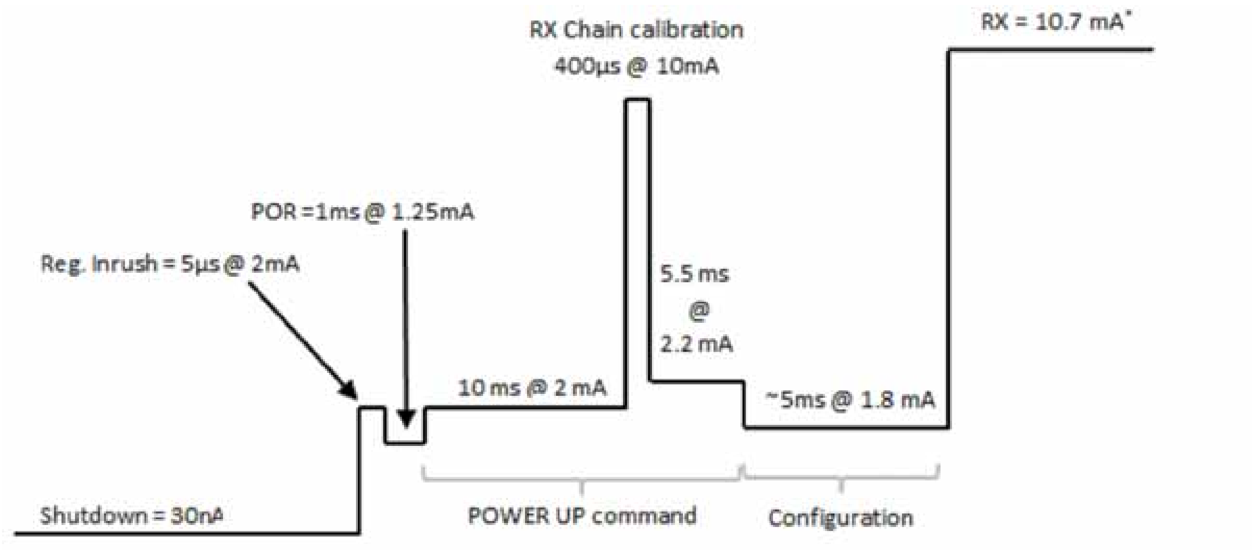

Figure 21. Supply Current versus Time Diagram from Shutdown to RX State

4.4. Radio Chip Waking Up

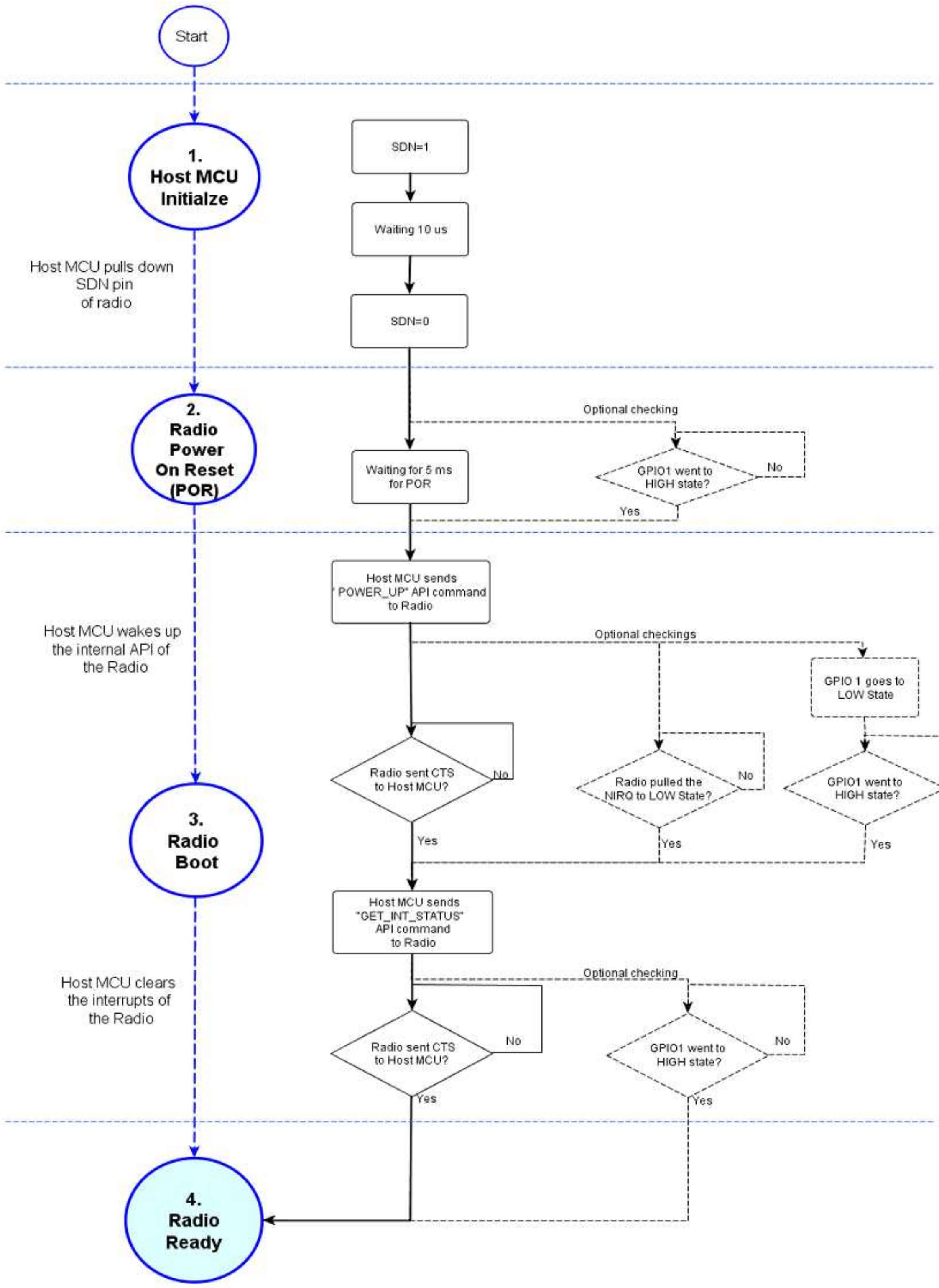
Start
1. Host MCU Initialze
Host MCU pulls down SDN pin of radio
SDN=1
Waiting 10 us
SDN=0
2. Radio Power On Reset (POR)
Optional checking
Waiting for 5 ms for POR
GPIO1 went to HIGH state?
No
Yes
Host MCU wakes up the internal API of the Radio
Host MCU sends "POWER_UP" API command to Radio
Optional checkings
GPIO 1 goes to LOW State
Radio sent CTS to Host MCU?
No
Radio pulled the NIRQ to LOW State?
No
GPIO1 went to HIGH state?
No
Yes
Yes
Yes
3. Radio Boot
Host MCU clears the interrupts of the Radio
Host MCU sends "GET_INT_STATUS" API command to Radio
Optional checking
Radio sent CTS to Host MCU?
No
GPIO1 went to HIGH state?
No
Yes
Yes
4. Radio Ready

Figure 22. Radio Wake Up Process

First the radio is in off state. After the SDN pin is pulled low, the radio wakes up and performs a Power on Reset, which takes a maximum of 5 ms until the chip is ready to receive commands on the SPI bus. GPIO1 pin goes high when the radio is ready for receiving SPI commands. During the reset period, the radio cannot accept any SPI command.

There are two ways to determine if the chip is ready to receive SPI commands after a reset event: 1) Use a timer in the host microcontroller to wait for this period or 2) Connect the GPIO1 pin of the radio to the host MCU and poll the status of this pin. During the power on reset, the GPIO1 remains in low state. Once the reset is finished, the radio sets GPIO1 to high state. Next, the radio device has to be sent to active mode by issuing a "POWER_UP" command via the SPI interface, which takes approximately 5 ms to be completed. This process can be monitored in three ways: The first option is to poll the CTS over the SPI; the second option is to detect that the radio pulled down the NIRQ pin, and the third option is to detect that the radio pulled up the GPIO1 pin. Once the device is in active mode, the host MCU must clear the interrupt flags from the radio.

4.5. EZConfig and Configuration Options

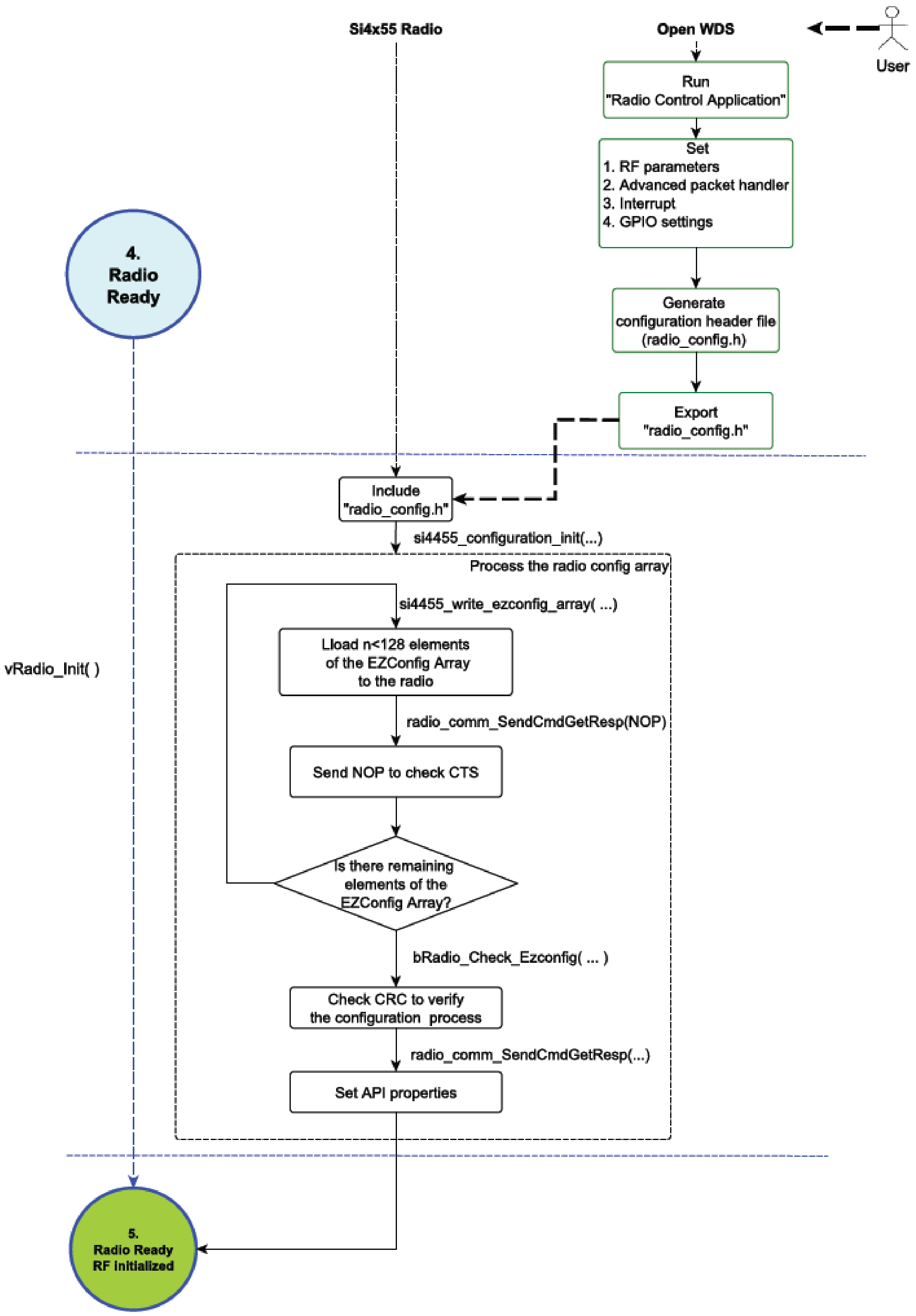
Si4x55 Radio
Open WDS
User
Run "Radio Control Application"
Set
1. RF parameters
2. Advanced packet handler
3. Interrupt
4. GPIO settings
Generate configuration header file (radio_config.h)
Export "radio_config.h"
4. Radio Ready
Include "radio_config.h"
si4455_configuration_init(...)
Process the radio config array
si4455_write_ezconfig_array(...)
Lload n<128 elements of the EZConfig Array to the radio
vRadio_Init( )
radio_comm_SendCmdGetResp(NOP)
Send NOP to check CTS
Is there remaining elements of the EZConfig Array?
bRadio_Check_Ezconfig(...)
Check CRC to verify the configuration process
radio_comm_SendCmdGetResp(...)
Set API properties
5. Radio Ready RF Initialized

Figure 23. Radio RF Initialization Process