C8051F930 Wireless Software Development Kit User's Guide

1. Introduction

The Silicon Labs Wireless Product Software Development Board, MSC-DBSB8, is designed to help engineers develop code for the Silicon Lab's EZRadio® and EZRadioPRO™ wireless products using the Silicon Labs C8051F9xx microcontroller platform.

The C8051F9xx Wireless Software Development Board (MSC-DBSB8) is designed for code development. A second platform, the WDS Loadboard, may also be purchased allowing for exhaustive RF lab based testing. The Loadboard can be bought under the part number MSC-DKLB2 but also within the ISM-DK3 kit.

Figure 1. MSC-DBSB8 Software Development Board (SDB)

Figure 2. MSC-DBLB2 (Not Included) Testing Platform for controlled Lab Tests (Loadboard)

Both boards come with the Silicon Labs standard 40-pin socket for connecting standard EZRadio® and EZRadioPRO™ evaluation testcards such as the Si4432-DKDB1. The onboard C8051F930 comes preloaded with sample firmware to demonstrate a packet-based wireless link between two systems.

The MSC-DBSB8 C8051F9xx software development board includes:

- One 40-pin socket for EZRadio and EZRadioPRO testcards
- C8051F930 microcontroller preloaded with demonstration software
- Standard debug connector for Silicon Labs C8051 programming and debugging
- 4 buttons and 4 LEDs for custom purposes
- LCD display for setup parameters and information display
- RS232 interface via a 9-pin DSUB male connector
- USB type B connector with Silicon Labs CP2102 USB > Serial Converter onboard
- On board 3.3 V PSU
- 5 x 19 through hole breadboard area for customer's application

Copyright © 2009 by Silicon Laboratories

TABLE OF CONTENTS

Section **Page**

1. Introduction 1
2. Power Supply 5
2.1. On Board PSU 5
2.2. External PSU 5
2.3. Powered by USB Port 5
3. System Introduction: MSC-DBSB8 ICD Connector 6
4. System Introduction: Schematic (MSC-DBSB8) 8
5. Typical Testboard Schematic (Si443x Testcard) 10
6. Using the SDB with a Standard Testcard 11
7. Radio Evaluation 12
7.1. Demonstration Mode 12
7.1.1. Packet Error Rates (PER) 12
7.1.2. Screen 2: Setting Up the RF Parameters 14
7.1.3. Screen 3: Setting up Further RF Parameters 14
7.1.4. Understanding Antenna Diversity and Where to Use It 16
7.1.5. Packet Length 17
7.1.6. Max Packets 18
7.1.7. Screen 5: The Ready Screen 18
7.1.8. Running the Demonstration 20
7.2. Lab Mode 22
7.2.1. Transmitter Evaluation Setup 26
7.2.2. Receiver Evaluation Setup 26
7.2.3. Transmitter Measurements 26
7.2.4. Results (CW Tests) 29
7.2.5. PN9 Measurement 30
7.2.6. Results (PN9 Tests) 32
7.2.7. Receiver Measurements 33
7.2.8. Results (BER Test) 35
7.2.9. Packet Error Test 36
7.2.10. Results (PER Test) 38
7.3. Additional Information 39
7.3.1. USB Communications 39
7.3.2. Packet Structure 40
8. Custom Software Development 41
8.1. Program Structure 41
8.1.1. Basic Code Overview 45
8.2. Basic Hardware Connections 46
9. Main 47
9.1. Flow Chart Main ( ) 47
9.2. Main Source File 48
10. Si4432 53

10.1. Flow Chart 55
10.1.1. RF Packet Received( ) 55
10.1.2. RFTransmit( ) 55
10.2. Si4432 Header File 56
10.3. Si4432 Source File 60
11. C8051 67
11.1. C8051 Header File 67
11.2. C8051 Source File 70
12. Troubleshooting 73
Document Change List 74
Contact Information 76

2. Power Supply

The board has three power options. The user can select between these options by the supply source selector switch (SW1).

2.1. On Board PSU

The on board PSU supplies 3.3 VDC. In this mode, the board should be powered by a standard 9 V ac or 9–12 V dc adapter.

2.2. External PSU

In this mode, the board can be powered via the direct dc supply connector by an external PSU. Any supply voltage can be used in the 3.3–4 V range. Polarity is marked on the PCB.

2.3. Powered by USB Port

In this mode, the board can be powered via the USB connector.

Note: When using the white LED Flash option, it is recommend to use an alternative power supply.

3. System Introduction: MSC-DBSB8 ICD Connector

Table 1. Debug Connector

Pin #	Description
1	VDD (3.3 V)
2	GND
3	GND
4	P2.7
5	RESET
6	P2.7
7	RST/C2CK
8	—
9	GND
10	—

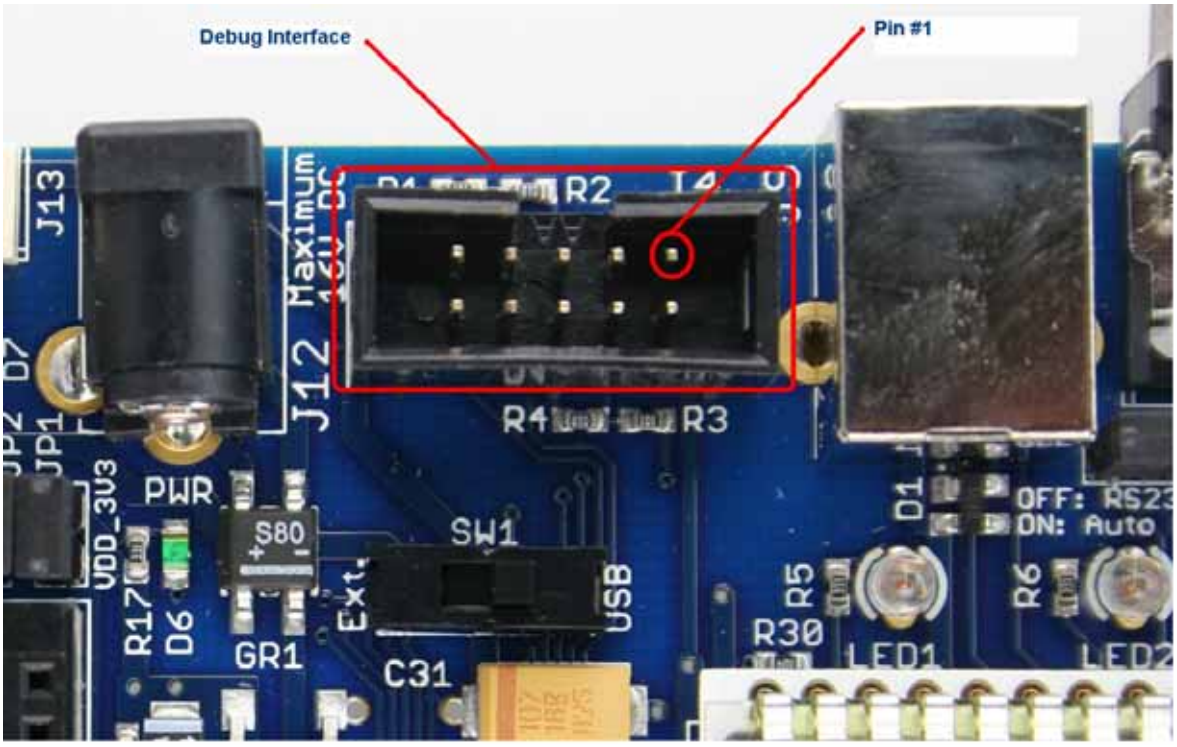


Figure 3. Debug Connector (Emulator and Programmer Interface)

Table 2. 40-Pin Testcard Connector (J5)

Pin #	Description	Pin #	Description
1	J6/1 (SPI_MOSI)	21	GND
2	J7/1	22	J15/1
3	J6/2 (SPI_SCK)	23	GND
4	J7/2	24	J15/2
5	J6/3 (RF_NSEL)	25	J8/1
6	J7/3	26	EBID port (SPI_MOSI)
7	J6/4	27	GND
8	J7/4	28	EBID port (SPI_MISO)
9	J6/5	29	J8/2
10	J7/5	30	EBID port (SPI_SCK)
11	J6/6	31	GND
12	J7/6 (RF_NIRQ)	32	EBID port (EE_NSEL)
13	J6/7 (PWRDN)	33	J8/3
14	J7/7(RF_NIRQ)	34	J15/3
15	J6/8 (GPIO)	35	GND
16	J7/8(SPI_MISO	36	J15/4
17	VDD (3.3 V)	37	J8/4
18	VDD (3.3 V)	38	J15/5
19	VDD (3.3 V)	39	GND
20	VDD (3.3 V)	40	J15/6

4. System Introduction: Schematic (MSC-DBSB8)

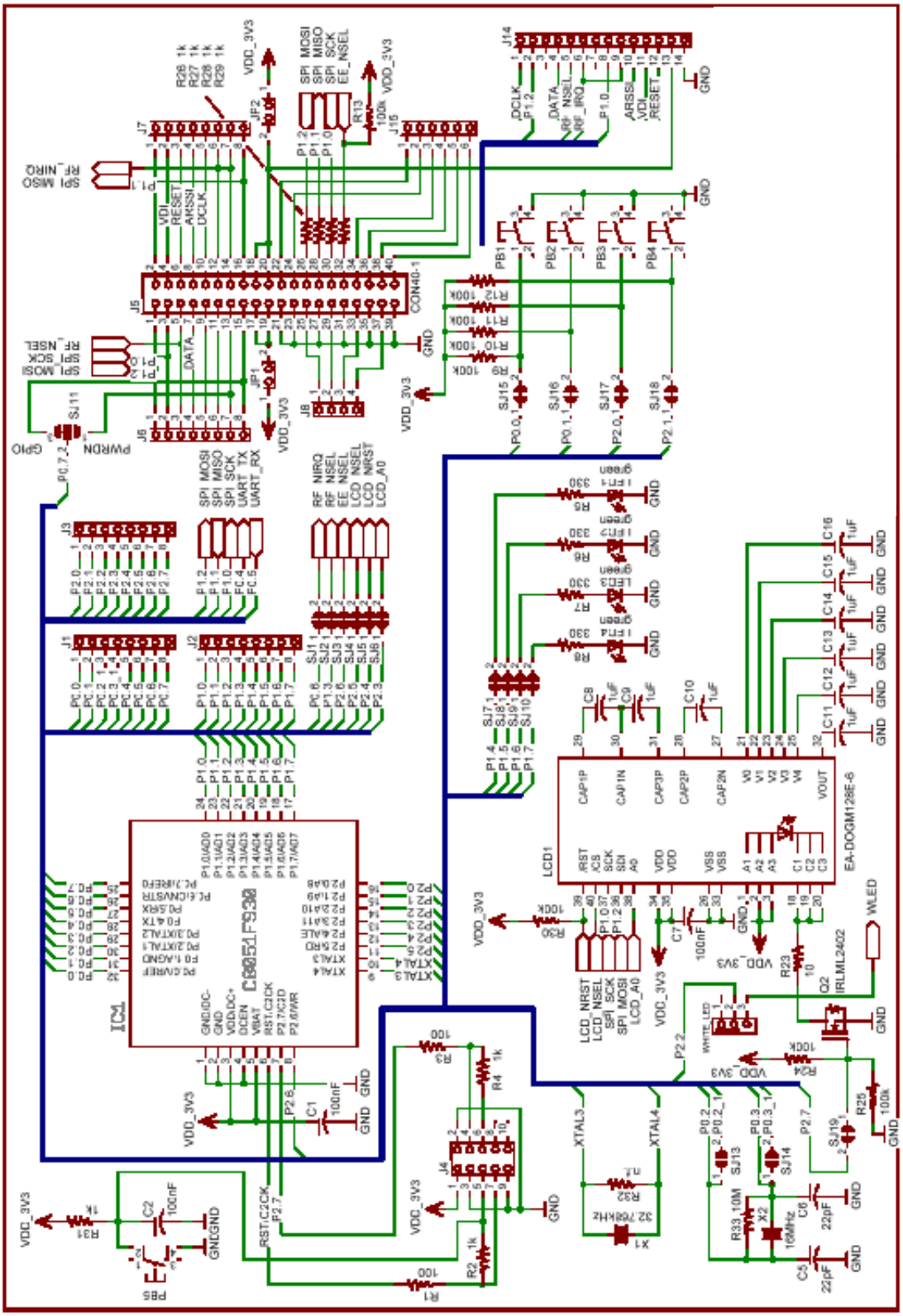


Figure 4. MSC-DBSB8 Schematic (1 of 2)

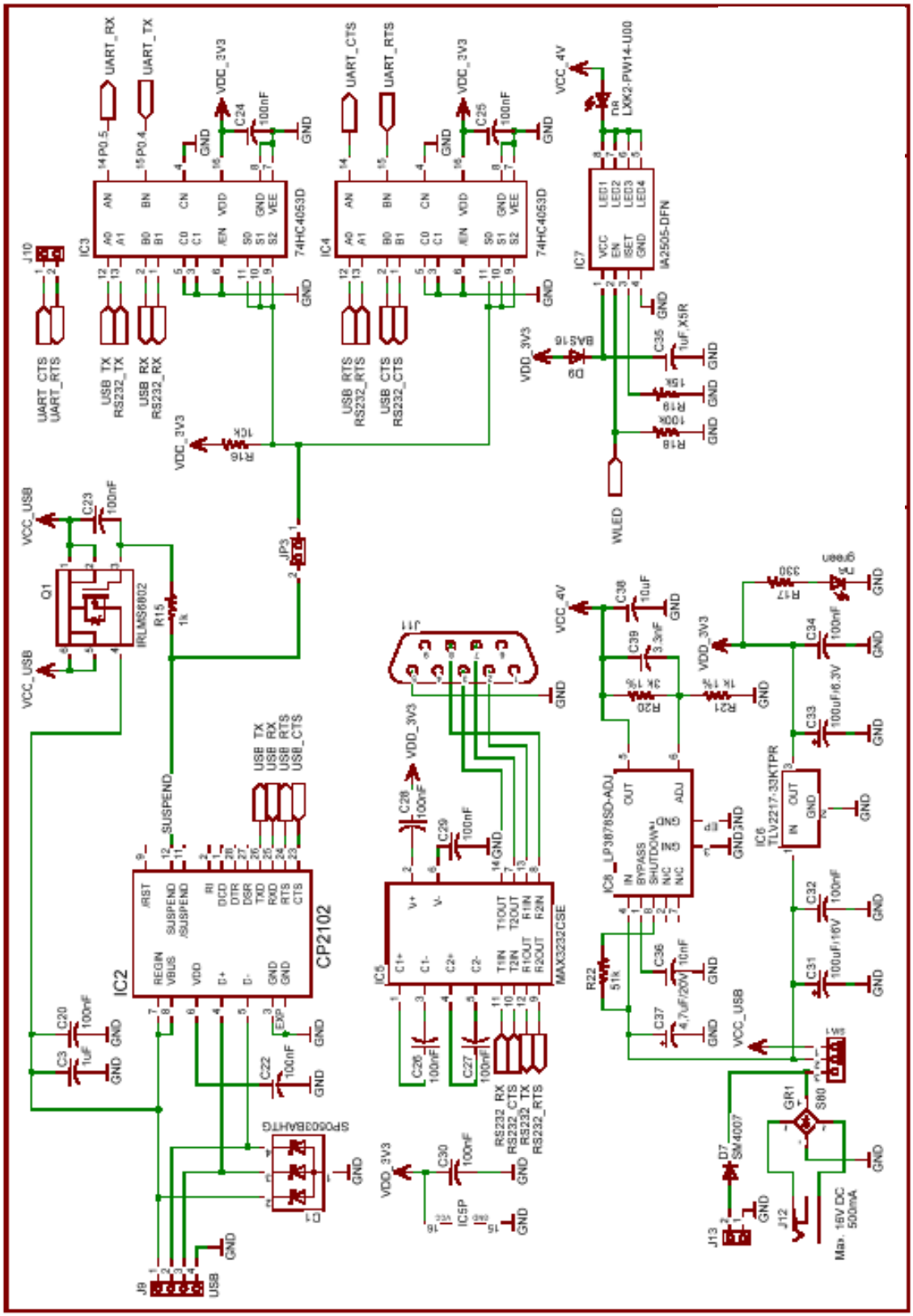


Figure 5. MSC-DBSB8 Schematic (2 of 2)

5. Typical Testboard Schematic (Si443x Testcard)

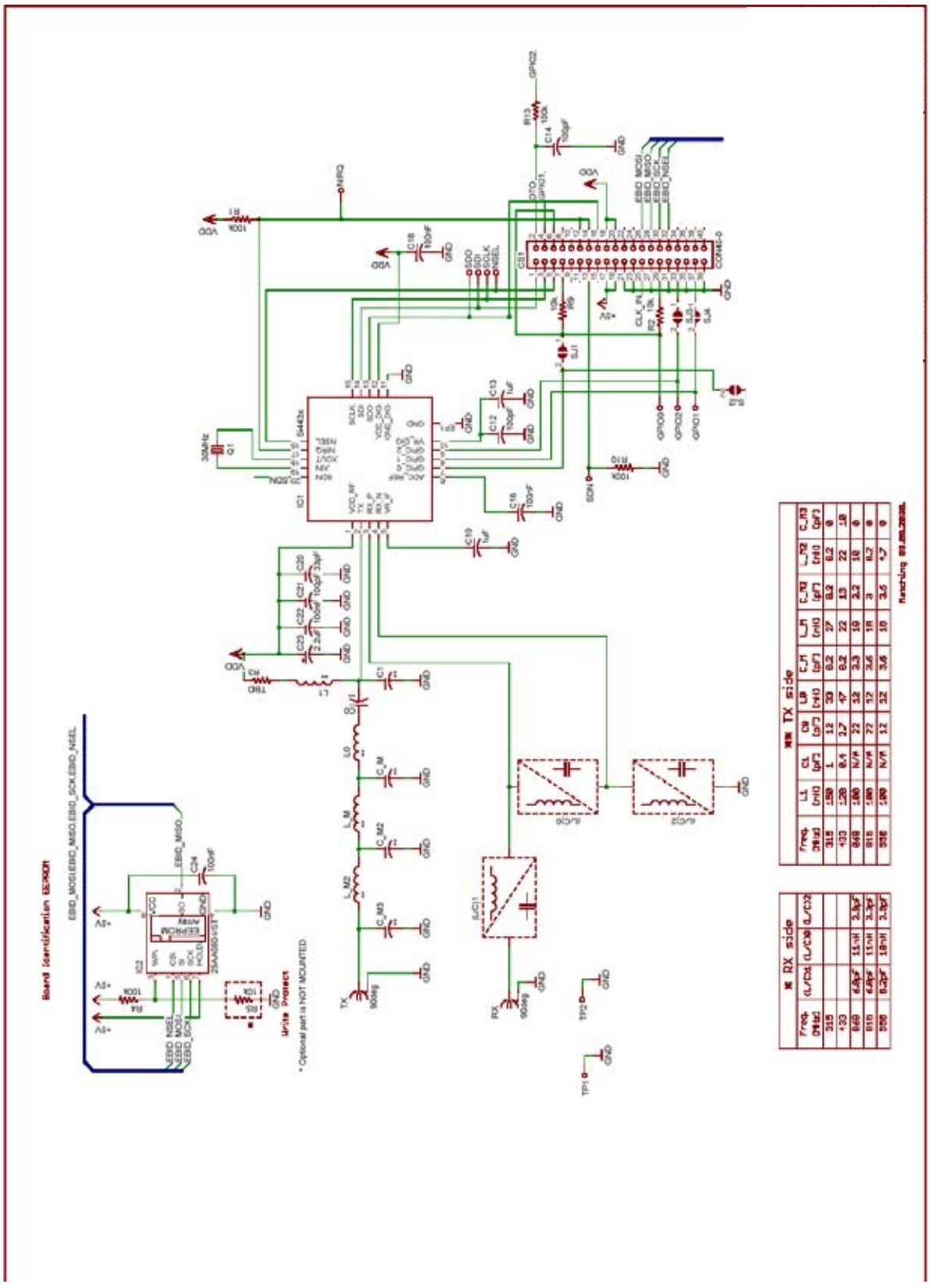


Figure 6. Si443x Testcard Schematic

6. Using the SDB with a Standard Testcard

The standard EZRadio or EZRadioPRO testcards that are typically plugged into the MSC-DBLB2 Loadboard when engineers are performing RF tests on the radio ICs can also be plugged into the 40pin socket on the Software Development Board (SDB), as demonstrated below.

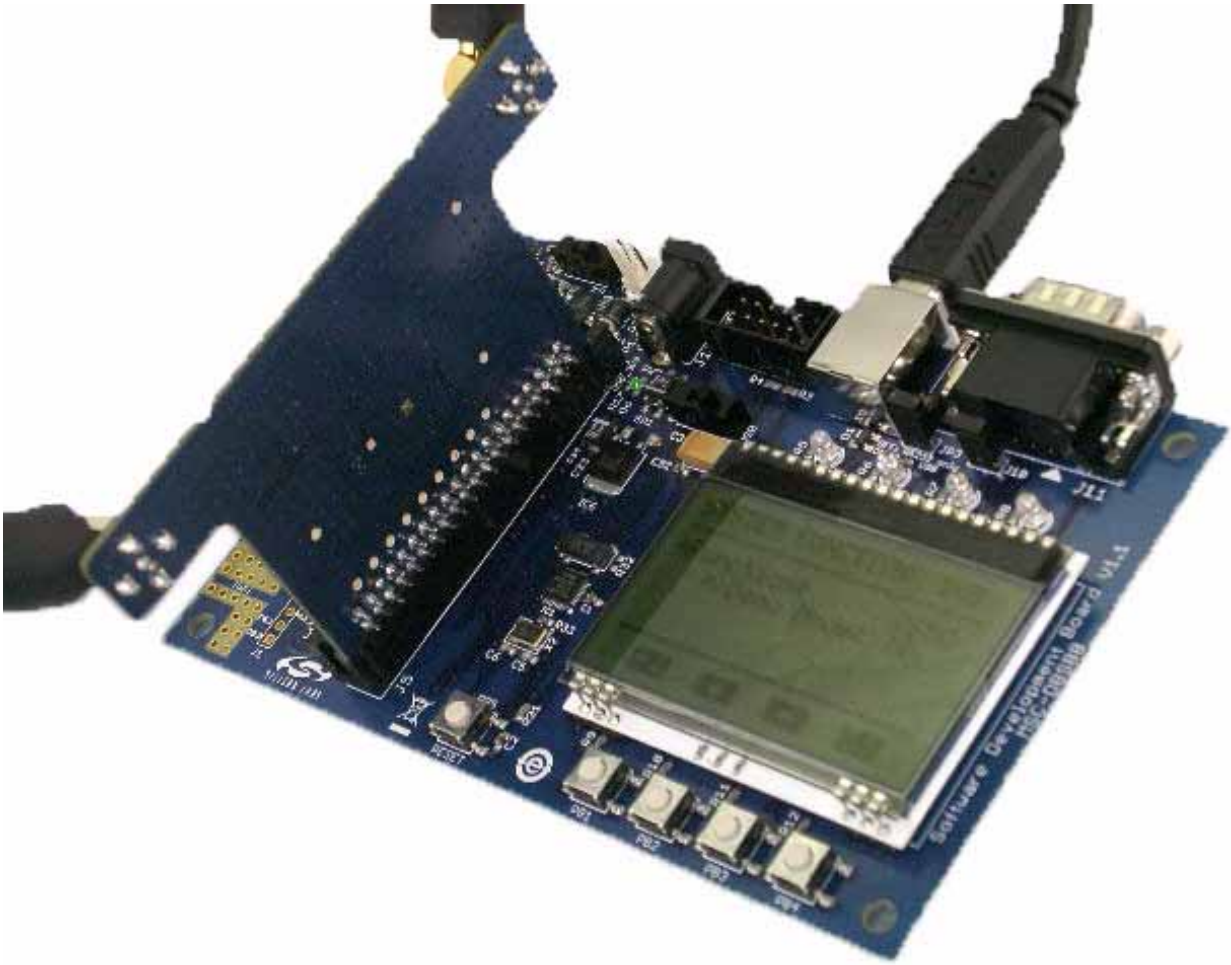


Figure 7. Software Development Board (MSC-DBSB8)
with a Standard Silicon Labs Testcard Installed

7. Radio Evaluation

7.1. Demonstration Mode

When shipped, the MSC-DBSB8 comes with example firmware, which is used to demonstrate the basic RF capabilities of Silicon Labs' RFIC. In the current public release of this firmware only the EZRadioPRO Si4432 transceiver is supported, later releases are intended to demonstrate the ever increasing number of products from Silicon Labs.

Newer firmware versions of the factory firmware may be available on the Silicon Labs website or via the WDS CDROM.

Introducing the PER demonstration (Version 3.xr Firmware)

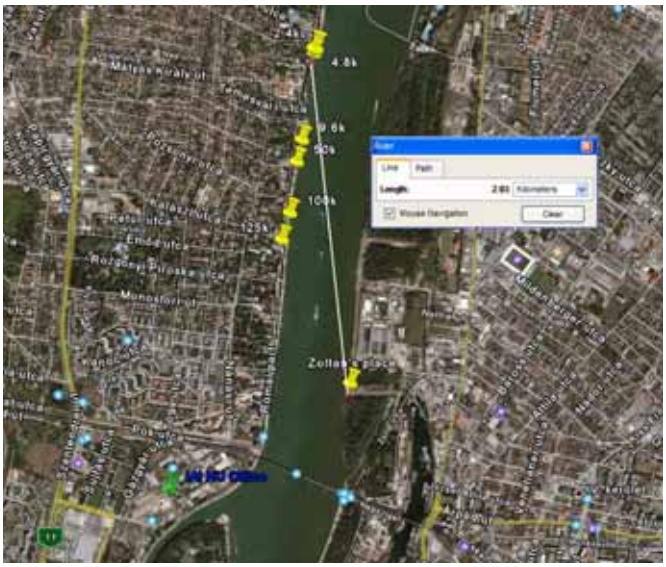
Reference firmware v3.xr is designed to show the Si443x in a packet error rate test demonstration. This firmware is preloaded on to the c8051F930 microcontroller, but it is also available on the WDS CD-ROMs in the SDB section. Source code to the Silicon Lab's firmware is also available in the same location, and is provided AS-IS for reference purposes.

7.1.1. Packet Error Rates (PER)

Packet Errors are common place in wireless communications. In real life applications, these errors are handled through the use of acknowledgements and retries. In the demonstration software such techniques are NOT implemented such that system designers can understand range limitations in various environments, allowing them to design robust protocols into their designs.

In order to use this demonstration, users should start the demo then move the two boards apart until packet errors can be seen to start appearing and the average PER is in the order of 5%. At this point, the PER demonstration should be cleared, and the test should be left alone to be re-run without being disturbed. It is expected that the PER % will be reduce since the environmental factors will be more constant. Re-run the previous steps until a 2–3% PER is found regularly at a given range. At this point, and with the current environmental conditions, you are at a range where a higher level protocol is required. Silicon Labs has found that at a low datarate and sending approximately 5000 packets, a 2–3 km range is attainable.

The demonstration shown below was performed along the Danube River in Budapest, Hungary using the Si4432.



Data rate	Range
2.4 kbps	2.05 km
4.8 kbps	2.03 km
9.6 kbps	1.5 km
50 kbps	1.37 km
100 kbps	1.1 km
125 kbps	0.96 km

Figure 8. Example Range-vs.-Data Rate
Location: Danube River in Budapest, Hungary

Note: The following screen shots reference firmware version 3.5r, screen shots may differ slightly from the version you have received.

Figure 9. Welcome Screen

Users may by pass the introduction screen by pressing any of the pushbutton 1-4, alternatively wait until the screen times out and moves on itself.

The following screen allows users to select the mode of operation:

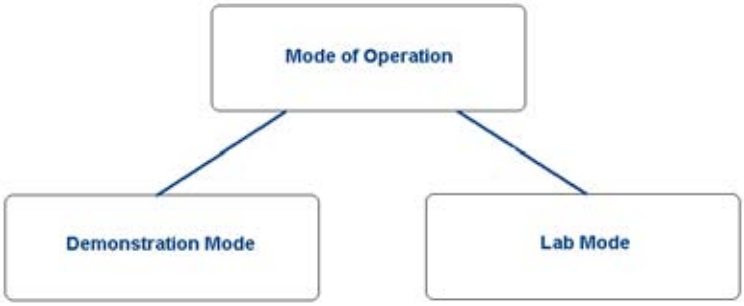


Figure 10. Operating Modes

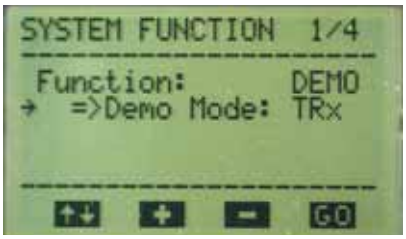


Figure 11. Screen 1: Demonstration Mode

Since the SDB's firmware recognized the Si4432 test card inserted into the 40 pin socket the appropriate modes of operation are presented on the menu system—in this case TRx (Transceiver). It is possible however to operate a transceiver in a RX (Receive) or TX (Transmit) mode also so menu features allow users to override the functionality. Menu's are driven through the push button's 1–4 under the LCD—the function of each button is shown on the screen.

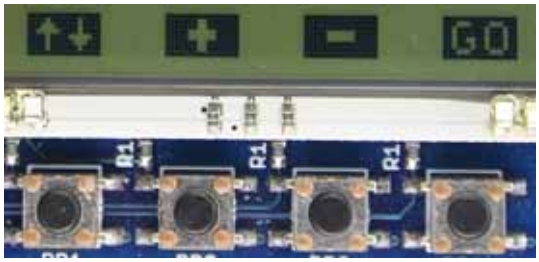


Figure 12. Push Buttons and LCD Labeling

The Up/Down button (PB1) moves the arrow up and down the LCD screen.

Figure 13. Active Menu Item Pointer

The arrow is used to highlight the function that will change when the user presses the '+' and '-' buttons. In the screen shown above, 'Function:' is highlighted, pressing the '+' and '-' buttons will switch the mode of operation between demonstration mode and lab mode.

Moving the arrow down to the ' =>Demo Mode: TRx' will allow the user to change between TRx, RX and TX functionality via the '+' and '–' buttons.

For the purpose of this introduction to the system we shall operate in 'Demonstration Mode'. Please ensure Demo mode is selected and TRx is enabled. The same selections should be made on both of the software development boards.

<Press the GO button>

7.1.2. Screen 2: Setting Up the RF Parameters

Figure 14. Screen 2: RF Parameters Menu

The 2nd screen sets up the RF parameters for the link. Adjustments are made by scrolling up and down the LCD using PB1, highlighting the relevant item to be changed and using the '+' and '-' buttons accordingly.

This introduction assumes the following settings:

Data Rate: 2.4 kbps

Modulation: GFSK

Frequency: 868.00 MHz

The same selections should be made on both of the software development boards.

<Press the GO button>

7.1.3. Screen 3: Setting up Further RF Parameters

On this screen, there may be a difference with respect to the screen shot available which is dependent on the testcard installed in to the 40 pin connector. Several testcard options are available for the SDBC-DK3 (see website for details). Typically, testcard variants include antenna diversity cards for use with antennas, single-ended Tx/Rx testcards (also intended for use with antennas), and split Tx/Rx testcards designed for use with coaxial cable to lab equipment (see "7.2. Lab Mode" on page 22 for more details).

Figure 15. Antenna Diversity Testcard (May be Ordered Separately)

Figure 16. Split TX and RX Testcard (Rx: Left SMA, Tx: Right SMA) for Use with Coaxial Cable and RF Test Equipment for Scientific RF Evaluation (May be Ordered Separately)

Figure 17. Single-Ended TX and RX Testcard

If an Antenna Diversity card is fitted then the 'Antenna Mode' option on the 3rd screen will be available, this line is automatically removed when non-antenna diversity cards such as the 4432-DKDB1 are inserted. With this options users have the ability to select antenna 1, antenna 2 or both (antenna diversity enabled). For typical operation select “1&2”.

This screen also allows users to setup the PA output levels of the of the radio. Figure 18 demonstrates an output power of +20 dBm selected.

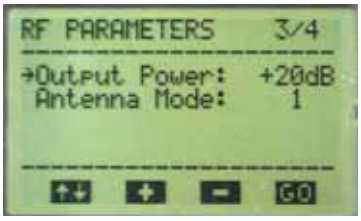


Figure 18. Screen 3: RF Parameters (Antenna Diversity Card Fitted)

7.1.4. Understanding Antenna Diversity and Where to Use It

Antenna Diversity is a technique that is use to improve the quality and reliability of a wireless link. The technique is particularly useful in city-like urban and indoor environments where clear lines of sight between the transmitter and the receiver cannot be achieved.

In environments where clear lines-of-sight (LOS) are not possible, signals reflect along multiple pathways (multipath) before being received, each reflection can introduce phase shifts, time delays, attenuations and distortions that can disrupt the quality of the signal which can cause problems for the receiver. Antenna diversity assists the receiver by allowing it to see the signal from two slightly different positions through the use of multiple antennas. Studies have shown that antenna diversity in both indoor and urban environments can recover 8–10 dB of the link budget that is usually lost to the environment when receivers use only single antenna implementations.

The technique however does not provide any major benefits in open environments and often this mis-understanding can cause confusion during range test exercises. In fact, due to the extra components used in the antenna switch it is possible that there maybe a small addition loss in the link budget thus reducing range. See Table 3 and Table 4.

Table 3. Illustrative Effects of Antenna Diversity in Indoor/Urban—Multipath Environments

	Single Antenna Implementation	Multiple Antenna Implementation
Transmit Power	+20 dBm	+20 dBm
Receive Sensitivity	–118 dBm	–118 dBm
Multipath Losses	–10 dBm	–2 dBm
Loss in Matching/Switch	–2 dBm	–3 dBm
Link Budget	126 dBm	133 dBm

Table 4. Effects of Antenna Diversity in Line of Sight (LOS)—Open Air Environments

	Single Antenna Implementation	Multiple Antenna Implementation
Transmit Power	+20 dBm	+20 dBm
Receive Sensitivity	–118 dBm	–118 dBm
Multipath Losses	0 dBm	0 dBm
Loss in Matching/Switch	–2 dBm	–3 dBm
Link Budget	136 dBm	135 dBm

It can be seen from Table 3 and Table 4 that while the effects of antenna diversity on LOS environments are negligible, the benefits in indoor/urban environments can significantly help create robust, higher quality robust links.

Figure 19. Screen 4: Setting Up the Node Parameters

The Self ID on each card should already be selected and is contained in the EBID eeprom on the testcard. The EBID is placed on the testcard so that our firmware and support GUI's (such as WDS) can recognize the characteristics of the board. The EBID is NOT required as part of the bill-of-materials in an end customers design. The EBID contains information such as the local ID, destination ID, matching network configuration and antenna/ test card configuration type.

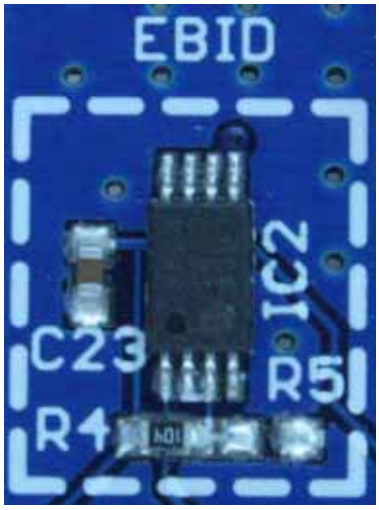


Figure 20. Test Card Characteristics EEPROM (EBID)

The SDBC-DK3 is initially configured with the same value for the destination ID and the self ID. In order to run the demo, you will need to configure the destination ID of the first board to that of the second and vice versa. This will allow the two SDB boards to communicate with each other. The ID is considered the address.

7.1.5. Packet Length

The node parameters screen also allows users to adjust the packet length so that they may:

1. Perform head-to-head comparisons with competitive radios
2. Learn the effects of packet length with respect to data-rate and robustness

Many things affect the robustness of a radio link and often the lack of understanding of some of these variables can skew test results quite substantially.

A good example of when this misunderstanding takes place is during head to head comparisons of different radio IC suppliers. By using the 'packet length' option in the menu, users may adjust the length of the packet to match such that they are looking at the same packet structure during their tests. (See section 7.7 for further details on the packet structure)

Once designers are satisfied with their head-to head comparisons of the radio IC's having based their tests on similar structures then designers can use the packet length option to experiment with their protocol. Packet length can have many tradeoffs with respect to power savings, robustness, data rates, processing overheads, etc.

Consider the following:

- During the transmission of long packets, there is an increased chance that a disturbance may occur somewhere along that packet—thus the need to implement good CRC checks.
- During the transmission of short packets, there is an increased chance that the entire packet may be lost during a disturbance—thus the need to implement more retries.
- Transmission of long/short packets with a slow data rates have good easily recognizable 1's and 0's but by comparison to fast data rates have a greater 'on time' and may use more power but retries may be less (in turn saving power).
- Transmission of long/short packets with a fast data rate may have a less 'on time' but retires may be greater, antenna diversity may help reduce the multipath effects.

The trade offs in radio applications are many and the list above is by no means the only possible scenarios, every application has its own list of acceptable trade offs and through the use of menu options such as data-rate, packet length and antenna diversity engineers can learn to best understand what options work for them and their environments.

7.1.6. Max Packets

So that designers can scientifically qualify the aforementioned trade offs, careful experimentation in applicable application-like environments is recommended, the 'Max. Packets' menu option allows designers to select the number of packets used to generate a Packet Error Rate (PER) result.

- A large number of packets (1000–9999) allows for a good averaged result but can take time particularly at low data rates.
- A small number of packets (100) allows for a quick environmental assessment to be made prior to an exhaustive test.

7.1.6.1. Example—Typical Usage

The designer will arrive at their test site and run a 100 packet test experiment prior to exhaustive testing. If the results are in accordance with previous tests then the environment is similar to those of the previous occasion. If there is a substantial difference in the results of a 100 packet test with any previous occasions results then new environmental factors are playing into tests and these should be recorded as comments so that results from exhaustive testing is better understood.

<Press the GO button>.

7.1.7. Screen 5: The Ready Screen

The ready screen on the LCD is the final step before starting your experimentation. The ready screen labels the LEDs 1–4 according to the function they will perform. In this demonstration they are TX, RX, Antenna 1 and Antenna 2. Note if you have disabled any of the antenna's or are using a none antenna diversity card then the associated antenna is not represented by the LEDs.

Figure 21. Non-Antenna Diversity Testcard

Note: Only one antenna highlighted on the top row and NO-ANTDIV shown in the second line of text.

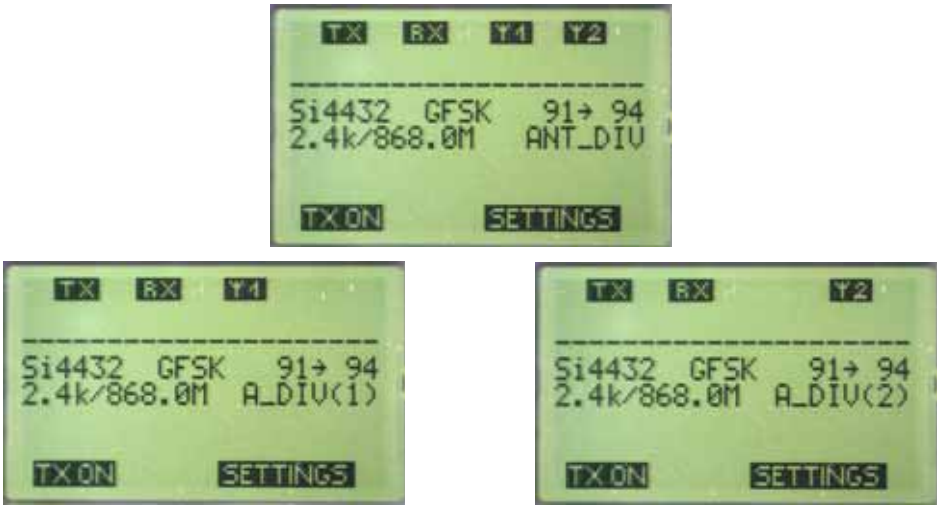


Figure 22. Antenna Diversity Testcard

Note: Depending on antenna selection the relevant antennas are shown on the top row and either ANTDIV, A_DIV(1), or A_DIV(2) is shown in the second line of text.

The ready screen is designed to allow you to review the settings on both boards. In the example shown above, we illustrate the following configuration:

- Our first board with ID 091 is sending its message to board with the ID 094
- At 2.4 kbps
- With frequency 868.00 MHz

Using the ready screen, we can see that there is an error on the split TX/RX board in that it is configured with a selfID of 054, this does not match the DestinationID on the antenna diversity board. From the 'ready screen' we can update the antenna diversity board accordingly. To do this, we can press PB3 or PB4, which are highlighted as 'SETTINGS' where we can re-run the setting accordingly.

Once the settings are correct we can run the demonstration accordingly.

If the user has the ability to see both ends of the link then a white LED driver is made available on the SDB with which you can enable a high brightness LED to make visual confirmation easier of the remote board.

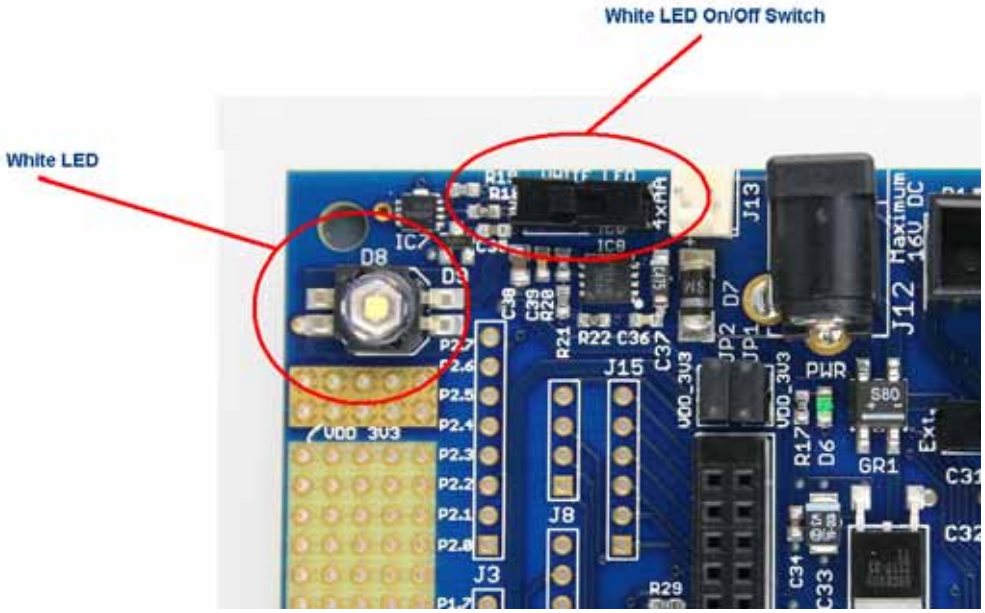


Figure 23. White LED Control

This feature is only manually enabled since the brightness of the LED may be distracting when on desk operation is being implemented.

7.1.8. Running the Demonstration

Longer tests provide better averages, but in the interest of time, this demonstration sends only 1000 packets. Users that modify the code can send as many or as few packets as they wish. The fact only 1000 packets are sent can cause higher PER percentages since most of the dropped packets will occur while the user is setting up the demonstration because he himself will absorb much of the radiation and add to multipath and fading effects. As users become more familiar with radio it is highly recommended a greater number of packets be sent such that a better average can be generated.

To run the demonstration, place one SDB in a fixed location. (If testing an antenna diversity system, place the antenna diversity unit in the fixed location while using a single-ended testcard as the roaming testcard). The SDB kept in the fixed location should be considered the “base unit”. This unit will act as the base unit. On the mobile unit, the unit with the split TX and RX, press 'TX ON' and walk away from the base unit until the PER settles around 5-7%. When the PER has settled and all 1000 packets are sent press the CLEAR button and run the test again but avoid being to close to the demo during the second test. If the test completes with 0-2% PER then the test should be run again with a greater range, if 4–6% then run again but with a lower range. Once you reliably get 2-3% at the end of the series of tests then, in that environment this can be considered the typical range in an application with limited error handling - as is the purpose of this experiment.

Good protocols and handling of dropped packets enable users to get much greater ranges.

During early experimentation, users may notice that the LCD and LEDs show information that represents antenna strength.

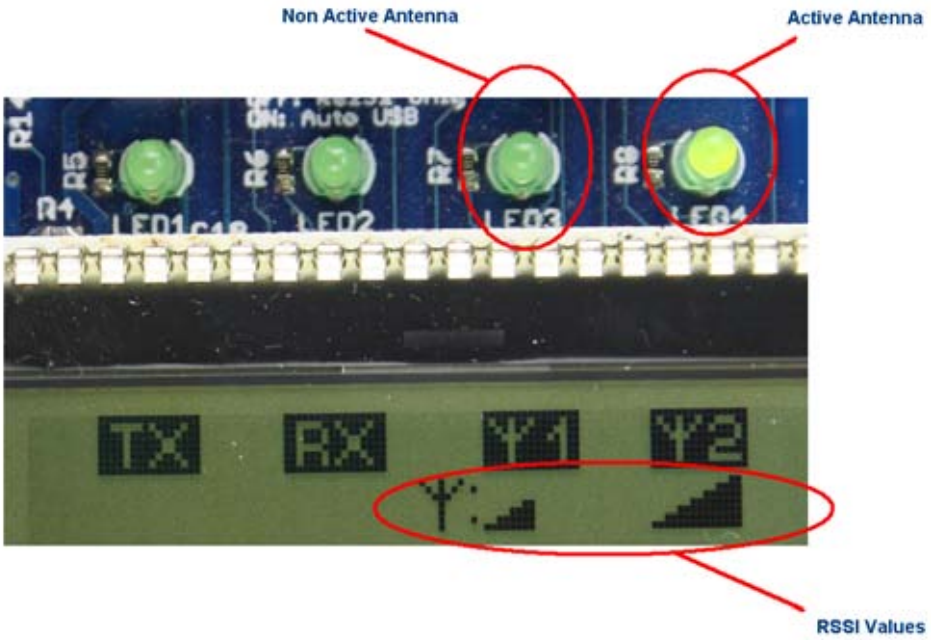


Figure 24. Active Antenna and RSSI Indications

7.2. Lab Mode

The lab mode is intended for users who want to evaluate the performance of the Silicon Labs RFICs, supported through the shipping factory firmware on the SDB platform.

Lab mode is intended so users can perform simple evaluations, such as:

- Transmitter Evaluation
  - Output power
  - Spectrum analysis
- Receiver Evaluation
  - BER Sensitivity
  - PER Sensitivity
  - Receiver parameters:
    - Automatic Frequency Control
    - Blocking
    - Selectivity

Using lab mode, users can independently evaluate transmit and receive performance. Table 5 lists the test cards available for ordering.

Table 5. Test Cards Available for Ordering

Type	Matching Network Configuration	Part Number
Transceivers	High band	4432 – DKDB1
		4431 – DKDB1
	Low band	4432 – DKDB5
		4431 – DKDB5
Receivers	High band	4330 – DKDB1
	Low band	4330 – DKDB5
Transmitters	High band	4032 – DKDB1
		4031 – DKDB1
	Low band	4032 – DKDB5
		4031 – DKDB5

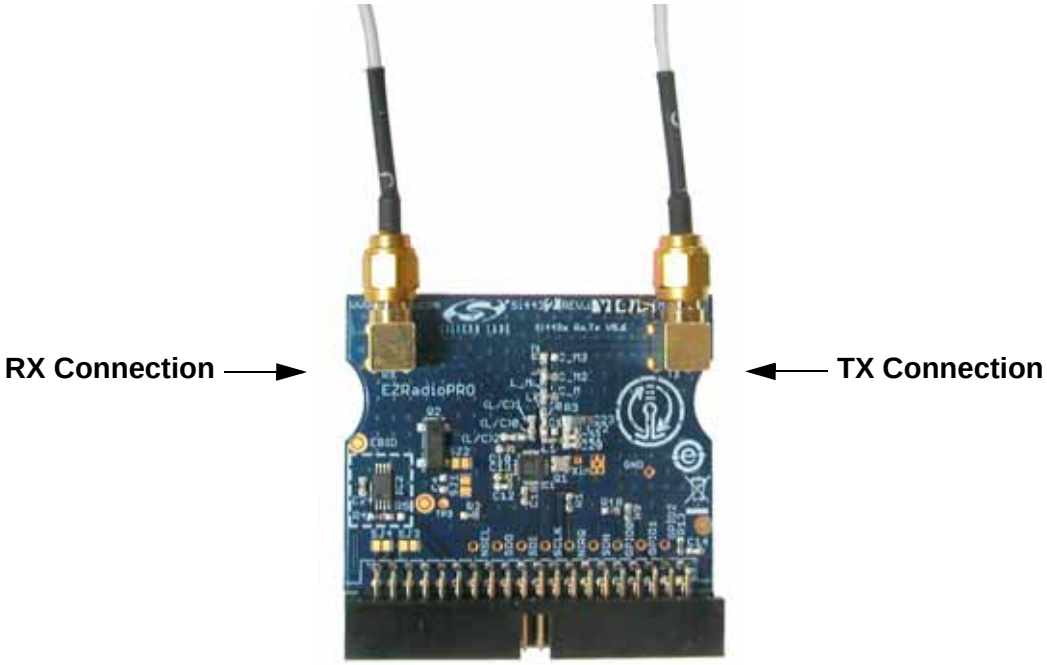


Figure 25. 4432-DKDB1 - Split TX/RX Antenna Card Using Coaxial Cable

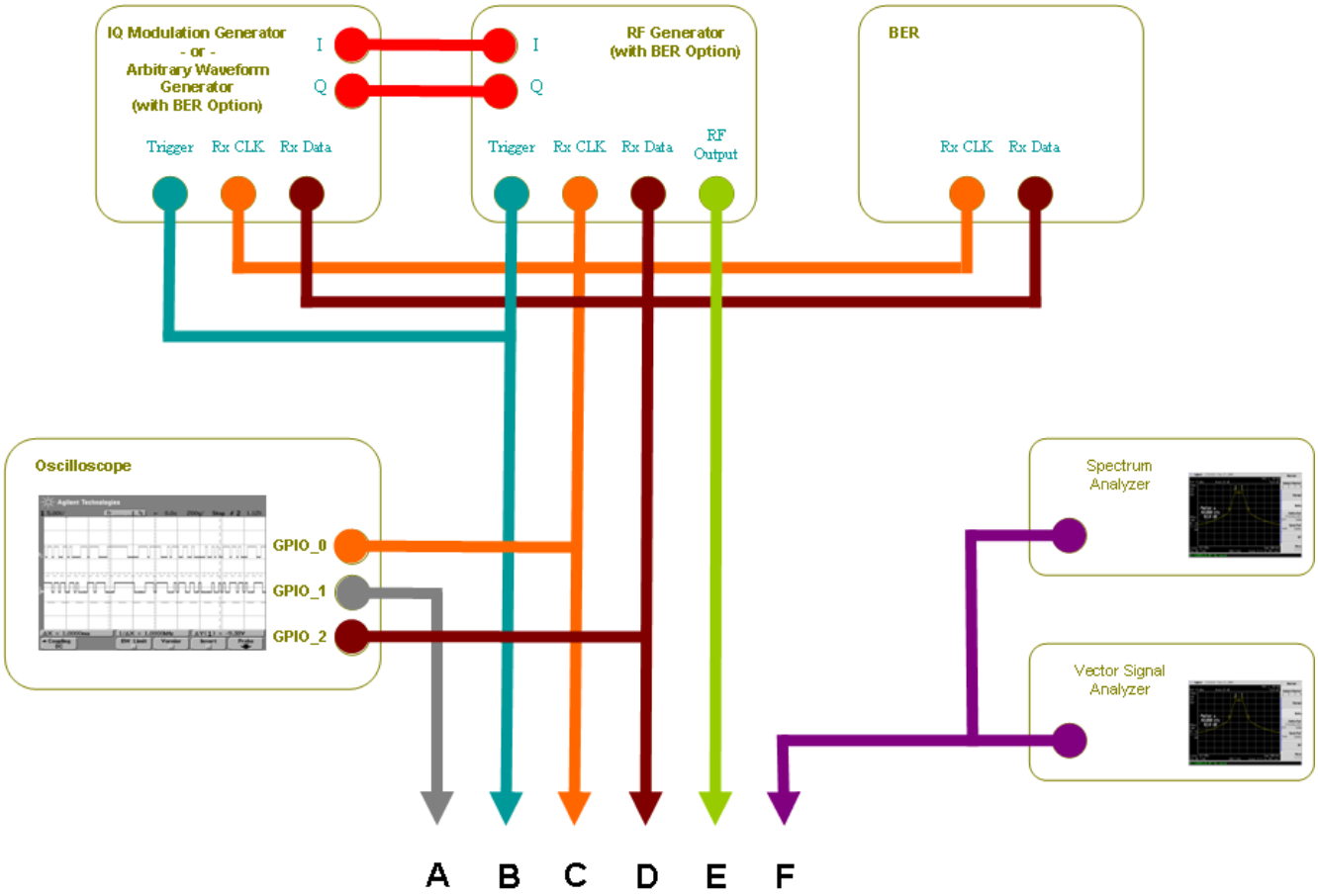


Figure 26. Lab Equipment Connection Diagram

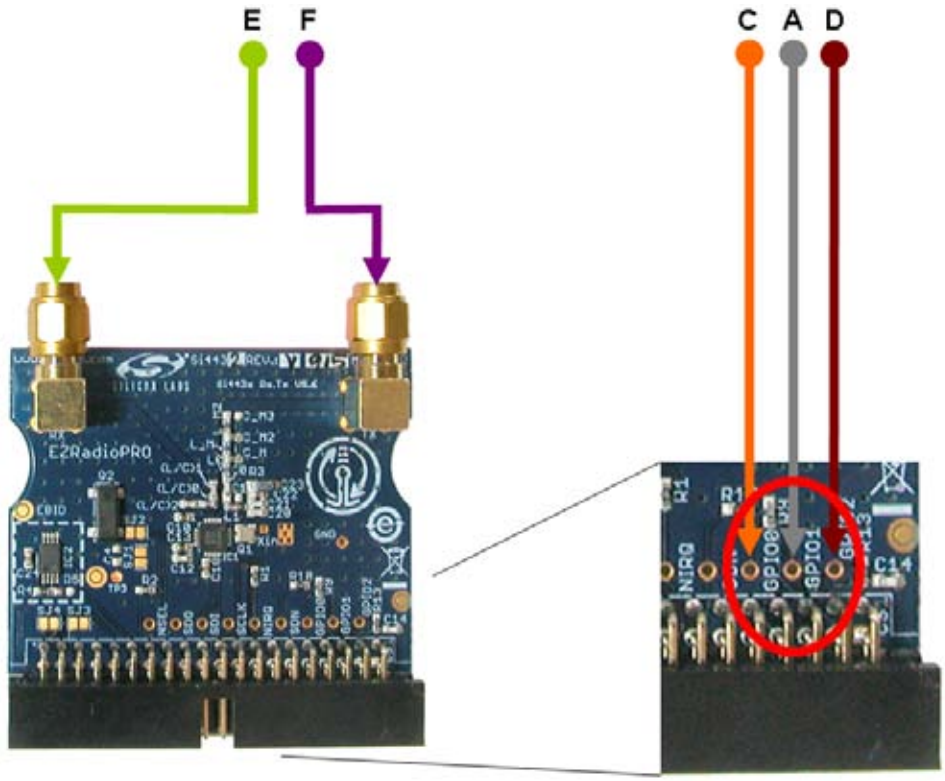


Figure 27. Test Card Connection Diagram

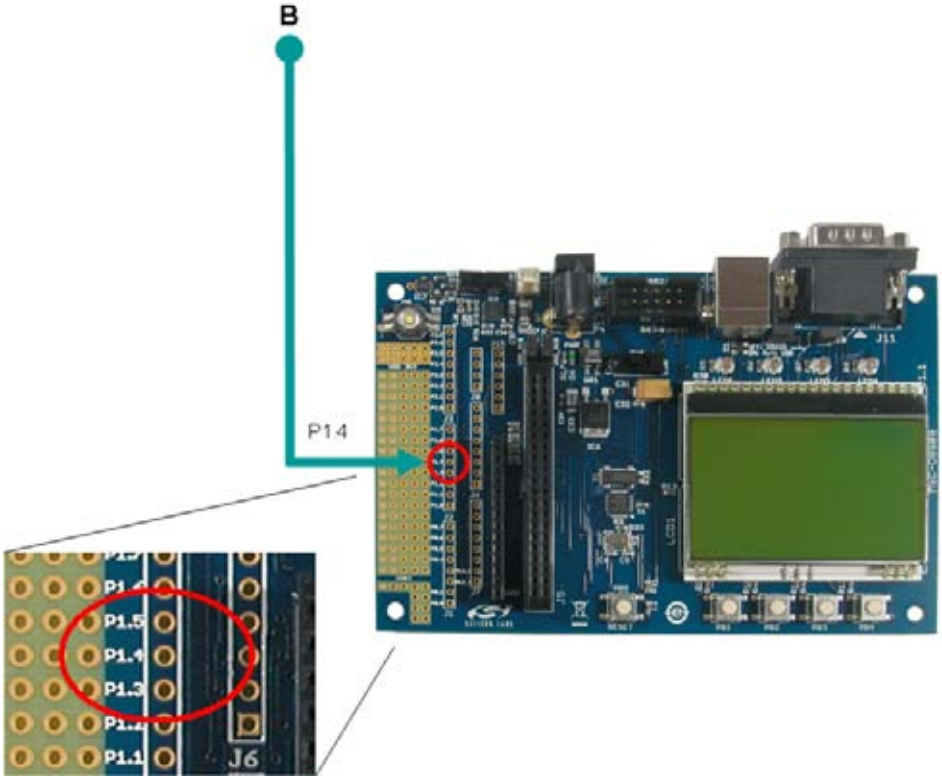


Figure 28. SDB Connection Diagram

7.2.1. Transmitter Evaluation Setup

The transmitter output of the 4432-DKDB1 test board can be connected to a spectrum analyzer in order to evaluate output power and spectrum plots. Alternatively the transmitter output may be connected to a vector signal analyzer to evaluate conditions such as freq vs time.

The 4432-DKDB1 testcard also provides access to the radio's GPIO which can be used as test-points for the radios internal signals - see diagram above.

7.2.2. Receiver Evaluation Setup

Receiver evaluation can be performed by connecting the receiver port of the testcard to an RF signal generator. The RF generator may use a data source from an external IQ generator or from its internal memory depending on its feature set, often RF generators have a PN9 pattern option.

The 4432-DKDB1 testcard also provides access to the radio's GPIO which can be used as test-points for the radios internal signals - see test card connection diagram.

Two modes are typically used during evaluations:

1. Direct Mode: In this mode, data is continuously sent via a source such as a PN9 generator.
2. Packet Mode: In this mode, the data source is customized in a defined packet structure.
   Typically: Preamble + Sync_Word + Data payload + CRC.

Example of a 20 byte packet that can be received by the SDB firmware:

```
Preamble:
        10101010101010101010101010101010101010101010101010101010101010
Sync:   0010110111010100
Date:   00010100001101000011000000001001011001100010111110011101010101
        01010111011000010100111100101011110101000100001111010011010000010
        11110111100110100111000010010001111000011
CRC :   1101011110011000
```

7.2.3. Transmitter Measurements

7.2.3.1. CW Lab Mode

Using the CW Lab Mode, users may evaluate the following:

1. Output Power
2. Frequency Offset in Transmitter Output
3. Phase Noise

7.2.3.2. Test Method

Figure 29. Setup Screen (1 of 4)

1. Ensure “Lab Mode” is selected as the operating function.
2. Select CW.
3. Press <GO> to move on from this screen.

Figure 30. Setup Screen (2 of 4)

1. Select the appropriate frequency.
   When evaluating with CW, data rate and modulation have no effect.
2. Press <GO> to move on from this screen.

Figure 31. Setup Screen (3 of 4)

1. Select the appropriate output power required for the test.
2. Press <GO> to move on from this screen.

Note: If an alternate testcard is used, such as the antenna diversity test cards, users may see slightly different screen shots than those shown. Users must turn off the diversity function by selecting “antenna 1” and connecting to the appropriate antenna connector using 50 Ω coaxial cable.

Figure 32. Setup Screen (4 of 4)

1. Parameters on setup screen 4 are not relevant to CW evaluations. Silicon Labs recommends leaving them at their default values.
2. Press <GO> to move on from this screen.

In Figure 33, the runtime screen will summarize the current valid settings.

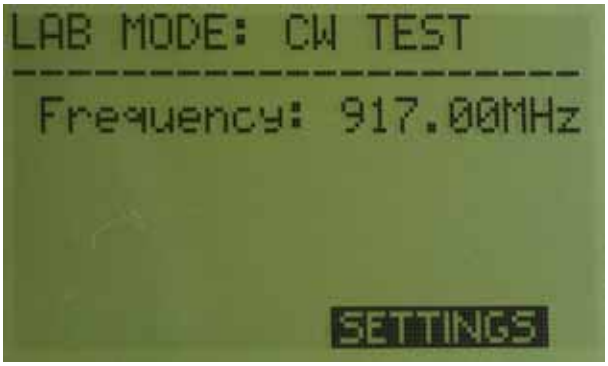


Figure 33. Runtime Screen

7.2.4. Results (CW Tests)

7.2.4.1. Output Power

1. Set the center frequency of spectrum analyzer to the frequency under test.
2. Set span to 10 MHz.
3. Measure the TX output power on displayed plot.

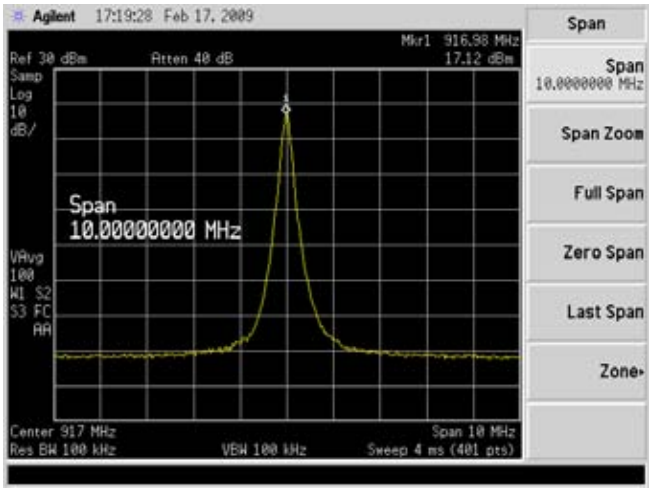


Figure 34. Spectrum Plot Showing CW Output at 917 MHz

7.2.4.2. Frequency Offset at Transmitter Output

1. Set the center frequency of spectrum analyzer to the frequency under test.
2. Set span to 100 kHz.
3. Measure the frequency offset between the expected frequency as selected in the menu and the actual TX output frequency.

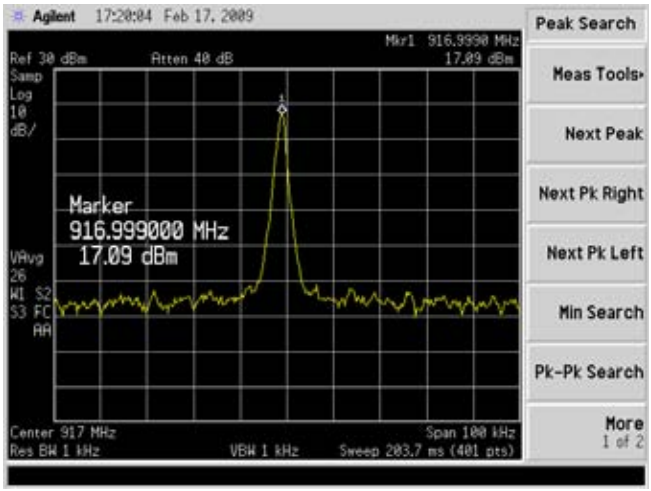


Figure 35. Typical Spectrum Plot Using a Silicon Labs Branded Testcard

In Figure 35, it can be seen that the frequency error is less than 1 kHz.

Silicon Labs testcards are designed to have a maximum frequency error of < 5 kHz.

7.2.4.3. Phase Noise

1. Set the Spectrum analyzer to “Phase Noise”.
2. Set the center frequency of spectrum analyzer to the frequency selected.
3. Set the spectrum analyzer to the desired span (typically from 100 Hz to 10 MHz span).

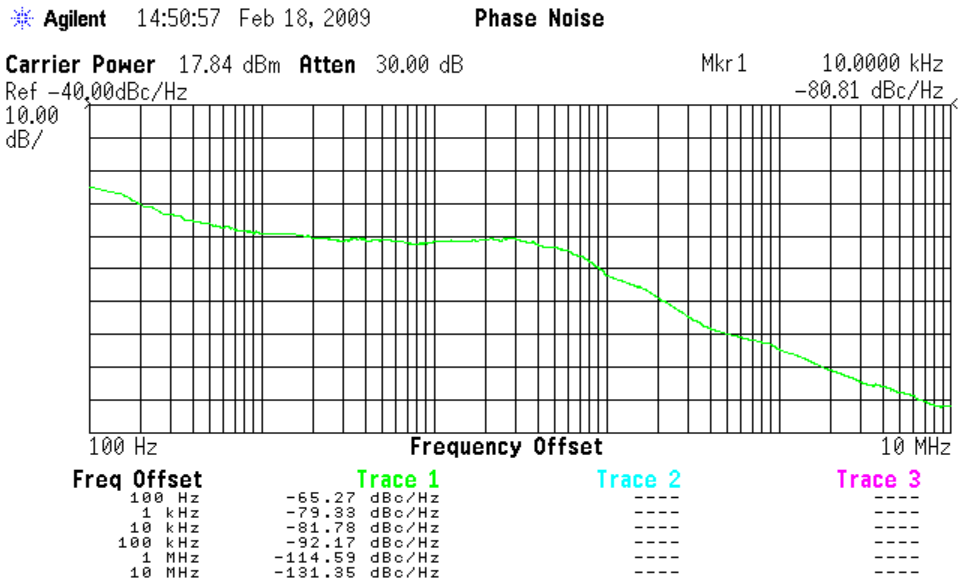


Figure 36. Typical Phase Noise Plot at 917 Mhz

7.2.5. PN9 Measurement

Using the PN9 Lab Mode, users may evaluate the following:

1. Tx output spectrum
2. Transmitter spectral mask

7.2.5.1. Test Method

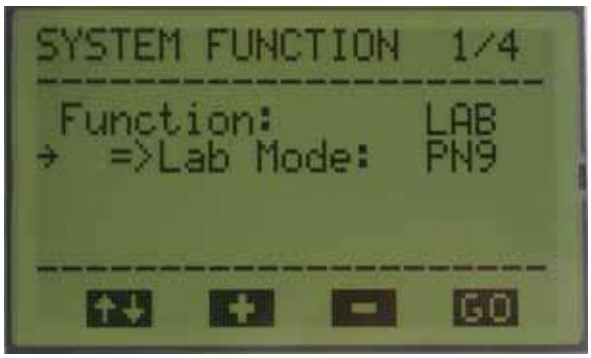


Figure 37. Setup Screen (1 of 4)

1. Ensure “Lab Mode” is selected as the operating function.
2. Select PN9.
3. Press <GO> to move on from this screen.

Figure 38. Setup Screen (2 of 4)

1. Select the appropriate frequency, data rate, and modulation.
2. Press <GO> to move on from this screen.

Figure 39. Setup Screen (3 of 4)

1. Select the desired output power.
2. Press <GO> to move on from this screen.

Figure 40. Setup Screen (4 of 4)

1. Parameters in Figure 40 are not relevant to PN9 evaluations. Silicon Labs recommends leaving them at their default values.
2. Press <GO> to move on from this screen.

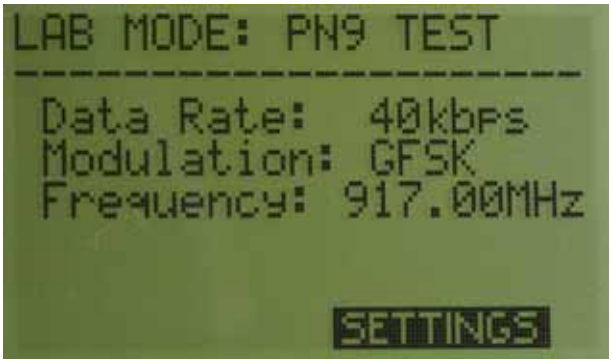


Figure 41. Runtime Screen

In Figure 41, the runtime screen will summarize the current valid settings.

7.2.6. Results (PN9 Tests)

7.2.6.1. TX Output Spectrum

1. Set the center frequency of spectrum analyzer to the frequency under test.
2. Set span to 500 kHz and observe the TX spectrum.

7.2.6.2. Evaluation of TX Spectral Mask

Using the PN9 mode, users can observe the TX spectrum to evaluate FCC/ETSI compliance.

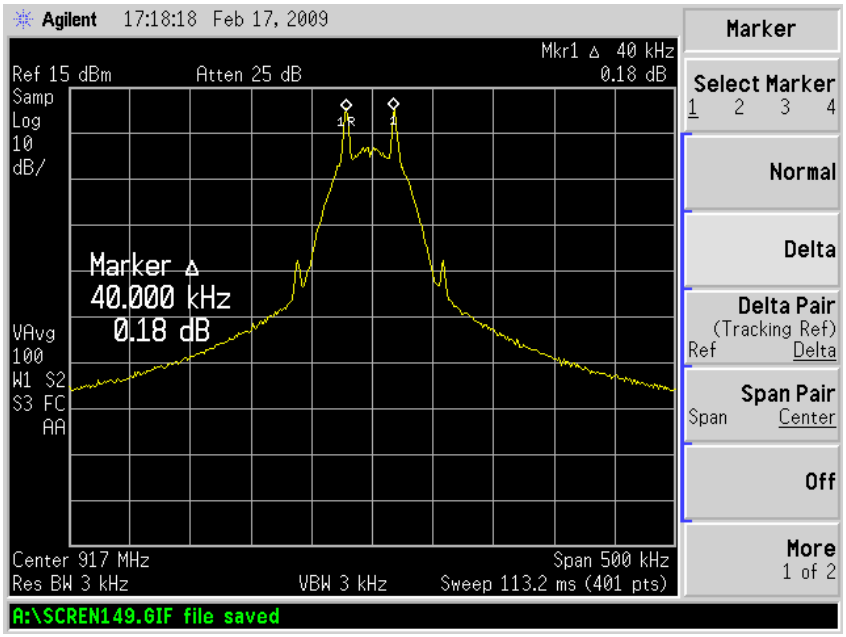


Figure 42. Spectrum Plot Demonstrating 40K Data Rate, 40K Deviation, GFSK Modulation

7.2.7. Receiver Measurements

7.2.7.1. Bit Error Rate Test

Using the BER Lab Mode users may evaluate the following:

1. BER Sensitivity.
2. Direct mode operation using a continuous data streams
3. Receiver modem parameters:
    i. Automatic Frequency Control
    ii. Blocking
    iii. Selectivity

7.2.7.2. Test Method

1. Set frequency, modulation type, data rate and deviation parameters on the RF signal generator.
2. Select the desired data source for the RF generator (e.g. PN9)
3. Connect the receiver's input to the signal generator's output.

Figure 43. Setup Screen (1 of 4)

1. Ensure Function is set to “Lab”
2. Set Lab Mode to “BER”
3. Press <GO> to move on from this screen.

Figure 44. Setup Screen (2 of 4)

1. Selections here should match those previously entered into the RF Signal Generator
2. Press <GO> to move on from this screen.

Figure 45. Setup Screen (3 of 4)

1. Parameters on setup screen 3 are not relevant to BER evaluations. Silicon Labs recommends leaving them at their default values.
2. Press <GO> to move on from this screen.

Figure 46. Setup Screen (4 of 4)

1. Parameters on setup screen 4 are not relevant to BER evaluations. Silicon Labs recommends leaving them at their default values
2. Press <GO> to move on from this screen

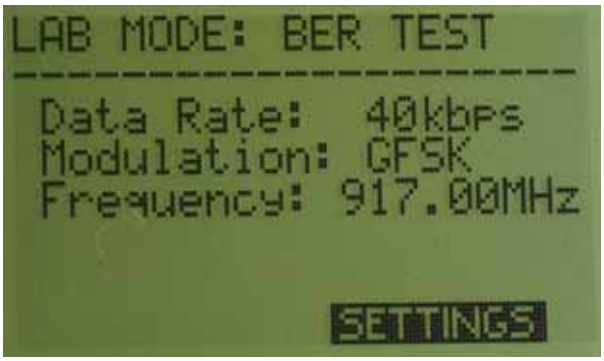


Figure 47. Runtime Screen

In the Figure 47, the runtime screen will summarize the current valid settings.

7.2.8. Results (BER Test)

7.2.8.1. BER Sensitivity Evaluation

BER results will be shown on the BER instrument or recorded by either RF generators or IQ modulators that have a BER option installed.

The top trace in Figure 48 demonstrates TX_Data as sent by a transmitter and the bottom trace is the data received on the GPIO pin. Please note that there is an expected shift caused by a delay between the data at the transmitter and the data received by the receiver.

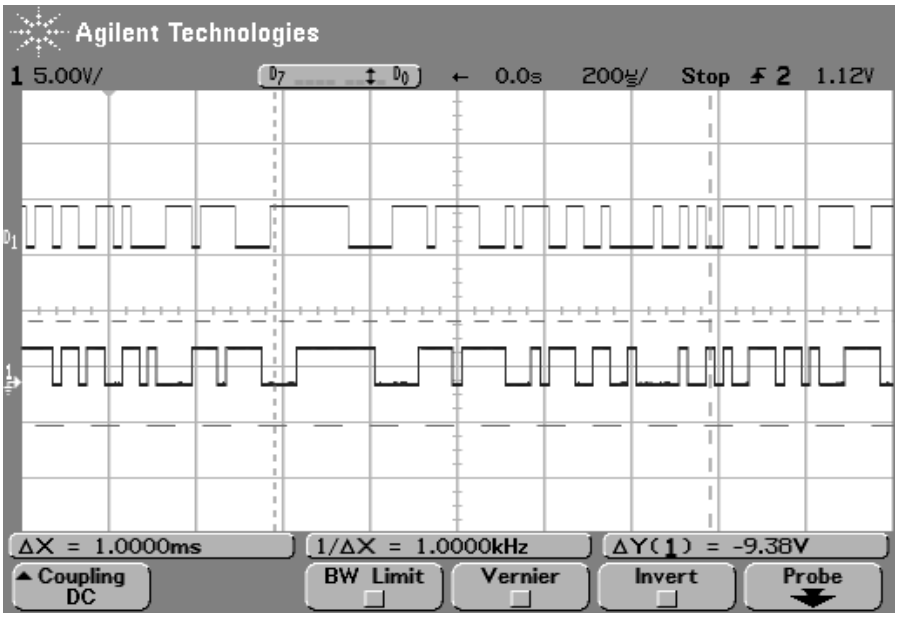


Figure 48. TX Data Sent and Received

7.2.8.2. Other Receiver Measurements

Using the test setup described above, users may also perform Automatic Frequency Control (AFC), Blocking and Selectivity tests. All the required parameters are controlled by the external RF generator.

7.2.9. Packet Error Test

Using the PER Lab Mode users may evaluate the following:

1. PER Sensitivity
2. FIFO mode using predefined packet structures (see data sheet for further details)
3. Receiver modem parameters:
    i. Automatic Frequency Control
    ii. Blocking
    iii. Selectivity
4. Connect the receiver's input to the signal generator's output.

7.2.9.1. Test Method

1. Set frequency, modulation type, data rate and deviation parameters on the RF signal generator.
2. Program a predefined packet into the generator using the format:
(preamble + sync_word(2DD4H) + data + CRC).
3. Set the signal generator to external single trigger mode.
4. The software development board (SDB) generates a trigger on test point P1.4, this should be connected to the RF signal generator's external trigger input. The P1.4 pin will enable the signal generator to send one packet for each trigger.

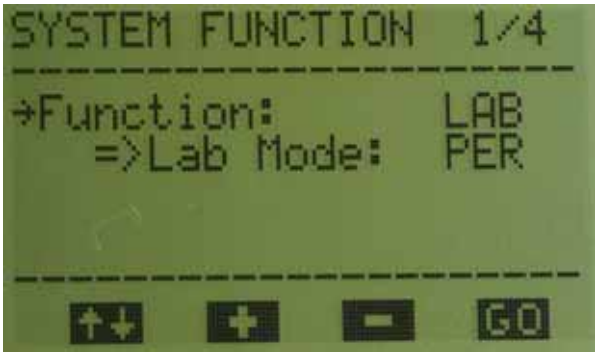


Figure 49. Setup Screen (1 of 5)

1. Ensure Function is set to “Lab”
2. Set Lab Mode to “PER”
3. Press <GO> to move on from this screen.

Figure 50. Setup Screen (2 of 5)

1. Selections here should match those previously entered into the RF Signal Generator.
2. Press <GO> to move on from this screen.

Figure 51. Setup Screen (3 of 5)

1. Parameters on setup screen 4 are not relevant to PER evaluations. Silicon Labs recommends leaving them at their default values.
2. Press <GO> to move on from this screen.

Figure 52. Setup Screen (4 of 5)

1. Ensure the Packet length matches that programmed in the RF signal generator.
2. Select number of packets to be received by the receiver using the Max. Packets field. The appropriate number of triggers will be sent by the SDB according to this setting.
3. Press <GO> to move on from this screen.

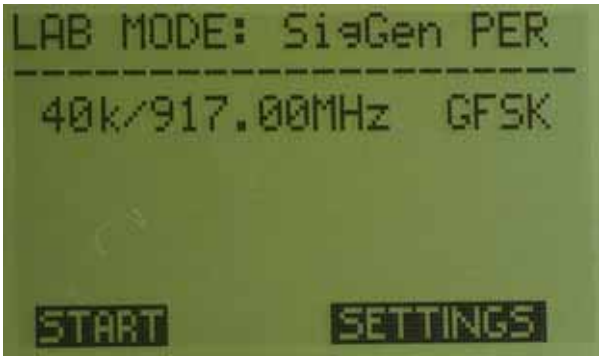


Figure 53. Setup Screen (5 of 5)

1. Press <Start> to commence packet error rate evaluation.
   This will start generating pulses on P1.4 that are used to trigger the RF signal generator. The RF generator will send one packet for every external trigger. As the radio is set to receive mode it is waiting for a specific pre-programmed packet to be arrive at the pre-programmed frequency, modulation and data rate.

Figure 54. Runtime Screen

In Figure 54, the runtime screen will summarize the current valid settings.

Notes:

1. TR = Trigger sent on P1.4.
2. MP = Missed packets
3. PER = Packet error rate

7.2.10. Results (PER Test)

7.2.10.1. PER Measurement

The result of PER measurement is shown on the LCD display. Connecting to the GPIO and the trigger the user can see the following signals.

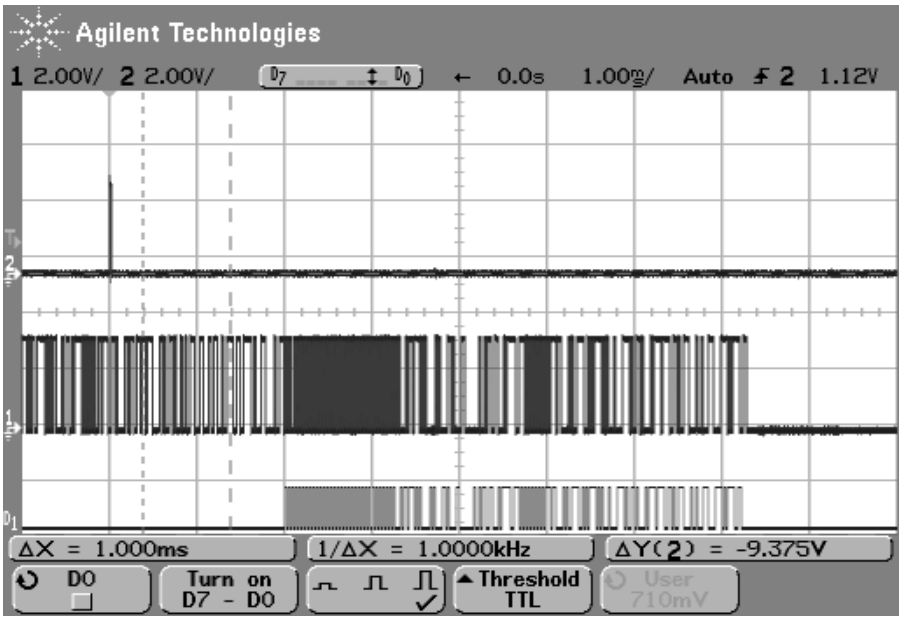


Figure 55.

In Figure 55, the top trace shows the trigger provided on P1.4 of the SDB, the middle trace shows the received data and the bottom trace shows the transmitted data from the RF signal generator.

In the plot we can see the receiver is turned on before the packet arrived. Once the valid packet is received the radio will return to tune mode. The radio will switch to receive mode prior to sending the next trigger. This is repeated for the value set in “Max. Packets”.

7.2.10.2. Other Receiver Measurements:

Using the test setup described above, users may also perform Automatic Frequency Control (AFC), Blocking and Selectivity tests. All the required parameters are controlled by the external RF generator.

7.3. Additional Information

7.3.1. USB Communications

To enable greater analysis of the data information regarding the test is sent out over the USB and can be viewed via a serial terminal emulator such as the WDS Terminal Emulator found on the WDS CDROM.

To configure the MSC-DBSB8 software development board to communicate with a PC via the USB port, a virtual serial port driver needs to be installed on the PC.

When the MSC-DBSB8 is connected, you may be prompted to install the Virtual COM port driver.

This driver can be found on the WDS CDROM.

The Virtual COM port settings of the software development board are as follows:

- Data rate is 19.2 kbps
- 1 stop bit
- No parity bit
- No handshake

If USB to virtual serial port driver is installed correctly, when the software development board is connected to PC by USB port and the WDS Terminal Emulator is running, test results like following can be seen in Figure 56.

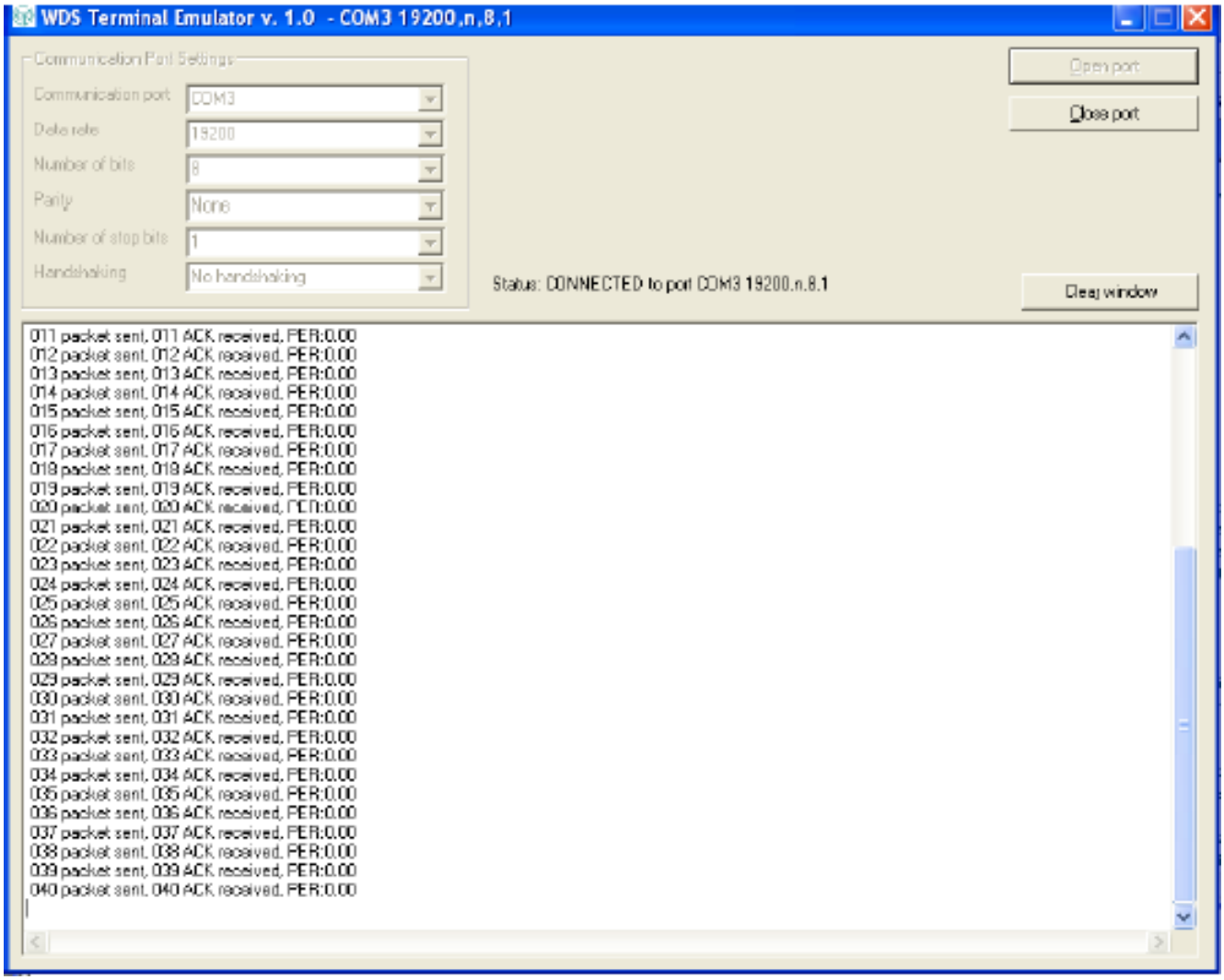


Figure 56. Figure 3:Test Result Displayed by USB Virtual COM Port

7.3.2. Packet Structure

The packet structure used by this demonstration is very simple but is not much different than a typical packet found in many RF applications today.

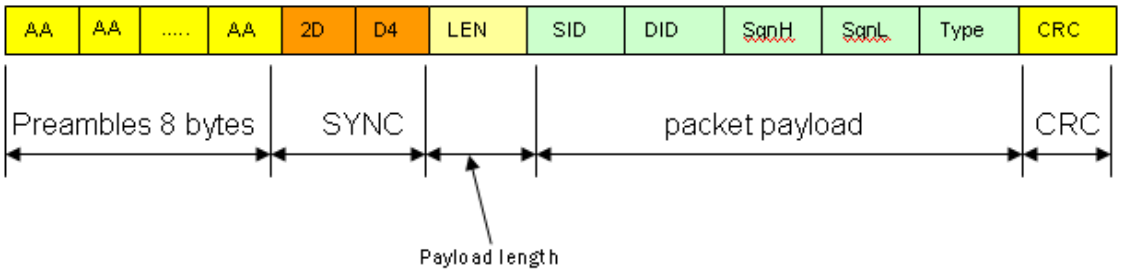


Figure 57. Packet Format Defined in the Packet Error Rate Test

8. Custom Software Development

Initially the SDBC-DK3 Software Development Kit offers the ability to become acquainted with the basic capabilities of the EZRadioPRO product family, however, the kit is also designed to be used for basic code development on any of the RF products offered by Silicon Laboratories. By design, the kit offers two modes of operation; demonstration mode or lab mode.

The source code to the factory firmware is available on the WDS CDROM but may be somewhat complex for use as reference code. To aid in software development the following chapter maybe used to illustrate basic code segment to create RF links using the EZRadioPRO platform.

The code set forth in the following chapter demonstrates a simple push button application and is based upon the EZRadioPRO Si443x transceiver using the C8051F930 microcontroller.

8.1. Program Structure

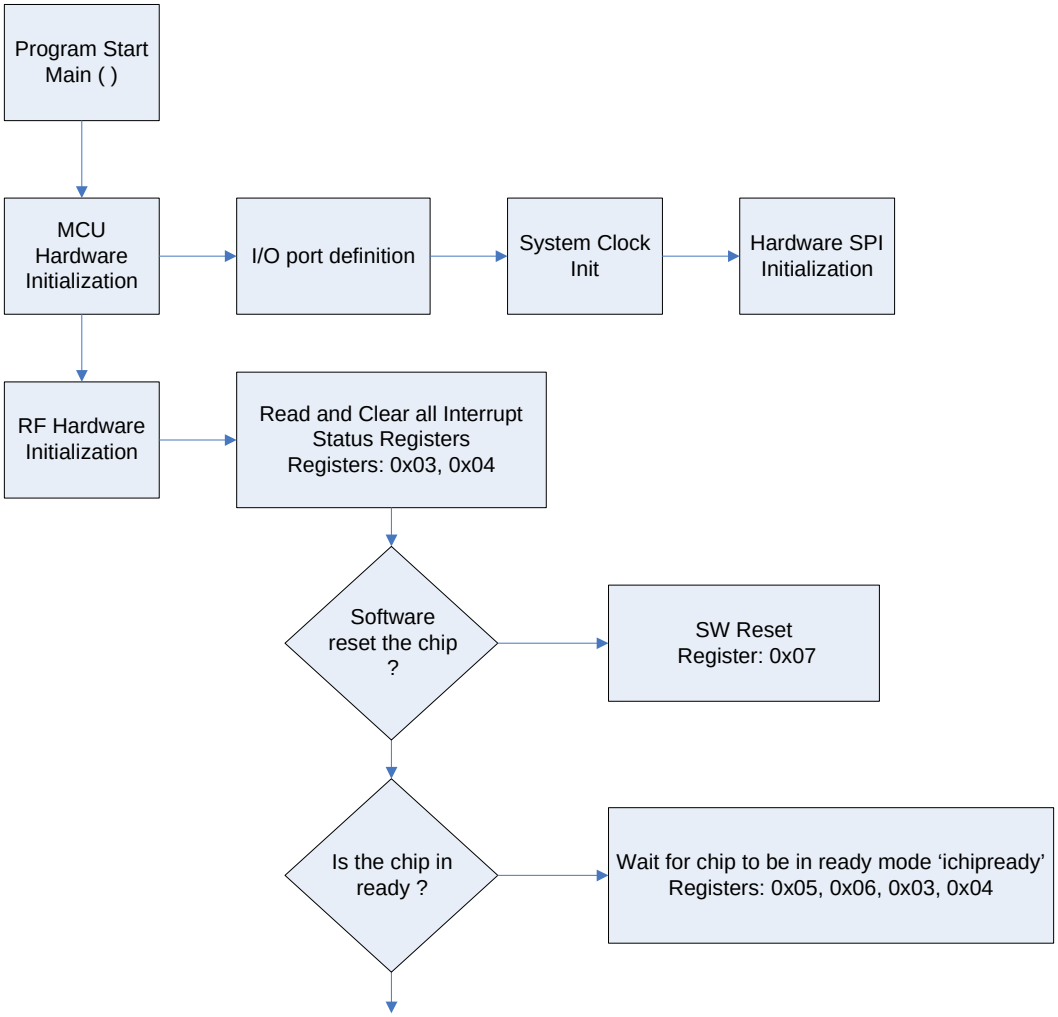


Figure 58. Basic Program Structure Block Diagram (1 of 4)

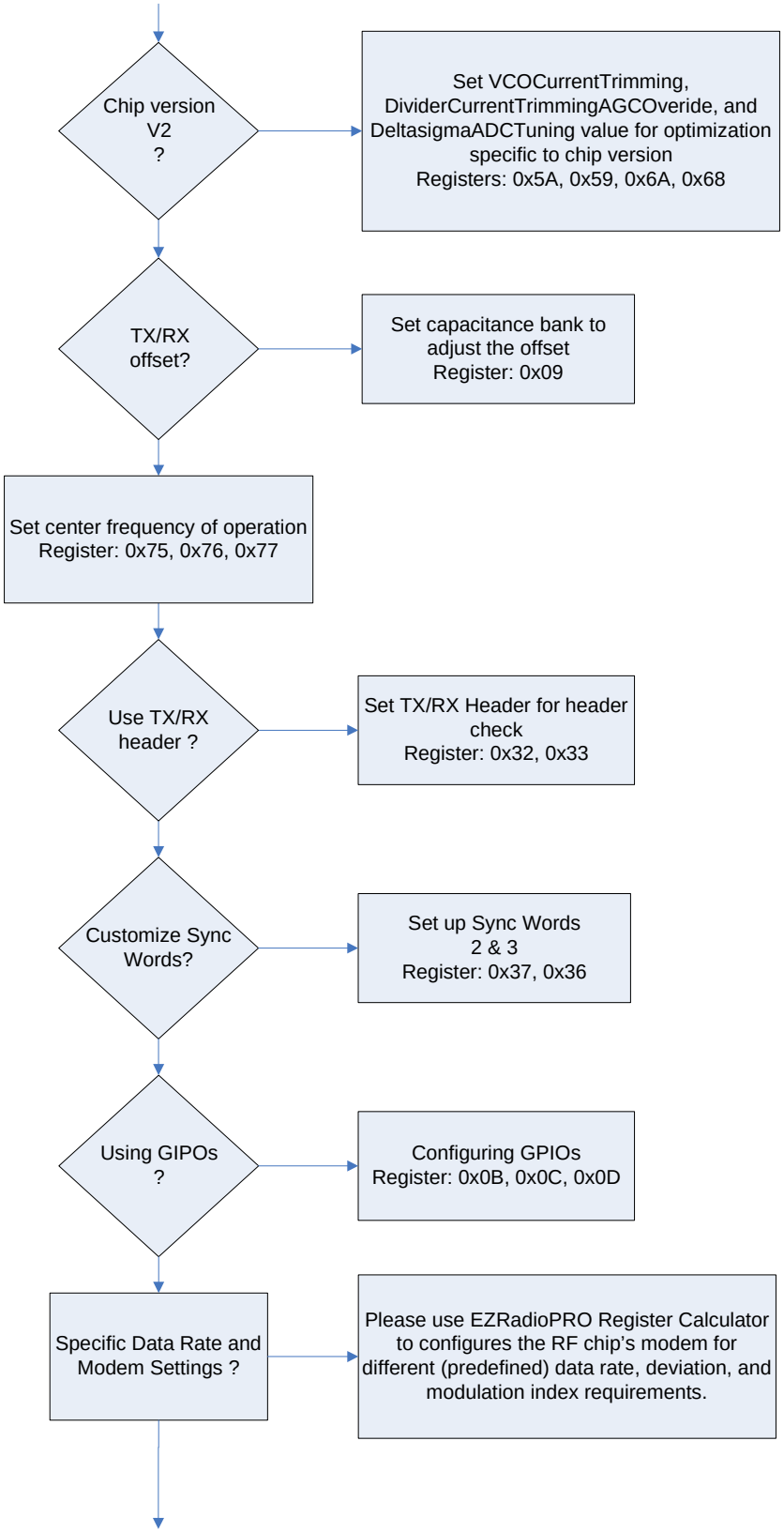


Figure 59. Basic Program Structure Block Diagram (2 of 4)

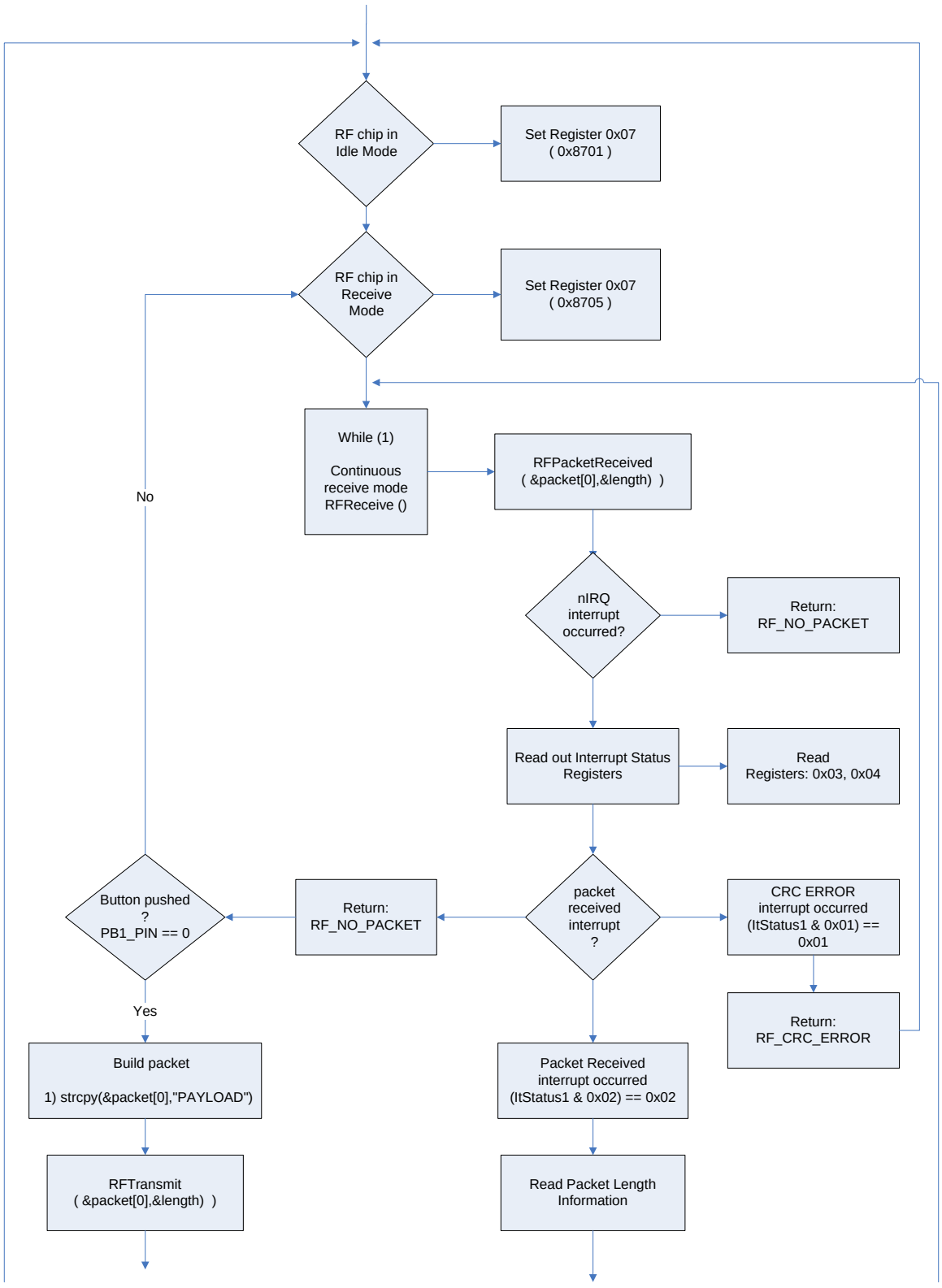


Figure 60. Basic Program Structure Block Diagram (3 of 4)

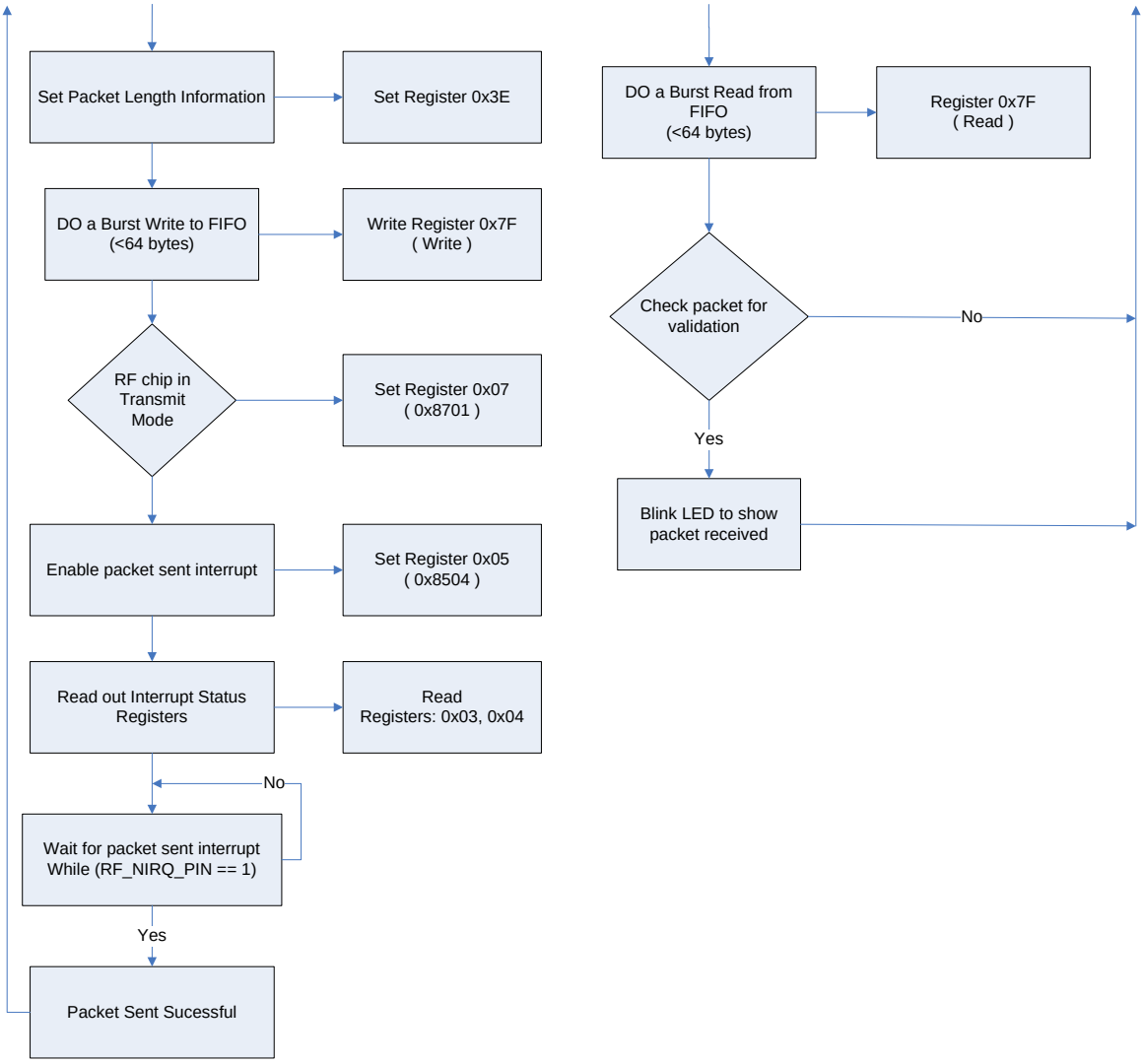


Figure 61. Basic Program Structure Block Diagram (4 of 4)

8.1.1. Basic Code Overview

```
Main ()                                                          (main.c)
   Hardware Initialization
      MCU hardware, system clock setup, and I/O init
         Hardware SPI pin definition                             (C8051.h)
                  nSEL and nIRQ pin definition
         SPI read/write function protocol
         i.e.,
            #define SYSCLK       (16000000L/2)
            #define SPI_CLOCK    (SYSCLK/4)

            //RF chip
            SBIT(RF_NSEL_PIN, SFR_P1, 3);
            SBIT(RF_NIRQ_PIN, SFR_P0, 6);

            //SPI port
            SBIT(SPI_MISO_PIN, SFR_P1, 1);
            SBIT(SPI_MOSI_PIN, SFR_P1, 2);
            SBIT(SPI_SCK_PIN, SFR_P1, 0);

         Hardware SPI setup                                      (C8051.c)
         nSEL and nIRQ pin setup
         SPI read/write functions
         i.e.,
            void SetHwMasterSpi(void)
            {
             SPI1CFG   = 0x40;                                   //Master SPI, CKPHA=0, CKPOL=0
             SPI1CN    = 0x00;                                   //3-wire Single Master, SPI enabled
             SPI1CKR   = (SYSCLK/(2*SPI_CLOCK))-1;
             SPI1EN          = 1;                                // Enable SPI1 module

             //set nSEL pins to high
             RF_NSEL_PIN = 1;
             }

         RF chip hardware and I/O init
         RF Parameters definition                                (Si4432.h)
         i.e.,
           //define the default radio frequency
           #define FREQ_BAND_SELECT 0x75                         //frequency band select
           #define NOMINAL_CAR_FREQ1 0xBB                        //default carrier frequency: 915 MHz
           #define NOMINAL_CAR_FREQ2 0x80

         RF hardware setup and parameters setting                (Si4432.c)
         i.e.,
            // set frequency
            SpiRfWriteAddressData((REG_WRITE | FrequencyBandSelect), FREQ_BAND_SELECT);
            SpiRfWriteAddressData((REG_WRITE | NominalCarrierFrequency1), NOMINAL_CAR_FREQ1);
            SpiRfWriteAddressData((REG_WRITE | NominalCarrierFrequency0), NOMINAL_CAR_FREQ2);

   RF chip in continuous receive mode                            (main.c)
      Check incoming data for valet packet
         Blink LED for valid packet
      Response to Push button command
         Send Data Packet out
```

8.2. Basic Hardware Connections

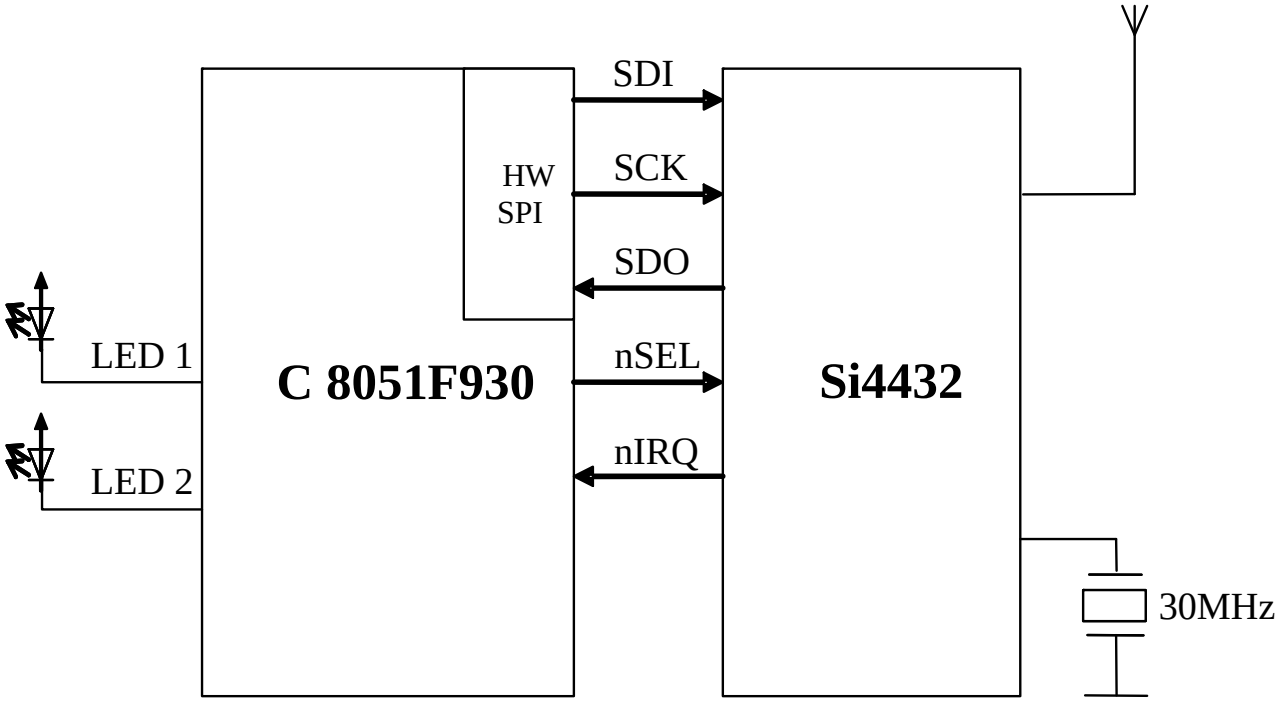


Figure 62. Basic Hardware Connections

9. Main

The main module main.c should include the main () function that is called upon startup. The main function should first call several initialization routines and then the main program loop itself. Many of the initialization and internal functions may be specific to the MCU hardware.

In this example the main loop sets the RF device into a continuous receive mode and wait for any incoming packet(s). In addition, it also polls for a push button event by user. If a button is pressed, then it'll sends a payload out using the internal FIFO and packet handler features of the EZRadioPRO device before returning to a continuous receive mode.

9.1. Flow Chart Main ( )

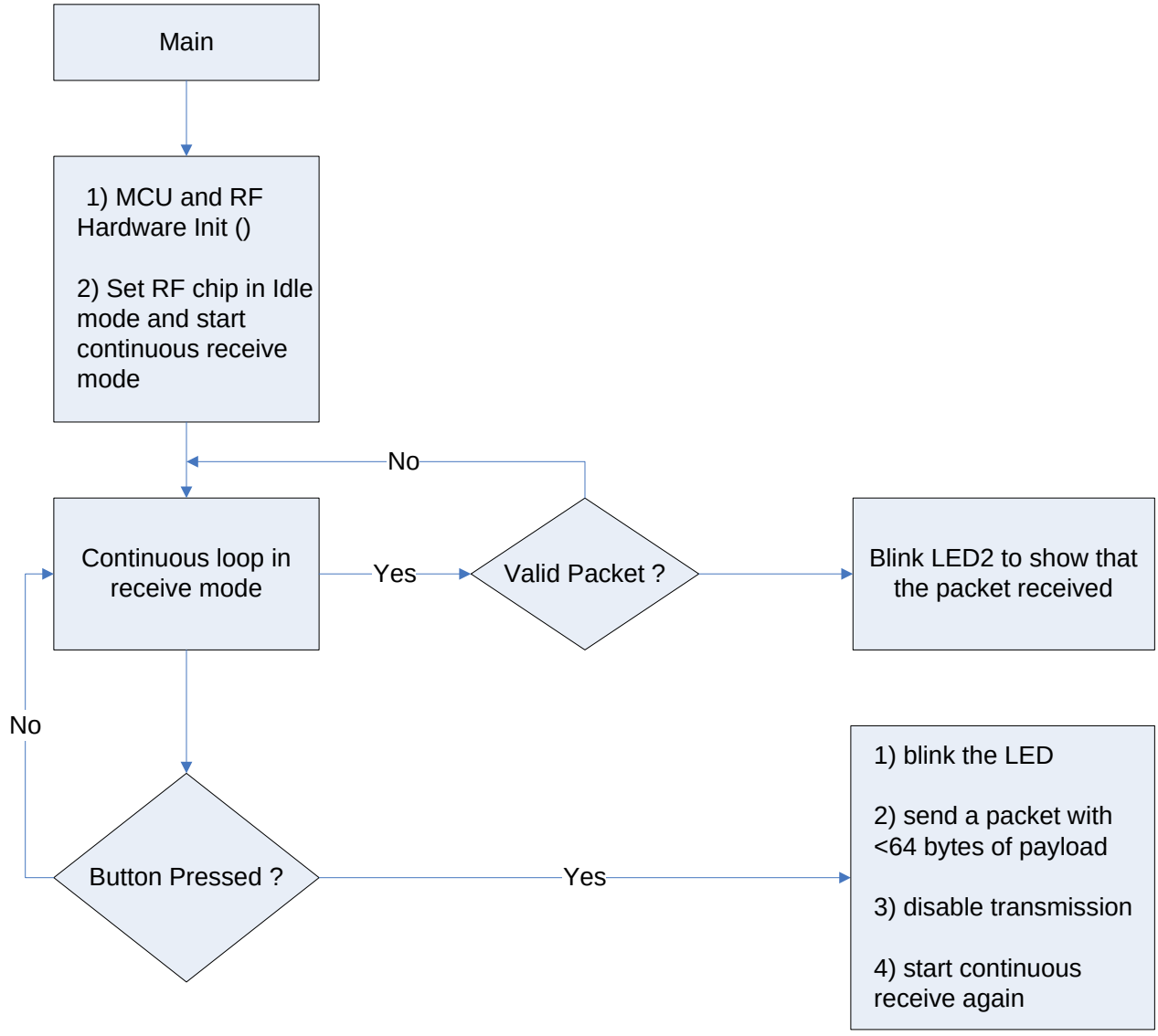


Figure 63. Flow Chart Main()

9.2. Main Source File

```
/*****************************************************************************
*
*         FILE --- MAIN.C
*
*         DESCRIPTION
*         This is the main file of the project.
*
*         CREATED
*         Silicon Laboratories Hungary Ltd
*
*         COPYRIGHT
*         Copyright 2008 Silicon Laboratories, Inc.
*         http://www.silabs.com
*
*****************************************************************************/

/*---------------------------------------------------------------------------
                              INCLUDE
---------------------------------------------------------------------------*/
#include "C8051.h"
#include "Si4432.h"

/*---------------------------------------------------------------------------
                        FUNCTION PROTOTYPES
---------------------------------------------------------------------------*/
void Hw_Init(void);
void delay_ms(uint8 delay);

/* The real program starts here.                                            */
/* After power-on, the first two tasks are the init of the MCU and the software development board.   */
/* The main loop starts after that. While (1) means that it is a never ending loop.                  */

/*---------------------------------------------------------------------------
                              MAIN PROGRAM
---------------------------------------------------------------------------*/

void main (void)
{
  idata uint8 packet[MAX_PAYLOAD_LENGTH];
  idata uint8 length;

        Hw_Init();                          // initialize the MCU and the SW Development board
        RfInitHw(DR4800BPS_DEV45KHZ);       // initialize the Si4432
        RFIdle();                           // set the radio into IDLE state
        RFReceive();                        // start continuous receive
```

The foreground loop continuously polls the nIRQ pin of the receiver. If the nIRQ is active (low), the microcontroller starts a status read. Then reads out the data packets from the FIFO.

```
while (1)                                                   // stay in receiving mode
{
        switch ( RFPacketReceived(&packet[0],&length) )     // check the status packet reception
        {
                case RF_NO_PACKET:              // CHIP is in RX mode, but no preamble detected
yet
```

If Button#1 is pressed, the LED1 will blink, and both the synthesizer and the power amplifier (PA) will be turned on. Then the packet will be built, and transmitted via the FIFO. Once complete, the power amplifier will be turned off and the system will return to receive mode.

```
if ( PB1_PIN == 0 )                     // On PB1, a packet send is initiated
{
        while(PB1_PIN == 0);            // wait for release of the button
        LED1_PIN = 1;                   // blink the LED
        length = 7;                     // send a packet (64 bytes payload)
        strcpy(&packet[0],"PAYLOAD");   // set packet content
        RFIdle();                       // disable receiving
        RFTransmit(&packet[0],length);  // start packet transmission
        LED1_PIN = 0;                   // release the LED
        RFIdle();                       // disable transmission
        RFReceive();                    // start continuous receive again
}
break;
```

At this point, the program is tests the packet length prior to a direct packet validation, blinking LED2 if expected packet data is received.

```
case RF_PACKET_RECEIVED:                // a packet received
RFIdle();                               // disable the receiver
if ( length == 7 )                      // check packet content is valid
        {
                if ( memcmp(&packet[0], "PAYLOAD", 7) == 0 )
                {
                        LED2_PIN = 1;   // blink LED2 if packet received
                        delay_ms(100);
                        LED2_PIN = 0;
                }
        }
        RFReceive();                    // restart continuous receive
        break;
```

Receiver will discard corrupted data packet and restart in continuous receive mode.

```
                                    case RF_CRC_ERROR:          // packet received with wrong CRC
                                    RFIdle();                   // disable receiver
                                    RFReceive();                // start continuous receive
                                    break;

                                    default:
                                    break;
                        }
                }
}


/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +
 +      FUNCTION NAME:      void Init(void)
 +      DESCRIPTION:        This function configures the HW
 +      INPUT:              None
 +      RETURN:             None
 +      NOTES:              None
 +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

void Hw_Init(void)
{
        uint16 i;

  // Disable the Watchdog Timer
        PCA0MD &= ~0x40;
        PCA0MD = 0x00;

        DisableGlobalIt();
        //I/O PORT INIT
  // P0.0  -  Skipped, Open-Drain, Digital
  // P0.1  -  Skipped, Open-Drain, Digital
  // P0.2  -  Skipped, Open-Drain, Analog
  // P0.3  -  Skipped, Open-Drain, Analog
  // P0.4  -  TX0 (UART0), Push-Pull,  Digital
  // P0.5  -  RX0 (UART0), Open-Drain, Digital
  // P0.6  -  Skipped, Open-Drain, Digital
  // P0.7  -  Skipped, Open-Drain, Digital
  // P1.0  -  SCK  (SPI1), Push-Pull,  Digital
  // P1.1  -  MISO (SPI1), Open-Drain, Digital
  // P1.2  -  MOSI (SPI1), Push-Pull,  Digital
  // P1.3  -  Skipped, Push-Pull,  Digital
  // P1.4  -  Skipped, Push-Pull,  Digital
  // P1.5  -  Skipped, Push-Pull,  Digital
  // P1.6  -  Skipped, Push-Pull,  Digital
  // P1.7  -  Skipped, Push-Pull,  Digital
  // P2.0  -  Skipped, Open-Drain, Digital
  // P2.1  -  Skipped, Open-Drain, Digital
  // P2.2  -  Skipped, Push-Pull,  Digital
  // P2.3  -  Skipped, Push-Pull,  Digital
  // P2.4  -  Skipped, Push-Pull,  Digital
  // P2.5  -  Skipped, Push-Pull,  Digital
  // P2.6  -  Skipped, Push-Pull,  Digital
  // P2.7  -  Skipped, Push-Pull,  Digital
```

```
    P0MDIN      =       0xF3;
    P0MDOUT     =       0x10;
    P0SKIP      =       0xCF;
    P1MDIN      =       0xFF;
    P1MDOUT     =       0xFD;
    P1SKIP      =       0xF8;
    P2MDIN      =       0xFF;
    P2MDOUT     =       0xFC;
    P2SKIP      =       0xFF;
    SFRPAGE     =       CONFIG_PAGE;
    P0DRV       =       0x10;
    P1DRV       =       0xFD;
    P2DRV       =       0xFC;
    SFRPAGE     =       LEGACY_PAGE;
    XBR0        =       0x01;
    XBR1        =       0x40;
    XBR2        =       0x40;

        // set inputs
        P0      |= 0xE3;        //Set P0 inputs
        P1      |= 0x02;        //Set P1 inputs
        P2      |= 0x03;        //Set P2 inputs

        //default I/O port
        LED1_PIN                = 0;
        LED2_PIN                = 0;
        LED3_PIN                = 0;
        LED4_PIN                = 0;
        BLED_PIN                = 0;
        LCD_NSEL_PIN            = 1;
        LCD_A0_PIN              = 0;
        LCD_RESET_PIN           = 0;

        // Oscillator init: external XTAL (16MHz), SYSCLK=XTAL/2
        OSCXCN    = 0x77;

        // 1ms delay for XTAL stabilization
        for(i=0;i<500;i++);
        while ((OSCXCN & 0x80) == 0);
        CLKSEL    = 0x01;

        //Initialize SPI
        SetHwMasterSpi();

        LED1_PIN = 1;
        delay_ms(5);
        LED2_PIN = 1;
        delay_ms(5);
        LED1_PIN = 0;
        delay_ms(5);
        LED2_PIN = 0;
}
```

```
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +
 +      FUNCTION NAME:      void delay_ms(void)
 +      DESCRIPTION:        This function generates milliseconds delay
 +      INPUT:              Number of milliseconds
 +      RETURN:             None
 +      NOTES:              None
 +
 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

void delay_ms(uint8 delay)
{
        xdata uint8 i;
        xdata uint16 j;

        for(i=0;i<delay;i++)
                for(j=0;j<8000;j++);            //delay 1ms
}
```

10. Si4432

The Si4432.c module contains code for all Si4432 related RF functions including RF setup parameters; Status Read, Transmit, Receive, and Idle state. There is a global variable (a table) 'RfSettings', which contains the preset modem parameters for each set of different data rates. These settings can be modified for other application specific settings using values calculated based on the data sheet or through the EZRadioPRO Register Calculator (available on WDS CDROM). It is suggested to change an entire line in the table if a new setting is desired

The RfInitHw () function initializes RF chip registers, I/O ports, timer, and IT routines needed by the RF stack. This function has to be called in the power-on routine. Application specific parameters include: frequency band, carrier frequency, TX/RX headers, sync words, modem setting, test bus and GPIO pin configurations.

Some of the core settings are listed below:

1. Read interrupt status to release the pending interrupts
2. SW reset -> wait for POR interrupt
3. Disable all ITs, except Chip Ready -- 'ichiprdy'
4. Set the non-default Si4432 registers
   - Set VCO
   - Set the AGC
   - Set ADC reference voltage to 0.9V
   - Set capacitance bank to adjust for adjust crystal PPM accuracy and TX/RX offsets
   - Reset digital testbus, disable scan test
   - Select nothing to the Analog Testbus
   - Set center frequency
   - Disable RX-TX headers
   - Set the sync word
   - Set GPIOs functionality
   - Set modem and RF parameters according to the selected DATA rate

The RFSetRfParameters() function configures the both the Tx and Rx RF parts of the radio for different (predefined) data rates, deviations and modulation index requirements. This sets up all of the modem settings in addition to the packet handler, CRC, preamble, and preamble detection threshold.

Note: The modem setting is a very important part of the RF parameters configuration. To simplify; there is a table in the code to provide common parameter values for a number of data rate configurations, this can be seen below. The values shown have been derived using the EZRadioPRO Register Calculator, available on the WDS CDROM or via the data sheet.

```
//=========================================================================
// This table contains the modem parameters for different data rates. See the comments for more details

code uint8 RfSettings[NMBR_OF_SAMPLE_SETTING][NMBR_OF_PARAMETER] =          // revV2
{
// IFBW, COSR, CRO2, CRO1, CRO0, CTG1, CTG0, TDR1, TDR0, MMC1, FDEV,AFC, ChargepumpCT
{0x01, 0x83, 0xc0, 0x13, 0xa9, 0x00, 0x05, 0x13, 0xa9, 0x20, 0x3a, 0x40, 0x80},      //DR: 2.4kbps, DEV:+-36kHz, BBBW: 75.2kHz
{0x04, 0x41, 0x60, 0x27, 0x52, 0x00, 0x0a, 0x27,  0x52, 0x20,  0x48, 0x40,  0x80},   //DR: 4.8kbps, DEV: +-45kHz, BBBW: 95.3kHz
{0x91, 0x71, 0x40, 0x34, 0x6e, 0x00, 0x18, 0x4e,  0xa5, 0x20,  0x48, 0x40,  0x80},   //DR: 9.6kbps, DEV: +-45kHz, BBBW:112.8kHz
{0x12, 0xc8, 0x00, 0xa3, 0xd7, 0x01, 0x13, 0x51,  0xec, 0x20,  0x13, 0x40,  0x80},   //DR: 10kbps,  DEV: +-12kHz, BBBW: 41.7kHz
{0x13, 0x64, 0x01, 0x47, 0xAE, 0x04, 0x46, 0xa3, 0xd7, 0x20, 0x13, 0x40, 0x80},     //DR: 20kbps,  DEV: +-12kHz, BBBW: 45.2kHz
{0x02, 0x64, 0x01, 0x47, 0xae, 0x05, 0x21, 0x0A, 0x3D, 0x00, 0x20, 0x40, 0x80},     //DR: 40kbps,  DEV: +-20kHz, BBBW: 83.2kHz
{0x05, 0x50, 0x01, 0x99, 0x9A, 0x06, 0x68, 0x0C, 0xCD, 0x00, 0x28, 0x40, 0x80},     //DR: 50kbps,  DEV: +-25kHz, BBBW:112.8kHz
{0x9A, 0x3C, 0x02, 0x22, 0x22, 0x07, 0xFF, 0x19, 0x9A, 0x00, 0x50, 0x00,0xC0},     //DR: 100kbps, DEV: +-50kHz, BBBW: 208 kHz
{0x89, 0x5e, 0x01, 0x5D, 0x86, 0x02, 0xAB, 0x20, 0xc5, 0x00, 0x66, 0x00,0xC0},     //DR: 128kbps, DEV:+-64kHz, BBBW:269.3kHz
};
//=========================================================================
```

Table 6. Registers

Bits	Register Name	Register Address
IFBW:	IF Filter Bandwidth	0x1C
COSR:	Clock Recovery Oversampling Ratio	0x20
CRO2:	Clock Recovery Offset 2	0x21
CRO1:	Clock Recovery Offset 1	0x22
CRO0:	Clock Recovery Offset 0	0x23
CTG1:	Clock Recovery Timing Loop Gain 1	0x24
CTG0:	Clock Recovery Timing Loop Gain 1	0x25
TDR1:	TX Data Rate 1	0x6E
TDR0:	TX Data Rate 0	0x6F
MMC1:	Modulation Mode Control 1	0x70
FDEV:	Frequency Deviation	0x72
AFC:	AFC Loop Gear Shift Override	0x1D
ChargepumpCT:	Charge Pump Current Trimming Override	0x58

The *RFIdle()* function sets the transceiver and the RF stack into an IDLE state independent of the actual state of the RF stack. It disables the transmit/receive mode and all the interrupts. It then reads the interrupt status registers from the radio and clears the IT flags.

The *RFTransmit()* function starts packet transmission and ensures packets are sent successfully.

The *RFReceive()* function enables packet reception by enabling the receiver and setting up the relevant interrupts prior to and reading the interrupt status registers.

The *RFPacketReceived()* function checks whether the packet is received or not. It reads the data packet from the FIFO if all packet handlers and the CRC are correct.

10.1. Flow Chart

10.1.1. RF Packet Received( )

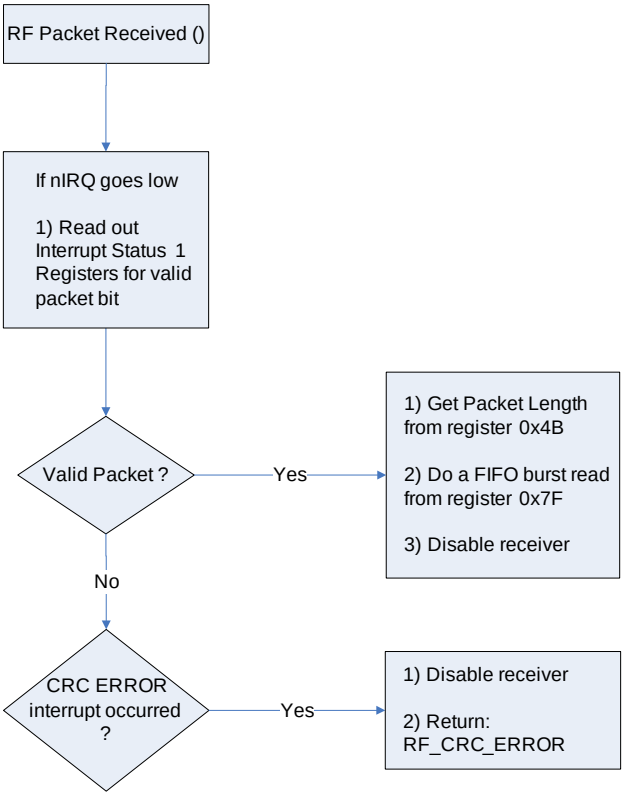


Figure 64.

10.1.2. RFTransmit( )

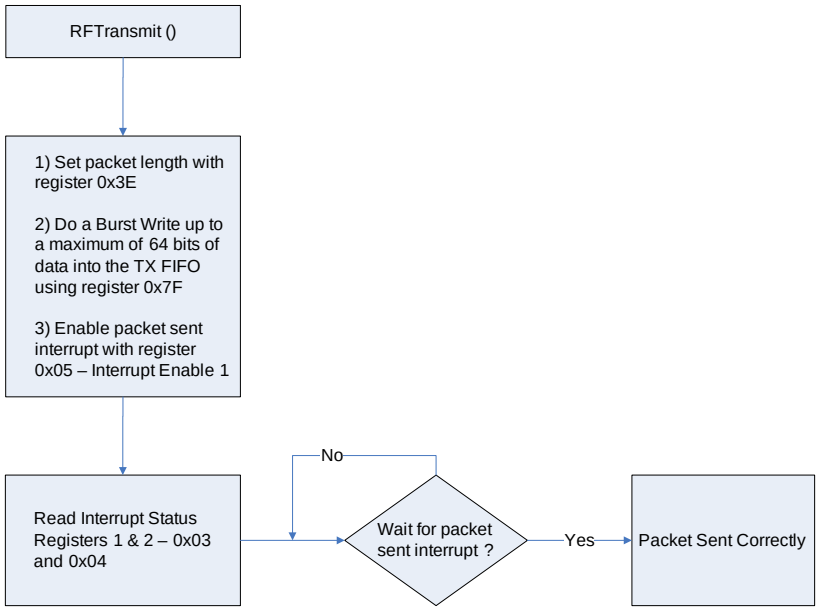


Figure 65.

10.2. Si4432 Header File

```
/*****************************************************************************************
**
**        FILE --- Si4432.h
**
**        DESCRIPTION
**        Header files for Si4432 usage, contains RF specific definition and type declaration
**
**        CREATED
**        Silicon Laboratories Hungary Ltd
**
**        COPYRIGHT
**        Copyright 2008 Silicon Laboratories, Inc.
**        http://www.silabs.com
**
*****************************************************************************************/

#ifndef Si4432_H
#define Si4432_H

#include "C8051.h"

          /* ======================================== *
           *        APPLICATION SPECIFIC DEFINITIONS          *
           * ======================================== */

// define the default radio frequency
#define FREQ_BAND_SELECT 0x75                   // frequency band select
#define NOMINAL_CAR_FREQ1 0xBB                  // default carrier frequency: 915 MHz
#define NOMINAL_CAR_FREQ2 0x80
//packet settings
#define PREAMBLE_LENGTH (4)                     // 4 byte preamble
#define PD_LENGTH (2)                           // preamble detection threshold in nibbles
```

The max length of the received data packet is defined here (in data bytes).

```
#define MAX_PAYLOAD_LENGTH (64)

          /* ======================================== *
           *                D E F I N I T I O N S                 *
           * ======================================== */

// definitions for register usage
#define  REG_READ                               (0x00)
#define  REG_WRITE                              (0x80)
#define  NMBR_OF_SAMPLE_SETTING                 (9)
#define  NMBR_OF_PARAMETER                      (13)
```

```
        /* =======================================*
         *            TYPE DECLARATION           *
         * =======================================*/

// RF stack enumerations
typedef enum _RF_ENUM
{
        RF_OK                           =       0x00,   // function response parameters
        RF_ERROR_TIMING                 =       0x01,
        RF_ERROR_PARAMETER              =       0x02,
        RF_PACKET_RECEIVED              =       0x03,
        RF_RX_FIFO_ALMOST_FULL          =       0x04,
        RF_NO_PACKET                    =       0x05,
        RF_CRC_ERROR                    =       0x06,
} RF_ENUM;

typedef enum _RF_SAMPLE_SETTINGS
{                                       // Data Rate; Freq Deviation; Receiver Bandwidth
        DR2400BPS_DEV36KHZ     = 0,     // DR = 2.4kbps;   Fdev = +-36kHz;   BBBW = 75.2kHz;
        DR4800BPS_DEV45KHZ     = 1,     // DR = 4.8kbps;   Fdev = +-45kHz;   BBBW = 95.3kHz;
        DR9600BPS_DEV45KHZ     = 2,     // DR = 9.6kbps;   Fdev = +-45kHz;   BBBW = 112.8kHz;
        DR10000BPS_DEV12KHZ = 3,        // DR = 10kbps;    Fdev = +-12kHz;   BBBW = 41.7kHz;
        DR20000BPS_DEV12KHZ = 4,        // DR = 20kbps;    Fdev = +-12kHz;   BBBW = 45.2kHz;
        DR40000BPS_DEV20KHZ = 5,        // DR = 40kbps;    Fdev = +-20kHz;   BBBW = 83.2kHz;
        DR50000BPS_DEV25KHZ = 6,        // DR = 50kbps;    Fdev = +-25kHz;   BBBW = 112.8kHz;
        DR100000BPS_DEV50KHZ = 7,       // DR = 100kbps;   Fdev = +-50kHz;   BBBW = 208kHz;
        DR128000BPS_DEV64KHZ = 8,       // DR = 128kbps;   Fdev = +-64kHz;   BBBW = 269.3kHz;
} RF_SAMPLE_SETTINGS;

typedef enum _RF_REG_MAP                // These settings are for silicon Rev-V2
{
 DeviceType                             = 0x00,
 DeviceVersion                          = 0x01,
 DeviceStatus                           = 0x02,
 InterruptStatus1                       = 0x03,
 InterruptStatus2                       = 0x04,
 InterruptEnable1                       = 0x05,
 InterruptEnable2                       = 0x06,
 OperatingFunctionControl1              = 0x07,
 OperatingFunctionControl2              = 0x08,
 CrystalOscillatorLoadCapacitance       = 0x09,
 MicrocontrollerOutputClock             = 0x0A,
 GPIO0Configuration                     = 0x0B,
 GPIO1Configuration                     = 0x0C,
 GPIO2Configuration                     = 0x0D,
 IOPortConfiguration                    = 0x0E,
 ADCConfiguration                       = 0x0F,
 ADCSensorAmplifierOffset               = 0x10,
 ADCValue                               = 0x11,
 TemperatureSensorControl               = 0x12,
 TemperatureValueOffset                 = 0x13,
 WakeUpTimerPeriod1                     = 0x14,
 WakeUpTimerPeriod2                     = 0x15,
 WakeUpTimerPeriod3                     = 0x16,
 WakeUpTimerValue1                      = 0x17,
 WakeUpTimerValue2                      = 0x18,
 LowDutyCycleModeDuration               = 0x19,
 LowBatteryDetectorThreshold            = 0x1A,
 BatteryVoltageLevel                    = 0x1B,
 IFFilterBandwidth                      = 0x1C,
```

```
AFCLoopGearshiftOverride                          = 0x1D,
  AFCTimingControl                                = 0x1E,
  ClockRecoveryGearshiftOverride                  = 0x1F,
  ClockRecoveryOversamplingRatio                  = 0x20,
  ClockRecoveryOffset2                            = 0x21,
  ClockRecoveryOffset1                            = 0x22,
  ClockRecoveryOffset0                            = 0x23,
  ClockRecoveryTimingLoopGain1                    = 0x24,
  ClockRecoveryTimingLoopGain0                    = 0x25,
  ReceivedSignalStrengthIndicator                 = 0x26,
  RSSIThresholdForClearChannelIndicator           = 0x27,
  AntennaDiversityRegister1                       = 0x28,
  AntennaDiversityRegister2                       = 0x29,
  DataAccessControl                               = 0x30,
  EZmacStatus                                     = 0x31,
  HeaderControl1                                  = 0x32,
  HeaderControl2                                  = 0x33,
  PreambleLength                                  = 0x34,
  PreambleDetectionControl                        = 0x35,
  SyncWord3                                       = 0x36,
  SyncWord2                                       = 0x37,
  SyncWord1                                       = 0x38,
  SyncWord0                                       = 0x39,
  TransmitHeader3                                 = 0x3A,
  TransmitHeader2                                 = 0x3B,
  TransmitHeader1                                 = 0x3C,
  TransmitHeader0                                 = 0x3D,
  TransmitPacketLength                            = 0x3E,
  CheckHeader3                                    = 0x3F,
  CheckHeader2                                    = 0x40,
  CheckHeader1                                    = 0x41,
  CheckHeader0                                    = 0x42,
  HeaderEnable3                                   = 0x43,
  HeaderEnable2                                   = 0x44,
  HeaderEnable1                                   = 0x45,
  HeaderEnable0                                   = 0x46,
  ReceivedHeader3                                 = 0x47,
  ReceivedHeader2                                 = 0x48,
  ReceivedHeader1                                 = 0x49,
  ReceivedHeader0                                 = 0x4A,
  ReceivedPacketLength                            = 0x4B,
  AnalogTestBus                                   = 0x50,
  DigitalTestBus                                  = 0x51,
  TXRampControl                                   = 0x52,
  PLLTuneTime                                     = 0x53,
  CalibrationControl                              = 0x55,
  ModemTest                                       = 0x56,
  ChargepumpTest                                  = 0x57,
  ChargepumpCurrentTrimming_Override              = 0x58,
  DividerCurrentTrimming                          = 0x59,
  VCOCurrentTrimming                              = 0x5A,
  VCOCalibration_Override                         = 0x5B,
  SynthesizerTest                                 = 0x5C,
  BlockEnableOverride1                            = 0x5D,
  BlockEnableOverride2                            = 0x5E,
  BlockEnableOverride3                            = 0x5F,
  ChannelFilterCoefficientAddress                 = 0x60,
  ChannelFilterCoefficientValue                   = 0x61,
  CrystalOscillator_ControlTest                   = 0x62,
  RCOscillatorCoarseCalibration_Override          = 0x63,
  RCOscillatorFineCalibration_Override            = 0x64,
  LDOControlOverride                              = 0x65,
```

```
  DeltasigmaADCTuning1                                  = 0x67,
  DeltasigmaADCTuning2                                  = 0x68,
  AGCOverride1                                          = 0x69,
  AGCOverride2                                          = 0x6A,
  GFSKFIRFilterCoefficientAddress                       = 0x6B,
  GFSKFIRFilterCoefficientValue                         = 0x6C,
  TXPower                                               = 0x6D,
  TXDataRate1                                           = 0x6E,
  TXDataRate0                                           = 0x6F,
  ModulationModeControl1                                = 0x70,
  ModulationModeControl2                                = 0x71,
  FrequencyDeviation                                    = 0x72,
  FrequencyOffset                                       = 0x73,
  FrequencyChannelControl                               = 0x74,
  FrequencyBandSelect                                   = 0x75,
  NominalCarrierFrequency1                              = 0x76,
  NominalCarrierFrequency0                              = 0x77,
  FrequencyHoppingChannelSelect                         = 0x79,
  FrequencyHoppingStepSize                              = 0x7A,
  TXFIFOControl1                                        = 0x7C,
  TXFIFOControl2                                        = 0x7D,
  RXFIFOControl                                         = 0x7E,
  FIFOAccess                                            = 0x7F,
} RF_REG_MAP;

        /* ======================================== *
         *      F U N C T I O N  P R O T O T Y P E S      *
         * ======================================== */

RF_ENUM RfInitHw(U8 data_rate);
RF_ENUM RFSetRfParameters(RF_SAMPLE_SETTINGS setting);
RF_ENUM RFIdle(void);
RF_ENUM RFTransmit(uint8 * packet, uint8 length);
RF_ENUM RFReceive(void);
RF_ENUM RFPacketReceived(uint8 * packet, uint8 * length);

#endif
```

10.3. Si4432 Source File

```
/*****************************************************************************************
**
**        FILE --- Si4432.c
**
**        DESCRIPTION
**        Contains all Si4432 RF functions
**
**        CREATED
**        Silicon Laboratories Hungary Ltd
**
**        COPYRIGHT
**        Copyright 2008 Silicon Laboratories, Inc.
**        http://www.silabs.com
**
*****************************************************************************************/

#include "C8051.h"
#include "Si4432.h"

/*---------------------------------------------------------------------------*/
/*                              GLOBAL variables                             */
/*---------------------------------------------------------------------------*/

// This table contains the modem parameters for different data rates. See the comments for more details

code uint8 RfSettings[NMBR_OF_SAMPLE_SETTING][NMBR_OF_PARAMETER] =          // revV2
{
// IFBW, COSR, CRO2, CRO1, CRO0, CTG1, CTG0, TDR1,   TDR0, MMC1, FDEV, AFC, ChargepumpCT
{0x01, 0x83, 0xc0, 0x13, 0xa9, 0x00, 0x05, 0x13, 0xa9, 0x20, 0x3a, 0x40, 0x80},      //DR: 2.4kbps, DEV:+-36kHz, BBBW: 75.2kHz
{0x04, 0x41, 0x60, 0x27, 0x52, 0x00, 0x0a, 0x27, 0x52, 0x20, 0x48, 0x40, 0x80},      //DR: 4.8kbps, DEV: +-45kHz, BBBW: 95.3kHz
{0x91, 0x71, 0x40, 0x34, 0x6e, 0x00, 0x18, 0x4e, 0xa5, 0x20, 0x48, 0x40, 0x80},      //DR: 9.6kbps, DEV: +-45kHz, BBBW:112.8kHz
{0x12, 0xc8, 0x00, 0xa3, 0xd7, 0x01, 0x13, 0x51, 0xec, 0x20, 0x13, 0x40, 0x80},      //DR: 10kbps,  DEV: +-12kHz, BBBW: 41.7kHz
{0x13, 0x64, 0x01, 0x47, 0xAE, 0x04, 0x46, 0xa3, 0xd7, 0x20, 0x13, 0x40, 0x80},      //DR: 20kbps,  DEV: +-12kHz, BBBW: 45.2kHz
{0x02, 0x64, 0x01, 0x47, 0xae, 0x05, 0x21, 0x0A, 0x3D, 0x00, 0x20, 0x40, 0x80},      //DR: 40kbps,  DEV: +-20kHz, BBBW: 83.2kHz
{0x05, 0x50, 0x01, 0x99, 0x9A, 0x06, 0x68, 0x0C, 0xCD, 0x00, 0x28, 0x40, 0x80},      //DR: 50kbps,  DEV: +-25kHz, BBBW:112.8kHz
{0x9A, 0x3C, 0x02, 0x22, 0x22, 0x07, 0xFF, 0x19, 0x9A, 0x00, 0x50, 0x00, 0xC0},      //DR: 100kbps, DEV: +-50kHz, BBBW: 208 kHz
{0x89, 0x5e, 0x01, 0x5D, 0x86, 0x02, 0xAB, 0x20, 0xc5, 0x00, 0x66,0x00, 0xC0},      //DR: 128kbps, DEV:+-64kHz, BBBW:269.3kHz
};

idata uint8 ItStatus1,ItStatus2;


/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +
 +        FUNCTION NAME:      void RfInitHw(void)
 +        DESCRIPTION:        Initializes the used I/O pins, SPI and timer peripherals,
 +                            IT routines needed for the RF stack
 +        RETURN:             None
 +        NOTES:              1) Has to be called in the power-on routine
 +                            2) It initializes the RF chip registers
 +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

RF_ENUM RfInitHw(U8 data_rate)
{
```

```
RF_NSEL_PIN    = 1;

// initialize I/O port directions

ItStatus1 = SpiRfReadRegister(InterruptStatus1);                    // read interrupt status
ItStatus2 = SpiRfReadRegister(InterruptStatus2);                    // SW reset   -> wait for POR interrupt
SpiRfWriteAddressData((REG_WRITE | OperatingFunctionControl1), 0x80);
                                                                    // Enable the POR interrupt
while ( RF_NIRQ_PIN == 1);                                          // Wait for the POR interrupt

// disable all ITs, except 'ichiprdy'
SpiRfWriteAddressData((REG_WRITE | InterruptEnable1), 0x00);
SpiRfWriteAddressData((REG_WRITE | InterruptEnable2), 0x02);
ItStatus1 = SpiRfReadRegister(InterruptStatus1);
ItStatus2 = SpiRfReadRegister(InterruptStatus2);

// set the non-default Si4432 registers
// set VCO
SpiRfWriteAddressData((REG_WRITE | VCOCurrentTrimming), 0x7F);
SpiRfWriteAddressData((REG_WRITE | DividerCurrentTrimming), 0x40);

// set the AGC
SpiRfWriteAddressData((REG_WRITE | AGCOverride2), 0x0B);

// set ADC reference voltage to 0.9V
SpiRfWriteAddressData((REG_WRITE | DeltasigmaADCTuning2), 0x04);
```

The default value on power up should be able to oscillate the crystal. Based on the crystal and PCB capacitance, these cap banks can be used to tune the TX/RX offset.

```
// set  cap. bank
SpiRfWriteAddressData((REG_WRITE | CrystalOscillatorLoadCapacitance), 0xD7);

// reset digital testbus, disable scan test
SpiRfWriteAddressData((REG_WRITE | DigitalTestBus), 41);//0x00);

// select nothing to the Analog Testbus
SpiRfWriteAddressData((REG_WRITE | AnalogTestBus), 0x0B);
```

Important: The band selector command (Configuration Command) should be sent prior to the receiver command since once band selection has been achieved, the synthesizer should be calibrated. Calibration can be done by turning off and on the receiver chain using the receiver command. In the current application the receiver chain is continuously turned on.

```
// set frequency
SpiRfWriteAddressData((REG_WRITE | FrequencyBandSelect), FREQ_BAND_SELECT);
SpiRfWriteAddressData((REG_WRITE | NominalCarrierFrequency1), NOMINAL_CAR_FREQ1);
SpiRfWriteAddressData((REG_WRITE | NominalCarrierFrequency0), NOMINAL_CAR_FREQ2);

// disable RX-TX headers,
SpiRfWriteAddressData((REG_WRITE | HeaderControl1), 0x00 );
SpiRfWriteAddressData((REG_WRITE | HeaderControl2), 0x02 );

// set the sync word
```

```
SpiRfWriteAddressData((REG_WRITE | SyncWord3), 0x2D);
        SpiRfWriteAddressData((REG_WRITE | SyncWord2), 0xD4);
```

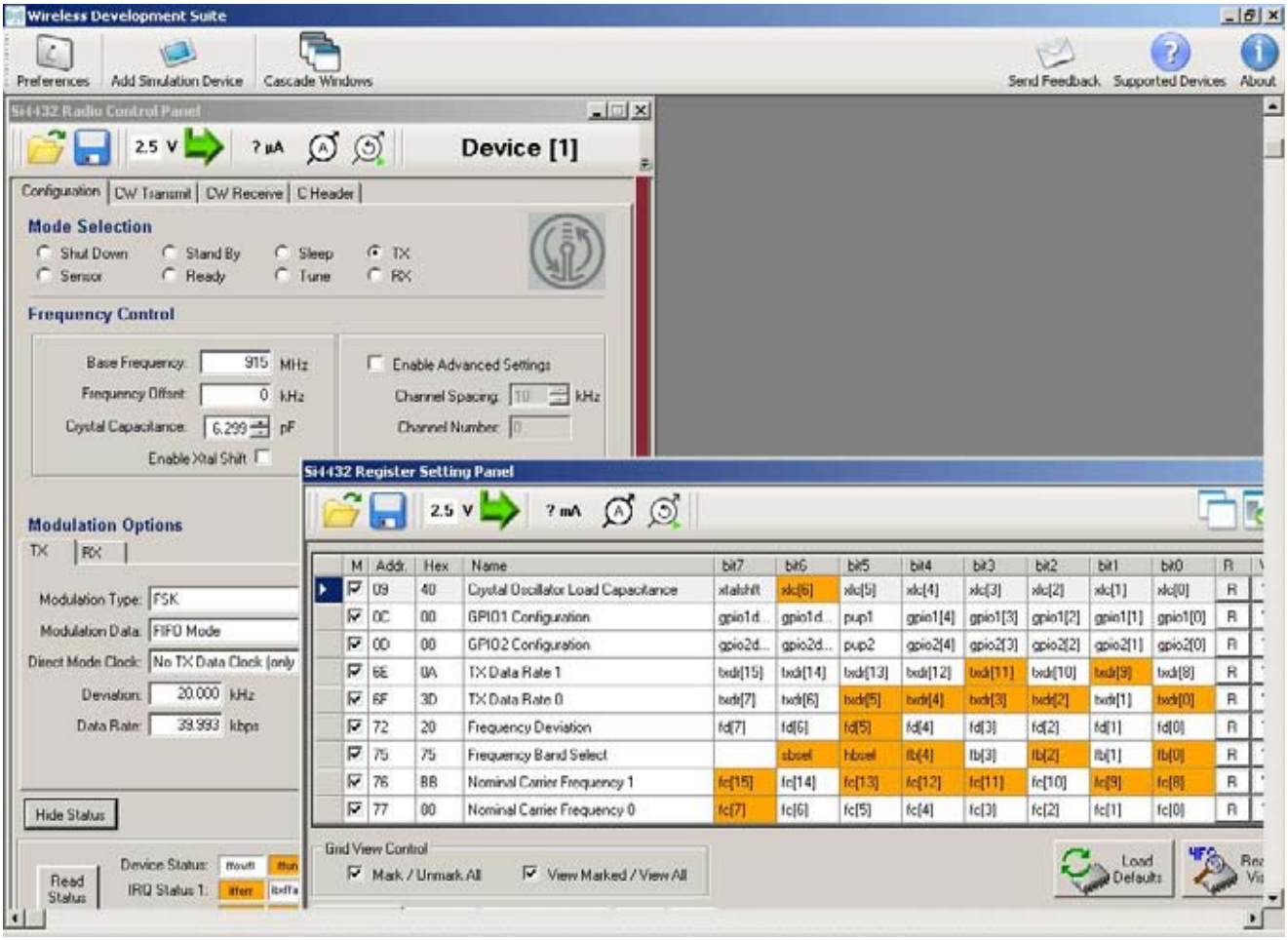


Figure 66.

GPIO definitions

```
        // set GPIO0 to RX DATA
        SpiRfWriteAddressData((REG_WRITE | GPIO0Configuration), 0x14);

        // set GPIO1 to TX State & GPIO2 to RX State
        SpiRfWriteAddressData((REG_WRITE | GPIO1Configuration), 0x12);
        SpiRfWriteAddressData((REG_WRITE | GPIO2Configuration), 0x15);
```

Next, define your RF parameters based on application specific data rate, deviation, receive baseband bandwidth etc.,

```
        // set modem and RF parameters according to the selected DATA rate
        RFSetRfParameters(data_rate);
        return RF_OK;
}
```

```
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +
 +       FUNCTION NAME:      RF_ENUM RFSetRfParameters (RF_SAMPLE_SETTINGS setting)
 +       DESCRIPTION:        This function configures the RF part of the chip (both TX and RX)
 +                           for different (predefined) data rate, deviation and modulation index
 +                           requirements.
 +       RETURN:             RF_OK: The operation was successful
 +                           RF_ERROR_PARAMETER: Invalid parameter, operation is ignored.
 +       NOTES:
 +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

RF_ENUM RFSetRfParameters(RF_SAMPLE_SETTINGS setting)
{
        // setup the internal digital modem according the selected RF settings (data rate)
        SpiRfWriteAddressData((REG_WRITE | IFFilterBandwidth), RfSettings[setting][0] );
        SpiRfWriteAddressData((REG_WRITE | ClockRecoveryOversamplingRatio), RfSettings[setting][1]);
        SpiRfWriteAddressData((REG_WRITE | ClockRecoveryOffset2), RfSettings[setting][2]);
        SpiRfWriteAddressData((REG_WRITE | ClockRecoveryOffset1), RfSettings[setting][3]);
        SpiRfWriteAddressData((REG_WRITE | ClockRecoveryOffset0), RfSettings[setting][4]);
        SpiRfWriteAddressData((REG_WRITE | ClockRecoveryTimingLoopGain1), RfSettings[setting][5]);
        SpiRfWriteAddressData((REG_WRITE | ClockRecoveryTimingLoopGain0), RfSettings[setting][6]);
        SpiRfWriteAddressData((REG_WRITE | TXDataRate1), RfSettings[setting][7]);
        SpiRfWriteAddressData((REG_WRITE | TXDataRate0), RfSettings[setting][8]);
        SpiRfWriteAddressData((REG_WRITE | ModulationModeControl1), RfSettings[setting][9]);
        SpiRfWriteAddressData((REG_WRITE | FrequencyDeviation), RfSettings[setting][10]);
        SpiRfWriteAddressData((REG_WRITE | AFCLoopGearshiftOverride), RfSettings[setting][11]);
        SpiRfWriteAddressData((REG_WRITE | ChargepumpCurrentTrimming_Override), RfSettings[setting][12]);

        // enable packet handler & CRC16
        SpiRfWriteAddressData((REG_WRITE | DataAccessControl), 0x8D);
        SpiRfWriteAddressData((REG_WRITE | ModulationModeControl2), 0x63);

        // set preamble length & detection threshold
        SpiRfWriteAddressData((REG_WRITE | PreambleLength), (PREAMBLE_LENGTH << 1));
        SpiRfWriteAddressData((REG_WRITE | PreambleDetectionControl), ( PD_LENGTH << 4));
        SpiRfWriteAddressData((REG_WRITE | ClockRecoveryGearshiftOverride), 0x03);

        return RF_OK;
}
```

```
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +
 +       FUNCTION NAME:      RF_ENUM RFIdle(void)
 +       DESCRIPTION:        Sets the transceiver and the RF stack into IDLE state,
 +                           independently of the actual state of the RF stack.
 +       RETURN:             RF_OK:          The operation was successful
 +       NOTES:
 +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

RF_ENUM RFIdle(void)
{
        // disable transmitter and receiver
        SpiRfWriteAddressData((REG_WRITE | OperatingFunctionControl1), 0x01);

        // disable all ITs
        SpiRfWriteAddressData((REG_WRITE | InterruptEnable1), 0x00);
        SpiRfWriteAddressData((REG_WRITE | InterruptEnable2), 0x00);

        // read the interrupt status registers from the radio to clear the IT flags
        ItStatus1 = SpiRfReadRegister(InterruptStatus1);
        ItStatus2 = SpiRfReadRegister(InterruptStatus2);

        return RF_OK;
}


/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +
 +       FUNCTION NAME:      RF_ENUM RFTransmit(uint8 * packet, uint8 length)
 +       DESCRIPTION:        Starts packet transmission
 +       INPUT:              MESSAGE structure
 +       RETURN:             RF_OK:          The packet sent correctly
 +
 +       NOTES:
 +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

RF_ENUM RFTransmit(uint8 * packet, uint8 length)
{
        uint8 temp8;

        // set packet length
        SpiRfWriteAddressData((REG_WRITE | TransmitPacketLength), length);

        for(temp8=0;temp8<length;temp8++)
        {
                SpiRfWriteAddressData((REG_WRITE | FIFOAccess),packet[temp8]);
        }


        // enable transmitter
        SpiRfWriteAddressData((REG_WRITE | OperatingFunctionControl1), 0x09);

        // enable the packet sent interrupt only
        SpiRfWriteAddressData((REG_WRITE | InterruptEnable1), 0x04);

        // read interrupt status registers
        ItStatus1 = SpiRfReadRegister(InterruptStatus1);
```

```
ItStatus2 = SpiRfReadRegister(InterruptStatus2);

        // wait for the packet sent interrupt
        while(RF_NIRQ_PIN == 1);

        // packet is sent correctly
        return RF_OK;
}


/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +
 +      FUNCTION NAME:      RF_ENUM RFReceive(void)
 +      DESCRIPTION:        Starts packet reception
 +      INPUT:              None
 +      RETURN:             RF_OK:          The operation was successful
 +      NOTES:
 +
 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

RF_ENUM RFReceive(void)
{
        // enable receiver chain
        SpiRfWriteAddressData((REG_WRITE | OperatingFunctionControl1), 0x05);

        // enable the wanted ITs
        SpiRfWriteAddressData((REG_WRITE | InterruptEnable1), 0x13);
        SpiRfWriteAddressData((REG_WRITE | InterruptEnable2), 0x00);

        // read interrupt status registers
        ItStatus1 = SpiRfReadRegister(InterruptStatus1);
        ItStatus2 = SpiRfReadRegister(InterruptStatus2);

        return RF_OK;
}
```

```
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +
 +      FUNCTION NAME:      RF_ENUM RFPacketReceived (uint8 * packet, uint8 * length)
 +      DESCRIPTION:        Check whether the packet received or not.
 +      INPUT:              Pointers for storing data and length
 +      RETURN:             RF_PACKET_RECEIVED:     Packet received
 +                          RF_NO_PACKET:           Packet is not yet received
 +                          RF_CRC_ERROR:           Received a packet with CRC error
 +      NOTES:
 +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

RF_ENUM RFPacketReceived (uint8 * packet, uint8 * length)
{
        xdata uint8 i;

  // Check if IT occurred or not
        if( RF_NIRQ_PIN == 0 )
        {
        /* check what caused the interrupt */
                // read out IT status register
                ItStatus1 = SpiRfReadRegister(InterruptStatus1);
                ItStatus2 = SpiRfReadRegister(InterruptStatus2);

                // packet received interrupt occurred
                if( (ItStatus1 & 0x02) == 0x02)
                {
                        // read buffer
                        *length = SpiRfReadRegister(ReceivedPacketLength) ;
                        for(i=0;i<*length;i++)
                        {
                                *packet++ = SpiRfReadRegister(FIFOAccess);
                        }

                        // disable receiver
                        SpiRfWriteAddressData((REG_WRITE | OperatingFunctionControl1), 0x01);
                        return RF_PACKET_RECEIVED;
                }

                // CRC ERROR interrupt occurred
                if( (ItStatus1 & 0x01) == 0x01 )
                {
                        // disable receiver
                        SpiRfWriteAddressData((REG_WRITE | OperatingFunctionControl1), 0x01);
                        return RF_CRC_ERROR;
                }
        }
        return RF_NO_PACKET;
}
```

11. C8051

The C8051.c module contains all the low level, 8051 dependent functions. The code mostly comprises of hardware SPI setup and SPI read/write function calls.

The SetHwMasterSpi() function initializes the 3-wire HW SPI port. This does not control the nSEL pin. The nSEL pin is controlled separately by RF_NSEL_PIN.

The SpiWrite() function sends data through the SPI port (8 bits length). The nSEL pin is controlled separately by RF_NSEL_PIN.

The SpiReadWrite() function sends and reads data via the SPI port (8 bits length). The nSEL pin is controlled separately by RF_NSEL_PIN.

The SpiRfWriteAddressData() function sends data through the SPI port (16 length - 8 bits address, 8 bits data). This function controls the nSEL pin.

The SpiRfWriteAddressData() function reads the current value of the register. This function controls the nSEL pin.

11.1. C8051 Header File

```
/*****************************************************************************************
*
**
**        FILE --- C8051.h
**
**        DESCRIPTION
**        Contains the 8051 specific declarations, IO declarations, type declarations
**
**        CREATED
**        Silicon Laboratories Hungary Ltd
**
**        COPYRIGHT
**        Copyright 2008 Silicon Laboratories, Inc.
**        http://www.silabs.com
**
*****************************************************************************************
/

#ifndef C8051_H
#define C8051_H

#include <compiler_defs.h>          // compiler declarations
#include <C8051F930_defs.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
            /* ========================================= *
             *              TYPE DECLARATION             *
             * ========================================= */

//Only these types of variables are used in this software
#undef    uint8
#undef    sint8
#undef    uint16
#undef    sint16
#undef    uint32
#undef    sint32

#define   uint8      unsigned char
#define   sint8      signed char
#define   uint16     unsigned short
#define   sint16     signed short
#define   uint32     unsigned long
#define   sint32     signed long

typedef struct
{
unsigned int bit0 : 1;
unsigned int bit1 : 1;
unsigned int bit2 : 1;
unsigned int bit3 : 1;
unsigned int bit4 : 1;
unsigned int bit5 : 1;
unsigned int bit6 : 1;
unsigned int bit7 : 1;
} reg;

typedef union
{
  reg testreg;
  uint8 adat;
}reg_union;

typedef union
{
  reg_union bytes[2];
  uint16 adat;
}reg16_union;

            /* =========================================*
            *              DEFINITIONS                  *
            * =========================================*/

#undef    TRUE
#undef    FALSE
#undef    INPUT
#undef    OUTPUT

#define   TRUE                  (1)
#define   FALSE                 (0)
#define   INPUT                 (1)
#define   OUTPUT                (0)
```

I/O definitions. The RF_NSEL_PIN and RF_NIRQ_PIN port are created separately as the Hardware SPI ports use only 3-wires.

```
//I/O pin definitions
SBIT(LED1_PIN,              SFR_P1, 4);
SBIT(LED2_PIN,              SFR_P1, 5);
SBIT(LED3_PIN,              SFR_P1, 6);
SBIT(LED4_PIN,              SFR_P1, 7);
SBIT(BLED_PIN,              SFR_P2, 2);
SBIT(PB1_PIN,               SFR_P0, 0);
SBIT(PB2_PIN,               SFR_P0, 1);
SBIT(PB3_PIN,               SFR_P2, 0);
SBIT(PB4_PIN,               SFR_P2, 1);

//RF chip
SBIT(RF_NSEL_PIN,           SFR_P1, 3);
SBIT(RF_NIRQ_PIN,           SFR_P0, 6);

//SPI port
SBIT(SPI_MISO_PIN,          SFR_P1, 1);
SBIT(SPI_MOSI_PIN,          SFR_P1, 2);
SBIT(SPI_SCK_PIN,           SFR_P1, 0);

//Test card EEPROM
SBIT(EE_NSEL_PIN,           SFR_P2, 6);

//LCD
SBIT(LCD_NSEL_PIN,          SFR_P2, 5);
SBIT(LCD_A0_PIN,            SFR_P2, 3);
SBIT(LCD_RESET_PIN,         SFR_P2, 4);
SBIT(LCD_BL_PIN,            SFR_P2, 7);

#define SYSCLK              (16000000L/2)      // SYSCLK frequency in Hz
#define  SPI_CLOCK          (SYSCLK/4)
#define  EnableGlobalIt()   EA = 1
#define  DisableGlobalIt()  EA = 0

        /* ======================================== *
        *       FUNCTION PROTOTYPES                *
        * ======================================== */

void SetHwMasterSpi(void);
void SpiWrite(uint8 spi_in);
uint8 SpiReadWrite(uint8 spi_in);
void SpiWriteByte(uint8 spi_in);
void SpiRfWriteAddressData(uint8 address, uint8 d);
uint8 SpiRfReadRegister(uint8 address);
uint8 SpiReadByteFromTestcardEEPROM(uint16 address);
void SpiWriteByteToTestcardEEPROM(uint16 address, uint8 d);
void SpiReadSegmentFromTestcardEEPROM(uint16 start_address, uint8 * d, uint8 length);

#endif
```

11.2. C8051 Source File

```
/*****************************************************************************************
**      FILE --- C8051.c
**
**      DESCRIPTION
**      Contains all the low level, 8051 dependent functions
**
**      CREATED
**      Silicon Laboratories Hungary Ltd
**
**      COPYRIGHT
**      Copyright 2008 Silicon Laboratories, Inc.
**      http://www.silabs.com
**
*****************************************************************************************/


#include "C8051.h"

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +
 +      FUNCTION NAME:      void SetHwMasterSpi(void)
 +      DESCRIPTION:        Initialize the HW SPI port
 +      INPUT:              Data
 +      RETURN:             None
 +      NOTES:              It doesn't control the nSEL pin
 +
 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

void SetHwMasterSpi(void)
{
        SPI1CFG   = 0x40;                                   //Master SPI, CKPHA=0, CKPOL=0
        SPI1CN    = 0x00;                                   //3-wire Single Master, SPI enabled
        SPI1CKR   = (SYSCLK/(2*SPI_CLOCK))-1;
        SPI1EN    = 1;                                      // Enable SPI1 module

        //set nSEL pins to high
        RF_NSEL_PIN = 1;
}
```

```
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +
 +       FUNCTION NAME:      void SpiWrite(uint8 spi_in)
 +       DESCRIPTION:        Sends 8 bits length data through the SPI port
 +       INPUT:              Data
 +       RETURN:             None
 +       NOTES:              It doesn't control the nSEL pin
 +
 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

void SpiWrite(uint8 spi_in)
{
        SPI1DAT = spi_in;                                   //write data into the SPI register
        while( SPIF1 == 0);                                 //wait for sending the data
        SPIF1 = 0;                                          //clear interrupt flag
}


/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +
 +       FUNCTION NAME:      uint8 SpiReadWrite(uint8 data)
 +       DESCRIPTION:        Sends and read 8 bits length data through the SPI port
 +       INPUT:              Data
 +       RETURN:             Received byte
 +       NOTES:              It doesn't control the nSEL pin
 +
 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

uint8 SpiReadWrite(uint8 spi_in)
{
        SPI1DAT = spi_in;                                   //write data into the SPI register
        while( SPIF1 == 0);                                 //wait for sending the data
        SPIF1 = 0;                                          //clear interrupt flag
        return SPI1DAT;                                     //read received bytes
}


/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +
 +       FUNCTION NAME:      void SpiRfWriteAddressData(uint8 address, uint8 data1)
 +       DESCRIPTION:        Sends 16 length data through the SPI port (address and data)
 +       INPUT:              Address - register address
 +                           Data - 8bit data
 +       RETURN:             None
 +       NOTES:              It controls the nSEL pin
 +
 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

void SpiRfWriteAddressData(uint8 address, uint8 d)
{
        RF_NSEL_PIN = 0;
        SpiWrite(address);
        SpiWrite(d);
        RF_NSEL_PIN = 1;
}
```

```
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +
 +       FUNCTION NAME:      uint8 SpiReadRegister(uint8 address)
 +       DESCRIPTION:        Read a register of the radio
 +       INPUT:              Address - register address
 +       RETURN:             Value of the register
 +       NOTES:              It controls the nSEL pin of the radio
 +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

uint8 SpiRfReadRegister(uint8 address)
{
        uint8 temp8;

        RF_NSEL_PIN = 0;
        SpiReadWrite( address );
        temp8 = SpiReadWrite( 0x00 );
        RF_NSEL_PIN = 1;
        return temp8;
}
```

12. Troubleshooting

Q1: My Software Development Board (SDB) displays an error message on startup.

A1: Factory firmware is designed to operate with officially approved testcards. The EBID (see Figure 20, “Test Card Characteristics EEPROM (EBID),” on page 17) contain an authentication code to enable the use of the testcard with the factory firmware.

Note: The EBID is only used in conjunction with factory firmware. EBID restrictions are not implemented by default in customer firmware.

Figure 67. Error Message

The standard error message highlights

1. A missing testcard
2. A missing EEPROM
3. An invalid EEPROM authentication code

In addition, the firmware revision is highlighted in order for technical support to assist you.

Q2: After the Silicon Labs splash screen (which contains the firmware revision), there is an additional screen shown before the setup menu's - What is this for?

A2: Authentication codes in the EBID enable the Silicon Labs to qualify a factory firmware build to a particular testcard. If you received these message screens then a testcard is either an engineering testcard or is a specially modified testcard for specific customers. Customers that have opened a technical support request and that have special requirements may have received modified testcards—in this event, a notification will be displayed.

DOCUMENT CHANGE LIST

Revision 0.2 to Revision 0.3

- Added Lab Mode instructions.
- Added software programmers guide.
- Updated "7.1.3. Screen 3: Setting up Further RF Parameters" on page 14.
- Updated "7.1.8. Running the Demonstration" on page 20.
- Updated "7.2. Lab Mode" on page 22.
- Added Table 5, “Test Cards Available for Ordering,” on page 22.

Revision 0.3 to Revision 0.4

- Updated SDBC package kit to reflect new contents.
  - Antenna diversity and split card no longer supplied in the kit.

NOTES:

CONTACT INFORMATION

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
Tel: 1+(512) 416-8500
Fax: 1+(512) 416-9669
Toll Free: 1+(877) 444-3032

Email: wireless@silabs.com
Internet: www.silabs.com

The information in this document is believed to be accurate in all respects at the time of publication but is subject to change without notice. Silicon Laboratories assumes no responsibility for errors and omissions, and disclaims responsibility for any consequences resulting from the use of information included herein. Additionally, Silicon Laboratories assumes no responsibility for the functioning of undescribed features or parameters. Silicon Laboratories reserves the right to make changes without further notice. Silicon Laboratories makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Silicon Laboratories assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. Silicon Laboratories products are not designed, intended, or authorized for use in applications intended to support or sustain life, or for any other application in which the failure of the Silicon Laboratories product could create a situation where personal injury or death may occur. Should Buyer purchase or use Silicon Laboratories products for any such unintended or unauthorized application, Buyer shall indemnify and hold Silicon Laboratories harmless against all claims and damages.

Silicon Laboratories and Silicon Labs are trademarks of Silicon Laboratories Inc.

Other products or brandnames mentioned herein are trademarks or registered trademarks of their respective holders.

OCEAN CHIPS
Океан Электроники
Поставка электронных компонентов

Компания «Океан Электроники» предлагает заключение долгосрочных отношений при поставках импортных электронных компонентов на взаимовыгодных условиях!

Наши преимущества:

- Поставка оригинальных импортных электронных компонентов напрямую с производств Америки, Европы и Азии, а так же с крупнейших складов мира;
- Широкая линейка поставок активных и пассивных импортных электронных компонентов (более 30 млн. наименований);
- Поставка сложных, дефицитных, либо снятых с производства позиций;
- Оперативные сроки поставки под заказ (от 5 рабочих дней);
- Экспресс доставка в любую точку России;
- Помощь Конструкторского Отдела и консультации квалифицированных инженеров;
- Техническая поддержка проекта, помощь в подборе аналогов, поставка прототипов;
- Поставка электронных компонентов под контролем ВП;
- Система менеджмента качества сертифицирована по Международному стандарту ISO 9001;
- При необходимости вся продукция военного и аэрокосмического назначения проходит испытания и сертификацию в лаборатории (по согласованию с заказчиком);
- Поставка специализированных компонентов военного и аэрокосмического уровня качества (Xilinx, Altera, Analog Devices, Intersil, Interpoint, Microsemi, Actel, Aeroflex, Peregrine, VPT, Syfer, Eurofarad, Texas Instruments, MS Kennedy, Miteq, Cobham, E2V, MA-COM, Hittite, Mini-Circuits, General Dynamics и др.);

Компания «Океан Электроники» является официальным дистрибьютором и эксклюзивным представителем в России одного из крупнейших производителей разъемов военного и аэрокосмического назначения «**JONHON**», а так же официальным дистрибьютором и эксклюзивным представителем в России производителя высокотехнологичных и надежных решений для передачи СВЧ сигналов «**FORSTAR**».

AVIC JONHON

«JONHON» (основан в 1970 г.)

Разъемы специального, военного и аэрокосмического назначения:

(Применяются в военной, авиационной, аэрокосмической, морской, железнодорожной, горно- и нефтедобывающей отраслях промышленности)

«FORSTAR» (основан в 1998 г.)

ВЧ соединители, коаксиальные кабели, кабельные сборки и микроволновые компоненты:

(Применяются в телекоммуникациях гражданского и специального назначения, в средствах связи, РЛС, а так же военной, авиационной и аэрокосмической отраслях промышленности).

FORSTAR®

Телефон: 8 (812) 309-75-97 (многоканальный)
Факс: 8 (812) 320-03-32
Электронная почта: ocean@oceanchips.ru
Web: http://oceanchips.ru/
Адрес: 198099, г. Санкт-Петербург, ул. Калинина, д. 2, корп. 4, лит. А