AN1211.01

SX1211 Wireless Star Network with FHSS

Table of Contents

1 Introduction

The purpose of this application note is to document an SX1211 based Wireless Star Network (**WSN**) using a frequency hopping spread spectrum protocol (FHSS).

The application note consists of this document in addition to a zip file containing both hardware and software source files made available for every user to be able to customize this generic application to their specific needs.

1.1 SX1211 transceiver

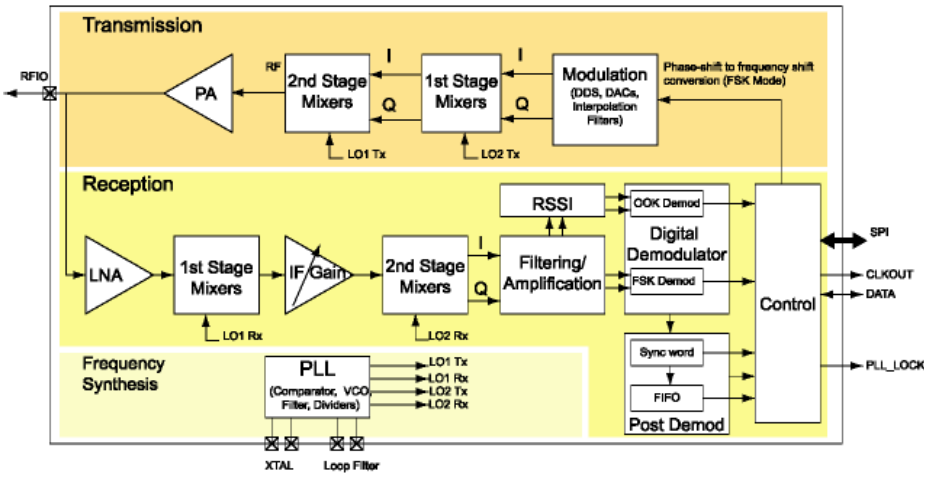


Figure 1: SX1211 Block diagram

The SX1211 is a low cost single-chip transceiver operating in the frequency ranges from 863-870, 902-928 MHz and 950-960 MHz. The SX1211 is optimized for very low power consumption (3mA in receiver mode). Its highly integrated architecture allows for minimum external component count while maintaining design flexibility. All major RF communication parameters are programmable and most of them may be dynamically set. It complies with European (ETSI EN 300-220 V2.1.1) and North American (FCC part 15.247 and 15.249) regulatory standards.

With small modifications to the SX1211 register configuration, the SX1212 transceiver may be supported by this FHSS firmware for operation on the 300 to 510MHz band.

- Low Rx power consumption: 3mA
- Low Tx power consumption: 25 mA @ +10 dBm
- Good reception sensitivity: down to -107 dBm at 25 kb/s in FSK, -113 dBm at 2kb/s in OOK
- Programmable RF output power: up to +12.5 dBm in 8 steps
- Packet handling feature with data (de)whitening and automatic CRC calculation
- Wide RSSI (Received Signal Strength Indicator) dynamic range, 70dB from Rx noise floor
- Bit rates up to 200 kb/s, NRZ coding
- On-chip frequency synthesizer
- FSK and OOK modulation
- Incoming sync word recognition
- Built-in Bit-Synchronizer for incoming data and clock synchronization and recovery (RX mode)
- 5 x 5 mm TQFN package
- Optimized Circuit Configuration for Low-cost applications

2 Application Description

2.1 Overview

As illustrated in *Figure* 2 below, the application is a simple star network consisting of one Master and 4 Slaves.

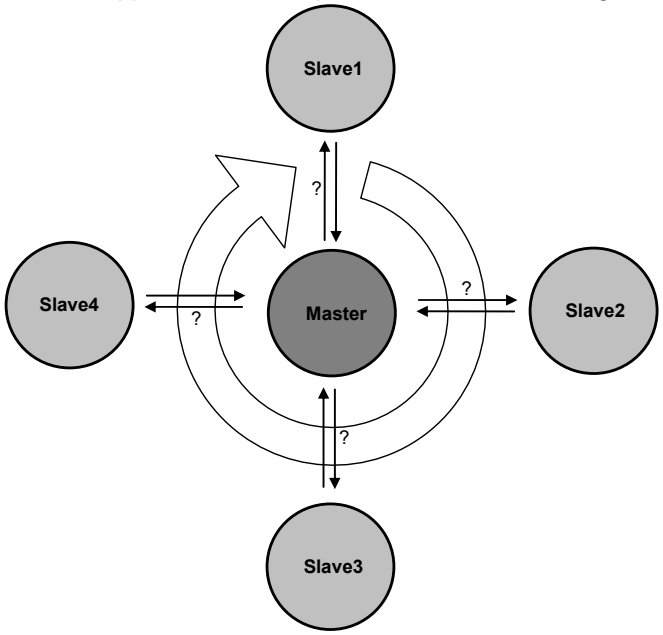


Figure 2: WSN application overview

2.2 Master Duties

The Master is the central node of the network. In the example hardware it is powered through the USB port of the PC to which it is connected.

The Master's tasks are:

- Synchronizing the network at system start-up (time + frequency hop).
- Monitoring the status of each Slave by sequentially checking their status to detect a loss of synchronization or an alarm.
- Ensuring synchronization in a noisy environment.
- Re-synchronizing Slaves which have lost sync.
- Processing a Slave's alarm (in this example the information is sent out the UART).

2.3 Slaves Duties

The Slave is battery powered and fitted with an alarm signal which can, for example, be the output of a smoke detector (simulated by a DIP switch in our case). Its tasks will be:

- Synchronizing to the Master at start-up and re-synchronize if needed.
- Monitoring its alarm signal and report status when requested by the Master.
- Keeping the synchronization in a noisy environment.
- Consuming as little power as possible to save batteries.

3 Hardware Description

3.1 Block Diagram

Every node of the network (Master or Slave) is based on the "WSN Node" hardware described below.

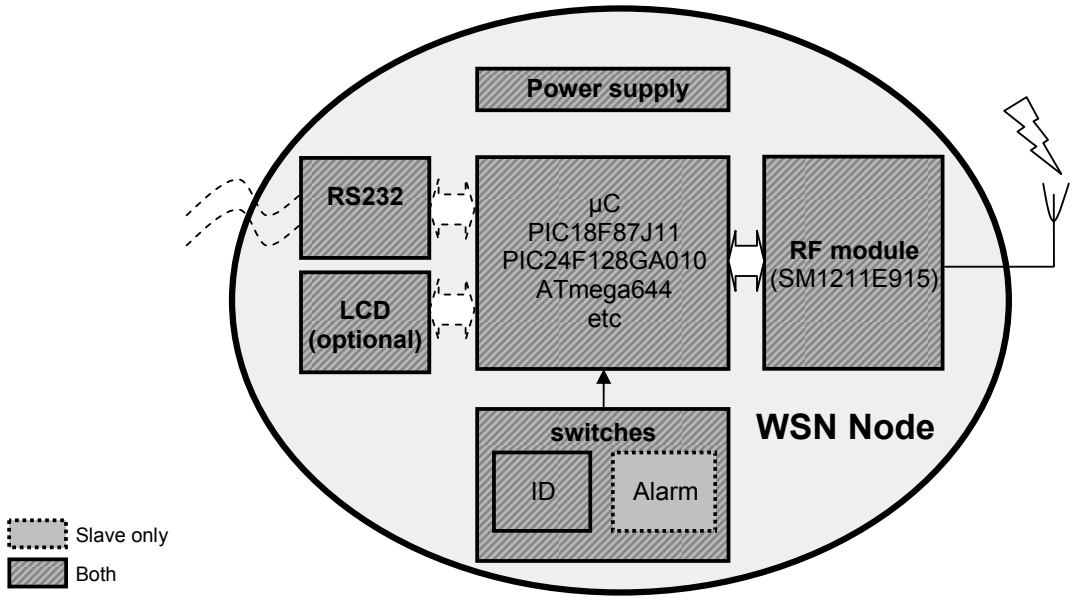


Figure 3: WSN node hardware block diagram

3.2 Hardware Platform

The supplied source code allows the hardware platform to be implemented microcontroller platforms from the following vendors:

- Atmel
- Microchip:

3.2.1 Atmel AVR

3.2.1.1 Using the Semtech-AVR demo board design files

The demo platform can be manufactured from the design files (schematics, layout) located in "\Hardware" folder of the zip file. The module can be manufactured "as is", or customized as required (form factor, etc).

The design has been made with Altium Designer 6.7.

Please note that the SM1211E915 (one per node) that plugs onto the demo board must be ordered separately.

Note that the SM1211 reference design file is available as a download from the Semtech website and can be used to easily integrate the RF part on a unique PCB, together with the µC and the other parts of the WSN node hardware.

3.2.1.2 Using Atmel tools

Standard tools are available from the IC vendors and can be interconnected together by following the provided schematics of the demo board as a model.

Here is an example list of the main standard tools to be used to build a WSN node:

- RF module : Semtech SM1211E915
- µC : Atmel STK500 (www.atmel.com)

- USB bridge (Master only, if connected to a PC) : FTDI DLP2232M-G (www.ftdichip.com)

These two options will provide equivalent hardware and, consequently, in the rest of the document we will only refer to the "Demo board" as hardware platform.

3.2.2 Atmel AVR Demo board overview

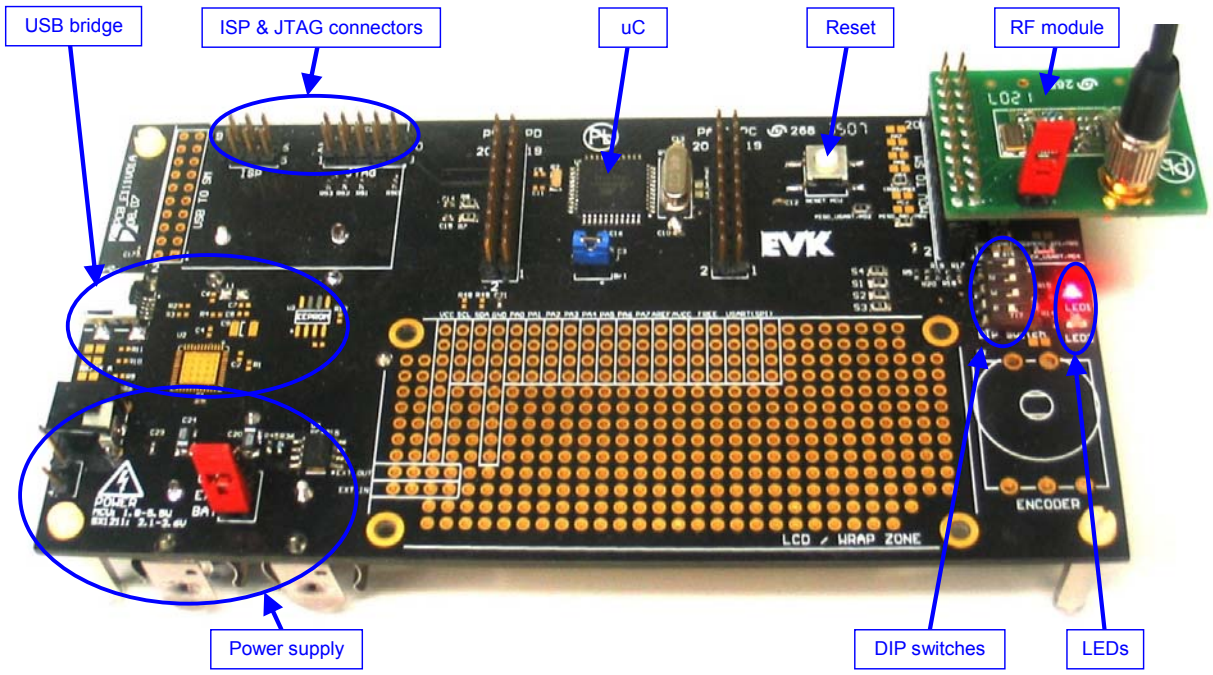


Figure 4: WSN demo board overview

3.2.3 Atmel Demo board features

3.2.3.1 Power supply

The board power supply is jumper-selectable between 3 sources:

- Batteries (3.6V Li-on or 2x1.5V AA),
- USB (regulated on-board to 3.3V)
- External (via 2pins 2.54mm header or jack connector, max 3.6V)

Please refer to the demo board design files for more details.

3.2.3.2 USB bridge

The USB bridge section is built around the FTDI FT2232. In the demo it is used by the Master node to create a communication path between the µC and a PC with UART protocol.

Please refer to the demo board design files and/or FT2232 datasheet for more details.

3.2.3.3 µC

The µC is the ATmega644P from Atmel. Some of its features are listed below:

- Voltage supply range : 1.8v – 5.5v
- Memory : 64K flash/2K EEPROM/4K SRAM
- consumption : 167nA – 2.7mA @ >8MHz
- 1 MIP / MHz

Please refer to the demo board design files and/or ATmega644P datasheet for more details.

3.2.4 Microchip

The two supported platforms are PIC18 and PIC24. Evaluation boards for these platforms can be purchased.

An adaptor board is used to connect the SM1211E915 into the provided daughter card socket on the microchip evaluation boards.

Figure 5 - PIC18 Explorer board (DM183032)

On the PIC18 Explorer board, the SM1211 adapter board plugs into J3 PICtail™ connector, labelled "PICDEM PIC18". The plug-in PIC18F87J11 "PIM" must be used to operate I/O pins at 3.3v. The on-board CPU uses 5v I/O. See Section 7 for wiring diagram.

The binary file wsn_fhss_pic18.hex is provided for those who want to program the firmware without compiling from source. This firmware is built for use with SM1211E915.

Figure 6 - Explorer 16 development board (DM240001) for PIC24

On the Explorer 16 board, the SM1211 adapter board plugs into J5 PICtail+™ connector. See Section 8 for wiring diagram.

On these boards, the operating mode (Slave vs. Master) is changed by pushing the right-most push-button nearest the PICtail+™ connector. The current operating mode can been seen on the top line of the LCD display. The current status of the firmware is displayed on the 2nd line of the LCD. This same status is printed out on the RS232.

The binary file wsn_fhss_pic24.hex is provided for those who want to program the firmware without compiling from source. This firmware is built for use with SM1211E915.

3.2.5 RF modules

All RF modules use the same 20pin 2x10 0.100" pin connector.

3.2.5.1 SM1211E915 RF module

The RF module is the SM1211E915, based on the SX1211

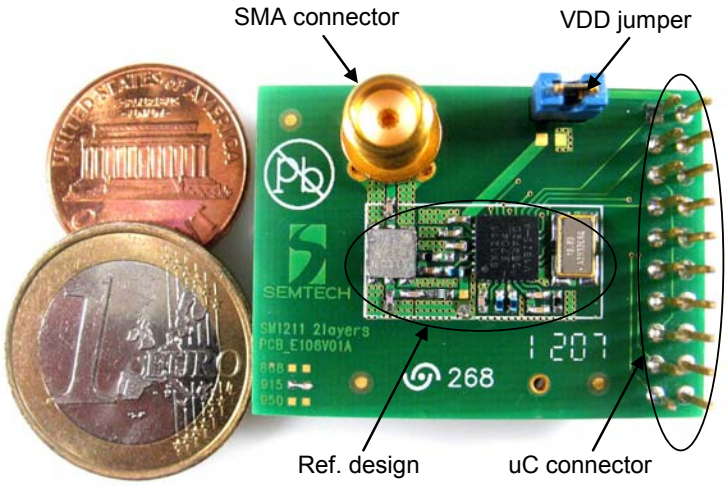


Figure 7: SM1211 RF module overview

Please refer to the demo board design files, SM1211 User's guide and SX1211 Datasheet, available from the Semtech website, for more details.

3.2.6 Atmel AVR board DIP switches

DIP switches S2, S3, and S4 are used to configure, at power-up, which board will be the Master and which will be configured as Slaves.

DIP switch S1 is used only on the Slaves to simulate the alarm.

S4	S3	S2	Node function
0	0	0	Master
0	0	1	Slave N°1
0	1	0	Slave N°2
0	1	1	Slave N°3
1	0	0	Slave N°4
1	0	1	Free
1	1	0	Free
1	1	1	Free

S1	Alarm status
0	Off
1	On

Figure 8: DIP switches truth tables

Please refer to the demo board design files for more details.

3.2.7 Atmel AVR board ISP and JTAG connectors

These connectors, associated with corresponding interface hardware, are used for on-board µC software programming, development and debug.

Please refer to the demo board design files and STK500 user's guide for more details.

3.2.8 Atmel AVR board Reset button

This button performs a Reset of the µC; it is referred to later in the document.

Please refer to the demo board design files for more details.

3.2.9 Atmel AVR board LEDs

LED1 reflects the synchronization status while LED2 reflects the alarm status.

Please refer to the demo board design files for more details.

4 Software Description

4.1 Protocol definition

4.1.1 Overview

The protocol is based on two phases:

4.1.1.1 Synchronization:

At power-up all nodes are preconfigured with correct IDs but are not synchronized on the same radio channel. The purpose of this phase is for the Master to establish contact with every Slave to initialize the network. This is achieved via a broadcast command. The master re-enters synchronization if any of the four slaves fail to reply to the master.

4.1.1.2 Dialog:

Once all nodes are synchronized, the dialog phase can start. The Master addresses each of the 4 Slaves one after the other to check their status (synchronization+ alarm) and do the relevant action. Dialog frequency is changed at every cycle.

All the nodes of the network (Master + Slaves) know and share the same predefined tables of frequencies as well as the pseudo-random order in which they must be accessed (described in more details later in the document).

50 radio channels are used. The same 50 radio channels are used by both synchronization and dialog messages. The two states are orthogonal due to the use of unique node addresses for each message type.

4.1.2 Synchronization phase

4.1.2.1 Master (see Figure 15 flow diagram)

1) The Master starts the synchronization phase by broadcasting its frequency hopping number at a predefined fast pace (“fast hopping” 8ms each).
2) After sending the sync message on all 50 channels, the Master considers that all the Slaves waiting for synchronization are now synchronized. The length in time of this synchronization period depends on the data rate, the size of the packet, the number of frequencies in the synchronization band (50 in this implementation), and the slow hopping rate of the Slaves.
3) Although the master can send synchronization message and hop in under 6ms, the hop rate is throttled to 8ms to ensure consistency across microprocessor types and different clock rates. 50 radio channels are covered in 400ms: 8 * 50 = 400.
4) The Master then broadcasts a synchronization end message `SYNC_END` to indicate to the Slaves that dialog phase is about to start (after power-up) or resume (in case synchronization phase was re-entered following a synchronization loss of a Slave during dialog session).
5) Synchronization phase ends when the Master finally broadcasts the meeting point for all nodes of the network, i.e. the frequency hopping number of the dialog session to begin (after power-up, it will be the first one of the dialog frequencies) or to resume (after synchronization loss in dialog session, it is the next dialog frequency expected before re-entering the synchronization phase). This last frequency hopping number sent right after `SYNC_END` is also referred to `radio_channel_dialog`.

4.1.2.2 Slave (see Figure 16 flow diagram)

1) When a Slave enters synchronization phase (i.e. after power-up or after a synchronization loss in a dialog phase) it tries to receive a message on the whole synchronization frequency band, jumping from one channel to the other at a predefined slow rate (“slow hopping”) to allow the fast hopping Master to “catch” it.
2) Once the Slave has received a broadcast packet from its Master it is considered synchronized and will follow the Master at the same predefined fast pace (“fast hopping”) until it receives the synchronization end message `SYNC_END` which indicates to the Slave that dialog phase is about to start (after power-up)

or resume (in case synchronization phase was re-entered following a synchronization loss of the Slave during dialog session).

3) Synchronization phase ends for the Slave when it receives the next frequency number to be used to start or resume dialog session (`radio_channel_dialog`).

The synchronization phase is illustrated in the following diagram.

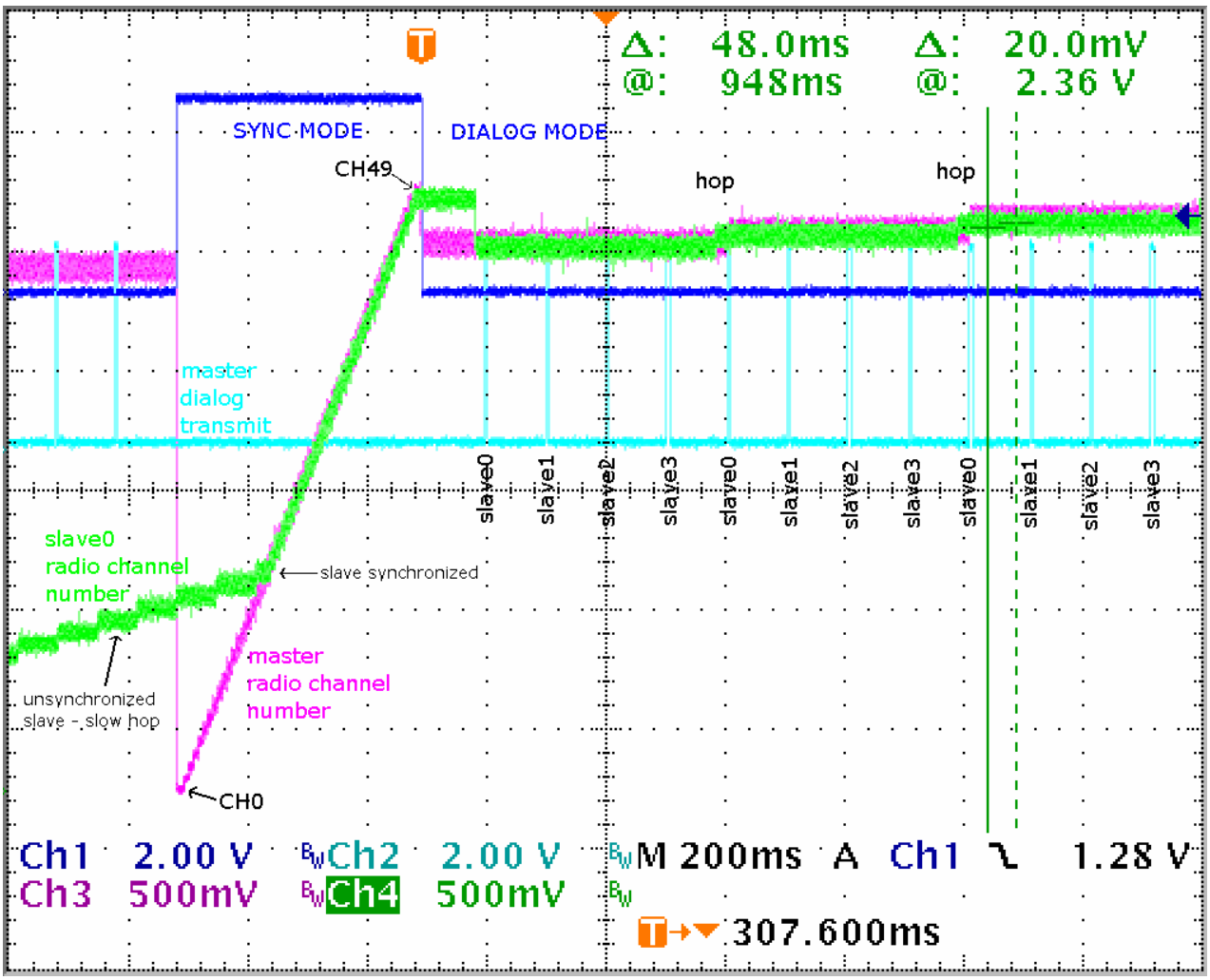


Figure 9: Synchronization phase overview

The magenta and green traces are analog outputs from the microcontroller representing the current radio channel being used. This oscilloscope screenshot shows an unsynchronized slave (Slave0) becoming synchronized with the master. The blue trace indicates the current mode, SYNC or DIALOG. The cyan trace indicates the master is transmitting in dialog mode. The green and magenta analog traces are updated with current hop number when the radio transceiver is commanded to a new frequency. None of these signals are required in the operation of the system, they only serve as an indication for diagnostic purpose.

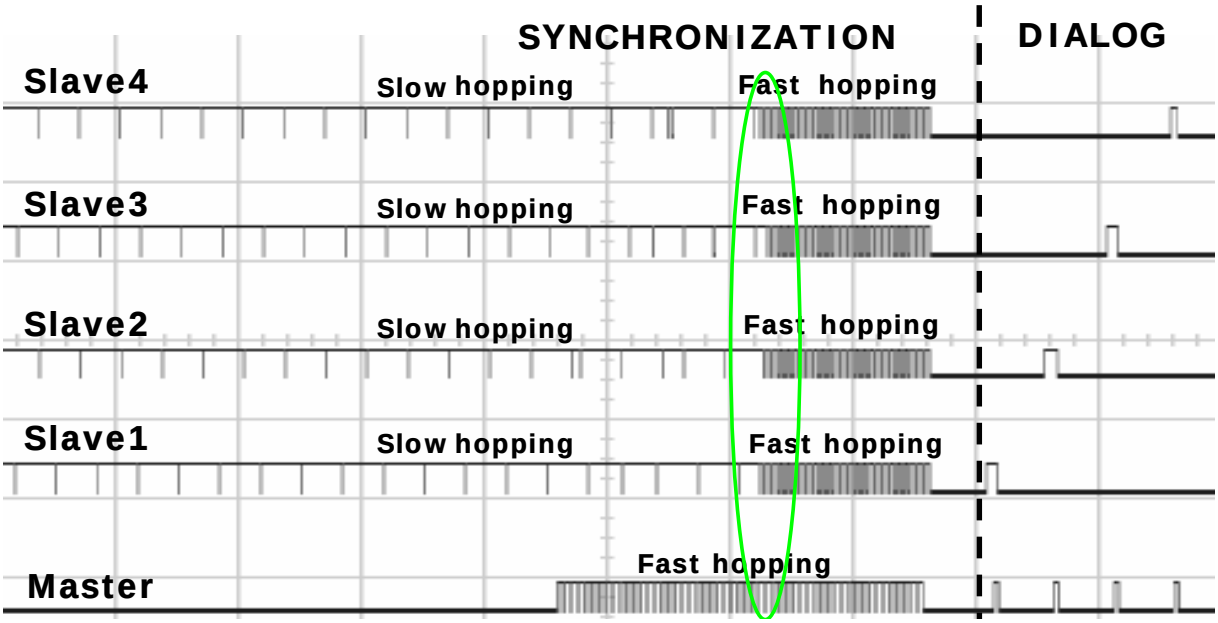


Figure 10: Synchronization phase at power-up

4.1.3 Dialog phase

4.1.3.1 Master

1. The Master will send a status "?" request to each Slave one after the other: Slave1 at t =0, Slave2 at t= 100ms, Slave3 at t=200ms, Slave 4 at t=300ms, Slave1 at t=400ms etc... Every time the Master completes a cycle of 400ms it will hop to the next dialog frequency. Note that the actual time of "100ms" is 101.5625ms due to the 256Hz rate of the 8-bit timer, for a total of 406.25ms per cycle.
2. Every time a Slave is interrogated the Master will expect an acknowledgment (alarm status)
3. If an alarm status is received as expected the Master will handle it (reported to the PC via UART)
4. The master attempts status request only once on the same radio channel because the slave uses the time of reception of status request to synchronize in time when to expect to be polled again on the next radio channel. At the next cycle the Master will try again to address the missing Slave.
5. If after a certain predetermined number of cycles of retry (3 in this implementation) the Master still hasn't received an answer, it will consider the Slave as unsynchronized and send, during next the cycle, a "Re-sync" command to the synchronized Slaves to indicate to them that they should sleep during the next cycle while Master will re-enter synchronization phase (Cf. above) to try to re-synchronize the missing Slave and bring it back into the network.
6. If any of the four slaves fails to respond to a status request (in dialog mode), the master sends message to all four slaves telling them synchronization mode "S" will now occur. All the synchronized slaves will then sleep for approx. 900ms, during which the master will be sweeping the channels with its synchronization broadcast. The slaves will then wake up at the correct time to receive the master status request in dialog mode. Only when all four slaves respond to status request will the master remain continuously in dialog mode.

4.1.3.2 Slave

1. Each Slave knows its position in the network and wakes up from sleep to Rx on the next dialog frequency every 400ms when it is supposed to receive a "Status?" request from the Master.
2. If the status "?" request is received successfully, the Slave will answer with its alarm status : "A" (alarm) or "K" (no alarm) and go back to sleep until its next wake-up slot in the next 400ms cycle
3. If the status "?" request is not received as expected, the Slave will, after a timeout, go back to sleep and retry to catch the "Status?" request at the next cycle.
4. If after a certain predetermined number of cycles of retry (1 in this implementation) the Slave still hasn't received a "Status?" command, it will consider itself unsynchronized and will re-enter the synchronization phase (Cf. above)

The dialog and Re-sync mechanisms are illustrated in figure below:

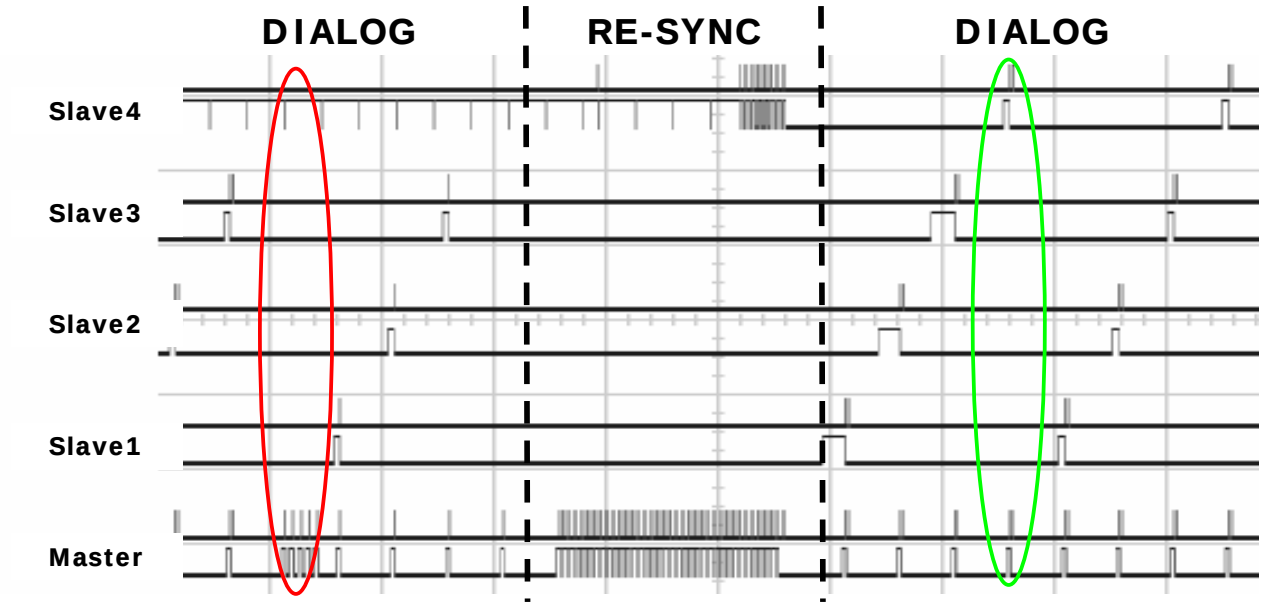


Figure 11: Slave4 has lost synchronization and is brought back to dialog phase by Master

4.1.4 Packet structure

4.1.4.1 Overview

The radio transceiver packet mode is used which greatly simplifies the packet definition and handling by the µC. Please refer to the SX1211 [1] for more details of the packet handler.

The generic packet structure of the WSN is described below and all nodes of the network (Master + Slaves) are configured internally to expect this same unique packet structure.

Preamble	***Sync Word (Network ID)***	***Length***	***Address (Node ID)***	***Payload (Data)***	***CRC***
0xAA,0xAA,0xAA,0xAA	0x69,0x81,0x7e,0x96	0x02	**0xXX**	**0xXX**	0xXX,0xXX

Figure 12: WSN packet structure

4.1.4.2 Preamble

Preamble length is configurable in a RF transceiver register. It is automatically generated at the beginning of each packet and is necessary for the receiver to synchronize its demodulator and be able to recover the following data properly.

4.1.4.3 Sync Word (Network ID)

Sync Word length and value are configurable in RF transceiver registers. Sync bytes are automatically generated after preamble and, associated with Sync Word filtering in the receiver, ensures that only the nodes (Master + Slaves) belonging to the same network (i.e. which have the same network ID) will be able to communicate together.

Any packet received with a different network ID (i.e. sync word) will be ignored.

Please refer to the SX1211 [1] for more details about the use of Sync Word filtering.

4.1.4.4 Packet Length

Although the length of all packets exchanged in this protocol is defined as constant (0x02), variable length packet format is used for potential future customization of the protocol. Packet format is configurable in RF transceiver registers.
Note that the value of the length byte does not include the length byte itself in the count.

Please refer to the SX1211 [1] for more details about the use of variable length packet format.

4.1.4.5 Address (Node ID)

Every node of the network is given a unique address depending on the configuration of the DIP switches and is configured accordingly in the `Node_Adrs` register in the RF transceiver. When transmitting the packet, the sender must mention the address of the recipient by sending the corresponding address. Associated with address filtering, this ensures that only the recipient node(s) will receive the information.

Any packet received with a node ID different from 0x00 (Broadcast) or its internally configured address will be ignored.

When the slaves are in sync mode (trying to acquire synchronization), the slave node is configured to only receive the broadcast address 0x00. When the slave enters dialog mode, the slave radio is reconfigured to only receive its own slave address (0x02 to 0x05). This allows both SYNC and DIALOG messages to share the same 50 radio channels.

Please refer to the SX1211 datasheet [1] for more details about the use of address filtering.

Address field can take the following values:

- 0x00: Broadcast (from Master to all Slaves)
- 0x01: Master
- 0x02: Slave 1
- 0x03: Slave 2
- 0x04: Slave 3
- 0x05: Slave 4

4.1.4.6 Payload (Data)

The Data field contains the information to be communicated and can take the following values depending on the situation and who is sending the packet:

From Master:

- [0-150]: Frequency hop number (current or next if following `SYNC_END` frame)
- [0xFA]: `SYNC_END`. This message is sent by the Master to indicate the end of the synchronization phase (i.e. all Slaves assumed synchronized)
- [‘?’]: “Status?”. This message is sent regularly to the Slaves during the dialog phase to check their alarm and indirectly also their synchronization status.
- [‘S’]: “Re-Sync”. This message is broadcast to all Slaves when at least one of them has lost synchronization. The Master will then re-enter synchronization to recover the missing Slave while synchronized Slaves will sleep during the next cycle.

From Slave:

- [‘K’]: “OK, no alarm”. This message is one of the two answers possible to a “Status?” message sent by the Master. It means that no alarm is set on this node (and also indirectly that the Slave is still synchronized).
- [‘A’]: “Alarm”. This message is one of the two answers possible to a “Status?” message sent by the Master. It means that an alarm is set on this node (i.e. DIP switch ON in this demo).

4.1.4.7 CRC

Data integrity control is performed via 2 bytes CRC which is automatically calculated, appended in Tx and checked in Rx by the packet handler of the RF transceiver. Packets received with a wrong CRC will be ignored.

Please refer to the SX1211 [1] for more details about the use of CRC filtering.

4.2 Code structure and implementation

4.2.1 Project structure overview

4.2.1.1 Platform independent source code

These files are contained within the `wsn_port` subdirectory. They implement frequency hopping network on Semtech RF transceiver devices , and don't depend on any particular CPU or compiler.

- **FHSSapi.c**: contains routines for master-slave synchronization.
- **FHSSapi.h**: declares functions and variables exported from FHSSapi.c to other files.
- **SX1211driver.c**: implements functions to operate the SX1211 transceiver.
- **SX1211driver.h**: defines SX1211 register addresses / constants.
- **Transceiver.h**: exports function prototypes to operate an RF transceiver, and defines constants for function return values and RF operating mode (sleep, RX, TX, etc).
- **WSN.c**: the main software, calls initialization, implements master and slave dialog routines.
- **Platform.h**: function prototypes for required CPU-dependant functionality, such as UART, SPI, and TIMER routines. The implementation of the functions is specific to each CPU, but the function prototype is constant across all platforms.

4.2.1.2 Platform dependant source code

These files are contained in a subdirectory under the `processors` subdirectory. The subdirectory will be named appropriately for the particular CPU for which it implements. These files implement the UART, SPI and TIMER functionality specific to the CPU being used. Additionally, the files supporting the compiler or IDE for the CPU will be contained under this subdirectory, such as Makefile or project/workspace files.

4.2.1.2.1 Required CPU-specific header files

- Types.h: defines the type for uint8_t and uint16_t, or includes the file which defines them.
- Cpu.h: defines macros for reading constants from program memory for Harvard architecture, and CPU initialization and watchdog stopping function prototypes.
- Timers.h: defines registers for accessing timers
- Io_port_mapping.h: defines registers for accessing pins on the CPU

4.2.2 Detailed descripton of frequency hopping source files.

4.2.2.1 FHSSapi.c

This file is written to be platform-independent; it does not depend on any particular µC.
The frequency hopping function `Fhss_Hop()` is specific to the RF transceiver used.
FHSSapi.c is agnostic to the transceiver being used.

- Function `uint8_t Sync_fhss(void)`

This function implements the synchronization phase for both master and slave. The return is the synchronization state (`NOT_SYNC`, `SYNC_END`, etc). The master sweeps all 50 channels with a broadcast message at a rate of 8ms per channel. The slave will continue to run this function until it has become synchronized to a master.

Please refer to the commented source code for more details.

4.2.3 RF transceiver-specific drivers: SX1211driver.c

This source file provides the functions to initialize the RF transceiver, and support transmitting and receiving RF messages. The SPI functions in spi.c are used to communicate with the transceiver. These drivers are written for any µC, however the SPI dependancy is specific to the µC being used.

The transceiver to be used is selected at compile time. Only one RF transceiver driver may be compiled in. No auto-detection of the RF transceiver is made at run-time. The transceiver driver compiled in must match the transceiver RF module plugged into the CPU.

The SX1211 driver is enabled by defining the pre-processor directive `SX1211`.

4.2.3.1 Function `void Fhss_Hop(uint8_t *hop_count)`

This function hops the RF transceiver to the next radio channel by applying frequency PLL values to the transceiver, then incrementing the hop_count. The function argument is provided to allow the caller to use separate count variables for synchronization and dialog mode hopping to prevent synchronization procedure from altering the current dialog channel.

- ***SX1211 hopping***

The registers R1, P1, and S1 are updated on each hop, including the band of operation: 902-915 or 915-928. Hopping only occurs with the RF transceiver in standby mode. The PLL_LOCK pin on the SX1211 is cleared before loading the new frequency for the purpose of observation of this pin during operation. When the transceiver has arrived at the new frequency, the PLL_LOCK will restore to HIGH logic level. To reduce SPI overhead, the NSS_CONFIG remains asserted throughout the hop procedure. The SX1211 does not require NSS_CONFIG to toggle between each register access.

- **SX1211driver.c Frequency tables**

A table of 50 radio channels is used for both synchronization and dialog. Due to the integer-N type PLL, the Semtech frequency calculator SX1211FreqCalc.exe is used to generate this table. The multichannel tab in the GUI is used with band edges of 903 and 927MHz to ensure compliance with FCC regulations. Each entry in the table contains R, P, and S values, as well as a value of the MCParam_Band. Support for crystal frequencies of 12.8MHz and 14.7456MHz are included in the SX1211driver.c.

The 300MHz and 400MHz UHF bands may be supported with minimal modification for using with SX1212 transceiver. These bands are not FCC compliant for frequency hopping, but may be permitted outside USA.

The pre-processor directive `SX1211_CRYSTAL_HZ` is defined with 12800000 or 14745600 to select the frequency table to be used. The choice of crystal frequency to use primarily depends on the RF bitrate desired. The SX1211 can be used with any crystal frequency between 9 and 15MHz.

Please refer to the commented source code for more details.

4.2.3.2 Function `uint8_t SendRfFrame(uint8_t *buffer, uint8_t size, uint8_t Node_Adrs, char immediate_rx)`

This function builds and sends an RF packet via the RF transceiver. This function blocks until the transceiver raises the Tx_done signal.

When the flag `immediate_rx` set to `TRUE` will cause the radio to immediately switch to receive mode instead of standby after transmit is complete. The purpose is to support reception of an acknowledgement to the sent message when the unit replying may be a faster CPU, perhaps causing reception to fail due to receiver not being enabled quick enough after transmission.

The packet engine receives the valid message by itself without CPU intervention, once the device is put into reception mode.

4.2.3.3 Function `uint8_t ReceiveRfFrame(uint8_t *buffer, uint8_t *size, uint16_t Timeout)`

This function facilitates reception of RF packets. It is a non-blocking state-machine; it must be called repeatedly until a return value other than `RX_RUNNING` is returned. On the initial call to this function, the transceiver is set up into receive mode. When reception is complete, `OK` will be returned, or `RX_TIMEOUT` if nothing was received within `Timeout` period. For the `Timeout` argument, the macro `HIRES_TIMEOUT()` is provided in timers.h to convert microseconds to the value needed for this `Timeout` argument.

If the RF transceiver was previously in receive mode from the SendRfFrame function, the initialization will be bypassed and only the timeout will be started.

4.2.4 WSN.c

This file is written to be platform-independent; it does not depend on any particular µC

4.2.4.1 Function `void OnMaster()`

This function implements the Master's duties (Sync, Dialog, Re-sync, communication with PC, etc).

Master dialog procedure described in section 4.1.3.1

See Figure 13 for flow diagram

4.2.4.2 Function `void OnSlave()`

This function implements the Slaves' duties (Sync, Dialog, Re-sync, alarm, etc).

Slave dialog procedure described in section 4.1.3.2

See Figure 14 for flow diagram

4.2.5 Platform-dependant functions

These functions are defined in files in a subdirectory under the `processors` directory. The subdirectory is named appropriately for the particular CPU for which it implements.

4.2.5.1 UART functions

- `void uart_init(void)`

Initializes UART. This function is called after initializing the transceiver to support CPU which is clocked from the RF transceiver clkout pin. Alternatively, the UART could be clocked from a crystal on the CPU.

- `void USART_send(const uint8_t *buffer, uint8_t size)`

This function sends bytes out the UART. This function is expected to be non-blocking, where bytes are loaded onto a circular buffer/fifo, and bytes are transmitted via interrupt.

- `void USART_send_str(const char *str)`

This is a convenience wrapper around USART_send() with strlen(), to simplify transmitting null-terminated strings.

4.2.5.2 TIMER functions

The header file timers.h provides macros `LOWRES_TIMEOUT()` and `HIRES_TIMEOUT()` to calculate the exact timer values needed for the following functionality.

- `void Wait(uint16_t compare_value)`

This function will block program execution for short periods using the high-resolution 16bit timer. The timer is expected to have a resolution of 2-3 microseconds and have a maximum delay value of approximately 65 milliseconds..

- `void EnableClock_HiRes(uint16_t compare_value)`

This non-blocking function initializes "high resolution" timer for expiration at a future time. The calling routine will poll `HIRES_COMPARE_B_FLAG` to check for expiration of this timer. The resolution and range requirements of this timer are the same as for the `Wait()` function: 16bits at approximately ~460KHz.

- `void go_sleep(void)`

This provides low-resolution, or long delays only for the slave. It is used to sleep both the RF transceiver and CPU when no communication will be taking place with this slave. A "low resolution" 8bit timer running at 256Hz will provide delays up to 1 second in 3.90625 millisecond steps. An ISR (found in timers.c) will wake up the CPU. The RF transceiver will put put into standby during this sleep period if the CPU main clock is driven by the RF transceiver clkout.

- `void EnableClock(uint16_t timeout)`

This non-blocking function initializes "low resolution" timer for expiration at a future time. The calling routine will poll `LOWRES_TIMER_FLAG` to check for expiration of this timer. The resolution and range requirement of this timer are the same as for the `go_sleep()` function: 8bits at 256Hz rate.

4.2.5.3 SPI functions

- `void SPIInit(void)`

This function initializes the SPI peripheral on the CPU to the configuration required by the RF transceiver. The pins NSS_CONFIG (and NSS_DATA for SX1211) will be configured as outputs and initialized to the unasserted HIGH state.

- `uint8_t SpiInOut (uint8_t outputByte)`

This function transfers a single byte over the SPI. This bus is full duplex, meaning that as a byte is being sent, a byte is also being received simultaneously. The pins NSS_CONFIG (or NSS_DATA for SX1211) will be handled by the caller in the RF transceiver driver.

4.3 Targeted µC platforms

The WSN software was tested on several different µC types to validate the portability of the source code. Only minimal effort should be required to build the software for other variants from these processor families, or to use a complete different µC vendor when similar peripheral capabilities exist.

All implementations on each CPU are compatible over the air, meaning (for example) that WSN-FHSS firmware running on the AVR ATmega644 can communicate to a PIC18 or PIC24, since all RF transceiver settings and firmware timings are identical across CPU platforms.

For each platform, the headers described in section 4.2.1.2.1 are provided, as well as drivers for SPI master port, UART and TIMERS (hi-res and low-res).

4.3.1 AVR ATmega644

Compiler used is GCC for AVR hosted on windows, called WinAVR. Supporting files are contained in the subdirectory `processors/atmega`.

Two crystals are used with this CPU: A high frequency (3.6864MHz) for the CPU clock, UART and 16bit “hires” timer1. A 32,768Hz watch crystal is used as clock source for “lowres” 8bit timer2.

The ATmega644 device has 64Kbytes of flash, yet less than 6Kbytes is needed by WSN-FHSS firmware, meaning a smaller AVR may be used.

The AVR family is Harvard architecture, meaning that an extra step is required to prevent constant variables from residing in RAM. On AVR this is called “pgmspace”. The frequency table, timing values, and strings are all constants which must be prevented from residing in RAM. In addition, the functions strcpy() and memcpy() have variants which support access to constant data in program flash. See `processors/atmega/cpu.h`

Three different peripherals of the µC can be used for SPI port: standard hardware SPI, USART0 SPI or USART1 SPI. By default USART1 SPI is used but it can be easily changed if needed by mounting 0R/NC resistors accordingly on the hardware(R25/R26, R29/R31, R46/R47 – Cf. schematics) and modifying the programmed SPI_MODE in SX1211driver.h as illustrated in figure below (SPI_MODE = N_SPI, U0_SPI or U1_SPI)

4.3.2 Microchip PIC18

The PIC18F87J11 is used on the PIC18 Explorer board (DM18032).

The PIC18F8722 installed on this board operates the I/O pins at 5V. All SM1211 boards must operate at 3.3V I/O. The plug in “PIM” board with the PIC18F87J11 will cause the board to operate all I/O at 3.3V. **Do not operate the RF transceivers with 5V I/O** from the board-mounted PIC18F8722.

To use the WSN-FHSS firmware with the PIC18 board, new firmware must be programmed using a programming device such as the ICD2 debugger. The MPLAB IDE (with MCC18 compiler) was used to build the firmware, and can also be used to operate the ICD2 to re-program the PIC18F87J11 on the Explorer board. The MPLAB workspace can be found in the WSN source-code at `processors/microchip/pic18f87j11/mplab/wsn_pic18.mcw`.

The WSN-FHSS firmware on this board will display operating mode (master/slave) on the top line of the LCD display. The 2nd line of the LCD display will show current activity, identically to that which is printed onto the RS232 port at 9600bps.

The operating mode of the firmware (Slave[0-3] vs. Master) is changed by pressing S2. This switch cannot generate an interrupt on the CPU (RA5 pin), meaning that in slave modes, when the CPU is sleeping in dialog mode, the switch must be held down for a longer period. In master mode, or unsynchronized slave, the S2 will respond immediately since the CPU is not sleeping.

The RB0 push-button is reserved for use by an RF transceiver IRQ pin.

An "analog" output, PWM-DAC is provided on the CCP4 output pin RG3. A resistor/capacitor filter can be installed on this pin to view the radio channel in use vs. time. This is only for diagnostic purpose of hopping software.

If building firmware using C18 compiler v3.3 or earlier, please verify the linker script used for the PIC18F87J11. Earlier tools had an error causing non-existent RAM to be used at address 0xf40 to 0xf59.

The relevant line of the linker script should end at address 0xf3f:

```
DATABANK   NAME=gpr15      START=0xF00          END=0xF3F
```

This applies to the files:

```
18f87j11.lkr      (extended instruction disabled, no debugger)
18f87j11_e.lkr    (extended instruction enabled)
18f87j11i.lkr     (for use with debugger)
18f87j11i_e.lkr   (using debugger, extended instruction enabled)
```

4.3.3 Microchip PIC24

The PIC24FJ128GA010 is used on the Explorer 16 board (DM240001).

To use the WSN-FHSS firmware with the PIC24 board, new firmware must be programmed using a programming device such as the ICD2 debugger. The MPLAB IDE (with C30 compiler) was used to build the firmware, and can also be used to operate the ICD2 to re-program the PIC24FJ128GA010 on the Explorer board.
The MPLAB workspace can be found in the WSN source-code at `processors/microchip/pic24fj128ga010/mplab/wsn_pic24.mcw`.

The WSN-FHSS firmware on this board will display operating mode (master/slave) on the top line of the LCD display. The 2nd line of the LCD display will show current activity, identically to that which is printed onto the RS232 port at 9600bps.

The operating mode of the firmware (Slave[0-3] vs. Master) is changed by pressing S4. This switch is polled, meaning that in slave modes, when the CPU is sleeping in dialog mode, the switch must be held down for a longer period. In master mode, or unsynchronized slave, the S4 will respond immediately since the CPU is not sleeping.

An "analog" output, PWM-DAC is provided on the OC3 output pin RD2, or pin 95 on the PICtail+™ connector. A resistor/capacitor filter can be installed on this pin to view the radio channel in use vs. time. This is only for diagnostic purpose of hopping software.

4.3.4 Renesas R8C/Tiny

Compiler used is NC30 with HEW IDE.

Supporting files are contained in the subdirectory processors/renesas.

The device used is R8C/1B with 16Kbytes of program flash.

No crystals are used with the CPU, the SX1211 clkout signal provides a clock to the XIN of the R8C/Tiny.

The “hires” 16bit support is provided by timer-C. The “lowres” 8bit support is provided by Timer-X prescaling for Timer-Z.

4.3.5 Texas Instruments MSP430

Compiler/IDE used is Code Composer Essentials (CCEv3).

Supporting files are contained in the subdirectory `processors/msp430`.

The .metadata subdirectory is deleted in the source archive to save space. After first opening the workspace, add back in the project: Project -> Open Existing Project, then browse to `/wsn_port/processors/msp430/wsn_fhss`.

The CPU operates from the clock provided by clkout from SX1211, driving XT2CLK. ACLK is provided by a 32,768Hz crystal.

The “hires” timer is provided by Timer-B, driven from the SX1211 clkout divided by 8. The “lowres” timer is provided by Timer-A, driven from the ACLK 32,768Hz crystal.

4.4 Software flow diagrams

Four diagrams are included to aid in reading the source code: OnMaster() and OnSlave() which are implemented in wsn.c, and one each Sync_Fhss() implementations for master and slave.

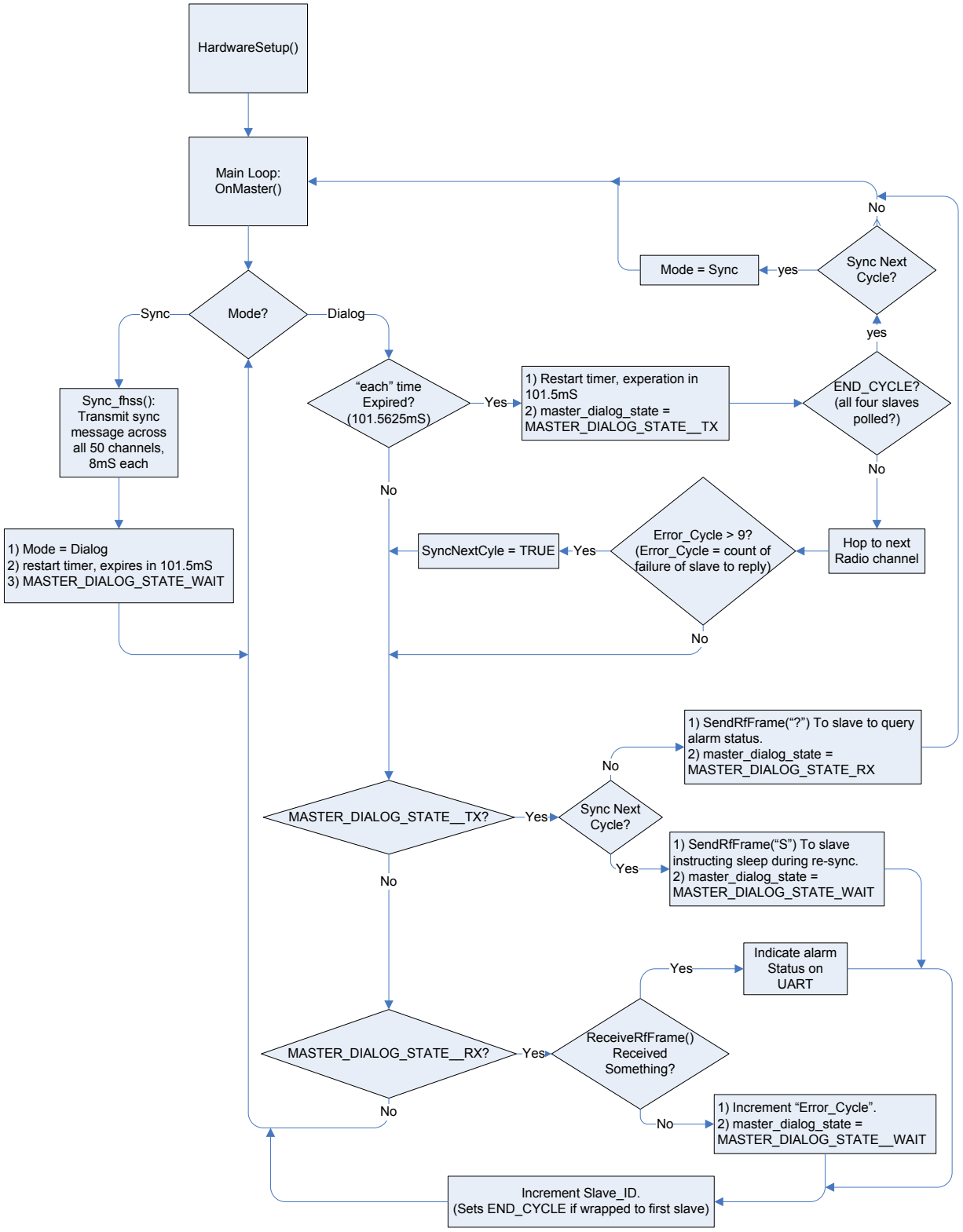


Figure 13 -- OnMaster() – calls Sync_Fhss() if any of the four slaves is not responding, and implements dialog mode.

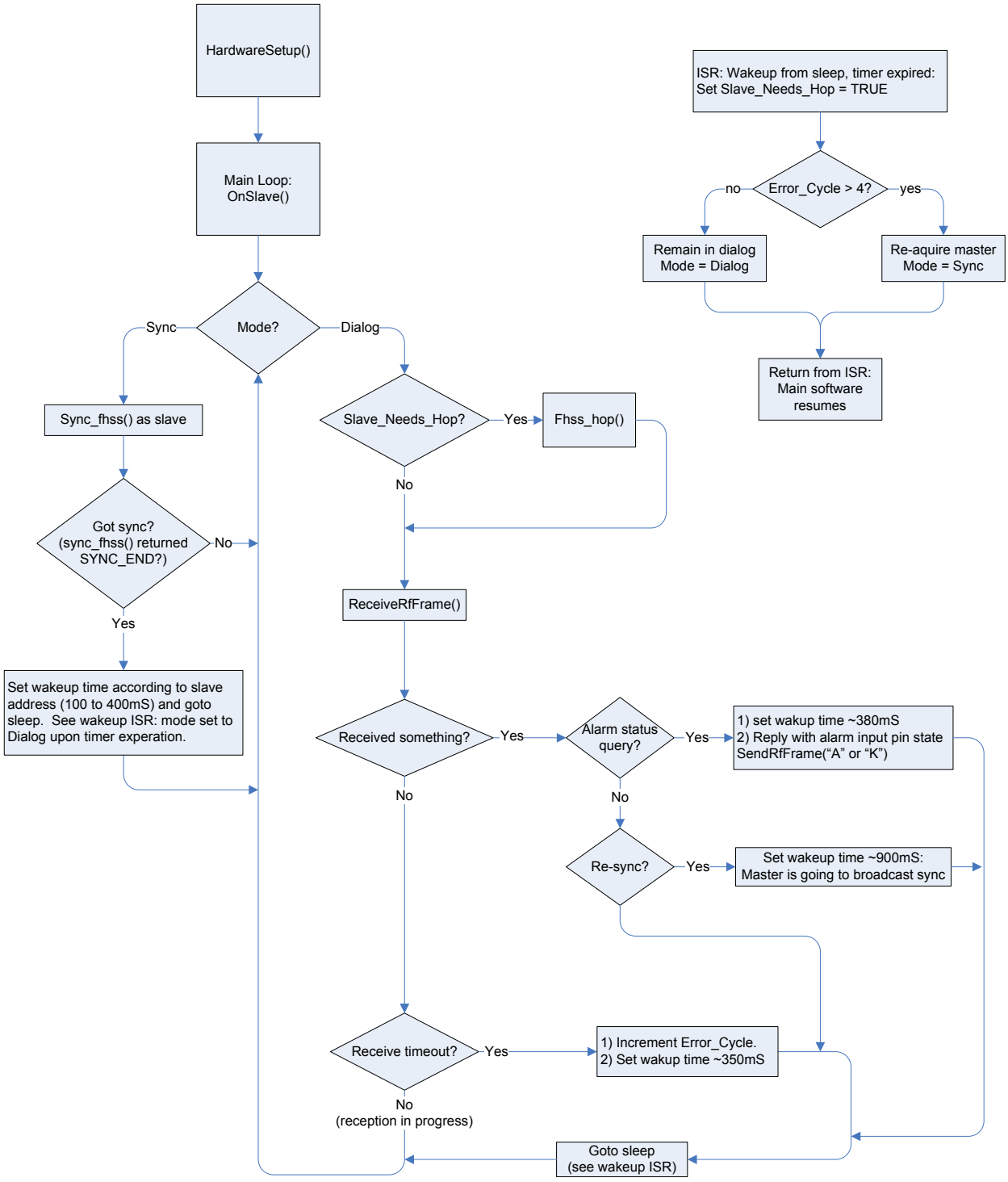


Figure 14 -- OnSlave() – calls Sync_Fhss() when not synchronized to the master, implements dialog mode as slave. Note the ISR in the upper-right corner is implement in a CPU-specific file.

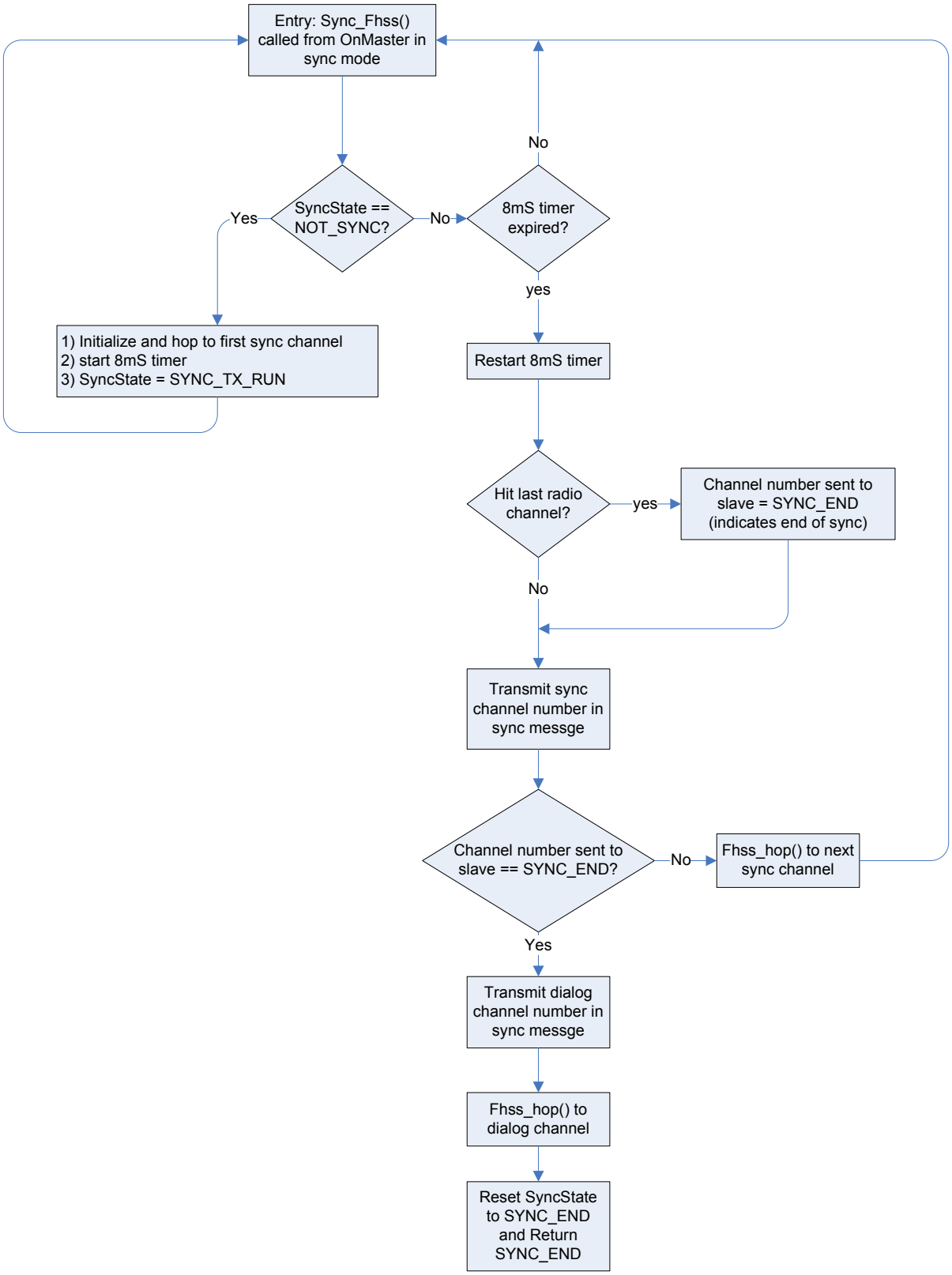


Figure 15 -- Sync_Fhss() as master – The sync messaging rate is throttled to 8ms, one sync message per radio channel, across 50 channels for 400ms total sync period.

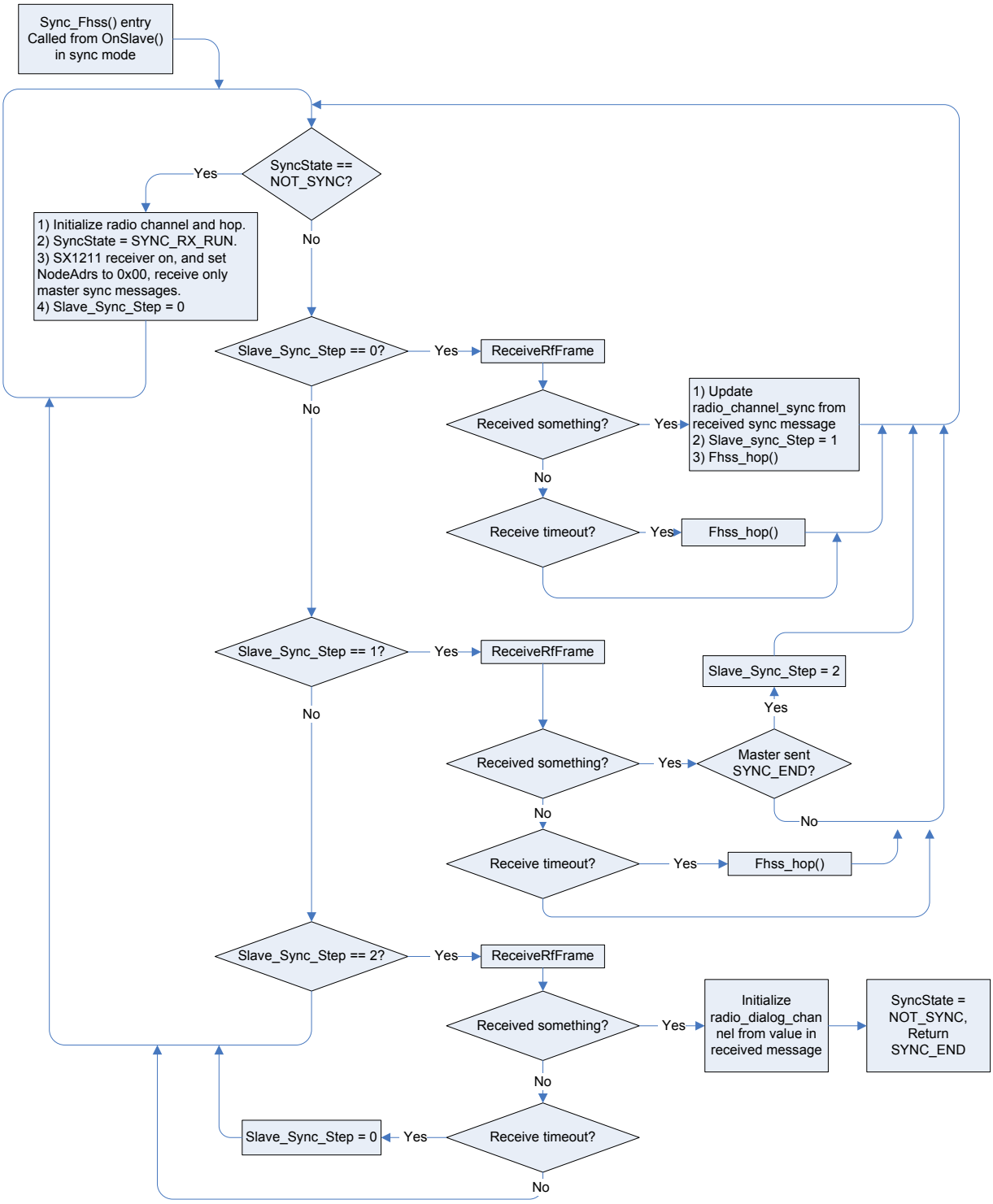


Figure 16 -- Sync_Fhss() as slave – responsible for acquiring sync messages from master, receives dialog radio channel number from last sync message.

4.5 Constraints, limitations and performance

4.5.1 Program and data memory footprint

Flash memory usage can vary on different µC platforms due to whether optimization level is targeting smaller program size or faster execution speed. Program size is expected to be under 6Kbytes, and RAM usage is expected to be under 500 bytes including any stack spaces.

4.5.2 µC requirements

If the WSN software is to be ported to another µC, the following requirements must be met when choosing the target reference:

- Equivalent data and program memory size bigger than footprint mentioned above.
- 1x USART, for the PC gateway (Master only).
- 1x 16bit timer "high resolution", to handle delays and timeouts.
- 1x 8bit timer "low resolution" which can be driven by an external low frequency crystal of 32.768 kHz. Configured for divide by 128 for 256Hz rate, 3.90625ms resolution.
- Timer **optional** 8bit (or 6bit) timer with one compare register capable of PWM output for diagnostics to indicate current radio channel on oscilloscope (PWM-DAC)
- Low power halt/sleep mode which can be interrupted by a timer. (for reduced slave power consumption)
- 1x Master SPI port, to communicate with the SM1211.
- I/Os (with PC connection): 21 minimum (SM1211, USB bridge, 2xLEDs, 4xDIP switch, low frequency crystal) + 11 optional (debug flags, emulation/programming modules, optional LCD, encoder).

4.5.3 Maximum number of Slaves

The hard limit for maximum number of slaves is 254.

The SX1211 Node_Adrs is used for addressing. Node_Adrs is 8bits, and address 0 is reserved for master broadcast. Each slave must have a unique Node_Adrs.

The trade-off is between the number of slaves in the system, and the cycle time of the master to poll all slaves. The more slaves added into the system will increase the cycle time. Additionally, larger payload size or reduced RF bit rate will increase the cycle time.

Conversely, larger (user data) bandwidth can be supported for fewer slaves.

4.5.4 SX1211 Slave average power consumption

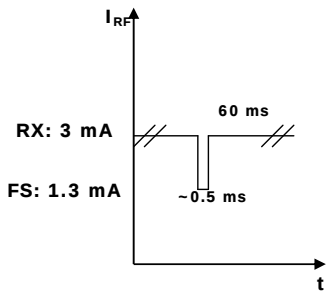
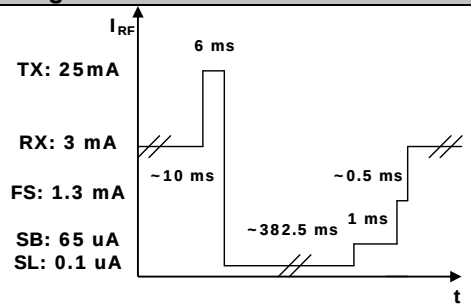
Synchronization (slow hopping)	Dialog
	
$I_{RF\_sync} \approx 2.99$ mA	$I_{RF\_dialog} \approx 0.45$ mA

Figure 17:SX1211 Slave node average power consumption

5 Demo User Guide

5.1 Installation

For the microchip PIC18 or PIC24 boards, each node needs one SM1211E915 and one PICtail(+) adapter board. WSN-FHSS firmware file must be programmed into each node of the network.
The firmware built for the appropriate RF transceiver must be used. The firmware only supports one transceiver type at run-time.

Even though the WSN is a 5-node network, it will operate correctly with a master and a single slave. When one of the four slaves is not responding, the master will regularly re-enter the sync mode.

5.1.1 PC software installation

On the microchip evaluation boards, sufficient status is displayed using the on-board LCD.
The UART is provided for platforms which do not have LCD, or for users who prefer status shown on UART.

The WSN-FHSS firmware uses a UART to transmit ASCII text, which allows a dumb terminal to monitor the master activity. You may use any dumb-terminal program of your choice, such as hyperterminal or teraterm, however some ANSI color codes are used to aid in the readability of the output.

The serial output from the master indicates the radio channel in use, and the status of each slave. The slave also outputs some diagnostic information, but this is only needed if diagnosing a slave.

5.2 Implementation

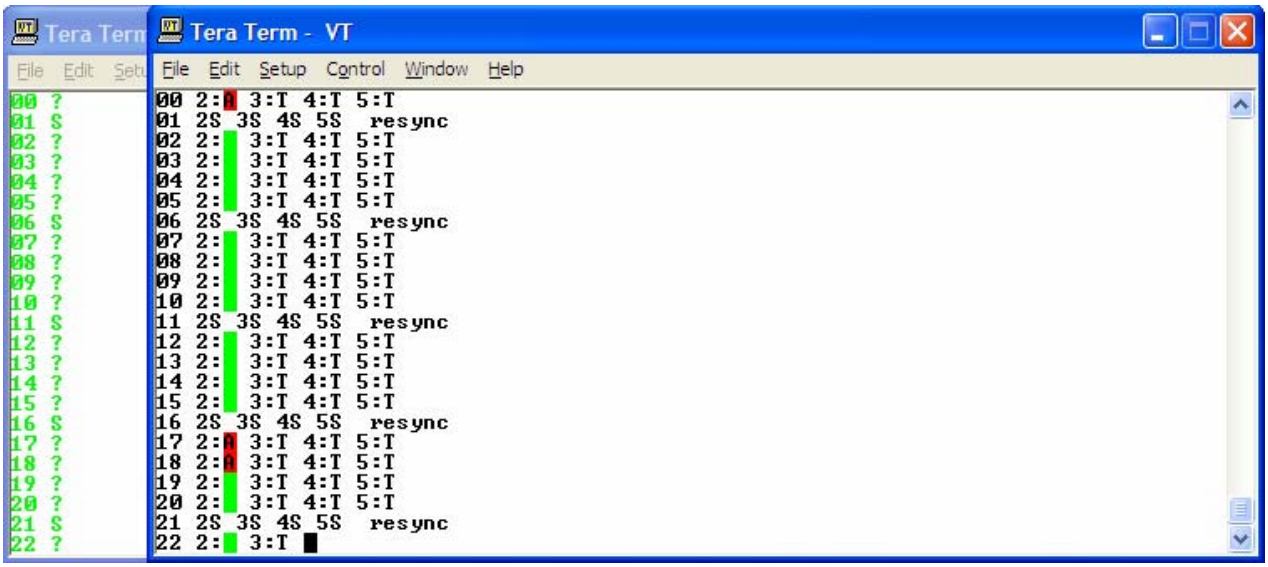


Figure 18: dumb-terminal connection to master and slave

A serial dumb-terminal is used to eliminate the need for specialized PC software.

The foreground window is connected to the master. The two-digit number printed at the start of line is the current radio channel. Then shown is <slave_node_address>:<reply_status> for each of the four slaves. The above diagram shows the first slave responding, but the other three slaves are not. 'T' indicates receive timeout, the red 'A' indicates the slave replying with alarm condition, and the green space indicates slave reply with no alarm.

The line "2S 3S 4S 5S" indicates that the resync command is being sent to all four slaves, indicating a synchronization cycle will begin. Slaves that receive this command will sleep for approximately 900ms while the master broadcasts synchronization for the slaves who are not responding. If all four slaves respond, the master will not enter the resync procedure, but simply remain continuously in dialog mode.

The terminal in the background is the slave. Just like the master, the current radio channel is printed at the start of each line. Next printed is the command received from the master. '?' is the status request, which the slave replies with alarm status. 'S' indicates a resync period will be performed by master, meaning to slave to sleep for approximately 900ms because this slave is already synchronized to the master.

The slave UART output is only for diagnostic purpose, and prints useful information when acquiring synchronization.

5.3 Compliance with FCC rules 47 CFR §15.247

5.3.1 Overview of the rules

The following summarizes the FCC rules as they apply to frequency hopping spread spectrum (FHSS) systems using the 902 to 928MHz United States ISM band.

It is recommended that the user refers to Government Printing Office website to download the latest revision of the Code of Federal Regulations (http://www.gpoaccess.gov/cfr/index.html)

§15.247(a)(1) requires that the 20dB bandwidth of the transmitted signal be less than the carrier (center) spacing between hopping channels. These hopping channels must be used in a pseudo-random order. Each of these hopping channels must be used equally on average. The receivers in the system shall have input bandwidths that correspond to that of the transmitted signal, and these receivers shall shift their frequency in synchronization with the transmitter(s) in the system.

§15.247(a)(1)(i) states that if the 20dB bandwidth is **less than** 250KHz, at least 50 hopping frequencies must be used. The average time of occupancy on any single frequency cannot be more than 400mS within a **20** second period.
§15.247(a)(1)(i) states that if the 20dB bandwidth is **greater than (or equal to)** 250KHz, at least 25 hopping frequencies must be used. The average time of occupancy on any single frequency cannot be more than 400mS within a **10** second period.

Note: A radio frequency is only occupied if a node in the system is transmitting. Nodes in receive or standby modes are not occupying a radio frequency.

§15.247(b)(2) permits up to +30dBm transmitter power when the above described rules are followed with at least 50 hopping channels, or +24dBm when at least 25 hopping channels are used. This permitted power is under the condition that antennas used have less than 6dB gain over an isotropic antenna. If more than 6dBi antenna is used, transmitter power must be reduced by the amount of antenna gain over 6dBi.

5.3.2 Hopping frequency assignments

50 radio channels are used with a constant spacing of 480 KHz between carrier/center frequencies. The lowest frequency is centered at 903.24MHz, and the highest at 926.76MHz. The random ordering of channels is obtained from http://random.org/sequences/ using a min and max of 0 and 49.

5.3.3 Transmitted modulation

The SX1211 transmits at a bit rate of 25 kbps bitrate with 50 KHz frequency deviation, when using the default 12.8MHz crystal of the SM1211 reference design.

5.3.4 Compliance description of hopping Algorithm

The following describes equal frequency usage, and how the hopping algorithm occupies each radio channel for less than 400ms in a 20 second period, using 50 hopping channels.

- Slave nodes in the system listen for synchronization messages from the master when the slaves are not previously synchronized. Once a slave has received the master synchronization message, it tracks the channels used by the master with identical timing. Slaves do not transmit until requested to by the master. Both slaves and master are programmed with identical hopping frequency tables.

- In synchronization mode, the master node sweeps 50 hopping channels at a rate of 8 milliseconds per channel. The transmitted message (occupancy) on each channel takes 4.16 milliseconds.. The total time spent in synchronization mode is 408ms for 50 hopping messages and one SYNC_END notification.
- In dialog mode, the master hops at a rate of 406.25 milliseconds. On each radio channel, each of the four slaves is contacted.
- Or, if any of the four nodes fail to respond, the system will re-enter synchronization mode for one sweep of all the channels with sync messages. The master cycles into synchronization mode every 2.54 seconds, with approx 2.13 seconds in dialog mode, and the other 0.4 seconds in the synchronization mode. Dialog mode hops five times between each synchronization mode, four attempts to contact each node and one notification that synchronization mode will begin.

5.3.5 Lab results of compliance test

These tests were performed on nodes operating at a 25,000bps bitrate from a 12.8MHz crystal on the SX1211.

- Carrier frequency separation: Measured 418 kHz
- 20 dB bandwidth: 285 kHz
- Band edge spurious: Within 100 kHz BW, spurious < -20 dBc
- Spurious conducted emissions: compliant

5.3.6 Permissible RF adjustments by hopping system developers

- Hopping channel assignments: Hopping system developers are encouraged to generate their own random channel sequence. Developers may also slightly reduce the channel spacing with care not to reduce it at or below the 20dB bandwidth of transmitted signal. The start and stop frequencies may be slightly shifted up or down with care not to exceed the band-edge spurious limit of -20dBc.

- Transmitted deviation: A wider deviation setting will increase occupied bandwidth. The 20dB bandwidth must not be increased to exceed the spacing of the hopping channels. The 20dB bandwidth cannot be reduced below 250 kHz if less than 50 hopping channels are used (this implementation uses 50 hopping channels).

- Power output: Transmitted power may be increased according to §15.247(b)(2) when the rules in §15.247(a)(1) are observed.

- Dialog-mode payload length: The developer can elect to increase the amount of payload transmitted with care not to exceed the 400ms total channel occupancy time (in a 20 second period with 50 channels). For example, a payload of 64 bytes would result in a message transmission time of 18 milliseconds per message.

5.3.7 Non FCC FHSS Implementation

For FHSS implementation in Canada using the 902 – 928 MHz band, please refer to Industry Canada / Industrie Canada RSS 210 (http://www.ic.gc.ca/eic/site/smt-gst.nsf/eng/sf01320.html)

With the CEPT countries please refer to ERC 70-03, available from the European Radiocommunications Office (http://www.erodocdb.dk/Docs/doc98/official/pdf/REC7003E.PDF), and ETS EN 300 220-1, available from the European Telecommunications Standards Institute (http://pda.etsi.org/pda/queryform.asp), for further information regarding the implementation of FHSS systems in the 863 – 870 MHz European ISM band.

6 References

- [1] RF transceiver Datasheets
  http://www.semtech.com/images/datasheet/sx1211.pdf
  http://www.semtech.com/images/datasheet/sx1231.pdf

- [2] TN8000.18 API
  http://www.semtech.com/images/datasheet/xe8000_tn_18_1200_api.pdf

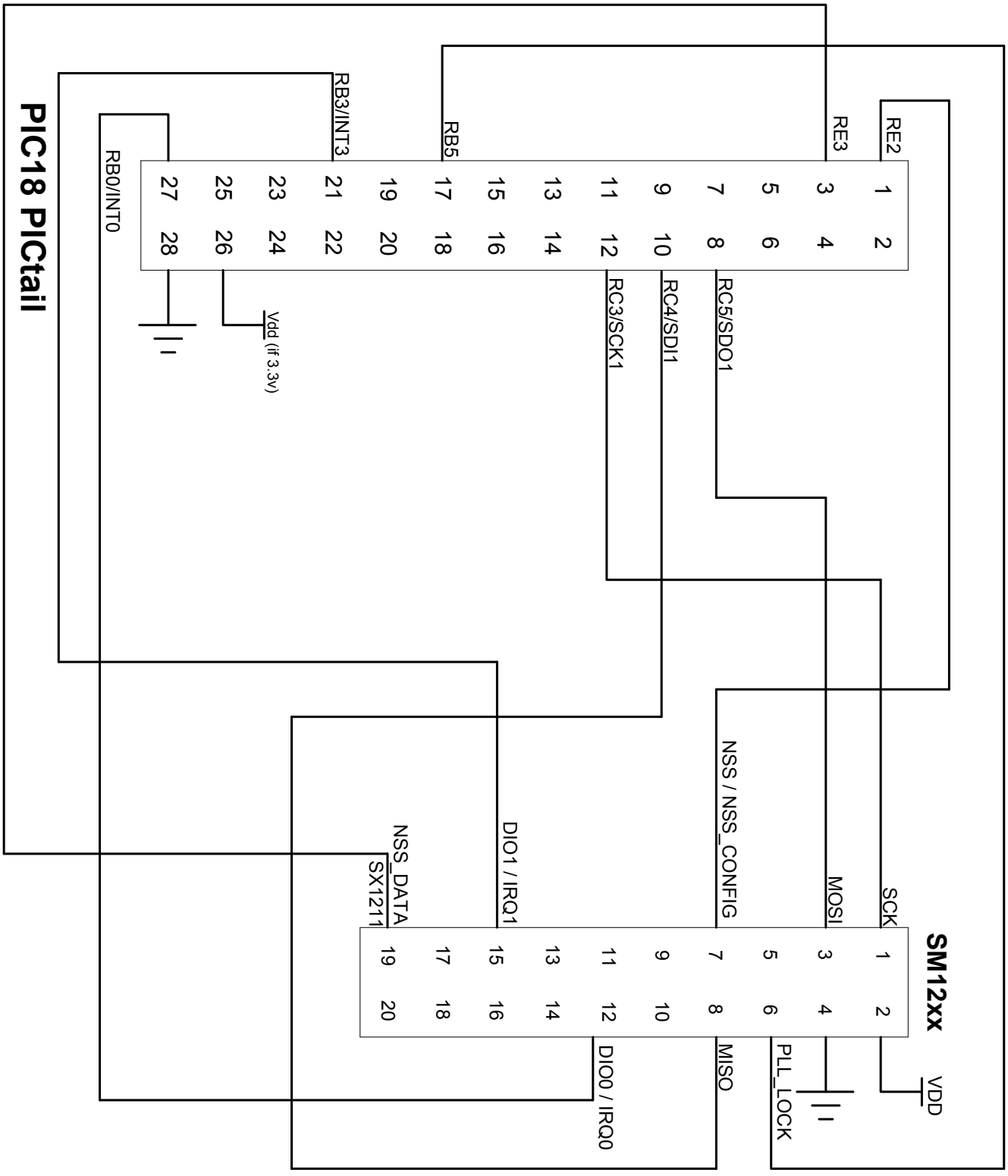
- [3] RF module user's guides
  http://www.semtech.com/images/datasheet/sm1211_users_guide_std.pdf
  http://www.semtech.com/images/datasheet/sm1231_user-guide.pdf

- [4] ATmega644P Datasheet
  http://www.atmel.com/dyn/products/product_card.asp?PN=ATmega644P

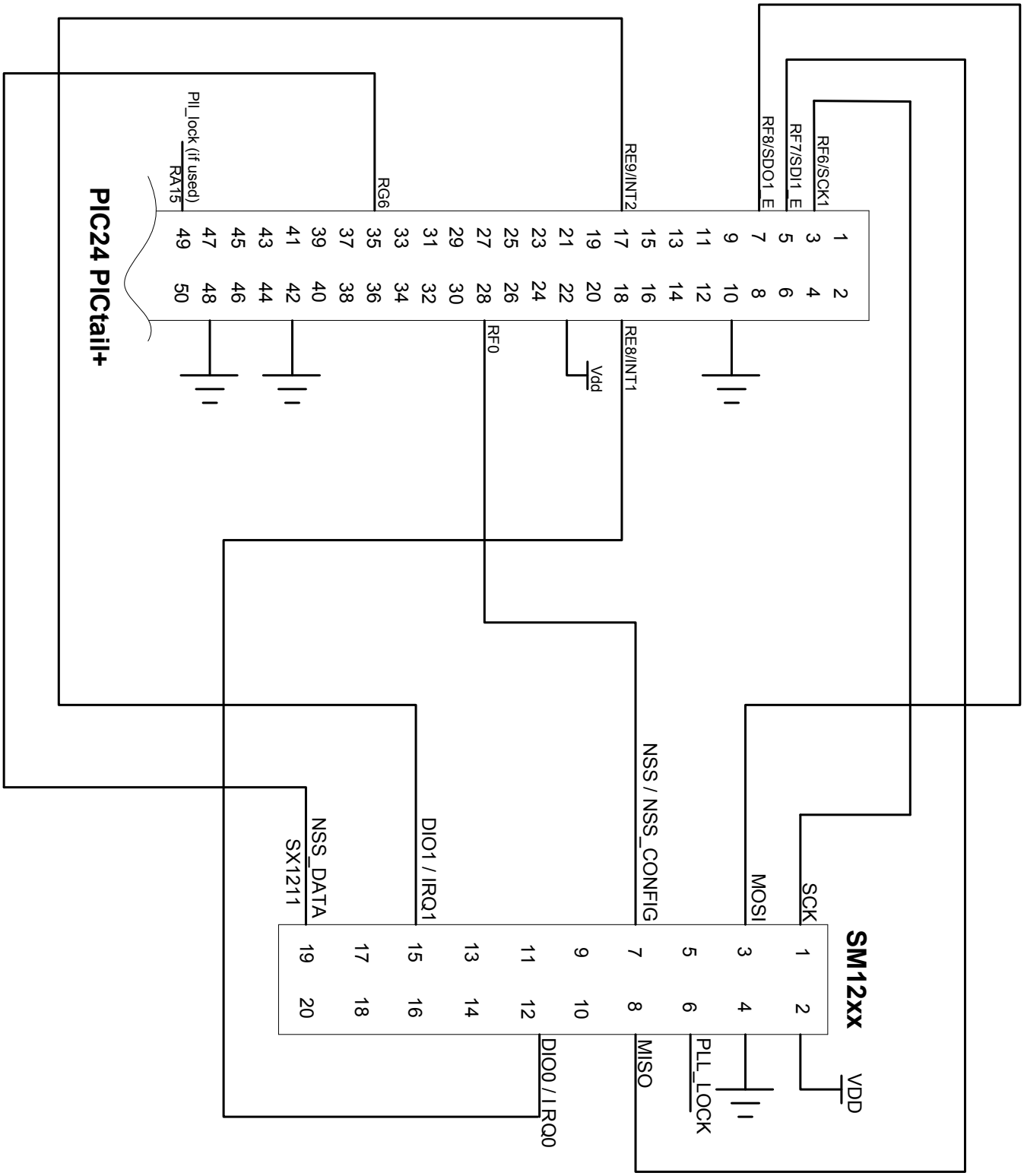
- [5] FT2232 Datasheet
  http://ftdichip.com/Products/FT2232C.htm

- [6] FCC rules for frequency hopping system
  http://edocket.access.gpo.gov/cfr_2007/octqtr/pdf/47cfr15.247.pdf

7 PIC18 wiring diagram



8 PIC24 wiring diagram



Contact Information

Semtech Corporation
Advanced Communications and Sensing Products Division
200 Flynn Road, Camarillo, CA 93012
Phone (805) 498-2111 Fax : (805) 498-3804