SMK900

Portia Radio Module

Datasheet

smartrek

Contents

1 Introduction 7

2 System 7

2.1 SMK900 System Overview . . . 7
2.2 Mesh Network Systems . . . 8
2.3 Frequency Hopping Implications . . . 8

3 Specifications 8

3.1 Broadcast Frame . . . 8
3.1.1 Dynamic parameters . . . 8
3.1.2 Broadcast Frame Structure . . . 10
3.2 SMK900 Addressing and Network Segregation . . . 11
3.3 Transparent and Protocol-Formatted Mode . . . 11
3.3.1 Transparent . . . 12
3.3.2 Protocol-formatted . . . 12

4 Virtual Machine 12

4.1 Description . . . 12
4.2 Virtual Machine Triggers . . . 12

5 Hardware 14

5.1 Serial Port . . . 14
5.2 Module Pin Out . . . 14
5.3 Power Supply and Input Voltage . . . 18
5.4 ESD and Transient Protection . . . 18
5.5 Antenna Connector . . . 20

6 Additional I2C Functions 20

7 Example VM User Script Using I2C Functions 21

8 Protocol-Formatted Messages 26

8.1 Protocol Formats . . . 26
8.2 Message . . . 27
8.3 Message Format Details . . . 27
8.3.1 Summary of message types . . . 27
8.4 EnterProtocolMode . . . 27
8.5 DeviceReset . . . 27
8.6 DeviceResetReply . . . 30

8.7 TxLongData 30
8.8 OTA 30
8.8.1 FASTWRITE 31
8.8.2 STARTTRANSFER 31
8.8.3 STOPTRANSFER 31
8.8.4 WRITE 31
8.8.5 READ 32
8.8.6 READFIRMWARECHECKOUTOK 32
8.8.7 GETMISSINGFLAGS 32
8.8.8 GETUNCHECKEDCRCPAGES 32
8.8.9 PRUNEVALIDPAGESBYCRC 33
8.8.10 READMETADATABUFFER 33
8.8.11 ACTIVATEMETADATAMAGICWORD 34
8.9 TxReduxData 34
8.10 RxDataPacket 34
8.11 RxReduxDataPacket 34
8.12 RXBcastInSniffedDataPacket 35
8.13 BcastUART2TrxBufferDone 36
8.14 RXBcastInSnifferAirDataPacket 36
8.15 RXBcastOutSnifferAirCmd 36
8.16 DynConfig 37
8.17 DynConfigReply 37
8.18 GetRegister 39
8.19 GetRegisterReply 39
8.20 SetRegister 39
8.21 SetRegisterReply 40
8.22 TransferConfig 40
8.23 TransferConfigReply 41
8.24 TXAirCmdWrapper 41
8.25 RXAirCommandWrapper 43

9 Configuration Registers **43**
9.1 Registers Description 46
9.1.1 addressBuf 46
9.1.2 addressBufLen 46
9.1.3 dyn 46
9.1.4 nwkID 46
9.1.5 hopTable 46
9.1.6 power 47

9.1.7 uart_bsel 47
9.1.8 nodeType 47
9.1.9 sleepMode 47
9.1.10 extSlpCtrlI2CAddress 47
9.1.11 extSlpCorrectionFactor 47
9.1.12 presetRF 47
9.1.13 cryptoData_qWord0 48
9.1.14 i2c 48
9.1.15 meshExecActiveFlag 48
9.1.16 sniffFlagsMask 48
9.1.17 enableNotificationFlagsMask 50
9.1.18 gpStorage_qWordx 50
9.1.19 index 50
9.1.20 valueRFLinks 50

10 Developer Kit 51

10.1 Portia Adapter Board 51
10.2 Arduino-Compatible Shield 51

11 Certification Information 51

List of Figures

1 Typical structure of periodic network broadcast cycles for the $DYN = 1, 2, 4, 1, 1, 5$ configuration . . . 11
2 Mesh Network Triggers . . . 12
3 External Triggers . . . 13
4 Module Through Hole Pinout . . . 15
5 Module USB Pinout . . . 17
6 Through Holes Mounting . . . 18
7 Module USB Pinout . . . 19

List of Tables

1 Specifications . . . 9
2 Virtual Machine Triggers . . . 13
3 Calculating the effective baud rate f_{BAUD} . . . 14
4 Standard baud rates list . . . 15
5 Module Pinout . . . 16
6 Module Pinout (continued) . . . 17
7 Additional I2C Functions . . . 20
8 Protocol-Formatted Common Header . . . 26
9 Packet Type Decoding . . . 27
10 Packet Type Decoding . . . 28
11 Packet Type Decoding (continued) . . . 29
12 EnterProtocolMode (special keyword command) . . . 29
13 DeviceReset (command) . . . 29
14 DeviceResetReply (reply) . . . 30
15 TxLongData (command) . . . 30
16 OTA FASTWRITE (command) . . . 31
17 OTA STARTTRANSFER (command) . . . 31
18 OTA STOPTRANSFER (command) . . . 31
19 OTA WRITE (command) . . . 32
20 OTA READ (command) . . . 32
21 OTA READFIRMWARECHECKOUTOK (command) . . . 32
22 OTA GETMISSINGFLAGS (command) . . . 33
23 OTA GETUNCKECKEDCRCPAGES (command) . . . 33
24 OTA PRUNEVALIDPAGESBYCRC (command) . . . 33
25 OTA READMETADATABUFFER (command) . . . 33
26 OTA ACTIVATEMETADATAMAGICWORD (command) . . . 34
27 TxReduxData (command) . . . 34
28 TxDataPacket (event) . . . 35
29 RxReduxDataPacket (event) . . . 35
30 RxBcastInSniffedDataPacket (event) . . . 35
31 BcastUART2TrxBufferDone (event) . . . 36
32 RXBcastInSnifferAirDataPacket (event) . . . 36
33 RXBcastOutSnifferAirCmd (event) . . . 37
34 DynConfig (command) . . . 38
35 DynConfigReply (reply) . . . 38
36 GetRegister (command) . . . 39

37 GetRegisterReply (reply) 39
38 SetRegister (command) 40
39 SetRegisterReply (reply) 40
40 TransferConfig (command) 41
41 TransferConfigReply (reply) 41
42 TXAirCmdWrapper (command wrapping another command for remoting) 42
43 RXAirCmdWrapper (event wrapping remote reply) 43
44 Register Table 44
45 Register Table (continued) 45
46 MeshExecActiveFlag Bit-Mask Register 49
47 Relevant Sniff Flags 49
48 Flags Enabling Special Notification UART Messages 50

1 Introduction

SMK900 transceivers provide for highly-reliable, long-range, and low power mesh networking radio applications. They use frequency hopping spread spectrum (FHSS) technology to ensure resistance to multipath fading and robustness, as well as for compliance with 900 MHz unlicensed band regulations in Canada and the US. The SMK900 supports a CTS-enabled serial port interface with data rates ranging from 1.2 to 230.4 kbps, with two possible modes of operation (transparent ASCII and protocol-formatted). For easy integration, error correction and buffering is all accomplished within the mesh controller module. A Virtual Machine is also available so that the user can leverage the module peripherals of the radio processor to perform operation such as signal processing and remote control devices. The module accepts multiple sleep synchronization clock sources: internal crystal, internal RC, or with an external I2C-line sleep controller for optimal sleep management. Key features include:

- Multipath fading resistance, with more than 51 frequency channels, 902.7 to 927.4 MHz
- Receiver protected by low-loss SAW filter for excellent receiver sensitivity and interference rejection
- Support for high-speed mesh networking applications
- Maximum range in free space exceeds 10 km (antenna height dependent)
- Typical range in forested areas between 250 and 500 m
- Transparent ASCII serial data mode for easiest integration available
- Advanced protocol-formated serial data mode available for maximum flexibility
- Accessibility to multiple analog and digital I/O via Virtual Machine
- Serial baud rates between 1.2 and 230.4 kbps
- AES 128 bits encryption available
- Module configuration stored in non-volatile memory
- Local and Over-the-air configuration for the radio
- Virtual Machine locally and remotely programmable at whim
- VM engine bytecode can be locally or remotely accessed/modified at whim via IDE

2 System

2.1 SMK900 System Overview

A SMK900 radio can be configured to operate in two main modes: gateway or node. A gateway controls a whole mesh network and functions as the main coordinator node, and its usual primary function is to bridge a host such as a PC, tablet, or internet gateway, with the rest of the mesh network. A node is a transceiver that acts as a repeater inside of the mesh network. The primary function of a node is to allow communication between an external devices and the gateway, or to serve as a bridge between analog and/or digital inputs/outputs such as for sensor arrays.

It is possible to configure any node so that it filters messages by MAC address, or configure it so that it acts like a sniffer relaying all messages transiting through the mesh network to the client circuitry via serial port.

2.2 Mesh Network Systems

The topology used by a SMK900 radio is that of a broadcast-only mesh network, with sleep-wake synchronization handled by the gateway. This mean that any SMK900 radio transmission sends a broadcast to the whole mesh network. There are no provisions for pure unicast messaging, and such needs are usually handled by integrator using a higher-level protocol sitting on top of the SMK900 protocol. The maximum of broadcast transmission of the same messages over the mesh network depend on the number of hops allowed.

Multiple independent mesh networks may coexist in the same physical space by configuring nodes with differing FHSS channels, different network IDs and/or different encryption patterns. Each of those mesh networks have to be controlled by its own gateway. The bridging between gateways is handled by the integrator.

2.3 Frequency Hopping Implications

The SMK900 uses the FHSS approach in order to ensure that co-located networks using different FHSS hopping tables (a.k.a channels) can coexist with graceful degradation of network performance as the number of conflicting networks overlapping increases. This however comes with the price that any node needs a certain time delay (called "seek time") in order to connect itself to its desired network (this delay can range between seconds to minutes, depending on the sleep interval between wake cycles for said network. For instance, at a sleep interval of 1 second, the seek time is of the order of 30 seconds to 1 minute, and this seek time increases linearly with said sleep interval.

3 Specifications

3.1 Broadcast Frame

In order to ensure synchronization of every node within a given network, the gateway always initiates a broadcast beacon periodically, which encompasses a broadcast parameters. This is the primary construct within which all nodes communicate. In particular, there is Time-Division Multiple Access (TDMA) slotting mechanism that specify unique slots, called broadcast phases, for outbound communication from a gateway to nodes, and inbound communication from a node back to a gateway.

3.1.1 Dynamic parameters

There are 6 parameters inside the broadcast beacon. They will be referred to in this integration guide by the moniker: dynamic parameters, or, in short, DYN. Configuration of those parameters usually take the form of a sequence of 6 bytes, with 1 byte per parameter, as follows:

$$DYN \leftarrow B_O, B_I, N_H, N_R, R, D \tag{1}$$

For each broadcast frame, the periodic delay between broadcast as well as its length being determined by:

B_O: number of gateway to node messages per broadcast (also called broadcast out phase

Table 1: Specifications

Absolute Maximum Rating	
Supply Voltage	-0.5 to 6.5V
All Input/Output Pins	-0.5 to 3.3V
Specification	Descriptions
Operating Temperature	-20 to 70C guaranteed for max hop count -40 to 85C guaranteed for half hop count
Storage Temperautre	-40 to 85C
Power Requirement	
Supply Voltage	3.3 to 5.5V
Transmit Current	130 mA peak
Receive Current	20 mA
Idle	30 uA peak
Transceiver	
Urban / Indoor / NLOS*	100 to 500m
Outdoor / LOS**	10km+
Transmit Power	Low: 50mW High 100mW
RF Data Rate	50 kbits
Number of Channels	5
Frequency Band	902 to 925 Mhz
Receiver Sensitivity	-110 dBm
Antenna Connector	U.Fl
Antenna Gain Maximum	3 dBi
Max Hop Count	31
Encryption OTA	AES 128 bits
Virtual Machine	
Memory	16 kBytes

* Not line of sight ** Line of sight

count),

B_I: number of node to gateway messages (also called broadcast in phase count),

N_H: the maximum number of hops for the network (i.e. maximum number of time a message can be relayed from node to node),

N_R: number of random-access specialized hop slots (called redux),

R: specialized slotting mechanism enabled

D: inverted sleep-wake duty cycle ratio D (min. value is 1),

The broadcast time can be then calculated as:

$$T_{BCAST}[msec] = 10(N_H(B_O + B_I) + N_R \cdot R) \tag{2}$$

The interval between each broadcast is:

$$T_{INTERVAL}[msec] = T_{BCAST} \cdot D \tag{3}$$

For the vast majority of applications, the default settings, where $B_O = B_I = 1$ and $R = 0$, are applicable, in which case the timing equations simplify to:

$$\begin{aligned} T_{BCAST}[msec] &= 20 \cdot N_H \\ T_{INTERVAL}[msec] &= T_{BCAST} \cdot D \end{aligned} \tag{4}$$

Allowable values for each of those are, with default values in **(boldface)**:

B_O: [(**1**), 2, 3, 4]

B_I: [(**1**), 2, 3, 4]

N_H: [1, 2, ... , 4, (**5**), 6, ..., 31]

N_R: [(**1**)]

R: [(**0**), 1]

D: [1, 2, 5, (**10**), 20, 40, 80]

Note that D is defined as $T_{INTERVAL}/T_{BCAST}$, and is thus the inverse of what is commonly defined as the network duty-cycle, which is defined as follows:

$$dutycycle[\%] = 100\%/D \tag{5}$$

Note that there are no parameters controlling any acknowledgement/retry cycles in case a message from one node to another fails to pass through. It is assumed that the onus of managing packet delivery failure occurrences fall upon the shoulder of the higher-level protocol as implemented by the integrator. The easiest way to do so, say for a mesh network with standard round-robin sensor polling, is simply to detect reply failure at the host side connected to the gateway, and retry polling for a number of times, before flagging a failure to user in the case a maximum number of retries has been hit. Note, however, that there is an internal CRC-based mechanism for controlling packet integrity, so any packet received can be assumed for the majority of applications as containing valid and uncorrupted information.

3.1.2 Broadcast Frame Structure

The following schematic shows the typical structure of periodic network broadcast cycles, for the following configuration $DYN = 1, 2, 4, 1, 1, 5$:

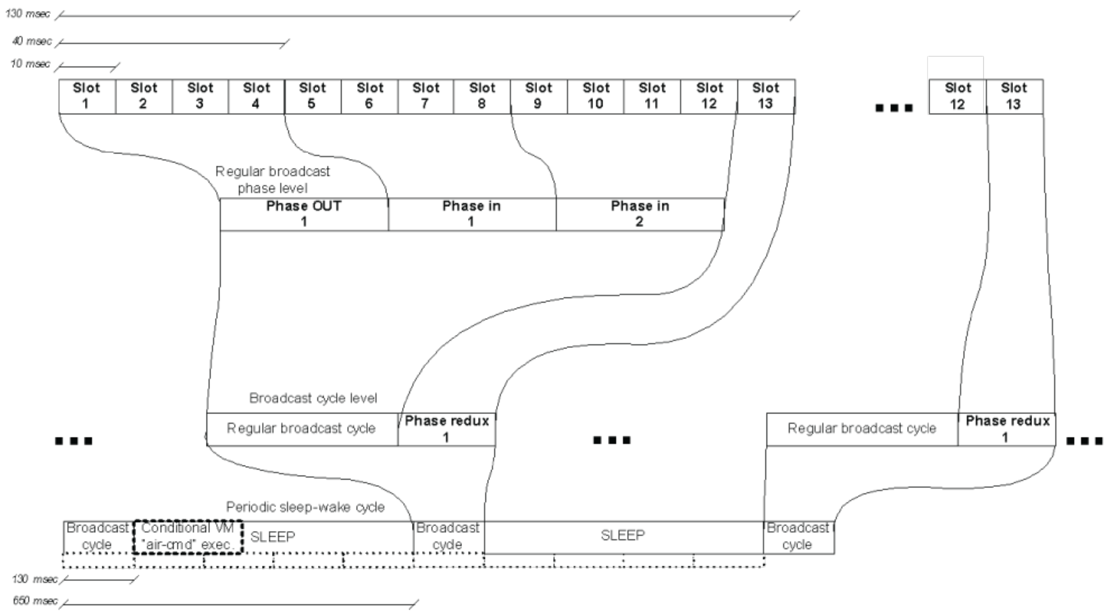

Figure 1: Typical structure of periodic network broadcast cycles for the $DYN = 1, 2, 4, 1, 1, 5$ configuration

3.2 SMK900 Addressing and Network Segregation

Each module has a unique factory-configured 3-byte address, called the MAC address. In the standard protocol-based serial data mode, this MAC address can be used to specify to which destination a message is intended, although this is not a necessary part of the protocol due to the fact that every message is treated as a broadcast to all nodes. The MAC address 0x000000 is treated as an invalid address. There is no broadcast address, in contrast with regular IEEE802.15.4 schemes (where a broadcast address is typically the highest possible address for a given number of bytes encoding said address). Note that in transparent serial data mode, MAC addresses are not used, and in this case the system behaves as a transparent point-to-multipoint system.

Every node also has a configurable network ID, called *NWKID*, between 0 and 7, which can be used to segregate multiple networks hopping on the same FHSS channel in order to reduce the impact of interference. Every node can also be encrypted using a unique network key, which not only secures a given mesh network, but also allows for more segregation between coexisting networks within the same physical space.

The primary means for network segregation is of course the selection of the FHSS channel via a configuration variable named *HOPTABLEID*. The current implementation allows for values between 0 and 5 (i.e. 6 different hop tables).

3.3 Transparent and Protocol-Formatted Mode

A SMK900 based network can be configured to use either the transparent or the protocol-formatted mode for the serial port interface.

3.3.1 Transparent

The transparent mode allows for basic, unsynchronized integration which emulate a simple point-to-multipoint serial link between a gateway and its nodes. In this case, the message is assumed to be of standard ASCII format, with the special ASCII terminator characters 0x13 (Carry) or 0x10 (Line Feed) are used as markers to trigger the end of a message stream, and thus to trigger transmission over radio waves.

3.3.2 Protocol-formatted

Protocol-formatted messages also referred to as Application Programing Interface (API) are discussed in the following sections. protocol-formatted messages usea start-of-message marker, followed by message length, message information type (or command byte), an optional string of MAC addresses, and finally, the payload.

4 Virtual Machine

4.1 Description

SMK900 radios feature an embedded Virtual Machine (VM) allowing Over-The-Air (OTA) firmware updates. This allows application-specific user scripting to control the internal modules of the SMK900 radio and ease interfacing with external devices. The main processor controls and manages wireless mesh operations and executes VM user scripts.

TheVirtual Machine is available when the serial protocol is configured to protocol-formatted mode (not to transparent mode). An IDE is available for Virtual Machine development, which includes basic compiler, disassembler, node configuration and VM upload/erase functions as well as direct node serial connection and OTA firmware node upload with a gateway.

4.2 Virtual Machine Triggers

Virtual Machine script execution is managed by the mesh network process. This is to ensure mesh operation priority over the Virtual Machine. VM execution is triggered by various events, which are defined in the table below.

The available triggers in the mesh network cycle are shown in the figures below.

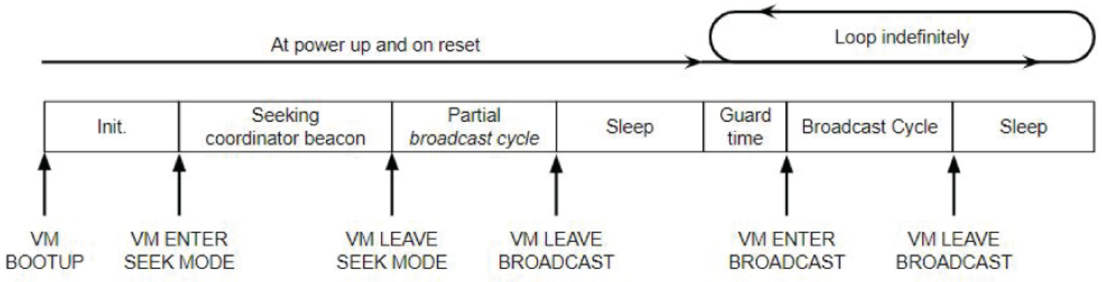

Figure 2: Mesh Network Triggers

Table 2: Virtual Machine Triggers

Trigger	Description
Bootup	Bootup system initialization
Enter Seek Mode	The transceiver is attempting communication with the gateway over the mesh network.
Leave Seek Mode	The transceiver has established a link to the mesh network.
Enter Broadcast	The transceiver wakes up to enter the broadcasting cycle
Leave Broadcast	Broadcasting cycle has ended and the transceiver is ready to enter sleep mode.
Serial Event	A message from a local serial device has been received by the transceiver. Note: The local serial device must read CTS low before sending.
Air Command	An execute air command message has been received. Typically, air command is used to control or read modules connected to the radio.

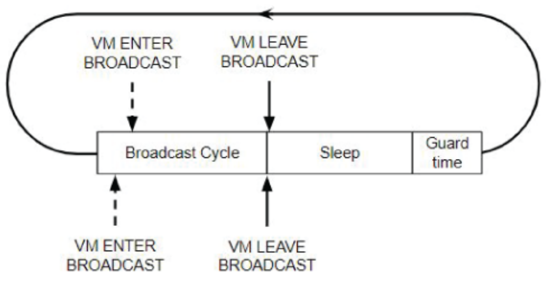

Figure 3: External Triggers

5 Hardware

The SMK900 module provides multiple application interfaces: a primary communication serial port (CTS enabled), a dedicated I2C port (Master mode only), and 13 generic digital I/O. The latter can be reconfigured to ADC (2x), to DAC (2x) or to PWM hardware signalling or clock generation (2x). The SMK900 transceiver can also use advanced peripherals such as hardware timers and event capture/compare within custom bytecode executed by its VM engine.

5.1 Serial Port

The host processor is tied to the SMK900 module over a full-duplex UART interface serial port with CTS pin hardware control. Baud rate is configurable from 1.2 to 230.4 kbps, with non standard baud rates also achievable (between the aforementioned boundaries). The serial port is configured for standard 8-bit data with no parity and 1 stop bit.

Baudrate configuration is done on the $uart_{bsel}$ register (register offset 0x06) This register can be split into: $BSEL = uart_{bsel}[0..11]$, and $BSCALE = uart_{bsel}[12...15]$. where $BSCALE$ is a 4-bit signed integer, ranging from -7 (0b1001) to +7 (0b0111). For positive values of $BSCALE$, the baud rate is prescaled by 2^{BSCALE}. or negative values the baud rate will use fractional counting, which increases resolution.

The formulae for calculating the effective baud rate f_{BAUD}:

Table 3: Calculating the effective baud rate f_{BAUD}

Conditions	Baud Rate (in baud or Hz)	BSEL Value
$BSCALE \geq 0$ $f_{BAUD} \leq 1MHz$	$f_{BAUD} = \frac{1 \cdot 10^6}{2^{BSCALE} \cdot (BSEL+1)}$	$BSEL = \frac{1 \cdot 10^6}{2^{BSCALE} \cdot f_{BAUD}} - 1$
$BSCALE < 0$ $f_{BAUD} \leq 1MHz$	$f_{BAUD} = \frac{1 \cdot 10^6}{(2^{BSCALE} \cdot BSEL)+1}$	$BSEL = \frac{1}{2^{BSCALE}} (\frac{1 \cdot 10^6}{f_{BAUD}} - 1)$

Thus, here is a list of standard baud rates and their corresponding suggested configuration values:

5.2 Module Pin Out

Electrical connections to the SMK900 are made through the I/O pads and through the I/O pins (depending on whether it is the SMT castellated or the through-hole version). The hardware I/O functions are detailed in Table 5 (note that the MCU alias is useful only when using advanced VM programming using accessible native MCU functionalities, and only the mutable generic I/O pins are available for such purposes, and are denoted by the principal name IO_x, where x is the generic pin number as used in VM programming).

The schematic block diagram for the through hole module pinout is shown at Figure 4, and the corresponding footpring at Figure 5. The surface mount pinout is shown at Figure **??**

Table 4: Standard baud rates list

Baud Rate (baud)	BSEL	BSCALE	$uart_{BSEL}$
2400	831 (0x33F)	-1 (0b1111)	0xF33F
4800	829 (0x33D)	-2 (0b1110)	0xE33D
9600	825 (0x339)	-3 (0b1101)	0xD339
19200	817 (0x331)	-4 (0b1100)	0xC331
38400	801 (0x321)	-5 (0b1011)	0xB321
57600	1047 (0x417)	-6 (0b1010)	0xA417
115200	983 (0x3S7)	-7 (0b1001)	0x93D7
125000	7 (0x007)	0 (0b0000)	0x0007
230400	428 (0x1AC)	-1 (0b1001)	0x91AC

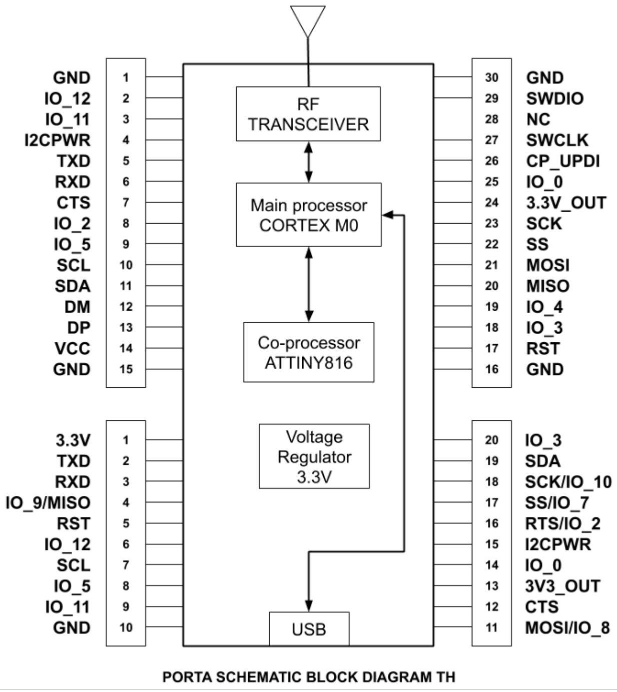

Figure 4: Module Through Hole Pinout

Table 5: Module Pinout

PIN TH	PIN XB	PIN SMD	NAME	ALIAS	I_O	DESCRIPTION
1	10	1, 13, 28	GND	-	-	Power supply and signal ground. Connect to the host ground.
2	6	9	IO_12	TX_LED	O(I)	Transmit LED pin. This pin activates only when a radio transmission is active
3	9	12	IO_11	BCAST_LED	O(I)	Broadcast LED pin. This pin activates/deactivates itself in order to mark the beginning and the end of a broadcast cycle.
4	15		I2C_PWR	-	O	I2C Power pin. Can be configured by changing the powerBusMode byte in the I2C configuration register. Allows to turn on/off an I2C peripheral connected to the module on demand when required (usually when a VM execution is running after a broadcast cycle).
5	2	5	TXD	-	O	Serial data output from the radio.
6	3	6	RXD	-	I	Serial data input to the radio.
7	12	15	/CTS	-	O	Host serial port CTS pin. When the line goes high, the host must stop sending data.
8	14		IO_2	RTS	I(O)	Generic I/O.
9	8	11	IO_5	DAC0	I(O)	Generic I/O. Alternately, hardware DAC channel 0.
10	7	10	I2C_SCL	-	O	I2C Master SCL clock pin. This pin should be pulled via resistor to a 3.3V high line (possibly the 3.3V_OUT pin).
11	19	22	I2C_SDA	-	I/O	I2C Master SDA data pin. This pin should be pulled via resistor to a 3.3V high line (possibly the 3.3V_OUT pin).
12	20	23	IO_1	-	I(O)	Generic I/O.
13			IO_6	DAC1	I(O)	Generic I/O. Alternately, hardware DAC channel 1.
14			VCC	-	I	Power supply input, +3.3 to +5.5 Vdc.
15	10	1, 13, 28	GND	-	-	Power supply and signal ground. Connect to the host ground.
16	10	1, 13, 28	GND	-	-	Power supply and signal ground. Connect to the host ground.
17	5	8	/RESET	-	I	Active low module hardware reset.
18			IO_3	ADC0	I(O)	Generic I/O. Alternately, hardware ADC channel 0.
19			IO_4	ADC1	I(O)	Generic I/O. Alternately, hardware ADC channel 1.
20	4	7	IO_9	MISO	I(O)	Generic I/O. Alternately, hardware SPI Master In Slave Out pin, or OC1A Timer C wave out Channel A.
21		14	IO_8	MOSI	I(O)	Generic I/O. Alternately, hardware SPI Master Out Slave In pin, or OC1B Timer wave out Channel B.

Table 6: Module Pinout (continued)

PIN TH	PIN XB	PIN SMD	NAME	ALIAS	I_O	DESCRIPTION
22	17	20	IO_7	/SS	I(O)	Generic I/O. Alternately, hardware SPI enable pin.
23	18		IO_10	SCK, MIRROR	I(O)	Generic I/O. Alternately, hardware SPI port clock, or MCU event mirror output pin.
24	1, 13	4, 16	3.3V	-	O	Stable low-power 3.3V output. Use with low-power devices only (< 10 mA average power consumption, < 20 mA peak consumption).
25		24	IO_0	ADC_EXT, REF		Generic I/O. Alternately, ADC external reference voltage pin. The voltage at this pin can be used by the ADCs as a reference for ratiometric measurements.
26			RSVD	-		
27			RSVd	-		Reserved pin. Leave unconnected.
28	10	1, 13, 28	GND	-		Connect the host circuit board ground plane.
29			RSVD	-		Reserved pin. Leave unconnected.
30	10	1, 13, 28	GND	-		Connect the host circuit board ground plane.
		2	DM	-	I/O	USB negative wire (white)
		3	DP	-	I/O	USB positive wire (green)

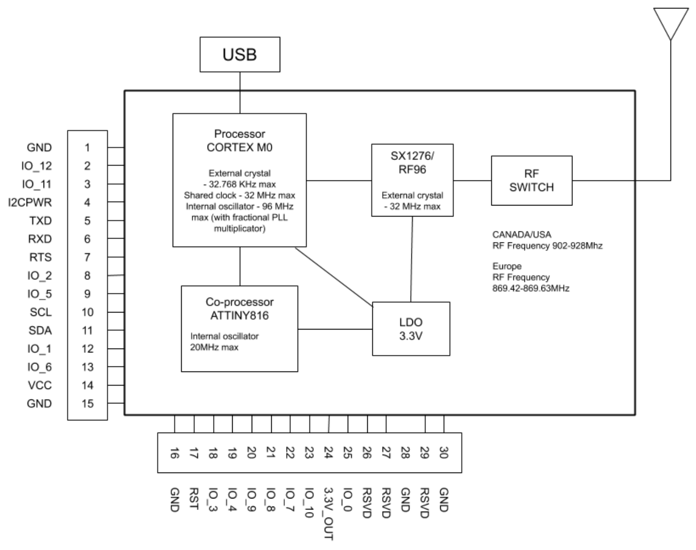

Figure 5: Module USB Pinout

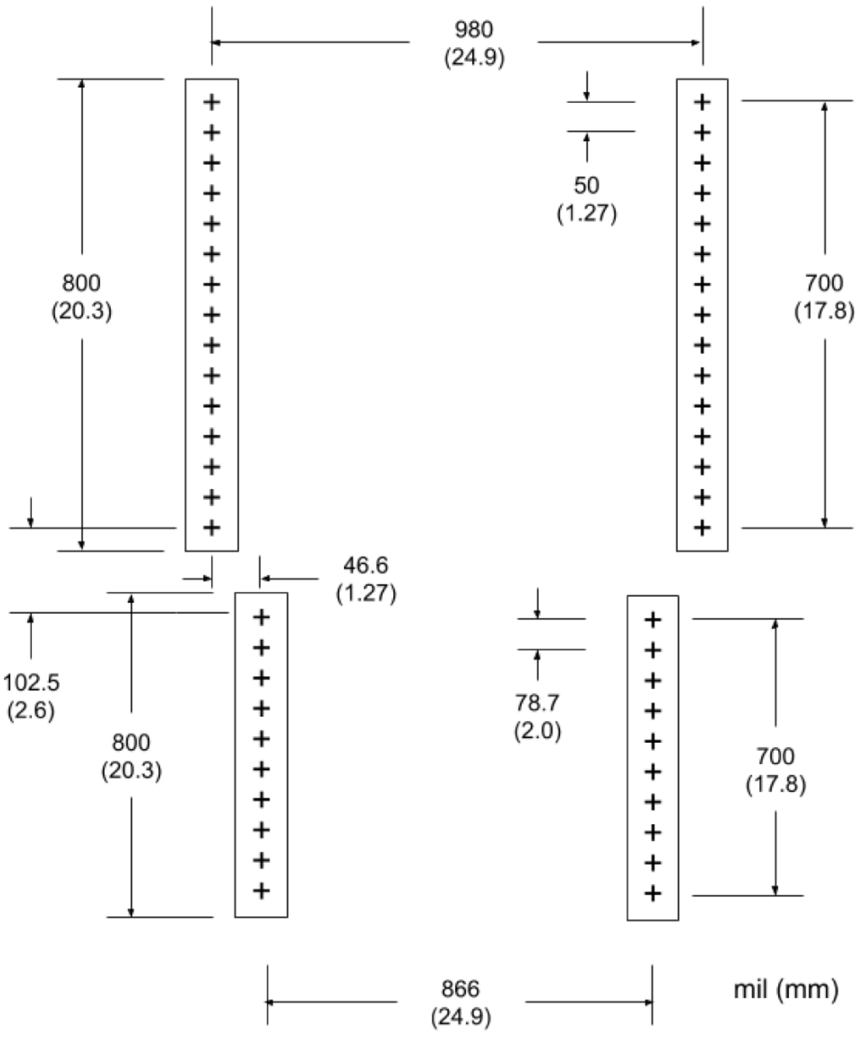

Figure 6: Through Holes Mounting

5.3 Power Supply and Input Voltage

SMK900 radio modules can operate from an unregulated DC input in the range of 3.3 to 5.5 V with a maximum ripple of 5% over the temperature range of -40 to 85 °C. Applying AC, reverse DC, or a DC voltage outside the range given above can cause damage and/or create a fire and safety hazard. Further, care must be taken so logic inputs applied to the radio stay within the voltage range of 0 to 3.3 V. Signals applied to the analog inputs must be in the range of 0 to ADC_EXT_REF (Pad/Pin 25) if the reference is used as such, else the range of 0 to VCC shall be used. Applying a voltage to a logic or analog input outside of its operating range can damage the SMK900 module.

5.4 ESD and Transient Protection

The SMK900 circuit boards are electrostatic discharge (ESD) sensitive. ESD precautions must be observed when handling and installing these components. Installations must be protected from electrical transients on the power supply and I/O lines. This is especially important in outdoor installations, and/or where connections are made to sensors with long leads. Inadequate transient protection can result in damage and/or create a fire and safety hazard.

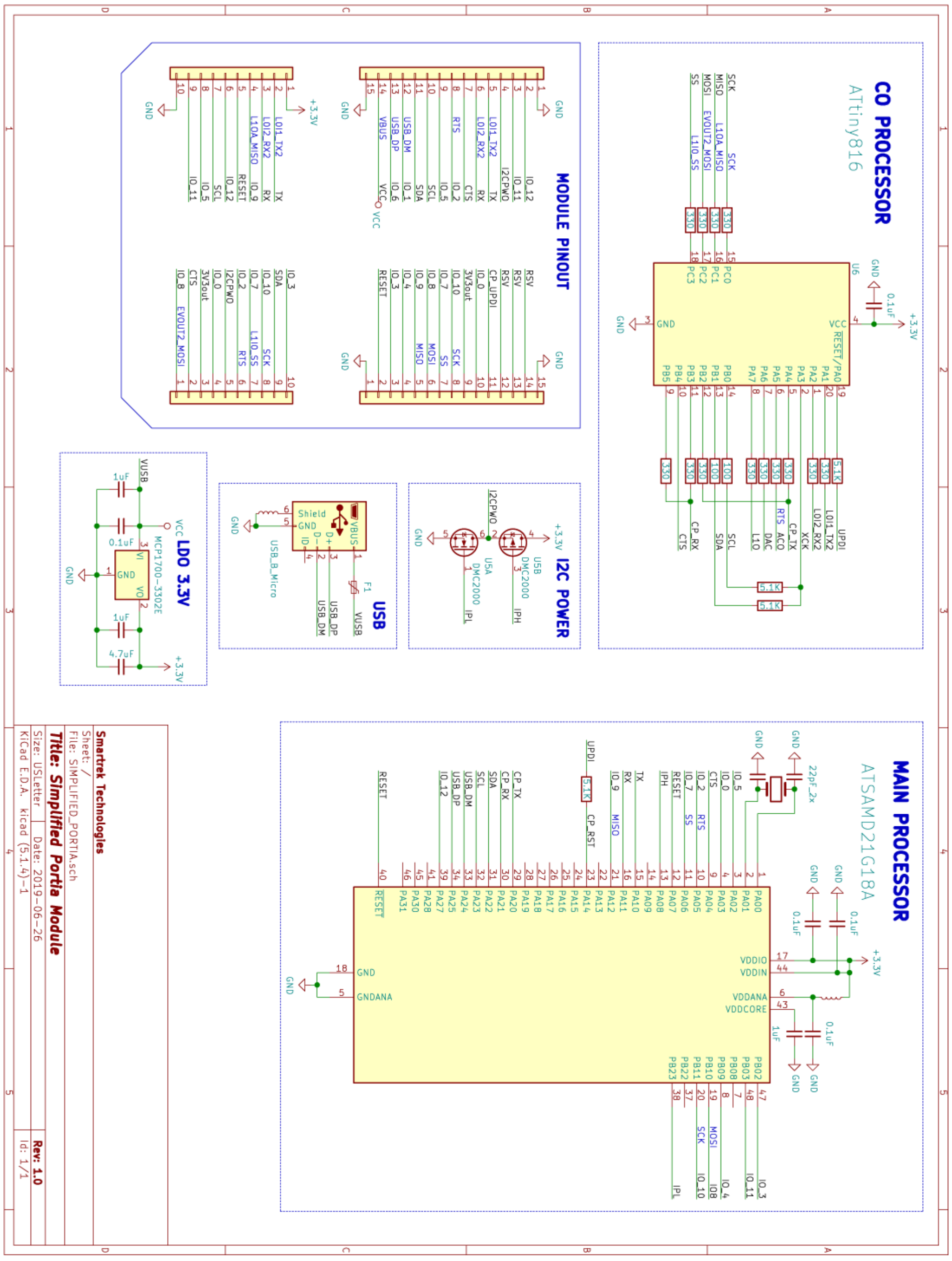
Figure 7: Module USB Pinout

In the case where low power consumption is desired, dedicated logic level converters, or equivalent FET circuitry can be used to achieve such specifications.

5.5 Antenna Connector

The antenna connector is a U.Fl type male connector which can either be mated to a PCB host board or directly to an antenna using the appropriate adapter. Impedance of all components from the connector up to the antenna part has to be 50 Ohms.

6 Additional I2C Functions

Table 7 lists additional I2C commands that are available for execution from a custom user VM script(see Section 4 for more details):

Table 7: Additional I2C Functions

Name	Type	Description	Master command byte stream	Expected slave answer byte stream
Read configuration	Read	Read in the following order: voltage (1 byte), RF channel (1 byte), I2C address (1 byte), expected reference voltage (2 bytes, Little-Endian byte ordering). Voltage is the input VCC voltage of the external sleep controller, using the internal chip FVR voltage reference. Precision expected of this voltage measurement is ±0.15 V, and is thus usually sufficient to evaluate battery pack status if standard alkaline batteries are used. The raw voltage value sent VRAW is in increments of 0.05V, and is an unsigned 8-bit integer. In other terms, $V[Volt] = VRAW \cdot 0.05$. The expected reference voltage REFVOLT is the actual reference voltage of the internal FVR used for ADC measurements, which defaults to 2048 mV (value stored in 16-bit as a mV value). In the case where there is a discrepancy in the voltage assessment of VCC by the external sleep controller V_{ext} and a calibrated measurement in laboratory V_{lab}, then the REFVOLT should be corrected in the following manner: $REFVOLT \leq REFVOLT \cdot V_{lab}/V_{ext}$	[(ADDR x 2)+1]	[VOLTAGE, RF channel, ADDR, REF voltage LS byte, REF voltage MS byte]
Change I2C address	Write	Write a new I2C address to the external sleep controller. Note that changes are effective without having to reboot the external sleep controller chip, less than 100 msec after the I2C command is sent on the I2C bus.	[ADDR x 2, 0x28, new I2C ADDR]	N/A
Change reference voltage	Write	Change the reference voltage used for calibrating the voltage measurements. Change is valid and enforced < 100 msec. after end of I2C command.	[ADDR x 2, 0x11, new REF voltage LS byte, new REF voltage MS byte]	N/A
Change RF channel	Write	Change the generic RF channel value via I2C. Change is valid and enforced < 100 msec. after end of I2C command.	[ADDR x 2, 0x18, new RF channel value]	N/A

Note that all byte streams are notated as a sequence of bytes in brackets [], and that ADDR is the current I2C address of the external sleep controller chip when the command is issued from the I2C master side.

7 Example VM User Script Using I2C Functions

The following is a VM example for a simple node that (1) at boot-up, queries the external sleep controller for the RF channel variable, and configures the transceiver NwkId and HopTable configuration registers accordingly (see section 7.3 for details about configuration registers); (2) when receiving an air command from a gateway, it queries the external sleep controller voltage, and sends this back as the gateway answer, in conjunction with the last known received packet RSSI value.

VM example!

```
#include "SMK900.evi"
//Note that SMK900.evi include code listing is appended as an annex of this document.
//This include defines transceiver-specific constants as well as available special-purpose
//transceiver functions. See section 8.2.1 for details.

#define RFCHANNEL_MAX_RETRIES_COUNT_STD (20)
#define SLEEPCTRL_I2C_ADDR_AND_0X7F_SHIFTL_1_OR_I2CMASTER_READMODE_gc (0x91)

function exec_bootup(){
   local rfChannel;
   local retryCount;

   local nwkid;
   local hoptableid;
   local i;
   retryCount=0;

   //Hardcode a delay of 100usec * 1000 * 60 = 6 seconds
   for(i=0;i<60;i++){
      Delay(1000);
   }

   while(retryCount<RFCHANNEL_MAX_RETRIES_COUNT_STD){
      if(I2C_Start( SLEEPCTRL_I2C_ADDR_AND_0X7F_SHIFTL_1_OR_I2CMASTER_READMODE_gc ) ){
         //Sleep controller answered NAK
         I2C_Stop();
         retryCount++;
         Delay(5000);//delay 500 msec before retrying
      }
      else{
         //Sleep controller answered ACK
         I2C_ReadAck(); //1st byte is voltage. We discard
         rfChannel = I2C_ReadNak(); //2nd byte is RF channel
         I2C_Stop();

         //Define NwkId and HopTable register values according to read RF channel value
         //stored in external sleep controller
         nwkid = rfChannel \% NWK_COUNT;
         hoptableid = HOPTABLE_50K_START + (rfChannel \% HOPTABLE_50K_COUNT);

         //REGISTER_xxx are VM-accessible mesh controller configuration parameters
         GetRegisterRAMBUF(8, REGISTER_NWKID);
```

```
                GetRegisterRAMBUF(9, REGISTER_HOPTABLE);
                if( (nwkid!=GetBuffer_U8(8)) || (hoptableid!=
                    GetBuffer_U8(9)) ){
                    SetBuffer(8,nwkid,1);
                    SetBuffer(9,hoptableid,1);
                    SetRegisterRAMBUF(8, REGISTER_NWKID);
                    SetRegisterRAMBUF(9, REGISTER_HOPTABLE);

                    TransferConfigEEPROM();
                    //force reset by letting watchdog expire (the mesh
                         controller
                    //has a safety watchdog that runs at all times,
                        and set by default at 8 sec)
                    while(1){}
                }

                break;
            }
        }
    }
```

VM example! (continued)

```
//Custom VM function executed when end node mesh controller
   receives an ”air cmd”, which is
//a command from coordinator node to execute a VM function
   over the air
function exec_aircmd(){
   local V;

   //time to read external sleep controller voltage using I2C
      bus
   if(I2C_Start(
      SLEEPCTRL_I2C_ADDR_AND_0X7F_SHIFTL_1_OR_I2CMASTER_READMODE_gc
       ) ){ //failure
      V=0;
   }
   else{
      V = I2C_ReadNak();
   }
   I2C_Stop();

   SetBuffer(0,GetRSSI(),1); //set first answer byte as the
      RSSI of the last known good packet received
   SetBuffer(1,V,1); //set 2nd answer byte as voltage fetched
      via I2C bus

   Send(2);
}
```

VM example! (continued)

```
//note: the content of airBuf will be : the air-wrapped
    command byte (for VM execution this will be either 0E or 8E
    , the former if no addr is sent back,
//the latter if addr has to be sent back.
//Then, the actual cmd byte(s), in this case we expect 1 byte,
     with ASCII char 0x41
//Total: 2 bytes

//ALIAS NAME............|.fn#.|ALIAS#|argc|.......argv
    ........|   Description
//GetAirBuf(r, i, len)..|.0...|0x74..|.4..|aliasID, r, i, len
      |  Copy
//mid(airbuf, i, len) into buffer #r: returns number of valid
    bytes in airbuf
//GetBuffer_16(r).......|.0...|0x22..|.2..|....aliasID, r
    .....|   Get buffer
//#r, 2 first bytes, as int16/uint16, and return it verbatim (
    as int16)

rxLen=GetAirBuf(0, 0, 2);
if(rxLen>=3){
   expectedCmd=(GetBuffer_16(1) & 0xFF);//expected state is in
        the 3rd byte!

   //Do something with received 'A' cmd
}
```

Listing 1: Receiving 'A' command

VM example! (continued)

```
//Common entry point for all VM executions: the subfunction to
    b executed is selected according to trigger
//type via GetExecType().
function main()
{
   local execType;

   execType = GetExecType();
   if(execType==MESHEXECTYPE_BOOTUP_bm){
      exec_bootup();
   }
   else if(execType==MESHEXECTYPE_AIRCMD_bm){
      exec_aircmd();
   }
}
```

8 Protocol-Formatted Messages

8.1 Protocol Formats

SMK900 module can work in one of two serial data modes - transparent or protocol. Transparent mode requires no data formatting, but cannot leverage the embedded node addressing schemes, nor can access configuration and VM upload/erase/check/execute functions. Thus, transparent mode is adapted mainly for simple drop-in serial wire replacement for point-to-multipoint applications.

Thus, a gateway that needs to send messages to a specific node, or a node replying to said gateway, must use protocol formatting if any advanced functionality other than simple ASCII message broadcast is needed, such as access to sensor I/O commands, configuration commands and replies, event monitoring, etc. All protocol-formatted messages have a common header as shown in Table 8

Table 8: Protocol-Formatted Common Header

0	1	2	3	4
SOP	Length (LSByte)	Length (MSByte)	PktType	Variable number of args...

The scale above is in bytes.

The Start-of-Packet (SOP) character, 0xFB, is used to mark the beginning of a protocol-formatted message and to assure synchronization in the event of a glitch on the serial port at startup.

This is followed by two length bytes, in Little-Endian ordering (lowest-significant byte first). This 16-bit value corresponds to the length of the remainder of the message following the length bytes themselves, i.e. the length of the entire message - 3.

The Packet Type (PktType) byte specifies the type of message. It is a bitfield-oriented specifier, decoded as shown in Table 10.

Table 9: Packet Type Decoding

Bit(s)	Meaning
7	Address send back request bit
6	Reserved for future use
5	Event - this bit is set to indicate an event message
4	Reply - this bit is set to indicate a message is a reply
3..0	Type - these bits indicate the message type

8.2 Message

Messages generated on the serial interface by the user are referred to as host messages, and have a PktType reply bit (4) cleared. Messages generated on the serial interface by the radio are referred to as reply or event messages, and have either bit 4 (replies) or bit 5 (event) of the PktType byte set. For most commands, there is a corresponding reply message, which is either an acknowledgement message.

Errors are usually flagged using event messages. Messages received by the radio and relayed back to user, such as node reply messages, are flagged as event messages as well. Note that for all quantities encoded using multi-byte, the byte ordering is Little-Endian, except for text strings. Little-Endian byte order places the lowest order byte in the left-most byte of the argument and the highest order byte in the right-most byte of the argument.

A command sent from the host to a module locally via serial port in order to initiate an action locally, or to read/write to the module locally, is the default type of message. However, there is also another type of message, which are called air command wrapped messages. Those are typically akin to local destination messages, but wrapped around a meta-message, which, combined with a destination MAC address, allows to execute said command at a remote transceiver node location with said MAC address as if that command was locally executed at this remote node location. Thus, it is possible to use the same command set (encapsulated within those meta-messages, called air command wrappers) to change the configuration and act on remote nodes in a straightforward manner. It is also possible to send air command wrapped messages to multiple MAC addresses (up to 4), if multi-phase mode is used (only possible when the number of broadcast in phases $B_I > 1$).

8.3 Message Format Details

8.3.1 Summary of message types

8.4 EnterProtocolMode

This command is used to enter protocol formatted mode from transparent serial mode via a special keyword (transceiver will reply with an EnterProtocolModeReply message).

8.5 DeviceReset

This resets the transceiver module (for local UART commands only, the transceiver will reply with a DeviceResetReply message). This command can be wrapped as an air command.

Table 10: Packet Type Decoding

Command	Reply	Event	Type	Direction	AirCmd Wrapped no MAC	AirCmd Wrapped w/ MAC	Gateway cmd	Node cmd
0x00	-	-	EnterProtocolMode	from Host	-	-	X	X
-	0x10	-	EnterProtocolModeReply	from Radio	-	-	X	X
0x01	-	-	ExitProtocolMode	from Host	-	-	X	X
0x02	-	-	DeviceReset	from Host	0x02	0x82	X	X
-	0x12	-	DeviceResetReply	from Radio	-	-	X	X
0x05	-	-	TXLongData	from Host	-	-	X	X
0x06	0x06	OTA	from Host	0x06	0x86	X	X	
0x07	-	-	TXReduxData	from Host	-	-		X
-	-	0x26	RXDataPacket	from Radio	-	-	X	X
-	-	0x28	RXReduxDataPacket	from Radio	-	-	X	
-	-	0x29	RXBcastInSniffedDataPacket	from Radio	-	-		X
-	-	0x2A	BcastUART2TrxBufferDone	from Radio	-	-	X	X
-	-	0x2B	RXBcastInSnifferAirDataPacket	from Radio	-	-		X
-	-	0x2C	RXBcastOutSnifferAirCmd	from Radio	-	-		X
0x0A	-	-	DynConfig	from Host	-	-	X	
-	0x1A	-	DynConfigReply	from Radio	-	-	X	
0x03	-	-	GetRegister	from Host	0x03	0x83	X	X
-	0x13	-	GetRegisterReply	from Radio	0x13	0x93	X	X
0x04	-	-	SetRegister	from Host	0x04	0x84	X	X
-	0x14	-	SetRegisterReply	from Radio	0x14	0x94	X	X
0x0B	-	-	TransferConfig	from Host	0x0B	0x8B	X	X
-	0x1B	-	TransferConfigReply	from Radio	0x1B	0x9B	X	X

Table 11: Packet Type Decoding (continued)

Command	Reply	Event	Type	Direction	tiny AirCmd Wrapped no MAC	tiny AirCmd Wrapped w/ MAC	Gateway cmd	Node cmd
0x0C	-	-	TXAirCmd-Wrapper	from Host	-	-	X	
-	-	0x2D	RXAirCmd-Wrapper	from Radio	-	-	X	
0x0D	-	-	VMFlash	from Host	0x0D	0x8D		X
-	0x1D	-	VM-FlashReply	from Radio	0x1D	0x9D		X
0x0E	-	-	VMExe-cute	from Host	0x0E	0x8E	X	X
-	0x1E	-	VMExe-cuteReply	from Radio	0x1E	0x9E	X	X
-	-	0x27	An-nounce/Er-ror	from Radio	0x27	0xA7	X	X

Table 12: EnterProtocolMode (special keyword command)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	0x08 0x00 = Number of bytes in message following this byte (Little-Endian)
0x03 - 0x10	Keyword	0x44 0x4E 0x54 0x43 0x46 0x47 0x00 0x00 = Keyword string

Table 13: DeviceReset (command)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	0x02 0x00 = Number of bytes in message following this byte (Little-Endian)
0x03	Packet Type	0x02 = DeviceReset
0x04	Reset Type	0x00 = Normal reset

8.6 DeviceResetReply

This is the reply message (only for local UART commands) after the transceiver receives a DeviceReset command. An equivalent air command reply wrapped accordingly can be sent out from a node back to its gateway if the latter sent an air command with DeviceReset as the wrapped command.

Table 14: DeviceResetReply (reply)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	0x01 0x00 = Number of bytes in message following this byte (Little-Endian)
0x03	Packet Type	0x12 = DeviceResetReply

8.7 TxLongData

Basic data send in command mode, that can span multiple consecutive broadcast phases. Note that no reply is tied to this command. In the case where the transmitter is a gateway, then all nodes directly or indirectly connected to that gateway via the mesh network that properly receives the broadcasted packet will send a corresponding RxDataPacket reply to their own hosts. In the case where the transmitter is a node, then only the gateway will send a corresponding RxDataPacket reply at reception of packet. If another node needs to monitor that signal, then the proper sniffer configuration register flags (RAM bank register sniffFlagsMask, flag BROADCASTIN_bm) must be written (see section 7.3), in which case the corresponding sniffed receive replies to host will be in the form of RXBcastInSniffedDataPacket.

Table 15: TxLongData (command)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	0x?? 0x?? = Number of bytes in message following this byte (Little-Endian)
0x03	Packet Type	0x05 = TxLongData
0x04	Start Broadcast Phase	0x00-0x03 = Start Broadcast Phase (range is in practice limited to B-1 where B is the number of Broadcast Out Phases if sender is a gateway or the number of Broadcast In Phases if sender is a node
0x05 - ...	Payload	Up to 20 x B bytes of data at 50kbit/sec 72 x B bytes of data at 100kbit/sec where B is the number of Broadcast Out Phases if sender is a gateway or the number of Broadcast In Phases if sender is a node

8.8 OTA

OTA commands allow for remote firmware updates. All OTA commands include a subcommand as the first byte of the payload. The subcommand also contains some metadata. The bit at `0x80` indicates whether the OTA command contains firmware (`0`) or VM code (`1`). The bit at `0x40` has a special meaning and indicates a fast write operation. In fast write mode, the command acts as a write command where the sequence id MSB is stored.

For instance, a very basic firmware update could go through the following steps:

1. Sending a `STARTTRANSFER` command to every node to be reprogrammed;
2. Use broadcasted `FASTWRITE` commands to upload the firmware;
3. Ensure no dropped pages with `GETMISSINGFLAGS` commands;
4. Perfom CRC checks with broadcast `PRUNEVALIDPAGESBYCRC` commands;
5. Ensure all CRC pages have been checked with `GETUNCHECKEDCRCPAGES` commands;
6. Write the metadata buffer with `WRITEMETADATABUFFER` commands;
7. Enable firmware upload to the desired target with `ACTIVATEMETADATAMAGICWORD` commands;
8. The nodes will perform the required update operation on reset.

8.8.1 FASTWRITE

Special command to write data starting at the seqence Id 0x(MSB)(LSB). Note that the subcommand value is used to hold the MSB of the sequence id.

Table 16: OTA FASTWRITE (command)

Subcommand	Command payload	Response payload
Sequence id msb	`[0x4(MSB), (LSB), Data]`	-

8.8.2 STARTTRANSFER

Enables an OTA transfer containing `pageCount` pages.

Table 17: OTA STARTTRANSFER (command)

Subcommand	Command payload	Response payload
0x00	`[0x00, pageCount:u16]`	`[0x00]`

8.8.3 STOPTRANSFER

Stops the device from accepting OTA commands.

Table 18: OTA STOPTRANSFER (command)

Subcommand	Command payload	Response payload
0x02	`[0x02]`	`[0x02]`

8.8.4 WRITE

Writes 16bytes long pages to the flash starting with page `startPage`. All pages that got written are returned in succession by this command. In most cases, the broadcast version of this command is more useful in order to update multiple devices at the same time.

A variant command, `WRITEMETADATABUFFER` performs the same write operation but on the metadata buffer, which contains firmware size, encryption salt and SHA256 hashes.

The normal write command uses the subcommand id `0x01`, while the `WRITEMETADATABUFFER` uses the subcommand id `0x0D`.

Table 19: OTA WRITE (command)

Subcommand	Command payload	Response payload
0x01	`[0x01, startPage:u16, data:[u8]]`	`[0x01, writtenPages:[u16]]`
0x0D	`[0x0D, startPage:u16, data:[u8]]`	`[0x0D, writtenPages:[u16]]`

8.8.5 READ

Reads `length` bytes from the flash starting at byte `byteOffset`.

Table 20: OTA READ (command)

Subcommand	Command payload	Response payload
0x03	`[0x03, length: u8, byteOffset:u24]`	`[0x03, data:[u8]]`

8.8.6 READFIRMWARECHECKOUTOK

Validates that the firmware in flash has been validated. This means that any previous OTA write operation has been successful and has been verified. This command only returns a single flag. Using the `GETUNCKECKEDCRCPAGES` command is better in this context since this command returns the location of the invalid pages.

Table 21: OTA READFIRMWARECHECKOUTOK (command)

Subcommand	Command payload	Response payload
0x10	`[0x10]`	`[0x10, firmwareOk:bool]`

8.8.7 GETMISSINGFLAGS

This command returns the missing scratch page byte indexes. A scratch page is a bit field representing the status of the `WRITE` operations. A single bit of scratch represents a single page of the write operation (16 bytes). When a write operation is performed, the corresponding scratch page bit is set. Each scratch page is a word of two bytes, representing a total page length of 256 flash bytes. When there are scratch pages that are not `0xff`, this means that there was an unsuccessful `WRITE` or `PLUNEVALIDPAGESBYCRC` operation on this page.

This command has two variations. The `STATICSIZE`, with id `0x06` returns a single non completed scratch word if any. The `VARIABLESIZE`, with id `0x07`, will fill the packet with as much non completed scratch word as it can.

8.8.8 GETUNCHECKEDCRCPAGES

This command returns the CRC pages that were not validated during a `PRUNEVALIDPAGESBYCRC` command. A CRC page corresponds to 256 bytes of firmware. That is 16 `WRITE` pages, or a single

Table 22: OTA GETMISSINGFLAGS (command)

Subcommand	Command payload	Response payload
0x06	`[0x06]`	`[0x06, scratchByteId: u16, scratchContent: u16]`
0x07	`[0x07]`	`[0x07, [scratchByteId: u16, scratchContent: u16]]`

scratch word. The command returns CRC pages id.

This command has two variations. The `STATICSIZE`, with id `0x12` returns a single non completed CRC check if any. The `VARIABLESIZE`, with id `0x13`, will fill the packet with as much non completed CRC checks as it can.

Table 23: OTA GETUNCKECKEDCRCPAGES (command)

Subcommand	Command payload	Response payload
0x12	`[0x12]`	`[0x12, crcId: u16]`
0x13	`[0x13]`	`[0x13, crcIds:[u16]]`

8.8.9 PRUNEVALIDPAGESBYCRC

This command performs CRC checks and marks the written pages as invalid if the check fails. This command is most useful in broadcast, as all nodes will be able to perform the checks at the same time. The `GETUNCHECKEDCRCPAGES` command can then be used to validate that every CRC check was actually performed.

This command contains the first CRC page to check, then a series of consecutive page CRC32. The command returns a operation status byte for each performed check. A CRC check failure will return a value of `0`, while a success will return a value of `1`. Other errors such as incomplete or non existant pages will return respectively a value of `2` and `3`.

Table 24: OTA PRUNEVALIDPAGESBYCRC (command)

Subcommand	Command payload	Response payload
0x09	`[0x09, firstPageId: u16, crc32Values:[u32]]`	`[0x12, crcId: u16, crcStatus: [u8]]`

8.8.10 READMETADATABUFFER

This command is used to read bytes from the metadata buffer.

Table 25: OTA READMETADATABUFFER (command)

Subcommand	Command payload	Response payload
0x0F	`[0x0F, offset: u16, length: u8]`	`[0x0F, offset: u16, length: u8, data: [u8]]`

8.8.11 ACTIVATEMETADATAMAGICWORD

This command enables special actions on reset regarding the firmware. The SMK900 firmware update magic word is `0x42424242`, this means that after writing this command, the firmware on this node will be updated on node reset from the flash content.

For this command to succeed, all CRC checks must have been done.

Table 26: OTA ACTIVATEMETADATAMAGICWORD (command)

Subcommand	Command payload	Response payload
0x0E	`[0x0E, magicWord: u32]`	`[0x0E, magicWord: u32, success: bool]`

8.9 TxReduxData

Special data send using the broadcast cycle redux phase. This command is only available for nodes (redux phase utilization is forbidden by gateway), and only if the dynamic configuration of the network has the redux phase enabled (see section 2.2 for more details). Note that no reply is tied to this command.

Table 27: TxReduxData (command)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	0x?? 0x?? = Number of bytes in message following this byte (Little-Endian)
0x03	Packet Type	0x07 = TxReduxData
0x04 - ...	Payload	Up to 20 bytes of data at 50kbit/sec 72 bytes of data at 100kbit/sec

8.10 RxDataPacket

Event message from transceiver to its host when it received a regular packet sent via TxLongData. Note that this packet type is also used as a special Broadcast End Marker, in which case the packet byte stream is hard-coded to the following: [0xFB 0x03 0x00 0x26 0xFF 0x00], a marker that is sent at the end of every broadcast cycle, before any air command is processed (and only if configuration register in RAM bank enableNotificationFlagsMask, flag CLOSEBROADCAST_bm, is set; see section 7.3 for details).

8.11 RxReduxDataPacket

Event message from transceiver to host when it receives a packet in the broadcast cycle redux phase. This is only available if the current dynamic configuration of the network allows a redux phase.

Table 28: TxDataPacket (event)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	0x?? 0x?? = Number of bytes in message following this byte (Little-Endian)
0x03	Packet Type	0x26 = RxDataPacket
0x04	Broadcast Phase ID	0x00-0x03 = Received packet Broadcast Phase ID; 0xFF = Broadcast End Marker (special case)
0x05	RSSI	0x00-0xFF = Received packet strength (for regular received packet); this value is forced to 0x00 if Broadcast Phase ID is 0xFF (Broadcast End Marker)
0x06 - ...	Payload	Up to 20 bytes of data at 50kbit/sec 72 bytes of data at 100kbit/sec

Table 29: RxReduxDataPacket (event)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	0x?? 0x?? = Number of bytes in message following this byte (Little-Endian)
0x03	Packet Type	0x28 = RxReduxDataPacket
0x04	RSSI	0x00-0xFF = Received packet strength
0x05 - ...	Payload	Up to 20 bytes of data at 50kbit/sec 72 bytes of data at 100kbit/sec

8.12 RXBcastInSniffedDataPacket

Event message from a node (exclusively) transceiver to host, when it receives a sniffed packet transmitted from another node towards the network gateway via command TxLongData. This message is only sent to transceiver host if the proper transceiver configuration register flag is set (RAM bank register sniffFlagsMask, flag BROADCASTIN_bm, see section 7.3).

Table 30: RxBcastInSniffedDataPacket (event)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	0x?? 0x?? = Number of bytes in message following this byte (Little-Endian)
0x03	Packet Type	0x29 = RXBcastInSniffedDataPacket
0x04	Broadcast Phase ID	0x00-0x03 = Received packet Broadcast Phase ID
0x05	RSSI	0x00-0xFF = Received packet strength
0x06 - ...	Payload	Up to 20 bytes of data at 50kbit/sec 72 bytes of data at 100kbit/sec

8.13 BcastUART2TrxBufferDone

Event message sent from transceiver to host after the UART transmit buffer is cleared and its corresponding data is transferred in RF packet buffers, ready to be transmitted out. This marker is always sent at the beginning of a broadcast, even if the UART transmit buffer is clear (only if configuration register in RAM bank enableNotificationFlagsMask, flag ENDUARTTRANSFERTOTXBUFFER_bm, is set; see section 7.3 for details).

Table 31: BcastUART2TrxBufferDone (event)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	0x01 0x00 = Number of bytes in message following this byte (Little-Endian)
0x03	Packet Type	0x2A = BcastUART2TrxBufferDone

8.14 RXBcastInSnifferAirDataPacket

Event message from a node (exclusively) transceiver to host, when it receives a sniffed packet transmitted from another node towards the network gateway which contains an air command reply of any kind (i.e. an air command reply sent back by a node after having received an air command sent by a gateway, the latter having sent it via command TXAirCmdWrapper). This message is only sent to transceiver host if the proper transceiver configuration register flag is set (RAM bank register sniffFlagsMask, flag BROADCASTIN_bm, see section 7.3). Also note that the entirety of the raw wrapped air command answer, including the sender node address bytes (if applicable), can be found within the payload of this message.

Table 32: RXBcastInSnifferAirDataPacket (event)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	0x?? 0x?? = Number of bytes in message following this byte (Little-Endian)
0x03	Packet Type	0x2B = RXBcastInSnifferAirDataPacket
0x04	Broadcast Phase ID	0x00-0x03 = Received packet Broadcast Phase ID
0x05	RSSI	0x00-0xFF = Received packet strength
0x06 - ...	Payload	Up to 20 bytes of data at 50kbit/sec 72 bytes of data at 100kbit/sec

8.15 RXBcastOutSnifferAirCmd

Event message from a node (exclusively) transceiver to host, when it receives a sniffed packet transmitted from a gateway towards another node which contains an air command of any kind (sent via TXAirCmdWrapper). This message is only sent to transceiver host if the proper transceiver configuration register flag is set (RAM bank register sniffFlagsMask, flag BROADCASTOUT_AIR_bm, see section 7.3). Also note that the entirety of the raw wrapped air command, including the requested node address bytes (if applicable, and possibly for multiple nodes), can be found within the payload of this message. Note that a supplementary parameter (Phase In Count) is provided in this message in order for the recipient to know how many MAC addresses are in said air command, which directly correlates with the current number of Broadcast In Phases of the network (parameter BI, see section 2.2).

Note that although is in effect theoretically possible for a node to extrapolate this information by retrieving the network dynamic configuration parameters, in order to get parameter BI instead on relying on the Phase In Count parameter of this event message, it is not a recommended practice. Indeed, it would be possible for a network to dynamically change its current dynamic configuration **after** the sniffed message arrives, but **before** the request for fetching the dynamic configuration is sent (by reading RAM bank register dyn; see section 7.3), thereby creating the low-level equivalent of a ”concurrency atomicity problem”. Thus, the Phase In Count parameter is provided to user in order to ensure atomicity when reading the current Broadcast In Phase count within the dynamic configuration parameters of the mesh network.

Table 33: RXBcastOutSnifferAirCmd (event)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	0x?? 0x?? = Number of bytes in message following this byte (Little-Endian)
0x03	Packet Type	0x2C = RXBcastOutSnifferAirCmd
0x04	Broadcast Phase ID	0x00-0x03 = Received packet Broadcast Phase ID
0x05	RSSI	0x00-0xFF = Received packet strength
0x06	Phase In Count	0x01-0x04 = Number of Broadcast In Phases at sniffed message reception
0x07 - ...	Payload	Up to 20 bytes of data at 50kbit/sec 72 bytes of data at 100kbit/sec

8.16 DynConfig

This is the main command for changing the dynamic configuration of a network, and can only be sent to a transceiver set as a gateway. After this command is sent, a reply will acknowledge the change request immediately if said request is valid (via a DynConfigReply message), and those changes will be applied to the gateway mesh configuration at the beginning of the next broadcast cycle (see section 2.2 for details about broadcast cycles, and for details about the DYN parameters), or will return an error event message if the set of DYN parameters is deemed invalid. Note that the new set of DYN parameters are then subsequently flooded to the whole of the mesh network via a gateway broadcast out packet.

8.17 DynConfigReply

This is the reply message corresponding to gateway command DynConfig, and is sent from gateway transceiver back to its host if the requested DYN configuration changes are deemed valid.

Table 34: DynConfig (command)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	0x07 0x00 = Number of bytes in message following this byte (Little-Endian)
0x03	Packet Type	0x0A = DynConfig
0x04	Phase Out Count	0x01-0x04 = Number of Broadcast Out Phases
0x05	Phase In Count	0x01-0x04 = Number of Broadcast In Phases
0x06	NumSeq	0x01-0x1F = Number of sequence slots (repeater hop count for mesh network)
0x07	NumSeqRedux	0x01 = Numer of sequence slots for redux phase (for the current firmware version only 1 value is supported)
0x08	ReduxEnable-Flag	0x00-0x01 = Redux phase enable flag
0x09	DutyCycleDiv	0x01 0x02 0x05 0x0A 0x014 0x28 0x50 = Duty Cycle inverted

Table 35: DynConfigReply (reply)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	0x01 0x00 = Number of bytes in message following this byte (Little-Endian)
0x03	Packet Type	0x1A = DynConfigReply

8.18 GetRegister

This is a configuration parameter register read command, in a certain register bank, and with a given offset (see section 7.3 for specific configuration register details). Note that the correct expected register size has to be provided as an argument, as the transceiver will use this to verify the validity of the GetRegister command (in case of an invalid size, it will return an error event message). This command can be wrapped as an air command.

Table 36: GetRegister (command)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	0x04 0x00 = Number of bytes in message following this byte (Little-Endian)
0x03	Packet Type	0x03 = GetRegister
0x04	Register Type	0x00=RAMBUF 0x01=RAM 0x02=EEPROM (configuration register bank selector)
0x05	Register Offset	0x00-0x14 = Configuration register offset in register bank
0x06	Register Size	0x01-0x08 = Requested configuration register size

8.19 GetRegisterReply

This is the reply corresponding to GetRegister command. An equivalent air command reply wrapped accordingly can be sent out from a node back to its gateway if the latter sent an air command with GetRegister as the wrapped command.

Table 37: GetRegisterReply (reply)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	(0x04 + N) 0x00 = Number of bytes in message following this byte (Little-Endian) where N is the Register Size parameter (see below)
0x03	Packet Type	0x13 = GetRegisterReply
0x04	Register Type	0x00=RAMBUF 0x01=RAM 0x02=EEPROM (configuration register bank selector)
0x05	Register Offset	0x00-0x14 = Configuration register offset in register bank
0x06	Register Size	0x01-0x08 = Returned configuration register size (N)
0x07 - ...	Register content	From 1 to 8 bytes of data (byte count is N): this is the content of register being read with variables or structures being stored in a Little-Endian manner

8.20 SetRegister

This is a configuration parameter register write command, in a certain register bank, and with a given offset (see section 7.3 for specific configuration register details; note that some registers are read-only). The correct expected register size has to be provided as an argument, as the transceiver will use this to verify the validity of the SetRegister command (in case of an invalid size, it will return an error event message). This command can be wrapped as an air command.

Table 38: SetRegister (command)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	(0x04 + N) 0x00 = Number of bytes in message following this byte (Little-Endian) where N is the Register Size parameter (see below)
0x03	Packet Type	0x04 = SetRegister
0x04	Register Type	0x00=RAMBUF (direct register writes disallowed for RAM and EEPROM banks)
0x05	Register Offset	0x00-0x14 = Configuration register offset in register bank
0x06	Register Size	0x01-0x08 = Configuration register size (N) of register to be written
0x07 - ...	Register content	From 1 to 8 bytes of data (byte count is N): this is the content of register being modified with variables or structures being stored in a Little-Endian manner

8.21 SetRegisterReply

This is the reply corresponding to SetRegister command. An equivalent air command reply wrapped accordingly can be sent out from a node back to its gateway if the latter sent an air command with SetRegister as the wrapped command.

Table 39: SetRegisterReply (reply)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	0x01 0x00 = Number of bytes in message following this byte (Little-Endian)
0x03	Packet Type	0x14 = SetRegisterReply

8.22 TransferConfig

This is a command for transferring content from one register bank to another bank (see section 7.3 for the definition of register banks), or to reset node to factory settings. Note that a factory reset might set the node address to another address than its current address, in which case the node address should be changed to its desired value immediately after a factory reset operation (either via local SetRegister UART command writing new values in RAMBUF bank register addressBuf as defined in section 7.3, followed by a TransferConfig command for RAMBUF to EEPROM, followed by a device reset, or via the equivalent VM execution commands, which are defined in section 5). This command can be wrapped as an air command.

Table 40: TransferConfig (command)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	0x02 0x00 = Number of bytes in message following this byte (Little-Endian)
0x03	Packet Type	0x0B = TransferConfig
0x04	Transfer Type	0x00 = RAM->TMP 0x01 = TMP->RAM 0x02 = TMP->EEPROM 0x03 = RESET TO FACTORY DEFAULTS

8.23 TransferConfigReply

This is the reply corresponding to TransferConfig command. An equivalent air command reply wrapped accordingly can be sent out from a node back to its gateway if the latter sent an air command with TransferConfig as the wrapped command.

Table 41: TransferConfigReply (reply)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	0x01 0x00 = Number of bytes in message following this byte (Little-Endian)
0x03	Packet Type	0x1B = TransferConfigReply

8.24 TXAirCmdWrapper

This command is used to wrap a local serial device command, so that it can be rerouted from a gateway to a given set of nodes, and executed there. This is useful when one needs to send a command to a node, without local access to its serial communication bus, in which case the command can be packaged within a TXAirCmdWrapper command, then sent via serial communication bus to a gateway transceiver controlling the network which is connected to the node in question. Said gateway then reroutes that command to the desired destination node(s) via the mesh network. The node(s) then read that command, as if it received it via its own local serial communication bus, except for the fact that all air commands are read by the node(s) only after a broadcast cycle ended, just before they go into sleep mode (if sleep mode is enabled for any particular node).

Any node which executes any command is usually expected to generate a reply. Because an air command is received via the mesh network, the corresponding reply is **not** transmitted over the serial bus to a host locally connected to that node, but is instead transmitted over the mesh network back to the gateway, which then proceeds to wrap the reply received in a RXAirCmdWrapper message. That wrapped reply message is then sent out to the host connected to that gateway transceiver.

Therefore, this command is only available for transceivers configured as gateways. If a node wants to sniff those special air command packets sent out from a gateway, then it needs to use the event marker RXBcastOutSnifferAirCmd (with proper concomitant configuration register flag set).

For a diagram detailing the whole flow of information and processing events related to it for air commands and the corresponding air replies, see section 7.2.23.

Table 42: TXAirCmdWrapper (command wrapping another command for remoting)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	0x?? 0x?? = Number of bytes in message following this byte (Little-Endian)
0x03	Packet Type	0x0C = TXAirCmdWrapper
0x04	Broadcast Phase ID	0x00-0x03 = Received packet Broadcast Phase ID

0x05	Wrapped Packet Type	IF using MULTI-PHASE mode:
		0x0F = multiphase mode
		ELSE:
		0x?? (not 0x0F) = any single command packet type byte for any command compatible with air command wrapping

0x06 - (0x06+N+1)	Destination MAC Address List	IF using MULTI-PHASE mode:
		N = M*BI bytes, which is the concatenated series of M MAC address bytes for each node being queried (total number of nodes queried is BI, i.e. corresponds to the Broadcast In Phase count of the network)
		ELSE:
		N = M bytes, which are the M MAC address bytes for the single node being queried

(0x06+N) - ...	Wrapped Partial Payload	IF using MULTI-PHASE mode:
		Wrapped command message, with Start-of-Packet and Length word removed
		ELSE:
		Wrapped command message, with Start-of-Packet and Length word and **Packet Type byte of that command** removed

8.25 RXAirCommandWrapper

This event message is a wrapper message for any replies from a node transmitted back to the gateway of a given mesh network (instead of being transmitted to a host locally connected to that node via serial communication bus), and thus is the corresponding message to TXAirCmdWrapper (see section 7.2.22 for details).

This reply message is only applicable for transceivers configured as gateways. If a node needs to sniff another node answer to an air command previously sent by gateway, then that node needs to use the event marker RXBcastInSnifferAirDataPacket (with proper concomitant configuration register flag set).

Table 43: RXAirCmdWrapper (event wrapping remote reply)

Byte offset	Field	Description
0x00	Start-of-Packet	0xFB = Indicates start of protocol formatted message
0x01 - 0x02	Length	0x?? 0x?? = Number of bytes in message following this byte (Little-Endian)
0x03	Packet Type	0x2D = RXAirCmdWrapper
0x04	Broadcast Phase ID	0x00-0x03 = Received packet Broadcast Phase ID
0x05	RSSI	0x00-0xFF = Received packet strength
0x06	Wrapped Packet Type	IF Packet Type byte of corresponding air command has its bit 7 set (i.e. if bit-masking that byte with 0x80 yields a non-zero result), then: 0x??, which is the reply Packet Type byte matching said air command, with its bit 7 set also ELSE: 0x??, which is the regular reply Packet Type byte matching said air command

0x07 - (0x07+N+1)	Node MAC Address	IF Packet Type byte of corresponding air command has its bit 7 set (i.e. if bit-masking that byte with 0x80 yields a non-zero result), then:
		N = M bytes, which are the M MAC address bytes for the node answering said air command
		ELSE:
		N = 0 bytes

(0x07+N)	Wrapped Partial Payload	Wrapped message, with Start-of-Packet and Length word and **Wrapped Packet Type byte of that wrapped reply message** removed

9 Configuration Registers

There are three configuration parameter banks: the TMP bank (also called RAMBUF), RAM bank, and EEPROM bank. The EEPROM allows a configuration that must stick through radio module reset or power-down to be stored in non-volatile memory. The RAM bank is copied from the EEPROM bank at boot-up, and is the main bank used by the mesh controller itself for its operations within its mesh network.

To control the mesh configuration registers, the SMK900 must be in protocol-formated mode. All three banks can be read via GetRegister command. Write operations (SetRegister) are only allowed to the TMP bank, and only to those registers. The proper procedure to change mesh configuration register, containing critical parameters such as the NwkID or the HopTable are changed, is as follows:

– Set registers in TMP bank to the desired value (using SetRegister command);
– Transfer TMP bank to EEPROM bank (using TransferConfig command);
– Reset the module to activate the changes into RAM bank (using DeviceReset command).

Here is a table summarizing the configuration parameter registers available to the user:

Table 44: Register Table

Register offset	Size (bytes)	Read-Only Flag	Register name	Sub-register name	Sub-register location bit offset	Sub-register range	Default value	Description
0	8		addressBuf	-	-		N/A	MAC address buffer
1	1		addressBufLen	-	-		3	MAC address buffer length
2	6	R	dyn	BO	0	1..4	1	Dynamic configuration Broadcast Out Phase Count
				BI	8	1..4	1	Dynamic configuration Broadcast In Phase Count
				NH	16	2..31	5	Dynamic configuration Number of Hops
				NR	24	1	1	Dynamic configuration Number of Hops for Redux Phase
				R	32	0..1	0	Dynamic configuration Redux Enable Flag
				D	40	1 2 5 10 20 40 80	10	Dynamic configuration Inverted Duty Cycle
3	1		nwkId	-	-	0..7	0	Network ID
4	1		hopTable	-	-	0..5	0	FHSS Hop Table selection
5	1		power	-	-	0..1	1	Power selection (0=LO 1=HI)
6	2		uart_bsel	-	-	1..65535	0x93D7	UART baudrate selector. See Serial Interface section for more details.
7	1		nodeType	-	-	0..1	1	Node Type (0=GATEWAY 1=NODE)
8	1		sleepMode	-	-	0..2	2	Sleep mode (0=IDLE 1=RC 2=EXTERNAL)
9	1		extSlpCtrlI2CAddress	-	-	0..127	0x48	Ext Slp Ctrl I2C address
10	2	R	extSlpCorrection Factor	-	-	0..65535	N/A	Correction Factor (current value)
11	1		presetRF	-	-	0..1	0	Preset RF configuration set (0=PRESET1_50K(default) 1=PRESET2_100K)
12	8		cryptoData_qWord0	-	-	0..$2^{64}-1$	0	Data encryption key first 8 bytes Little Endian
13	8		cryptoData_qWord1	-	-	0..$2^{64}-1$	0	Data encryption key last 8 bytes Little Endian
14	7		i2c	delayClockLen	0	0..65535	16	I2C delay for clock generation. The I2C max clock speed is: MaxSpeed[MHz] = 1 / (2 + 0.5*(delayClockLen)). Thus a value of 16 yields a 100KHz max speed and a value of 96 yields a 20KHz max speed. Use lower clock values when using higher value resistor pull-ups or when the capacitance charge on the I2C pins is higher than usual.
				srcPort	16	0	0	I2C source port (hard-coded to the I2C pins assigned to module for the current firmware revision)
				pullupEnabled Flag	24	0..1	1	I2C internal weak pull-up on I2C pins (0 = disabled 1 = enabled)
				powerBus Mode	32	0..3	1	I2C Power Bus mode. Values possible are: 0 = DISABLED (hi-impedance) 1 = NORMAL (pin state toggling between hi-impedance and VCC connection depending on VM execution and I2C commands - on if I2C command executed and togglable via VM commands) 2 = ALWAYSOFF (pin connected to ground) 3 = ALWAYSON (pin connected to VCC)
				powerBus_ powerUp Delay_msec	40	0..65535	0	I2C additional power-up delay for stabilization purpose (if needed)

Table 45: Register Table (continued)

Register offset	Size (bytes)	Read-Only Flag	Register name	Sub-register name	Sub-register location bit offset	Sub-register range	Default value	Description
15	1		meshExecActiveFlag	SERIAL_bm	0	0..1	1	VM Execution Trigger Enable Flag On Serial Cmd Receive Event
				AIRCMD_bm	1	0..1	1	VM Execution Trigger Enable Flag On AirCmd Receive Event
				BOOTUP_bm	2	0..1	1	VM Execution Trigger Enable Flag On Boot-Up Event
				ENTER SEEKMODE_bm	3	0..1	0	VM Execution Trigger Enable Flag On Enter Seek Mode Event
				LEAVE SEEKMODE_bm	4	0..1	0	VM Execution Trigger Enable Flag On Leave Seek Mode Event
				ENTER BROADCAST_bm	5	0..1	0	VM Execution Trigger Enable Flag On Enter Broadcast Event
				LEAVE BROADCAST_bm	6	0..1	0	VM Execution Trigger Enable Flag On Leave Broadcast Event
				ENABLEVM FLASHOP_bm	7	0..1	1	Enable VM Flash Operations Flag
16	1		sniffFlagsMask	BROADCASTIN _bm	0	0..1	0	Enable node sniffing of broadcast in phase messages from other nodes to gateway
				BROADCASTOUT _ AIR_bm	1	0..1	0	Enable node sniffing of broadcast out air commands to specific MAC addresses that do not match address of said sniffer node
17	1		enableNotification FlagsMask	CLOSE BROADCAST _bm	0	0..1	0	Enable broadcast end close event message report (via special RxDataPacket event message with Broadcast phase ID = 0xFF)
				ENDUART TRANSFERTO TXBUFFER_bm	1	0..1	0	Enable uart transfer to TX buffer event message report (via BcastUART2TrxBufferDone event message)
18	8		gpStorage_ qWord0	-	-	0..$2^{64}-1$	0	General purpose storage buffer bytes [0 7]
19	8		gpStorage_ qWord1	-	-	0..$2^{64}-1$	0	General purpose storage buffer bytes [8 15]
20	8		gpStorage_ qWord2	-	-	0..$2^{64}-1$	0	General purpose storage buffer bytes [16 23]
21	1		versionBundle. version	-	-	0..255	1	Firmware version of the module.
22	8		cryptoCfg			0..$2^{64}-1$	0	
23			cryptoCfg			0..$2^{64}-1$	0	
24		R	versionBundle. subVersion			0..255		Sub version of the firmware.
25		R	versionBundle. dbVersion			0..255		Database version of the firmware
26		R	versionBundle. partNumber Version					Hardware part number?
128	8		index			0..$2^{64}-1$	0	Index meta-register
137	2		valueRFLinks			0..$2^{16}-1$	N/A	Indexed read-only register readout for various RF health parameters

9.1 Registers Description

9.1.1 addressBuf

holds the address bytes for the transceiver. This buffer is always defined as having 8 bytes, and the address is held in a Little-Endian format. The number of bytes used to form the effective address is defined in subsequent register addressBufLen (register offset 0x01) from the left. For example an addressBuf = [0x08 0x15 0x02 0x04 0x00 0x00 0x00 0x00] with addressBufLen = 3 will yield a 3-byte MAC address of 2.21.8 (if one uses a 4-byte MAC convention, this can also be written as 0.2.21.8).

9.1.2 addressBufLen

holds the effective number of address bytes, as described above.

9.1.3 dyn

holds the dynamic configuration of the transceiver. This is the set of 6 parameters that determine the configuration of the broadcast cycles and sleep intervals of a given mesh network, as described in dynamic parameters section. This register is read-only, because only a transceiver configured as a gateway is allowed to change the dynamic configuration of the whole network it is attached to (and this is not done via asynchronous writing to the dyn register in the RAM bank directly, but is rather accomplished using a special command, which schedules a change action of the dyn parameters in the RAM bank in such a way that the change becomes effective at the beginning of the next broadcast cycle).

9.1.4 nwkID

holds the network ID of the network that the transceiver is allowed to receive and transmit to. This is used as a basic packet filter in such a way that any network using the same frequency hop sequence than the transceiver, but with a different configured network ID, will be effectively invisible to said transceiver. Range of valid nwkID is from 0 to 7 (i.e. 8 different possible values). A given network access key can be uniquely configured using a given set of 4 configuration registers: nwkID, hopTable, cryptoData_qWord0, and cryptoData_qWord1.

9.1.5 hopTable

holds the current hop sequence of the network intended to be connected to the transceiver. This variable ranges from 0 to 5 (i.e. 6 different possible values). Combined with nwkID, cryptoData_qWord0 and cryptoData_qWord1, this defines a unique network access key. Note that if one discounts the use of encryption keys, then it would be possible to combine hopTable and nwkID in such a way that the combined number becomes an extended "channel number". For instance, one could define an arbitrary configuration variable CHANNEL with the following mapping: hopTable = CHANNEL % 6 + HOPTABLE_OFFSET, and nwkID = CHANNEL % 8 + NWKID_OFFSET, where HOPTABLE_OFFSET and NWKID_OFFSET are arbitrarily chosen numbers. This would extend the maximum number of channels to effective.

9.1.6 power

holds two different power presets, one at high power (1 = HI: 158mW) and the other at low power (0 = LO: 40mW).

9.1.7 uart_bsel

baud rate selector register for the main UART communication bus. See section 3.0 for details on how to configure this register.

9.1.8 nodeType

defines the type of transceiver, which can either be a gateway (nodeType = 0) or an individual node of the mesh (nodeType = 1).

9.1.9 sleepMode

selects the active sleep clock in use with the transceiver. Idle mode (sleepMode = 0) is a setting where only the internal fast clock of the mesh controller is employed as the main sleep time base. WARNING: this setting is the highest power consumption mode, and is mainly suitable for transceivers wired on the electrical grid in some way, with high power availability off-grid. Examples include typically transceivers configured as gateways. RC mode (sleepMode = 1) is a setting where a lower power RC clock is used as the main sleep clock... This mode uses the higher-speed clock in order to recalibrate the RC clock operation from broadcast cycle to broadcast cycle, in order to compensate for clock frequency variation due to environmental changes such as temperature. Note that even with this compensation in place, it is not suggested to use this sleep mode for sleep intervals between broadcast cycles higher than 2 seconds. Finally, external sleep controller mode (sleepMode = 2) allows the use of a dedicated Smartrek external sleep controller chip in I2C slave mode in order to retro-fit ultra-low power consumption capabilities for longer sleep period of times (> 2 seconds, up to about 20 seconds). This chip is to be connected to the main transceiver using the transceiver I2C lines. Pull-up resistor values suggested for that I2C bus is 2 kOhms or less, and powered with an input voltage between 3.3 and 5.5 V. See section 6.3 for more details on how to integrate the external sleep controller chip.

9.1.10 extSlpCtrlI2CAddress

external sleep controller I2C address, if external sleep controller is selected as the main sleepMode mode of operation. Default factory-configured I2C address is 0x48 (note that the external sleep controller itself can have its own internal address changed via I2C command (see section 6.3 for more details).

9.1.11 extSlpCorrectionFactor

this is a read-only register indicating the current correction factor compensating for the external sleep controller crystal drift (see section 6.3 for more details).

9.1.12 presetRF

this is the current RF parameter preset for the transceiver (default value is presetRF = 0) and corresponds to a RF bitrate of 50 kbit/sec. Currently, it is the only suggested preset, and for this preset, the data payload of every packet is limited to 20 bytes. For the presetRF = 1 set of parameters, it is a (beta-stage) preset for 100 kbit/sec mode of operation with a somewhat larger maximum data payload size (72 bytes).

9.1.13 cryptoData_qWord0

AES-128 16-byte key least significant 8 bytes (Little-Endian). This register, combined with its counterpart cryptoData_qWord1, nwkId and hopTable, constitutes the set of registers for a given network access key.

9.1.14 i2c

register set for I2C bus configuration parameters. The following sub-registers are defined:

- delayClockLen - I2C maximum clock speed parameter, which defines the minimum duration of an I2C bit in the following fashion: $MaxSpeed[MHz] = 1/(2 + 0.5(delayClockLen))$;
- srcPort - I2C port selector (must be set to 0 in the current firmware revision);
- pullupEnabledFlag - enables or disables the internal resistor pull-ups for I2C port (those are very weak >10kOhm pull-ups, and will not satisfy the I2C rise-time specifications by default);
- powerBusMode - selects the operation mode for I2C Power Pin (module pin #4, see section 6.1). Values possible are: 0 = DISABLED (hi-impedance), 1 = NORMAL (pin state toggling between hi-impedance and VCC connection depending on VM execution and I2C commands - on if I2C command executed, and togglable via VM commands), 2 = ALWAYSOFF (pin connected to ground), 3 = ALWAYSON (pin connected to VCC). Note about NORMAL mode: every time the I2C bus is used in order for the mesh controller to interact with an external sleep controller, the I2C Power Pin will be connected to VCC temporarily, then shut down and set to high impedance when that transaction is done. For user-customized control of this pin, see section 5 for details;
- powerBus_powerUpDelay_msec - delay automatically applied after the I2C Power Pin (module pin #4, see section 6.1) is auto-connected to VCC, in milliseconds, for hardware stabilization purposes.

9.1.15 meshExecActiveFlag

bit-mask register in order to set up event trigger points that are to activate a VM user-made program. All bit flags are considered enabled when their binary value is 1, and disabled when their binary value is 0. The bit-mask constants corresponding to each trigger point as defined in section 5 are:

9.1.16 sniffFlagsMask

flags which, when activated, enable sniffing of some types of packets, either from a gateway to another node, or vice versa. The concept of sniffing is only relevant when using UART command mode, because transparent mode acts as a dumb point-to-multipoint serial link between all the transceivers. Moreover, this mode is only relevant for air commands when considering packets outgoing from a gateway to nodes, because only this mode uses automatic packet filtering at the destination transceiver site using the requested MAC address given by the gateway. Indeed, all regular long packet transmissions out in command mode are considered as being whole network broadcasts, and are received by default by all node transceivers in the mesh network. Here is a description table of the relevant flags:

Table 46: MeshExecActiveFlag Bit-Mask Register

Constant name	Bit position	Bit-mask value	Description
SERIAL_bm	0	0x01	Activate virtual machine when a serial command for VM Exec has been received locally (see section 5 for details) via the UART communication bus. In this case the VM command is executed on the spot when the serial command is received. Note that if the VM is executed during an active broadcast cycle the virtual machine engine will always yield to the mesh protocol controller software when the latter needs to execute timing-critical operations for synchronizing the transceiver with its mesh network as for any local serial commands (see section 3.0 for details).
AIRCMD_bm	1	0x02	Activate virtual machine in a node when an air command for VM Exec (i.e. command wrapped with an Air Cmd. Wrapper) from a gateway has been received via a gateway broadcast out phase (see section 5 for details). In this case the VM execution command is executed at the end of the broadcast cycle in which the air command was received like any other air command requests. Note that LEAVEBROADCAST_bm event marker will be activated before said VM will be executed if that event marker is enabled.
BOOTUP_bm	2	0x04	Activate virtual machine when the main boot sequence and configuration initialization of the transceiver finished executing.
ENTERSEEK-MODE_bm	3	0x08	Activate virtual machine when a node goes into seek mode (i.e. searches for an existing mesh network to latch itself onto).
LEAVESEEK-MODE_bm	4	0x10	Activate virtual machine when a node leaves seek mode (i.e. just found an existing mesh network to latch itself onto).
ENTER-BROAD-CAST_bm	5	0x20	Activate virtual machine when a transceiver (gateway or node) starts a broadcast cycle (i.e. leaves sleep/idle state in between broadcast cycles).
LEAVEBROAD-CAST_bm	6	0x40	Activate virtual machine when a transceiver (gateway or node) broadcast cycle just ended (i.e. about to go to sleep). Note that this event if enabled will be always triggered before any pending VM execution request via air command (see above AIRCMD_bm) will be executed.
ENABLEVM-FLASHOP_bm	7	0x80	Enable VM flash operations. This needs to be enabled for VM programming operations to be available and thus acts as a safety lock flag.

Table 47: Relevant Sniff Flags

Constant name	Bit position	Bit-mask value	Description
BROAD-CASTIN_bm	0	0x01	Enable node sniffing of broadcast in phase messages from other nodes to gateway. All broadcast in that was sent in command mode by another node be it an air command or a regular packet transmission will be read by the sniffing transceiver when this mode is activated.
BROAD-CASTOUT_AIR_bm	1	0x02	Enable node sniffing of broadcast out air commands to specific MAC addresses that do not match address of said sniffer node. Note that this mode only applies for air commands sent from a gateway out.

9.1.17 enableNotificationFlagsMask

flags enabling special notification UART messages for some useful events. Here is a description table of the relevant flags:

Table 48: Flags Enabling Special Notification UART Messages

Constant name	Bit pos-ition	Bit-mask value	Description
CLOSE-BROAD-CAST_bm	0	0x01	Enable broadcast end close event message report (via special RxDataPacket event message with Broadcast phase ID = 0xFF). If this is activated the following raw UART message will be sent from transceiver to the host in order to mark the end of a broadcast (this always occur before any air command is processed as explained in section 5):
			[0xFB 0x03 0x00 0x26 0xFF 0x00]
ENDUART-TRANSFER-TOTXBUFFER_bm	1	0x02	Enable UART transfer to TX buffer event message report (via BcastUART2TrxBufferDone event message). This marker will be sent from transceiver to the host in order to mark a transfer from internal UART buffer of a message to the main RF packet buffers in which case a new UART message can now be queued in the UART buffer. This event occurs at the very beginning of every broadcast cycle and is triggered even if no UART message is pending (in that case the byte transfer count internally to the transceiver is zero but the transfer event still occurs) The raw UART message sent to host for this event marker is:
			[0xFB 0x01 0x00 0x2A]

9.1.18 gpStorage_qWordx

gpStorage_qWord0, gpStorage_qWord1, gpStorage_qWord2 - 8 byte sized general purpose registers, that are usually employed as general storage space for a VM user program so that variables can be carried over from a given VM execution instance to the next (see section 5 for more details). Moreover, because those registers exist in both RAMBUF, RAM and EEPROM banks, the latter (EEPROM) can be used as non-volatile general purpose storage space for a given VM user program.

9.1.19 index

index - 8 byte sized indexing meta-register, employed to access or write to index-enabled registers. For instance, a given read-only register could allow user to read out more than one configuration or status parameter, depending on the value held by the index register.

9.1.20 valueRFLinks

valueRFLinks - 2 byte sized index register, read-only, which reads out health status of the end-node. The health status parameter being read out depends on the value currently held by the index register. The following readouts are provided:

– Index 0: scaledAveragedRSSI - Auto-rescaled (not raw) and low-pass filtered RSSI value (back-compatible with legacy SMK targets). This is the average RSSI value as seen by packets received by the end-node itself, and thus this value can be polled remotely via gateway in order to determine the approximate mesh network signal strengths at a given end-node location;
– Index 1: averagedRSSI - This is the non-rescaled, but low-pass filtered RSSI value (similar to Index 0, but with no back-compatibility);
– Index 2: hopX256 - The current hop distance of the end-node with respect to its gateway (a.k.a. how many hops is needed to reach it), times 256. This is a low-pass filtered value, in order to stabilize the measurement against RF instantaneous variabilities;

— Index 3: hop - The current hop distance of the end-node with respect to its gateway (a.k.a. how many hops is needed to reach it), times 1. This is a low-pass filtered value, in order to stabilize the measurement against RF instantaneous variabilities.

10 Developer Kit

10.1 Portia Adapter Board

The Portia Adapter Board is provided in the development kit in order to connect a Portia SMK900 radio module to a USB serial device such as a PC or laptop This is especially useful, when developing, to monitor the mesh network, to generate/modify/upload a Virtual Machine firmware. Additionally, the adapter board features a breadboard footprint compatible dual inline package (DIP). Note that two 15 positions male headers with 100 mil spacing have to be soldered .

10.2 Arduino-Compatible Shield

The Arduino-compatible Shield can be used for fast prototyping by leveraging the extensive Arduino library

11 Certification Information

SMK900 Certification

Smartrek Technologies Module

FCC ID: 2AP8V-SMK900

IC: 24079-SMK900

United State (FCC)

Note: This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

— Reorient or relocate the receiving antenna.
— Increase the separation between the equipment and receiver.
— Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
— Consult the dealer or an experienced radio/TV technician for help.

FCC Antenna Gain Restriction and MPE Statement: The SMK900 radio has been designed to operate with any dipole antenna of up to 3 dBi of gain. The antenna used for this transmitter must be installed to provide a separation distance of at least 20 cm from all persons and must not be co-located or operating in conjunction with any other antenna or transmitter.

IMPORTANT: The SMK900 Module has been certified by the FCC for use with other products without any further certification (as per FCC section 2.1091). Modifications not expressly approved by Smartrek Technologies could void the user's authority to operate the equipment.

IMPORTANT: OEMs must test final product to comply with unintentional radiators (FCC section 15.107 and 15.109) before declaring compliance of their final product to Part 15 of the FCC rules.

IMPORTANT: The RF module has been certified for remote and base radio applications. If the module will be used for portable applications, please take note of the following instructions the device must undergo SAR testing.

Warning!

OEM labeling requirements: As an Original Equipment Manufacturer (OEM) you must ensure that FCC labeling requirements are met. You must include a clearly visible label on the outside of the final product enclosure that displays the following content:

Contains FCC ID: 2AP8V-SMK900 The enclosed device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (i.) this device may not cause harmful interference and (ii.) this device must accept any interference received, including interference that may cause undesired operation.

Certification SMK900

Smartrek Technologies Module

FCC ID: 2AP8V-SMK900

IC: 24079-SMK900

États-Unis (FCC)

Remarque : Cet équipement a été testé et déclaré conforme aux limites d'un appareil numérique de classe B, conformément à la partie 15 des règlements de la FCC. Ces limites sont conçues pour fournir une protection raisonnable contre les interférences nuisibles dans une installation résidentielle. Cet équipement génère, utilise et peut émettre de l'énergie radiofréquence et, s'il n'est pas installé et utilisé conformément aux instructions, peut causer des interférences nuisibles aux communications radio. Cependant, il n'y a aucune garantie que des interférences ne se produiront pas dans une installation particulière. Si cet équipement cause des interférences nuisibles à la réception de la radio ou de la télévision, ce qui peut être déterminé en éteignant et en rallumant l'équipement, l'utilisateur peut tenter de corriger ces interférences par une ou plusieurs des mesures suivantes :

- Réorienter ou déplacer l'antenne de réception.
- Augmenter la distance entre l'équipement et le récepteur.
- Brancher l'équipement dans une prise de courant sur un circuit différent de celui auquel le récepteur est branché.
- Consulter le revendeur ou un technicien radio/TV expérimenté pour obtenir de l'aide.

Restriction de gain d'antenne FCC et déclaration MPE : Le module radio SMK900 a été conçu pour fonctionner avec n'importe quelle antenne dipôle jusqu'à 3 dBi de gain. L'antenne utilisée pour cet émetteur doit être installée à une distance de séparation d'au moins 20 cm de toutes les personnes et ne doit pas être placée ou utilisée conjointement avec une autre antenne ou un autre émetteur.

IMPORTANT : Le module SMK900 a été certifié par la FCC pour une utilisation avec d'autres produits sans autre certification (selon la section 2.1091 de la FCC). Toute modification non expressément approuvée par Smartrek Technologies pourrait annuler le droit de l'utilisateur d'utiliser l'équipement.

IMPORTANT : Les OEM doivent tester le produit final pour se conformer aux radiateurs non intentionnels (articles 15.107 et 15.109 de la FCC) avant de déclarer la conformité de leur produit final à la partie 15 des règles de la FCC.

IMPORTANT : Le module RF a été certifié pour les applications radio de base et à distance. Si le module doit être utilisé pour des applications portables, veuillez prendre note des instructions suivantes, l'appareil doit subir un test SAR.

Attention !

Exigences d'étiquetage OEM : En tant que fabricant d'équipement d'origine (OEM), vous devez vous assurer que les exigences d'étiquetage de la FCC sont respectées. Vous devez inclure une étiquette clairement visible à l'extérieur de l'enveloppe du produit final qui affiche le contenu suivant :

Contient l'ID FCC : 2AP8V-SMK900 L'appareil fourni est conforme à la partie 15 des règlements de la FCC. L'exploitation est assujettie à aux deux conditions : (i.) cet appareil ne doit pas causer d'interférences nuisibles et (ii.) cet appareil doit doit accepter toute interférence reçue, y compris des interférences susceptibles de provoquer un fonctionnement non désiré.

ISED (Innovation, Science and Economic Development Canada)

This device complies with Industry Canada license-exempt RSS standard(s). Operation is subject to the following two conditions: (1) this device may not cause interference, and (2) this device must accept any interference, including interference that may cause undesired operation of the device.

Le présent appareil est conforme aux CNR d'Industrie Canada applicables aux appareils radio exempts de licence. L'exploitation est autorisée aux deux conditions suivantes : (1) l'appareil ne doit pas produire de brouillage, et (2) l'utilisateur de l'appareil doit accepter tout brouillage radioélectrique subi, même si le brouillage est susceptible d'en compromettre le fonctionnement.

Labeling requirements: Similarly to FCC, labeling requirements for Industry Canada must be clearly visible label on the outside of the final product enclosure and must display the following text.

Contains IC: 24079-SMK900 The integrator is responsible for its product to comply with IC ICES-003 & FCC Part 15, Sub. B Unintentional Radiators. ICES-003 is the same as FCC Part 15 Sub. B and Industry Canada accepts FCC test report or CISPR 22 test report for compliance with ICES-003.