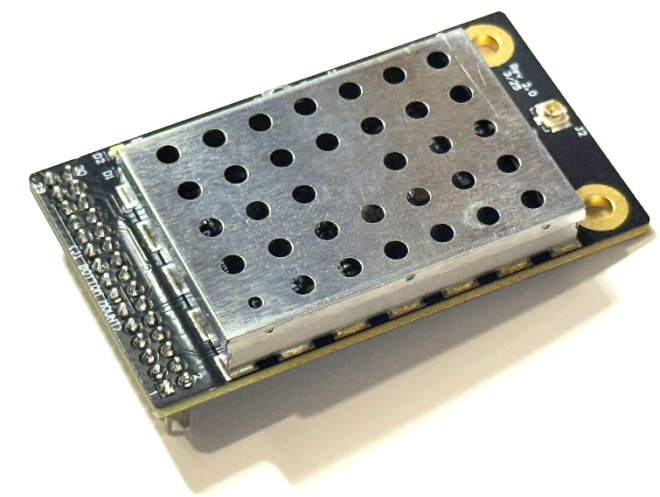
Aether RF
KM-100A
Integration Guide



Revision 1.2, 2/12/24

TABLE OF CONTENTS

1. Introduction

The **Aether RF *KM-100A*** is a versatile module designed to meet the demands of modern wireless communication. This compact device boasts a 30 dBm, 900 MHz ISM Band radio (902 MHz to 928 MHz), perfect for reliable long-distance communication between two modules. The Aether RF ***KM-100A*** is UART controlled, ensuring easy hardware and firmware integration with host systems, and operates on a single 5-volt DC power supply. Whether you are developing new wireless solutions or enhancing existing products with wireless capability, the Aether RF ***KM-100A*** is your go-to module for robust and flexible communication and control.

2. FCC Declaration

This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions:

(1) This device may not cause harmful interference, and

(2) this device must accept any interference received, including interference that may cause undesired operation.

This equipment has been tested and found to comply with the limits for a Class B digital device.

The following antennas are FCC certified for use with the module:

1. Aether RF: ARF-A100
2. TE Connectivity: ANT-916-CW-HWR-RPS
3. TE Connectivity: ANT-915-IPW1-RPS

Changes, modifications, or antennas not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

This module is FCC certified. When installed in a host device, the host must be labeled to display the FCC ID of the module. The label must be visible to the end user and must contain the following text:

Contains FCC ID: 2BGQ6KM100A

The end product must not be marketed as FCC certified unless the FCC ID of this module is clearly displayed on the host device. The integrator is responsible for ensuring that the host continues to comply with FCC Part 15 requirements.

FCC Radiation Exposure Statement

This equipment complies with FCC radiation exposure limits set forth for an uncontrolled environment. This equipment should be installed and operated with minimum distance 20cm between the radiator and your body.

This transmitter must not be co-located or operating in conjunction with any other antenna or transmitter. The antennas 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.

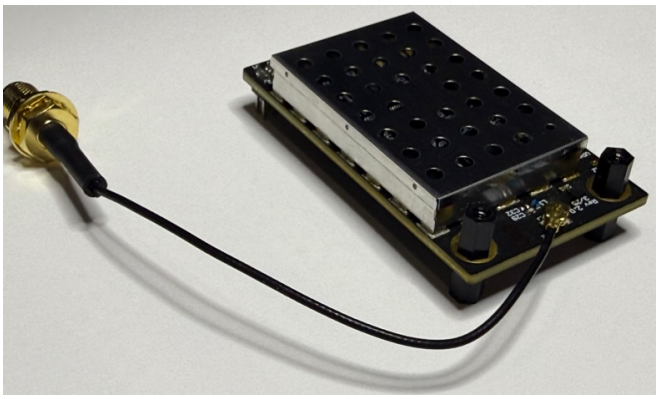
3. Installation

You'll need the following in order to install the module:

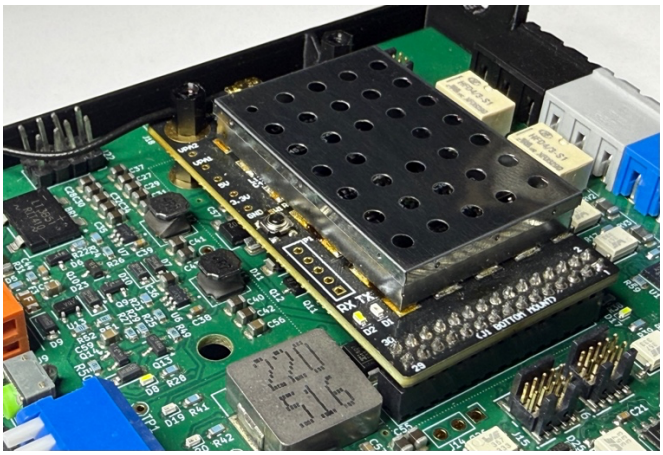
- One module
- One u.FL to RP-SMA adapter (comes installed on module)
- One of the below antennas:
    - Aether RF: ARF-A100
    - TE Connectivity: ANT-916-CW-HWR-RPS
    - TE Connectivity: ANT-915-IPW1-RPS
- Two 6mm Plastic standoffs , either screw type or snap in
- A host board (see Sections 6 and 7 for pinout and power requirments)
- A 6.5mm hole to mount the RP-SMA connector

Steps:

1. Make sure the host board is powered off
2. Install the standoffs on either the module or the host board

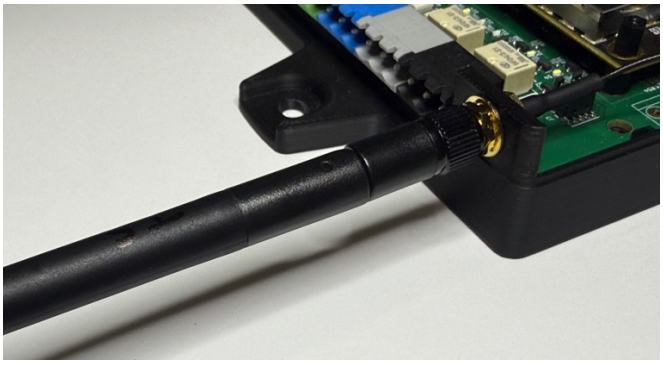


3. Plug in the module to the board

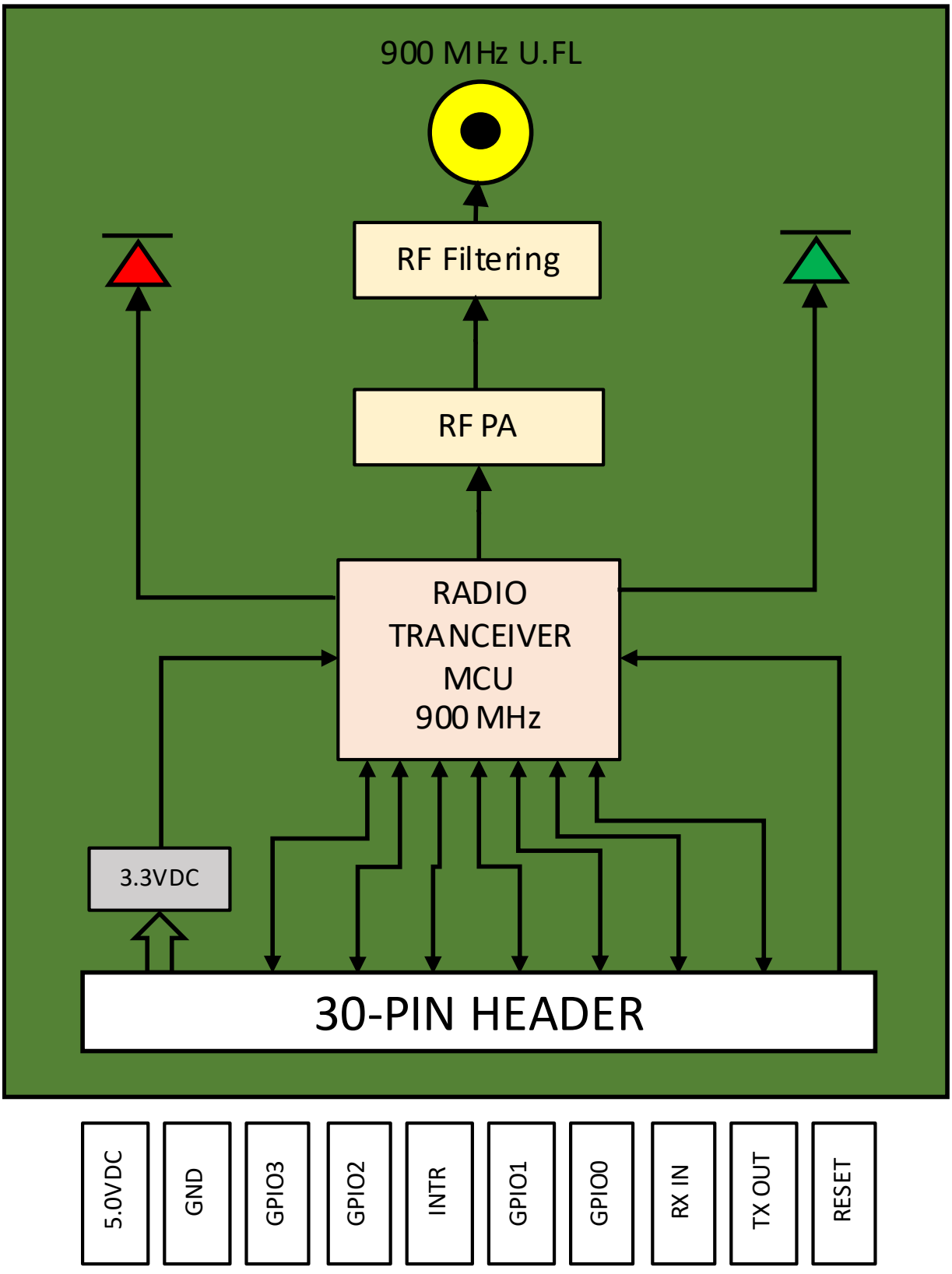


4. Route the RP-SMA connector through the hole
5. Secure the RP-SMA connector to the chassis with the provided nut and lock washer. The use of pliers is recommended to make sure the RP-SMA connector is tight.

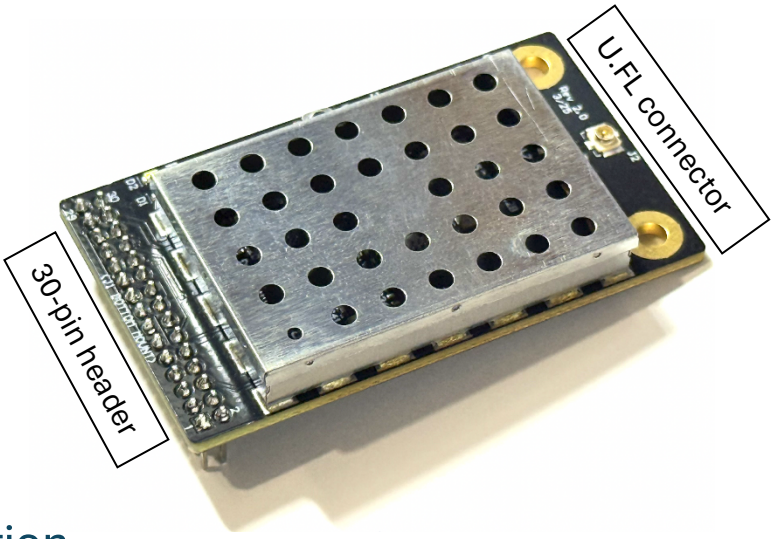
6. Install the antenna on to the RP-SMA connector. If the RP-SMA connector rotates when the antenna is installed, then tighten the RP-SMA connector nut.



4. Block Diagram



5. Connecting to the Wireless Module



6. Pin Description

The part required to be installed on the host PCB is TE Connectivity 3-1470209-8

PIN	FUNCTION	NOTE
1	GROUND	Module Power Ground
2	GPIO7	General Purpose I/O
3	GROUND	Module Power Ground
4	INTERRUPT	Hardware Interrupt from Module
5	NC	No Connect
6	NC	No Connect
7	GPIO2	General Purpose I/O
8	NC	No Connect
9	SWDIO	Programming Pin
10	NC	No Connect
11	SWDCLK	Programming Pin
12	RX	UART Receive Data to Module
13	NC	NC
14	TX	UART Transmit Data from Module
15	NC	NC
16	NC	No Connect
17	GPIO1	General Purpose I/O
18	GPIO6	General Purpose I/O
19	GPIO5	General Purpose I/O
20	RTS	Optional
21	GPIO4	General Purpose I/O
22	CTS	Optional
23	RESET	Hardware Reset to Module

24	NC	No Connect
25	GPIO3	General Purpose I/O
26	GPIO0	General Purpose I/O
27	GROUND	Module Power Ground
28	+5 VDC	
29	GROUND	Module Power Ground
30	+5 VDC	
J1	antenna connector	U.FL male

7. Detailed Pin Description

RESET

Connect to open collector or open drain host output pin and/or ICSP programming header. Internally pulled up to 3.3 VDC on module.

TX

UART TRANSMIT pin from Module to Host.

RX

UART RECEIVE Pin from Host to Module.

GPIO(n)

Configurable GPIO for general purpose use as Input or Output to/from Host.

RTS/CTS

Optionally configured for UART hardware signaling.

Interrupt

Active low output from Module to Host. Event configurable. Pulse width configurable.

GND

System Common GROUND

+5VDC

System Voltage Supply from Host

8. Command Format

Command Send

attention (uint8_t)	command (uint8_t)	argument0 (uint8_t)	argument1 (uint8_t)	argument..x (uint8_t)	terminate (uint8_t)

Attention = '~'

Command = 1 of 255 total

Argument(s) = argument0 -> argument..x, number of arguments is command dependent

Command terminator = 0x0a = '\n' = LF (line feed)

Command Response

attn	command	arg	LF
~	uint8_t	uint8_t	uint8_t

Attention = '~'

Command = 1 of 255 total

arg = ACK (0x06) or err_code

Response terminator = 0x0a = '\n' = LF (line feed)

UART ACK can be enabled/disabled.

9. Radio Module Commands

See current revision app_radio_commands.h for actual command byte and return error code enumerated values. Command values below are example values only. All 32 bytes are not required. Valid paypoad is 1-32 bytes.

SET RADIO TRANSMIT (ON)

Commands the radio to transmit.

(uint8_t cmd, uint8_t ret) set_radio_transmit (bool arg)

attn	cmd0	payload0	payload...	payload31	LF
~	set_transmit	byte	byte(s)	byte	uint8_t

Example:

0x7e	0x01	byte0	...bytes(n)	...byte31	0x0a

Notes:

Automatically terminates and sets radio to receive mode.

Configurable hardware interrupt when complete.

Response:

cmd, ACK (0x06) or err_code

SET RADIO TRANSMIT (OFF)

Commands the radio module to stop transmitting during a transmission. Note that the radio module automatically terminates transmit mode and returns to receive mode after packet is sent.

attn	cmd	arg	LF
~	set_transmit	false	uint8_t

Example:

0x7e	0x01	byte0	...bytes(n)	...byte31	0x0a

Notes:
Terminates an in progress transmission.

Response:

cmd, ACK (0x06) or err_code

SET RADIO RECEIVE MODE

Note that the radio module boots in receive mode and also automatically terminates transmit mode and returns to receive mode after packet is sent.

(uint8_t cmd, uint8_t ret) set_receive (bool arg)

attn	cmd	arg	LF
~	set_receive	true	uint8_t

Example:

0x7e	0x02	0x01	0x0a

Notes:
Terminates transmission if in progress.

Response:

cmd, ACK (0x06) or err_code

SET TRANSMIT PAYLOAD (Deprecated)

Sets radio module transmit buffer with data desired to be sent

(uint8_t cmd, uint8_t ret) set_tx_payload(uint8_t cmd, bool tx option, uint8_t len, char(s) data)

attn	cmd0	cmd1	len	payload0	payload…	payload31	LF
~	set_tx_payload	true/false	byte	byte	byte(s)	byte	uint8_t

Example:

0x7e	0x03	0x01	0x01	0x20	byte0	…byte31	0x0a

Notes:

Cmd1 = tx option = true = immediately transmit after setting data. Configurable hardware interrupt when complete. Automatically cleared after transmission.

Cmd1 = tx option = false = wait for *set_transmit* command to transmit.

Len and Payload = 1-32 bytes

Response:

cmd, ACK (0x06) or err_code

GET TRANSMIT PAYLOAD

Returns radio module transmit buffer

(uint8_t cmd, bool tx option, uint8_t length, char(s) data) get_tx_payload (uint8_t cmd)

attn	cmd	LF
~	uint8_t	uint8_t

Example:

0x7e	0x04	0x0a

Notes:
Returns current radio transmit buffer

Response:

Returns radio transmit buffer in *set_transmit_payload* format.

SET FREQUENCY HOPPING

Sets radio module frequency hopping mode and pseudo random hop table value

(uint8_t cmd, uint8_ret) set_frequency_hopping (bool mode, uint8_t hop_table, uint8_t data rate)

attn	cmd	arg0	arg1	arg2	LF
~	set_frequency_hopping	true	table	data rate	uint8_t

Example:

0x7e	0x08	0x01	0x20	0x0a

Notes:

arg0 = true = set to frequency hopping mode.

arg1 = pseudo random hopping table = 0 thru 199.

arg0 = data rate =

enum{

2400,

4800,

9600

};

Response:

cmd, ACK (0x06) or err_code

GET FREQUENCY HOPPING

Returns radio module frequency hopping mode and pseudo random hop table value

(uint8_t cmd, uint8_t cmd, ret = bool, uint8_t hop_table) get_frequency_hopping (uint8_t cmd)

attn	cmd	table	LF
~	get_frequency_hopping	uint8_t	uint8_t

Example:

0x7e	0x09	0x57	0x0a

Notes:

Response = frequency hopping = true/false. Current hop table in use.

SET SINGLE CHANNEL (deprecated)

Sets radio module to single channel (narrow band) mode and channel

(uint8_t cmd, uint8_t ret) set_single_channel(uint8_t channel)

attn	cmd	arg0	arg1	LF

~	set_single_channel	byte	byte	uint8_t

Example:

0x7e	0x0a	0x01	0x20	0x0a

Notes:

arg0 = true = sets to single frequency transmission (narrow band – non hopping). Receiving module(s) must be set to same channel if in narrow band mode.
arg1 = transmission channel frequency = 0 thru 49.

arg0 = false = disable single channel mode and switch to frequency hopping. arg1 = frequency hopping table. (see *set_frequency_hopping*)

Can be used if host application hopping channel, sequence control, and timing is required.
Response:

cmd, ACK (0x06) or err_code

GET SINGLE CHANNEL (deprecated)

Returns radio module single channel (narrow band) mode and channel

(uint8_t cmd, uint8_t channel, uint8_t hop_table) get_single_channel (uint8_t cmd)

attn	cmd	LF
~	get_single_channel	uint8_t

Example:

0x7e	0x0b

Notes:
Response:

Returns single_channel setting in *get_single_channel* format.

SET RADIO POWER

Sets radio module output power.

(uint8_t cmd, uint8_t ret) set_radio_power(uint8_t cmd, uint8_t power)

attn	cmd	arg	LF
~	set_radio_power	uint8_t	uint8_t

Example:

0x7e	0x0c	0x06	0x0a

Notes:
Sets output power register in radio transceiver IC. Value = 0 thru 255.

Value	Nominal Output (dBm)	Value	Nominal Output (dBm)	Value	Nominal Output (dBm)	Value	Nominal Output (dBm)
0	TBD	4	TBD	8	TBD	12	TBD
1	TBD	5	TBD	9	TBD	13	TBD
2	TBD	6	TBD	10	TBD	14	TBD
3	TBD	7	TBD	11	TBD	15	TBD

Response:

cmd, ACK (0x06) or err_code

Notes:

Maximum power output is 29.88 dBm. The nodule will not allow power any greater regardless of set_power setting.

GET RADIO POWER

Returns radio module output power setting

(uint8_t cmd, uint8_t power)get_radio_power (uint8_t cmd)

attn	cmd	LF
~	get_radio_power	uint8_t

Example:

0x7e	0x0d	0x0a

Notes:
Get output power register in radio transceiver IC. Value = 0 – 15.

Response:

Returns output power register in *set_radio_power* format.

SET AES ENCRYPTION

Enables or disables AES encryption, and sets AES128 or AES 256 mode.

(uint8_t cmd, uint_t ret) set_aes_encrytion (bool mode, uint8_t val)

attn	cmd	arg0	LF
~	set_aes_encryption	uint8_t	uint8_t

Example:

0x7e	0x0e	0x20	0x0a

Notes:

arg0 AES encryption values = 0x1f = AES128, 0xff = AES256, 0 = disable (default).

If AES is enabled, transmit packet size is fixed to 16 or 32 bytes accordingly.

Shorter packets sent using *set_tx_payload* will be '0' padded.

Response:

cmd, ACK (0x06) or err_code

GET AES ENCRYPTION

Returns AES encryption settings

(uint8_t cmd, bool mode, uint8_t val) get_aes_encryption (uint8_t cmd)

attn	cmd	LF
~	get_aes_encryption	uint8_t

Example:

0x7e	0x10	0x0a

Notes:

Get AES encryption settings.

Response:

Returns settings in set_aes_encryption format.

SET AES KEY

Sets AES encryption key values. Length depends on AES128 or AES256 setting

(uint8_t cmd, uint_t ret) set_aes_key (uint8_t cmd, byte0… byte15 or 31)

attn	cmd	arg0	payload0	payload…	payload15 or 31	LF
~	set_aes_key	uint8_t	byte	byte(s)	byte	uint8_t

Example:

0x7e	0x11	0x20	byte0	…byte 15 or byte31	0x0a

Notes:

arg0 = AES encryption values = 0x1f = AES128, 0xff = AES256

payload length = 16 or 32 bytes accordingly.

Response:

cmd, ACK (0x06) or err_code

GET AES KEY

Returns AES encryption key value. Length depnds on Length depends on AES128 or AES256 setting

(uint8_t cmd, byte0... byte15 or 31) get_aes_key (uint8_t cmd)

attn	cmd	LF
~	get_aes_key	uint8_t

Example:

0x7e	0x12	0x0a

Notes:

Returns settings in *set_aes_key* format. If no key set, 0x00 (1 byte) will be returned.

10. System Commands

SET LED

Sets or clears red LED or green LED. Also sets time in msec or persistent.

(uint8_t cmd, uint_t ret) set_led (uint8_t led, bool mode, uint8_t time)

attn	cmd	arg0	arg1	arg2	LF
~	set_led	uint8_t	bool	uint8	uint8_t

Example:

0x7e	0x13	0x01	0x01	0x64	0x0a

Notes:

arg0 = 0x01 or 0x02 = RED or GREEN LED, respectively

arg1= true or false = ON or OFF, respectively

arg2 = 0x01 -> 0xfe = ON/OFF time in milliseconds. 0xff = persistent ON/OFF

Response:

cmd, ACK (0x06) or err_code

SET_GPIO

Sets or clears specified GPIO pin. Also sets time in msec or persistent.

(uint8_t cmd, uint_t ret) set_gpio (uint8_t gpio, bool mode, uint8_t time)

attn	cmd	arg0	arg1	arg2	LF
~	set_gpio	uint8_t	uint8_t	uint8	uint8_t

Example:

0x7e	0x14	0x01	0x01	0x20	0x0a

Notes:

arg0 = 0x00 or 0x03 = GPIO pin number

arg1= 0x00 or 0x01 = LOW or HIGH, respectively

arg2 = 0x01 thru 0xfe = LOW/HIGH time in milliseconds. 0xff = persistent ON/OFF, 0x00 = invalid

Response:

cmd, ACK (0x06) or err_code

GET_GPIO

Returns GPIO output states in bit mapped lower nibble of return byte

(uint8_t cmd, uint8_t state) get_gpio (uint8_t cmd)

attn	cmd	LF
~	get_gpio	uint8_t

Example:

0x7e	0x15	0x0a

Notes:

Response:

Returns current state of GPIO outputs.

Return Example:

0x7e	0x15	0x05	0x0a

gpio0 = set

gpio1 = clear

gpio2 = set

gpio3 = clear

SET INTERRUPT GLOBAL

Enables or disables hardware interrupt pin

(uint8_t cmd, uint8_t ret) set_interrupt_global (bool arg)

attn	cmd	arg	LF
~	set_interrupt_global	bool	uint8_t

Example:

0x7e	0x16	0x01	0x0a

Notes:

arg = true or false = enable or disable hardware interrupt pin

Response:

cmd, ACK (0x06) or err_code

GET INTERRUPT GLOBAL

Returns setting (enabled or disabled) of hardware interrupt pin

(uint8_t cmd, uint8_t ret) get_interrupt_global (uint8_t cmd)

attn	cmd	LF
~	get_interrupt_global	uint8_t

Example:

0x7e	0x17	0x0a

Notes:

Get current global interrupt setting

Response:

Returns current global interrupt setting in *set_global_interrupt* format

SET INTERRUPT MASK

Sets enable or disable of radio module events that assert hardware interrupt pin

(uint8_t cmd, uint8_t ret) set_interrupt_mask (uint8_t cmd, uint8_t intr0, uint8_t intr1)

attn	cmd	arg0	arg1	LF
~	set_interrupt_mask	uint8_t	uint8_t	uint8_t

Example:

0x7e	0x18	0x06	0x00	0x0a

Notes:

Enables hardware interrupt for radio module events. bit mapped value = 0 thru 15.

bit Position (arg0)	Interrupt Event	bit Position (arg1)	Interrupt Event
0	RECEIVE_DATA	8	RESERVED
1	TRANSMIT_COMPLETE	9	RESERVED
2	RECEIVE_PREAMBLE_DETECT	10	RESERVED
3	RECEIVE_PREAMBLE_LOST	11	RESERVED
4	RECEIVE_SYNC_DETECT	12	RESERVED
5	RESERVED	13	RESERVED
6	RESERVED	14	RESERVED
7	RESERVED	15	RESERVED

Response:

cmd, ACK (0x06) or err_code

GET INTERRUPT MASK

Returns settings of radio module events that assert hardware interrupt pin

(uint8_t cmd, uint8_t ret) set_interrupt_mask (uint8_t cmd, uint8_t intr0, uint8_t intr1)

attn	cmd	LF
~	set_interrupt_mask	uint8_t

Example:

0x7e	0x19	0x0a

Notes:

Gets hardware interrupt mask settings for radio module events. bit mapped value = 0 thru 15.

Response:

Returns interrupt mask settings in *set_interrupt_mask* format

GET INTERRUPT

Returns radio module event(s) that asserted hardware interrupt pin

(uint8_t cmd, uint8_t intr0, uint8_t intr1) get_interrupt (uint8_t cmd)

attn	cmd	LF

~	get_interrupt	uint8_t

Example:

0x7e	0x20	0x0a

Notes:
Gets hardware interrupt flags asserted. Bit mapped value = 0 thru 15.

IMPORTANT: all interrupt flags are cleared after being read. If multiple interrupts are simultaneously asserted, ensure application handles all applicable interrupts.

Response:

Returns hardware interrupt value(s) in *set_interrupt_mask* format

SET_INTERRUPT_TIME

Sets time hardware interrupt pin will be asserted low on event. If set persistent, hardware interrupt pin will remain asserted low until *get_interrupt* is received.

(uint8_t cmd, uint_t ret) set_interrupt_time (uint8_t time)

attn	cmd	arg0
~	set_interrupt_time	uint8_t

Example:

0x7e	0x21	0x0a	0x0a

Notes:
arg0 = 0x01 thru 0xfe = assert LOW time in milliseconds. 0xff = persistent ON/OFF, 0x00 = invalid

Response:

cmd, ACK (0x06) or err_code

GET DEVICE UID

Returns radio module unique device ID

(uint8_t cmd, byte0… byte7) get_device_uid (uint8_t cmd)

attn	cmd	LF
~	get_device_uid	uint8_t

Example:

0x7e	0x22	0x0a

Notes:

Returns device unique ID in 8 byte payload. This value is factory lasered in the radio transceiver silicon and guaranteed unique within the scope of Aether RF radio modules.

Response Example:

0x7e	0x22	byte0 thru byte 7	0x0a

SOFT RESET

Performs radio module soft boot.

(uint8_t cmd, uint_t ret) soft_reset (uint8_t cmd)

attn	cmd	LF
~	soft_reset	uint8_t

Example:

0x7e	0x23	0x0a

Notes:

Soft Reboots radio module. Returns ACK when ready after boot sequence.

Response:
cmd, ACK (0x06) or err_code

SET SLEEP

Sets radio module lower power sleep mode.

(uint8_t cmd, uint_t ret) set_sleep_mode (uint8_t cmd)

attn	cmd	LF
~	set_sleep_mode	uint8_t

Example:

0x7e	0x24	0x0a

Notes:

Module can be awakened with any AURT command. Module will not receive radio commands in low power sleep mode.

Response:

cmd, ACK (0x06) or err_code

GET FIRMWARE VERSION

Gets radio module firmware version.

(uint8_t cmd, byte0... byte2) get_firmware_version (uint8_t cmd)

attn	cmd	LF
~	get_firmware_version	uint8_t

Example:

0x7e	0x25	0x0a

Notes:

Returns radio module firmware version in three bytes.

Response Example:

attn	cmd	arg0	arg1	arg2	LF
0x7e	0x25	byte0	byte1	byte2	0x0a

arg0 = major version number (example '1')

arg1 = minor version number (example '2')

arg3 = sub-minor (patch) version number (example '3')

11. Specifications

Dimensions	32mm(W) x 54.7mm(L) x 6mm(H)
Power Supply	+5 VDC @ TBD TX mA, RX mA
Low Power Sleep	TBD uA.
I/O Voltage	0 - 3.3 VDC
UART Interface	115,200, 8 bit, no parity, 1stop
900 MHz Output Power	+29.88 dBm
900 MHz Modulation/Protocol	Proprietary/Proprietary
900 MHz Connection	U.FL
Operating Temperature	TBD
Certifications	TBD

12. Resources

List development resources available.

radio_commands.h, includes for host application. Command and return code enums.

```
enum  EFR_COMMANDS
{
   // System Commands
   set_led = 1,
   set_gpio,
   get_gpio,
   set_interrupt_mask,
   get_interrupt_mask,
   ack, // 0x06
   get_interrupt,
   clear_interrupt,
   set_interrupt_global,
   no_lf, // 0x0a skip Line Feed
   get_interrupt_global,
   get_device_id,
   no_cr, // ox0d skip Carriage Return
   set_soft_reset,
   set_echo,
   set_sleep,
   get_fw_version,
   get_system_info,
   set_uart_ack,
```

```
  // Radio Commands
  set_freq_hopping,
  nack, // 0x21
  get_freq_hopping,
  get_hop_table,
  set_channel,
  get_channel,
  set_transmit,
  set_receive,
  set_tx_payload,
  get_tx_payload,
  get_rx_payload,
  set_aes_encrypt,
  get_aes_encrypt,
  set_aes_key,
  get_aes_key,
  get_radio_stats,
  clear_radio_stats,
  set_pa_enable,
  get_pa_enable,
  set_efr_power,
  get_efr_power,
  set_pa_bypass,
  get_pa_bypass,
};
```