

## StreamingTx Libraries

Generated by Doxygen 1.8.5

Wed Nov 29 2017 13:17:36



# Contents

<b>1</b>	<b>PCB layout from schematic.</b>	<b>1</b>
1.1	GPIO pins . . . . .	1
<b>2</b>	<b>format_string</b>	<b>3</b>
<b>3</b>	<b>Module Index</b>	<b>5</b>
3.1	Modules . . . . .	5
<b>4</b>	<b>Data Structure Index</b>	<b>7</b>
4.1	Data Structures . . . . .	7
<b>5</b>	<b>Module Documentation</b>	<b>9</b>
5.1	Analog to Digital Conversion . . . . .	9
5.1.1	Detailed Description . . . . .	9
5.1.2	Enumeration Type Documentation . . . . .	9
5.1.2.1	adc_channel . . . . .	9
5.1.3	Function Documentation . . . . .	9
5.1.3.1	adc_value . . . . .	9
5.2	Sound buzzer module . . . . .	11
5.2.1	Detailed Description . . . . .	11
5.2.2	Enumeration Type Documentation . . . . .	11
5.2.2.1	tune_index . . . . .	11
5.2.3	Function Documentation . . . . .	11
5.2.3.1	buzzer_tune . . . . .	11
5.3	Protocol logical channels . . . . .	12
5.3.1	Detailed Description . . . . .	12
5.3.2	Enumeration Type Documentation . . . . .	12
5.3.2.1	button_bits . . . . .	12
5.3.3	Function Documentation . . . . .	12
5.3.3.1	channel_value . . . . .	12
5.3.3.2	get_buttons . . . . .	13
5.4	Product configuration . . . . .	14
5.4.1	Detailed Description . . . . .	14

5.5	Cyclic Redundancy Check	15
5.5.1	Detailed Description	15
5.6	EEPROM reading/writing (NOT flash)	16
5.6.1	Detailed Description	16
5.6.2	Function Documentation	16
5.6.2.1	eeprom_read	16
5.6.2.2	eeprom_write	16
5.7	General Purpose Input/Output	17
5.7.1	Detailed Description	17
5.7.2	Enumeration Type Documentation	17
5.7.2.1	gpio_config	17
5.7.2.2	gpio_pins	18
5.7.3	Function Documentation	18
5.7.3.1	gpio_clear	18
5.7.3.2	gpio_config	18
5.7.3.3	gpio_get	19
5.7.3.4	gpio_set	19
5.7.3.5	gpio_toggle	19
5.8	SPI interface to radio chip	20
5.8.1	Detailed Description	20
5.8.2	Function Documentation	20
5.8.2.1	spi_force_chip_select	20
5.8.2.2	spi_read1	20
5.8.2.3	spi_read_registers	20
5.8.2.4	spi_transfer	20
5.8.2.5	spi_write	21
5.9	STM8 hardware interface	22
5.9.1	Detailed Description	22
5.10	Telemetry packet interface	23
5.10.1	Detailed Description	23
5.10.2	Enumeration Type Documentation	23
5.10.2.1	telem_type	23
5.10.2.2	tx_telem_type	23
5.11	Timer routines	24
5.11.1	Detailed Description	24
5.11.2	Function Documentation	24
5.11.2.1	timer_call_after_ms	24
5.11.2.2	timer_get_ms	24
5.11.2.3	timer_init	24
5.11.2.4	timer_irq	24

5.12	UART input/output	25
5.12.1	Detailed Description	25
5.13	Utility functions	26
5.13.1	Detailed Description	26
5.13.2	Function Documentation	26
5.13.2.1	chip_init	26
5.13.2.2	delay_ms	26
5.13.2.3	delay_us	27
5.13.2.4	printf	27
5.14	Beken BK2425 radio module	28
5.14.1	Detailed Description	31
5.14.2	Enumeration Type Documentation	31
5.14.2.1	anonymous enum	31
5.14.2.2	anonymous enum	31
5.14.2.3	BK_CONFIG_e	31
5.14.2.4	BK_FEATURE_e	32
5.14.2.5	BK_FIFO_STATUS_e	32
5.14.2.6	BK_PKT_TYPE_E	32
5.14.2.7	BK_SPI_CMD_e	32
5.14.2.8	BK_STATUS_e	33
5.14.2.9	CHANNEL_MHZ_e	34
5.14.2.10	ITX_SPEED_e	34
5.14.2.11	SPI_Flag_e	34
5.14.3	Function Documentation	35
5.14.3.1	beken_change_FCC_channel	35
5.14.3.2	beken_set_CW_mode	36
5.14.3.3	beken_start_factory_test	36
5.14.3.4	BK2425_Initialize	36
5.14.3.5	BK2425_SetSpeed	36
5.14.3.6	ChangeChannel	36
5.14.3.7	ChangeOutputPower	36
5.14.3.8	Get_Chip_ID	37
5.14.3.9	LookupChannel	37
5.14.3.10	NextChannelIndex	37
5.14.3.11	Receive_Packet	37
5.14.3.12	Send_Packet	37
5.14.3.13	SetChannelRange	38
5.14.3.14	SetRBank	38
5.14.3.15	SPI_Bank1_Read_Reg	38
5.14.3.16	SPI_Bank1_Write_Reg	38

5.14.3.17 SPI_Read_Reg . . . . .	38
5.14.3.18 SPI_Write_Buf . . . . .	39
5.14.3.19 SPI_Write_Cmd . . . . .	39
5.14.3.20 SPI_Write_Reg . . . . .	39
5.15 printf functions . . . . .	40
5.15.1 Detailed Description . . . . .	40
5.15.2 Function Documentation . . . . .	40
5.15.2.1 printf . . . . .	40
5.15.2.2 vprintf . . . . .	40
5.16 Main transmitter code . . . . .	41
5.16.1 Detailed Description . . . . .	41
5.16.2 Enumeration Type Documentation . . . . .	41
5.16.2.1 control_mode_t . . . . .	41
<b>6 Data Structure Documentation</b>	<b>43</b>
6.1 gpio_regs Struct Reference . . . . .	43
6.1.1 Detailed Description . . . . .	43
6.2 packetDataDevice_s Struct Reference . . . . .	43
6.2.1 Detailed Description . . . . .	44
6.3 packetDataDevice_s::packetDataDevice_u Union Reference . . . . .	44
6.3.1 Detailed Description . . . . .	44
6.4 packetDataDeviceCtrl_s Struct Reference . . . . .	44
6.4.1 Detailed Description . . . . .	45
6.5 packetDataDeviceID_s Struct Reference . . . . .	45
6.5.1 Detailed Description . . . . .	45
6.6 packetDataDrone_s Struct Reference . . . . .	45
6.6.1 Detailed Description . . . . .	46
6.7 telem_firmware Struct Reference . . . . .	46
6.7.1 Detailed Description . . . . .	46
6.8 telem_packet Struct Reference . . . . .	46
6.8.1 Detailed Description . . . . .	46
6.9 telem_play Struct Reference . . . . .	46
6.9.1 Detailed Description . . . . .	46
6.10 telem_status Struct Reference . . . . .	47
6.10.1 Detailed Description . . . . .	47
6.11 telem_tx_status Struct Reference . . . . .	47
6.11.1 Detailed Description . . . . .	48
<b>Index</b>	<b>49</b>

# Chapter 1

## PCB layout from schematic.

The schematic "Streaming and Streaming with GPS Drone button board" v0.1 says

### 1.1 GPIO pins

Port	Meaning
A1	BUTTON_STUNT
A2	BUTTON_VIDEO
B0	CH4 = ROLL (mode2) RightHorizontal
B1	CH3 = PITCH (mode2) RightVertical
B2	CH1 = THROTTLE (mode2) LeftVertical
B3	CH2 = YAW (mode2) LeftHorizontal
B4	PWR
B5	RADIO_PACTL
C1	BUTTON_GPS
C2	USER
C3	RADIO_IRQ
C4	RADIO_CS
C5	RADIO_SCK
C6	RADIO_MOSI
C7	RADIO_MISO
D0	BUTTON_MODE
D1	SWIM
D2	RADIO_CE
D3	LED_GPS
D4	BEEP
D5	UART_TX
D6	UART_RX
D7	LED_MODE
E5	BUTTON_LL
F4	VBAT_SENSE





## Chapter 2

# format\_string

The following formats are supported by printf

format	output type	argument-type
%d	decimal	int
%ld	decimal	long
%hd	decimal	char
%u	decimal	unsigned int
%lu	decimal	unsigned long
%hu	decimal	unsigned char
%x	hexadecimal	int
%lx	hexadecimal	long
%hx	hexadecimal	char
%o	octal	int
%lo	octal	long
%ho	octal	char
%c	character	char
%s	character	generic pointer



## Chapter 3

# Module Index

### 3.1 Modules

Here is a list of all modules:

Analog to Digital Conversion . . . . .	9
Sound buzzer module . . . . .	11
Protocol logical channels . . . . .	12
Product configuration . . . . .	14
Cyclic Redundancy Check . . . . .	15
EEPROM reading/writing (NOT flash) . . . . .	16
General Purpose Input/Output . . . . .	17
SPI interface to radio chip . . . . .	20
STM8 hardware interface . . . . .	22
Telemetry packet interface . . . . .	23
Timer routines . . . . .	24
UART input/output . . . . .	25
Utility functions . . . . .	26
Beken BK2425 radio module . . . . .	28
printf functions . . . . .	40
Main transmitter code . . . . .	41



## Chapter 4

# Data Structure Index

### 4.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">gpio_regs</a>	Declaration of how the hardware is laid out on STM8 processors (e.g . . . . .	43
<a href="#">packetDataDevice_s</a>	Data structure for data packet transmitted from device (controller) to host (drone) . . . . .	43
<a href="#">packetDataDevice_s::packetDataDevice_u</a>	< The variant part of the packets . . . . .	44
<a href="#">packetDataDeviceCtrl_s</a>	Data for packets that are not droneid packets Onair order = little-endian . . . . .	44
<a href="#">packetDataDeviceID_s</a>	Data for packets that are binding packets Onair order = little-endian . . . . .	45
<a href="#">packetDataDrone_s</a>	Data structure for data packet transmitted from host (drone) to device (controller) . . . . .	45
<a href="#">telem_firmware</a>	Telemetry packet for the command to write to new firmware . . . . .	46
<a href="#">telem_packet</a>	Telemetry packet from RX to TX . . . . .	46
<a href="#">telem_play</a>	Telemetry packet for the command to play a tune . . . . .	46
<a href="#">telem_status</a>	Telemetry status packet . . . . .	47
<a href="#">telem_tx_status</a>	Tx_status structure sent one byte at a time to RX . . . . .	47



## Chapter 5

# Module Documentation

### 5.1 Analog to Digital Conversion

#### Enumerations

- enum `adc_channel` { `STICK_ROLL` = 1, `STICK_PITCH` = 0, `STICK_THROTTLE` = 3, `STICK_YAW` = 2 }
- The meaning of each analog channel, assuming mode2 stick mapping.*

#### Functions

- void `adc_init` (void)  
*This function initialises the ADC module.*
- uint16\_t `adc_value` (uint8\_t chan)  
*This function returns the most recently converted data from a specified channel.*
- void `adc_irq` (void)  
*This is the interrupt routine for supporting ADC conversions.*

#### 5.1.1 Detailed Description

#### 5.1.2 Enumeration Type Documentation

##### 5.1.2.1 enum `adc_channel`

The meaning of each analog channel, assuming mode2 stick mapping.

#### Enumerator

- `STICK_ROLL`** Right joystick horizontal axis.
- `STICK_PITCH`** Right joystick vertical axis.
- `STICK_THROTTLE`** Left joystick vertical axis.
- `STICK_YAW`** Left joystick horizontal axis.

#### 5.1.3 Function Documentation

##### 5.1.3.1 uint16\_t `adc_value` ( uint8\_t *chan* )

This function returns the most recently converted data from a specified channel.

**Returns**

Returns the raw input value (not normalised).

**Parameters**

<i>chan</i>	Which channel are we interested in now. See <a href="#">adc_channel</a>
-------------	---



## 5.2 Sound buzzer module

### Enumerations

- enum [tune\\_index](#)  
*The index into the tune table.*

### Functions

- void [buzzer\\_init](#) (void)  
*Initialise the sound buzzer module.*
- void [buzzer\\_tune](#) (uint8\_t t)  
*Start playing the given tune number.*

#### 5.2.1 Detailed Description

#### 5.2.2 Enumeration Type Documentation

##### 5.2.2.1 enum [tune\\_index](#)

The index into the tune table.

#### 5.2.3 Function Documentation

##### 5.2.3.1 void [buzzer\\_tune](#) ( uint8\_t t )

Start playing the given tune number.

Only one tune can be played at a time

##### Parameters

<i>t</i>	The tune number. See <a href="#">tune_index</a>
----------	---

## 5.3 Protocol logical channels

Support radio protocol logical channels.

### Enumerations

- enum `button_bits` {  
`BUTTON_LEFT` = 0x01, `BUTTON_RIGHT` = 0x02, `BUTTON_LEFT_SHOULDER` = 0x04, `BUTTON_RIGHT_SHOULDER` = 0x08,  
`BUTTON_POWER` = 0x10, `BUTTON_MODE` = 0x20 }

*A bitset of the buttons on this controller.*

### Functions

- uint16\_t `channel_value` (uint8\_t chan)  
*Lookup a channel value required by the radio protocol.*
- uint8\_t `get_buttons` (void)  
*Return a byte that contains a bitset of pressed buttons.*

#### 5.3.1 Detailed Description

Support radio protocol logical channels.

#### 5.3.2 Enumeration Type Documentation

##### 5.3.2.1 enum button\_bits

A bitset of the buttons on this controller.

##### Enumerator

**`BUTTON_LEFT`** The left button.  
**`BUTTON_RIGHT`** The right button.  
**`BUTTON_LEFT_SHOULDER`** The left shoulder button.  
**`BUTTON_RIGHT_SHOULDER`** The right shoulder button.  
**`BUTTON_POWER`** The POWER button.  
**`BUTTON_MODE`** The MODE button.

#### 5.3.3 Function Documentation

##### 5.3.3.1 uint16\_t channel\_value ( uint8\_t chan )

Lookup a channel value required by the radio protocol.

##### Returns

An 11 bit channel output value.

## Parameters

<i>chan</i>	The index into the protocol channel
-------------	-------------------------------------

5.3.3.2 `uint8_t get_buttons ( void )`

Return a byte that contains a bitset of pressed buttons.

## Returns

[`button\_bits`](#) The union of all the currently pressed buttons, sampled right now.

## 5.4 Product configuration

### 5.4.1 Detailed Description

## 5.5 Cyclic Redundancy Check

Support calculating CRCs.

### Functions

- `uint8_t crc\_crc8` (`const uint8_t *p`, `uint16_t len`)  
*8-bit crc*
- `uint32_t crc\_crc32` (`const uint8_t *p`, `uint16_t len`)  
*a poor-mans crc32, re-using the crc16 table*

### 5.5.1 Detailed Description

Support calculating CRCs.

## 5.6 EEPROM reading/writing (NOT flash)

Support the rewritable EEPROM on the CPU (it has many more erase cycles than the flash)

### Functions

- void `eeeprom_write` (uint16\_t offset,uint8\_t value)  
*Write a byte to the EEPROM (must be unlocked)*
- uint8\_t `eeeprom_read` (uint16\_t offset)  
*Read a byte from the EEPROM - just uses normal address space.*
- void `eeeprom_unlock` (void)  
*Unlock the EEPROM memory before writing.*
- void `progmem_unlock` (void)  
*Unlock the program memory before writing.*
- void `eeeprom_lock` (void)  
*Lock the EEPROM memory after writing.*

### 5.6.1 Detailed Description

Support the rewritable EEPROM on the CPU (it has many more erase cycles than the flash)

### 5.6.2 Function Documentation

#### 5.6.2.1 uint8\_t eeeprom\_read ( uint16\_t offset )

Read a byte from the EEPROM - just uses normal address space.

#### Returns

The byte at that offset in the EEPROM

#### Parameters

<i>offset</i>	The offset of the data within EEPROM
---------------	--------------------------------------

#### 5.6.2.2 void eeeprom\_write ( uint16\_t offset, uint8\_t value )

Write a byte to the EEPROM (must be unlocked)

#### Parameters

<i>offset</i>	The offset of the data within EEPROM
<i>value</i>	The byte to write

## 5.7 General Purpose Input/Output

Support raw GPIO access.

### Data Structures

- struct `gpio_regs`

*Declaration of how the hardware is laid out on STM8 processors (e.g.*

### Enumerations

- enum `gpio_pins` {  
`GPIO_PORTA` = 0x000, `GPIO_PORTB` = 0x100, `GPIO_PORTC` = 0x200, `GPIO_PORTD` = 0x300,  
`GPIO_PORTE` = 0x400, `GPIO_PORTF` = 0x500, `GPIO_PORTG` = 0x600, `GPIO_PORTH` = 0x700,  
`GPIO_PORTI` = 0x800, `GPIO_PIN0` = (1 << 0), `GPIO_PIN1` = (1 << 1), `GPIO_PIN2` = (1 << 2),  
`GPIO_PIN3` = (1 << 3), `GPIO_PIN4` = (1 << 4), `GPIO_PIN5` = (1 << 5), `GPIO_PIN6` = (1 << 6),  
`GPIO_PIN7` = (1 << 7) }

*Definition of ports; one of these can be ored with one or more pin bits to refer to a collection of pins on a single port.*

- enum `gpio_config` {  
`GPIO_INPUT_FLOAT` = 0x0, `GPIO_INPUT_PULLUP` = 0x2, `GPIO_INPUT_FLOAT_IRQ` = 0x1, `GPIO_INPU-`  
`T_PULLUP_IRQ` = 0x3,  
`GPIO_OUTPUT_OPEN_DRAIN` = 0x0, `GPIO_OUTPUT_PUSHPULL` = 0x6, `GPIO_OUTPUT_OPEN_DRAIN-`  
`_FAST` = 0x5, `GPIO_OUTPUT_PUSHPULL_FAST` = 0x7,  
`GPIO_SET` = 0x10, `GPIO_CLEAR` = 0x20 }

*Configuration values, for `gpio_config`.*

### Functions

- void `gpio_config` (uint16\_t pins, enum `gpio_config` config)  
*Configure one or more pins on a port.*
- void `gpio_set` (uint16\_t pins)  
*Set one or more pins on a port high.*
- void `gpio_clear` (uint16\_t pins)  
*Set one or more pins on a port low.*
- void `gpio_toggle` (uint16\_t pins)  
*Toggle one or more pins on a port between high and low.*
- bool `gpio_get` (uint16\_t pin)  
*Get the current state of an input pin.*

#### 5.7.1 Detailed Description

Support raw GPIO access. This module is for configuring and using GPIO pins directly within the project.

#### 5.7.2 Enumeration Type Documentation

##### 5.7.2.1 enum `gpio_config`

Configuration values, for `gpio_config`.

#### Enumerator

**`GPIO_INPUT_FLOAT`** Input pin with no pullup.

***GPIO\_INPUT\_PULLUP*** Input pin with internal pullup resistor active.

***GPIO\_INPUT\_FLOAT\_IRQ*** Input pin with no pullup; generates IRQ.

***GPIO\_INPUT\_PULLUP\_IRQ*** Input pin with internal pullup resistor active; generates IRQ.

***GPIO\_OUTPUT\_OPEN\_DRAIN*** Output pin as open drain.

***GPIO\_OUTPUT\_PUSHPULL*** Output pin as push pull.

***GPIO\_OUTPUT\_OPEN\_DRAIN\_FAST*** Output pin as open drain with fast response.

***GPIO\_OUTPUT\_PUSHPULL\_FAST*** Output pin as push pull with fast response.

***GPIO\_SET*** Flag to set a GPIO.

***GPIO\_CLEAR*** Flag to clear a GPIO.

#### 5.7.2.2 enum gpio\_pins

Definition of ports; one of these can be ored with one or more pin bits to refer to a collection of pins on a single port.

Enumerator

***GPIO\_PORTA*** Port A.

***GPIO\_PORTB*** Port B.

***GPIO\_PORTC*** Port C.

***GPIO\_PORTD*** Port D.

***GPIO\_PORTE*** Port E.

***GPIO\_PORTF*** Port F.

***GPIO\_PORTG*** Port G.

***GPIO\_PORTH*** Port H.

***GPIO\_PORTI*** Port I.

***GPIO\_PIN0*** Pin 0 of a port.

***GPIO\_PIN1*** Pin 1 of a port.

***GPIO\_PIN2*** Pin 2 of a port.

***GPIO\_PIN3*** Pin 3 of a port.

***GPIO\_PIN4*** Pin 4 of a port.

***GPIO\_PIN5*** Pin 5 of a port.

***GPIO\_PIN6*** Pin 6 of a port.

***GPIO\_PIN7*** Pin 7 of a port.

### 5.7.3 Function Documentation

#### 5.7.3.1 void gpio\_clear ( uint16\_t pins )

Set one or more pins on a port low.

Assumes the port is configured for output.

Parameters

<i>pins</i>	One or more pins to set low on a single specified GPIO port. See <a href="#">gpio_pins</a>
-------------	--

#### 5.7.3.2 void gpio\_config ( uint16\_t pins, enum gpio\_config config )

Configure one or more pins on a port.



## Parameters

<i>pins</i>	One or more pins to configure on a single specified GPIO port. See <a href="#">gpio_pins</a>
<i>config</i>	The configuration format wanted for the specified pin(s)

**5.7.3.3** `bool gpio_get ( uint16_t pin )`

Get the current state of an input pin.

Assumes the port is configured for digital input.

## Returns

true if at least one specified GPIO pin is high (false if all are low).

## Parameters

<i>pin</i>	One or more pins to test on a single specified GPIO port. See <a href="#">gpio_pins</a>
------------	---

**5.7.3.4** `void gpio_set ( uint16_t pins )`

Set one or more pins on a port high.

Assumes the port is configured for output.

## Parameters

<i>pins</i>	One or more pins to set high on a single specified GPIO port. See <a href="#">gpio_pins</a>
-------------	---

**5.7.3.5** `void gpio_toggle ( uint16_t pins )`

Toggle one or more pins on a port between high and low.

Assumes the port is configured for output.

## Parameters

<i>pins</i>	One or more pins to toggle between high and low on a single specified GPIO port. See <a href="#">gpio_pins</a>
-------------	--

## 5.8 SPI interface to radio chip

### Functions

- void `spi_init` (void)  
*Initialise the SPI interface to the radio chip.*
- void `spi_write` (uint8\_t n,const uint8\_t \*buf)  
*Write an array of bytes to the SPI interface and ignore the read array.*
- uint8\_t `spi_read1` (void)  
*Read one byte from the SPI interface, writing 0 to it.*
- void `spi_transfer` (uint8\_t n,const uint8\_t \*sendbuf,uint8\_t \*recvbuf)  
*Transfer two arrays of bytes in both directions over the SPI interface.*
- void `spi_force_chip_select` (bool set)  
*Set or clear the chip select of the radio chip, but only once.*
- void `spi_read_registers` (uint8\_t reg,uint8\_t \*buf,uint8\_t len)  
*Read data from the SPI chip, using a 'register' to specify which data.*

### 5.8.1 Detailed Description

### 5.8.2 Function Documentation

#### 5.8.2.1 void spi\_force\_chip\_select ( bool set )

Set or clear the chip select of the radio chip, but only once.

##### Parameters

<i>set</i>	True on set, False on clear
------------	-----------------------------

#### 5.8.2.2 uint8\_t spi\_read1 ( void )

Read one byte from the SPI interface, writing 0 to it.

##### Returns

Returns the input byte.

#### 5.8.2.3 void spi\_read\_registers ( uint8\_t reg, uint8\_t \* buf, uint8\_t len )

Read data from the SPI chip, using a 'register' to specify which data.

##### Parameters

<i>reg</i>	The index of the 'register' on the SPI chip to read. Sent before reading the buffer.
<i>buf</i>	The buffer of bytes to read (must be at least len bytes in size).
<i>len</i>	The number of bytes to read in one transaction

#### 5.8.2.4 void spi\_transfer ( uint8\_t n, const uint8\_t \* sendbuf, uint8\_t \* recvbuf )

Transfer two arrays of bytes in both directions over the SPI interface.

## Parameters

<i>n</i>	The number of bytes to transfer in each direction over the SPI interface.
<i>sendbuf</i>	The array of bytes to write. If NULL then bytes of value 0 are sent.
<i>recvbuf</i>	A buffer array of bytes to store the data read from the SPI interface. If NULL then the read bytes are discarded.

5.8.2.5 void spi\_write ( uint8\_t *n*, const uint8\_t \* *buf* )

Write an array of bytes to the SPI interface and ignore the read array.

## Parameters

<i>n</i>	The number of bytes to write
<i>buf</i>	A pointer to the array of bytes to write

## 5.9 STM8 hardware interface

### 5.9.1 Detailed Description

## 5.10 Telemetry packet interface

### Data Structures

- struct [telem\\_status](#)  
*Telemetry status packet.*
- struct [telem\\_play](#)  
*Telemetry packet for the command to play a tune.*
- struct [telem\\_firmware](#)  
*Telemetry packet for the command to write to new firmware.*
- struct [telem\\_packet](#)  
*telemetry packet from RX to TX*
- struct [telem\\_tx\\_status](#)  
*tx\_status structure sent one byte at a time to RX.*

### Enumerations

- enum [telem\\_type](#) { [TELEM\\_STATUS](#) = 0, [TELEM\\_PLAY](#) = 1, [TELEM\\_FW](#) = 2 }  
*The type of telemetry packet.*
- enum [tx\\_telem\\_type](#) { [TXTELEM\\_RSSI](#) = 0, [TXTELEM\\_CRC1](#) = 1, [TXTELEM\\_CRC2](#) = 2 }  
*Type of telemetry data.*

#### 5.10.1 Detailed Description

#### 5.10.2 Enumeration Type Documentation

##### 5.10.2.1 enum telem\_type

The type of telemetry packet.

##### Enumerator

**TELEM\_STATUS** a [telem\\_status](#) packet  
**TELEM\_PLAY** command to play a tune  
**TELEM\_FW** command to update new firmware

##### 5.10.2.2 enum tx\_telem\_type

Type of telemetry data.

##### Enumerator

**TXTELEM\_RSSI** The data word is the RSSI.  
**TXTELEM\_CRC1** The data word is part 1 of the CRC.  
**TXTELEM\_CRC2** The data word is part 2 of the CRC.

## 5.11 Timer routines

### Functions

- void `timer_init` (void)  
*Initialise the 1ms timer on timer4.*
- void `timer_irq` (void)  
*The interrupt function for the timer IRQ.*
- uint32\_t `timer_get_ms` (void)  
*Get the current time since bootup.*
- void `timer_call_after_ms` (uint16\_t dt\_ms, timer\_callback\_t callback)  
*Request a callback after a number of milliseconds.*
- void `timer_delay_ms` (uint16\_t ms)  
*Busy loop to delay for some milliseconds, using the timer for accuracy.*

### 5.11.1 Detailed Description

### 5.11.2 Function Documentation

#### 5.11.2.1 void timer\_call\_after\_ms ( uint16\_t dt\_ms, timer\_callback\_t callback )

Request a callback after a number of milliseconds.

Only one callback can be active at a time.

#### Parameters

<i>dt_ms</i>	The time of the requested callback, in milliseconds
<i>callback</i>	The function to be called

#### 5.11.2.2 uint32\_t timer\_get\_ms ( void )

Get the current time since bootup.

#### Returns

Returns the number of milliseconds since bootup.

#### 5.11.2.3 void timer\_init ( void )

Initialise the 1ms timer on timer4.

#### 5.11.2.4 void timer\_irq ( void )

The interrupt function for the timer IRQ.

This is for Timer4

## 5.12 UART input/output

### Functions

- void `uart2_init` (void)  
*Initialise UART2 for output debugging.*
- void `uart2_write` (const char \*str)  
*Output a nul-terminated string to UART2.*
- void `uart2_putchar` (char c)  
*Output a single character to UART2.*

### 5.12.1 Detailed Description

## 5.13 Utility functions

Support utility functions such as chip setup, LED, timing and maths.

### Functions

- void `chip_init` (void)  
*Initialise the chip and PCB.*
- void `led_init` (void)  
*Initialise the LEDs.*
- void `led_green_set` (bool set)  
*Turn the green LED on or off as specified.*
- void `led_yellow_set` (bool set)  
*Turn the yellow LED on or off as specified.*
- void `led_green_toggle` (void)  
*Toggle the green LED on or off.*
- void `led_yellow_toggle` (void)  
*Toggle the yellow LED on or off.*
- void `delay_ms` (uint16\_t d)  
*Busy loop to wait a number of milliseconds (up to about 65 seconds) (empirically tuned on one CPU)  
The scale factor is precise to <1% accuracy if it is accurate*
- void `delay_us` (uint16\_t d)  
*Busy loop to wait a number of microseconds (up to about 65ms) (empirically tuned on one CPU)  
Only vaguely accurate since scale factor has no bits of resolution.*
- uint16\_t `get_random16` (void)  
*Simple 16 bit random number generator.*
- void `printf` (const char \*fmt,...)  
*Small implementation of the standard printf routine.*

### 5.13.1 Detailed Description

Support utility functions such as chip setup, LED, timing and maths.

### 5.13.2 Function Documentation

#### 5.13.2.1 void `chip_init` ( void )

Initialise the chip and PCB.

This function is specific to the hardware layout

#### 5.13.2.2 void `delay_ms` ( uint16\_t d )

Busy loop to wait a number of milliseconds (up to about 65 seconds) (empirically tuned on one CPU)

The scale factor is precise to <1% accuracy if it is accurate

#### Parameters

---



<i>d</i>	The number of milliseconds to wait
----------	------------------------------------

#### 5.13.2.3 void delay\_us ( uint16\_t *d* )

Busy loop to wait a number of microseconds (up to about 65ms) (empirically tuned on one CPU)

Only vaguely accurate since scale factor has no bits of resolution.

##### Parameters

<i>d</i>	The number of microseconds to wait
----------	------------------------------------

#### 5.13.2.4 void printf ( const char \* *fmt*, ... )

Small implementation of the standard printf routine.

##### Parameters

<i>fmt</i>	The format string. <a href="#">format_string</a>
------------	--

## 5.14 Beken BK2425 radio module

### Data Structures

- struct [packetDataDeviceCtrl\\_s](#)  
*Data for packets that are not droneid packets Onair order = little-endian.*
- struct [packetDataDeviceID\\_s](#)  
*Data for packets that are binding packets Onair order = little-endian.*
- struct [packetDataDevice\\_s](#)  
*Data structure for data packet transmitted from device (controller) to host (drone)*
- struct [packetDataDrone\\_s](#)  
*Data structure for data packet transmitted from host (drone) to device (controller)*

### Typedefs

- typedef struct [packetDataDeviceCtrl\\_s](#) packetDataDeviceCtrl  
*Data for packets that are not droneid packets Onair order = little-endian.*
- typedef struct [packetDataDeviceID\\_s](#) packetDataDeviceID  
*Data for packets that are binding packets Onair order = little-endian.*
- typedef struct [packetDataDevice\\_s](#) packetFormatTx  
*Data structure for data packet transmitted from device (controller) to host (drone)*
- typedef struct [packetDataDrone\\_s](#) packetFormatRx  
*Data structure for data packet transmitted from host (drone) to device (controller)*
- typedef enum [ITX\\_SPEED\\_e](#) ITX\_SPEED  
*The baud rate of the GFSK modulation.*
- typedef enum [SPI\\_Flag\\_e](#) SPI\_Flag\_TypeDef  
*Flags for the STM8 hardware SPI registers.*
- typedef enum [BK\\_SPI\\_CMD\\_e](#) BK\_SPI\_CMD  
*SPI register commands for the BK2425 and nrf24L01+ chips.*

### Enumerations

- enum [BK\\_PKT\\_TYPE\\_E](#) {  
  [BK\\_PKT\\_TYPE\\_INVALID](#) = 0, [BK\\_PKT\\_TYPE\\_CTRL](#) = 0x10, [BK\\_PKT\\_TYPE\\_AVAILABLE](#) = 0x11, [BK\\_PKT\\_TYPE\\_DISCONNECTED](#) = 0x12,  
  [BK\\_PKT\\_TYPE\\_PAIRING](#) = 0x13, [BK\\_PKT\\_TYPE\\_DRONE](#) = 0x14 }  
*The type of packets being sent between controller and drone.*
- enum { [COMMS\\_STATE\\_UNPAIRED](#), [COMMS\\_STATE\\_DISCONNECTED](#), [COMMS\\_STATE\\_PAIRING](#), [COMMS\\_STATE\\_PAIRED](#) }  
*Comms state.*
- enum [CHANNEL\\_MHZ\\_e](#) {  
  [CHANNEL\\_MIN\\_PHYSICAL](#) = 0, [CHANNEL\\_MAX\\_PHYSICAL](#) = 83, [CHANNEL\\_FCC\\_LOW](#) = 10, [CHANNEL\\_FCC\\_HIGH](#) = 72,  
  [CHANNEL\\_FCC\\_MID](#) = 41, [CHANNEL\\_TEST\\_MODE](#) = 41 }  
*Channel hopping parameters.*
- enum [ITX\\_SPEED\\_e](#) { [ITX\\_250](#), [ITX\\_1000](#), [ITX\\_2000](#) }  
*The baud rate of the GFSK modulation.*
- enum [SPI\\_Flag\\_e](#) {  
  [SPI\\_FLAG\\_BSY](#) = (uint8\_t)0x80, [SPI\\_FLAG\\_OVR](#) = (uint8\_t)0x40, [SPI\\_FLAG\\_MODF](#) = (uint8\_t)0x20, [SPI\\_FLAG\\_CRCERR](#) = (uint8\_t)0x10,  
  [SPI\\_FLAG\\_WKUP](#) = (uint8\_t)0x08, [SPI\\_FLAG\\_TXE](#) = (uint8\_t)0x02, [SPI\\_FLAG\\_RXNE](#) = (uint8\_t)0x01 }

*Flags for the STM8 hardware SPI registers.*

- enum `BK_SPI_CMD_e` {  
`BK_REG_MASK` = 0x1F, `BK_READ_REG` = 0x00, `BK_WRITE_REG` = 0x20, `BK_RD_RX_PLOAD` = 0x61,  
`BK_WR_TX_PLOAD` = 0xA0, `BK_W_ACK_PAYLOAD_CMD` = 0xA8, `BK_FLUSH_TX` = 0xE1, `BK_FLUSH_RX` = 0xE2,  
`BK_REUSE_TX_PL` = 0xE3, `BK_NOP` = 0xFF, `BK_CONFIG` = 0x00, `BK_EN_AA` = 0x01,  
`BK_EN_RXADDR` = 0x02, `BK_SETUP_AW` = 0x03, `BK_SETUP_RETR` = 0x04, `BK_RF_CH` = 0x05,  
`BK_RF_SETUP` = 0x06, `BK_STATUS` = 0x07, `BK_OBSERVE_TX` = 0x08, `BK_CD` = 0x09,  
`BK_RX_ADDR_P0` = 0x0A, `BK_RX_ADDR_P1` = 0x0B, `BK_RX_ADDR_P2` = 0x0C, `BK_RX_ADDR_P3` = 0x0D,  
`BK_RX_ADDR_P4` = 0x0E, `BK_RX_ADDR_P5` = 0x0F, `BK_TX_ADDR` = 0x10, `BK_RX_PW_P0` = 0x11,  
`BK_RX_PW_P1` = 0x12, `BK_RX_PW_P2` = 0x13, `BK_RX_PW_P3` = 0x14, `BK_RX_PW_P4` = 0x15,  
`BK_RX_PW_P5` = 0x16, `BK_FIFO_STATUS` = 0x17, `BK_DYNPD` = 0x1C, `BK_FEATURE` = 0x1D,  
`BK_PAYLOAD_WIDTH` = 0x1F, `BK2425_R1_WHOAMI` = 0x08, `BK2425_R1_12` = 0x0C }

*SPI register commands for the BK2425 and nrf24L01+ chips.*

- enum { `BK_CHIP_ID_BK2425` = 0x63 }
- enum `BK_STATUS_e` {  
`BK_STATUS_RBANK` = 0x80, `BK_STATUS_RX_DR` = 0x40, `BK_STATUS_TX_DS` = 0x20, `BK_STATUS_MAX_RT` = 0x10,  
`BK_STATUS_RX_MASK` = 0x0E, `BK_STATUS_RX_P_5` = 0x0A, `BK_STATUS_RX_P_4` = 0x08, `BK_STATUS_RX_P_3` = 0x06,  
`BK_STATUS_RX_P_2` = 0x04, `BK_STATUS_RX_P_1` = 0x02, `BK_STATUS_RX_P_0` = 0x00, `BK_STATUS_TX_FULL` = 0x01 }

*Meanings of the BK\_STATUS register.*

- enum `BK_FIFO_STATUS_e` { , `BK_FIFO_STATUS_TX_FULL` = 0x20, `BK_FIFO_STATUS_TX_EMPTY` = 0x10, `BK_FIFO_STATUS_RX_FULL` = 0x02, `BK_FIFO_STATUS_RX_EMPTY` = 0x01 }

*Meanings of the FIFO\_STATUS register.*

- enum `BK_CONFIG_e` {  
`BK_CONFIG_MASK_RX_DR` = 0x40, `BK_CONFIG_MASK_TX_DS` = 0x20, `BK_CONFIG_MASK_MAX_RT` = 0x10, `BK_CONFIG_EN_CRC` = 0x08,  
`BK_CONFIG_CRCO` = 0x04, `BK_CONFIG_PWR_UP` = 0x02, `BK_CONFIG_PRIM_RX` = 0x01 }

*Meanings of the BK\_CONFIG register.*

- enum `BK_FEATURE_e` { `BK_FEATURE_EN_DPL` = 0x04 }

*Meanings of the BK\_FEATURE register.*

## Functions

- void `SPI_Write_Cmd` (uint8\_t reg)  
*Write a single byte command to the SPI bus (e.g.*
- void `SPI_Write_Reg` (uint8\_t reg, uint8\_t value)  
*Writes value 'value' to register 'reg'.*
- uint8\_t `SPI_Read_Status` (void)  
*Read the status from the BK2425.*
- uint8\_t `SPI_Read_Reg` (uint8\_t reg)  
*Read one uint8\_t from BK2425 register 'reg' via SPI.*
- void `SPI_Write_Buf` (uint8\_t reg, const uint8\_t \*pBuf, uint8\_t length)  
*Writes contents of a buffer to BK2425 via SPI.*
- void `SwitchToRxMode` (void)  
*Switch the Beken radio to Rx mode.*
- void `SwitchToTxMode` (void)  
*Switch the Beken radio to Tx mode.*
- void `SwitchToIdleMode` (void)  
*Switch the Beken radio to Idle mode.*

- void [SwitchToSleepMode](#) (void)  
*Switch the Beken radio to Sleep mode.*
- void [SetRBank](#) (char \_cfg)  
*Set which register bank we are accessing on the Beken spi chip.*
- int [BK2425\\_GetSpeed](#) (void)  
*Return the current speed in kbps.*
- void [BK2425\\_Initialize](#) (ITX\_SPEED spd)  
*BK2425 initialization of radio registers.*
- void [BK2425\\_SetSpeed](#) (bool bFast)  
*Change between 250kbps and 2000kbps on the fly.*
- void [SPI\\_Bank1\\_Write\\_Reg](#) (uint8\_t reg, const uint8\_t \*pBuf)  
*Write a 32-bit Bank1 register.*
- void [SPI\\_Bank1\\_Read\\_Reg](#) (uint8\_t reg, uint8\_t \*pBuf)  
*Read a 32-bit Bank1 register.*
- void [ChangeChannel](#) (uint8\_t channelNumber)  
*Change the radio channel.*
- void [initBeken](#) (void)  
*Initialise the Beken chip ready to be talked to.*
- void [deinitBeken](#) (void)  
*DeInitialise the Beken chip after talking.*
- void [describeBeken](#) (void)  
*Describe our transmission parameters to the serial port for verification by the tester.*
- void [ChangeAddress](#) (PAIRADDR tmpaddress, uint8\_t rxch)  
*Change pipeline address.*
- void [ChangeAddressTx](#) (PAIRADDR tmpaddress, uint8\_t txch)  
*Change address.*
- void [ChangeOutputPower](#) (uint8\_t power)  
*Change the radio output power of the Beken radio chip.*
- void [IWDG\\_Kick](#) (void)  
*Kick the independant windowed watchdog so that it does not reset the CPU by timing out.*
- bool [Send\\_Packet](#) (uint8\_t type, const uint8\_t \*pbuf, uint8\_t len)  
*Fill the Bekens tx FIFO to send a packet.*
- uint8\_t [Receive\\_Packet](#) (uint8\_t rx\_buf[])  
*Read FIFO to read a packet.*
- void [FlushTx](#) (void)  
*Flush the Beken radio TX buffer.*
- uint8\_t [Get\\_Chip\\_ID](#) (void)  
*Get the Beken radio chip ID.*
- void [VerifyBekenChipID](#) (void)  
*Ensure that the chip id is good.*
- bool [SetChannelRange](#) (uint8\_t min, uint8\_t max)  
*Set the range of the channel indexes we are using.*
- uint8\_t [LookupChannel](#) (uint8\_t idx)  
*Convert a logical channel index into a physical channel.*
- uint8\_t [NextChannelIndex](#) (uint8\_t seq)  
*Channel hopping algorithm implementation.*
- void [beken\\_init](#) (void)  
*Initialise the Beken radio chip.*
- void [beken\\_irq](#) (void)  
*The IRQ routine that needs to be called on radio interrupts for the Beken chip.*
- void [beken\\_timer\\_irq](#) (void)

*The IRQ routine that needs to be called on timer interrupts for the Beken chip.*

- void `beken_start_bind_send` (void)  
*Start sending a binding packet.*
- void `beken_start_send` (void)  
*Start sending a control data packet.*
- void `beken_start_FCC_test` (void)  
*Start sending an FCC test packet.*
- void `beken_start_factory_test` (uint8\_t test\_mode)  
*Start sending an factory test packet.*
- void `beken_next_FCC_power` (void)  
*Set the next FCC power.*
- void `beken_set_CW_mode` (bool cw)  
*Go into continuous carrier wave send mode or normal mode.*
- void `beken_change_FCC_channel` (int8\_t change)  
*Change the FCC channel.*
- void `beken_FCC_toggle_scan` (void)  
*Toggle the FCC scan.*
- uint8\_t `get_tx_power` (void)  
*Get the current tx power.*
- int8\_t `get_FCC_chan` (void)  
*Get the current FCC channel.*
- uint8\_t `get_FCC_power` (void)  
*Get the current FCC power.*

### 5.14.1 Detailed Description

### 5.14.2 Enumeration Type Documentation

#### 5.14.2.1 anonymous enum

Comms state.

Enumerator

- COMMS\_STATE\_UNPAIRED** I have not paired.
- COMMS\_STATE\_DISCONNECTED** I have paired but am not connected.
- COMMS\_STATE\_PAIRING** Telling the drones I have accepted one of them.
- COMMS\_STATE\_PAIRED** Telling the drone I have received its accept.

#### 5.14.2.2 anonymous enum

Enumerator

- BK\_CHIP\_ID\_BK2425** The expected value of reading BK2425\_R1\_WHOAMI.

#### 5.14.2.3 enum BK\_CONFIG\_e

Meanings of the BK\_CONFIG register.

Enumerator

- BK\_CONFIG\_MASK\_RX\_DR** Mask interrupt caused by RX\_DR.

**BK\_CONFIG\_MASK\_TX\_DS** Mask interrupt caused by TX\_DS.  
**BK\_CONFIG\_MASK\_MAX\_RT** Mask interrupt caused by MAX\_RT.  
**BK\_CONFIG\_EN\_CRC** Enable CRC. Forced high if one of the bits in the EN\_AA is high.  
**BK\_CONFIG\_CRCO** CRC encoding scheme (0=8 bits, 1=16 bits)  
**BK\_CONFIG\_PWR\_UP** POWER UP.  
**BK\_CONFIG\_PRIM\_RX** Receive/transmit.

#### 5.14.2.4 enum BK\_FEATURE\_e

Meanings of the BK\_FEATURE register.

Enumerator

**BK\_FEATURE\_EN\_DPL** Dynamic packet length is enabled.

#### 5.14.2.5 enum BK\_FIFO\_STATUS\_e

Meanings of the FIFO\_STATUS register.

Enumerator

**BK\_FIFO\_STATUS\_TX\_FULL** The tx buffer has more than ? item.  
**BK\_FIFO\_STATUS\_TX\_EMPTY** The tx buffer has less than ? item.  
**BK\_FIFO\_STATUS\_RX\_FULL** The rx buffer has more than ? items.  
**BK\_FIFO\_STATUS\_RX\_EMPTY** The rx buffer has less than ? items.

#### 5.14.2.6 enum BK\_PKT\_TYPE\_E

The type of packets being sent between controller and drone.

Enumerator

**BK\_PKT\_TYPE\_INVALID** Invalid packet from empty packets or bad CRC.  
**BK\_PKT\_TYPE\_CTRL** (Tx->Drone) [ctrl] Packet type 3 = user control  
**BK\_PKT\_TYPE\_AVAILABLE** (Tx->Drone) [info] Packet type 5 = tx is available (and was either never paired or has been switched off and on again)  
**BK\_PKT\_TYPE\_DISCONNECTED** (Tx->Drone) [id] Packet type 6 = tx was connected and is now available  
**BK\_PKT\_TYPE\_PAIRING** (Tx->Drone) [id] Packet type 9 = tx is pairing to this address (normal comms speed - better range)  
**BK\_PKT\_TYPE\_DRONE** (Drone->Tx) Packet type 4 = drone command to tx (reply to ctrl)

#### 5.14.2.7 enum BK\_SPI\_CMD\_e

SPI register commands for the BK2425 and nrf24L01+ chips.

Enumerator

**BK\_REG\_MASK** The range of registers that can be read and written.  
**BK\_READ\_REG** Define read command to register (0..1F)

**BK\_WRITE\_REG** Define write command to register (0..1F)

**BK\_RD\_RX\_PLOAD** Define RX payload register address.

**BK\_WR\_TX\_PLOAD** Define TX payload register address.

**BK\_W\_ACK\_PAYLOAD\_CMD** (nrf: +pipe 0..7)

**BK\_FLUSH\_TX** Define flush TX register command.

**BK\_FLUSH\_RX** Define flush RX register command.

**BK\_REUSE\_TX\_PL** Define reuse TX payload register command.

**BK\_NOP** Define No Operation, might be used to read status register.

**BK\_CONFIG** 'Config' register address

**BK\_EN\_AA** 'Enable Auto Acknowledgment' register address

**BK\_EN\_RXADDR** 'Enabled RX addresses' register address

**BK\_SETUP\_AW** 'Setup address width' register address

**BK\_SETUP\_RETR** 'Setup Auto. Retrans' register address

**BK\_RF\_CH** 'RF channel' register address

**BK\_RF\_SETUP** 'RF setup' register address

**BK\_STATUS** 'Status' register address

**BK\_OBSERVE\_TX** 'Observe TX' register address (lost packets, retransmitted packets on this frequency)

**BK\_CD** 'Carrier Detect' register address

**BK\_RX\_ADDR\_P0** 'RX address pipe0' register address (5 bytes)

**BK\_RX\_ADDR\_P1** 'RX address pipe1' register address (5 bytes)

**BK\_RX\_ADDR\_P2** 'RX address pipe2' register address (1 byte)

**BK\_RX\_ADDR\_P3** 'RX address pipe3' register address (1 byte)

**BK\_RX\_ADDR\_P4** 'RX address pipe4' register address (1 byte)

**BK\_RX\_ADDR\_P5** 'RX address pipe5' register address (1 byte)

**BK\_TX\_ADDR** 'TX address' register address (5 bytes)

**BK\_RX\_PW\_P0** 'RX payload width, pipe0' register address

**BK\_RX\_PW\_P1** 'RX payload width, pipe1' register address

**BK\_RX\_PW\_P2** 'RX payload width, pipe2' register address

**BK\_RX\_PW\_P3** 'RX payload width, pipe3' register address

**BK\_RX\_PW\_P4** 'RX payload width, pipe4' register address

**BK\_RX\_PW\_P5** 'RX payload width, pipe5' register address

**BK\_FIFO\_STATUS** 'FIFO Status Register' register address

**BK\_DYNPD** 'Enable dynamic payload length' register address

**BK\_FEATURE** 'Feature' register address

**BK\_PAYLOAD\_WIDTH** 'payload length of 256 bytes modes register address

**BK2425\_R1\_WHOAMI** Register to read that contains the chip id.

**BK2425\_R1\_12** PLL speed 120 or 130us.

#### 5.14.2.8 enum BK\_STATUS\_e

Meanings of the BK\_STATUS register.

##### Enumerator

**BK\_STATUS\_RBANK** Register bank 1 is in use.

**BK\_STATUS\_RX\_DR** Data ready.

**BK\_STATUS\_TX\_DS** Data sent.  
**BK\_STATUS\_MAX\_RT** Max retries failed.  
**BK\_STATUS\_RX\_MASK** Mask for the receptions bit.  
**BK\_STATUS\_RX\_P\_5** Data pipe 5 has some data ready.  
**BK\_STATUS\_RX\_P\_4** Data pipe 4 has some data ready.  
**BK\_STATUS\_RX\_P\_3** Data pipe 3 has some data ready.  
**BK\_STATUS\_RX\_P\_2** Data pipe 2 has some data ready.  
**BK\_STATUS\_RX\_P\_1** Data pipe 1 has some data ready.  
**BK\_STATUS\_RX\_P\_0** Data pipe 0 has some data ready.  
**BK\_STATUS\_TX\_FULL** Tx buffer full.

#### 5.14.2.9 enum CHANNEL\_MHZ\_e

Channel hopping parameters.

Values are in MHz from 2400Mhz.

Enumerator

**CHANNEL\_MIN\_PHYSICAL** Minimum physical channel that is possible.  
**CHANNEL\_MAX\_PHYSICAL** Maximum physical channel that is possible.  
**CHANNEL\_FCC\_LOW** Minimum physical channel that will pass the FCC tests.  
**CHANNEL\_FCC\_HIGH** Maximum physical channel that will pass the FCC tests.  
**CHANNEL\_FCC\_MID** A representative physical channel.  
**CHANNEL\_TEST\_MODE** Frequency to use for testing.

#### 5.14.2.10 enum ITX\_SPEED\_e

The baud rate of the GFSK modulation.

Enumerator

**ITX\_250** 250kbps (slowest but furthest range)  
**ITX\_1000** 1000kbps (balanced)  
**ITX\_2000** 2000kbps (fastest hence least congested)

#### 5.14.2.11 enum SPI\_Flag\_e

Flags for the STM8 hardware SPI registers.

Enumerator

**SPI\_FLAG\_BSY** Busy flag  
**SPI\_FLAG\_OVR** Overrun flag  
**SPI\_FLAG\_MODF** Mode fault  
**SPI\_FLAG\_CRCERR** CRC error flag  
**SPI\_FLAG\_WKUP** Wake-up flag  
**SPI\_FLAG\_TXE** Transmit buffer empty  
**SPI\_FLAG\_RXNE** Receive buffer empty



### 5.14.3 Function Documentation

#### 5.14.3.1 void beken\_change\_FCC\_channel ( int8\_t *change* )

Change the FCC channel.

## Parameters

<i>change</i>	?
---------------	---

5.14.3.2 void beken\_set\_CW\_mode ( bool *cw* )

Go into continuous carrier wave send mode or normal mode.

## Parameters

<i>cw</i>	false=normal, true=carrier wave
-----------	---------------------------------

5.14.3.3 void beken\_start\_factory\_test ( uint8\_t *test\_mode* )

Start sending an factory test packet.

## Parameters

<i>test_mode</i>	The type of test to send.
------------------	---------------------------

5.14.3.4 void BK2425\_Initialize ( ITX\_SPEED *spd* )

BK2425 initialization of radio registers.

## Parameters

<i>spd</i>	The baudrate to modulate the transmission and reception at.
------------	---

5.14.3.5 void BK2425\_SetSpeed ( bool *bFast* )

Change between 250kbps and 2000kbps on the fly.

## Parameters

<i>bFast</i>	false=slow speed, true=fast speed
--------------	-----------------------------------

5.14.3.6 void ChangeChannel ( uint8\_t *channelNumber* )

Change the radio channel.

## Parameters

<i>channelNumber</i>	A physical radio channel. See <a href="#">CHANNEL_MHZ_e</a>
----------------------	---

5.14.3.7 void ChangeOutputPower ( uint8\_t *power* )

Change the radio output power of the Beken radio chip.

## Parameters

<i>power</i>	power value
--------------	-------------

5.14.3.8 `uint8_t Get_Chip_ID ( void )`

Get the Beken radio chip ID.

## Returns

BK\_CHIP\_ID\_BK2425

5.14.3.9 `uint8_t LookupChannel ( uint8_t idx )`

Convert a logical channel index into a physical channel.

## Returns

The physical channel, in MHz above 2400Mhz.

## Parameters

<i>idx</i>	The logical channel, as an index into a frequency hopping table.
------------	--

5.14.3.10 `uint8_t NextChannelIndex ( uint8_t seq )`

Channel hopping algorithm implementation.

Calculate the next channel to use for transmission and change to it

## Returns

The next value of the logical channel index.

## Parameters

<i>seq</i>	The current value of the logical channel index
------------	--

5.14.3.11 `uint8_t Receive_Packet ( uint8_t rx_buff[] )`

Read FIFO to read a packet.

## Returns

0 if no packet, 1 if packet read

## Parameters

<i>rx_buf</i>	The buffer to fill
---------------	--------------------

5.14.3.12 `bool Send_Packet ( uint8_t type, const uint8_t * pbuf, uint8_t len )`

Fill the Bekens tx FIFO to send a packet.

## Returns

True if ack overflow was set when send was requested.

## Parameters

<i>type</i>	WR_TX_PLOAD or W_TX_PAYLOAD_NOACK_CMD
<i>pbuf</i>	a buffer pointer
<i>len</i>	packet length in bytes

5.14.3.13 `bool SetChannelRange ( uint8_t min, uint8_t max )`

Set the range of the channel indexes we are using.

## Returns

true if we changed something

## Parameters

<i>min</i>	The minimum logical channel range
<i>max</i>	The maximum logical channel range

5.14.3.14 `void SetRBank ( char _cfg )`

Set which register bank we are accessing on the Beken spi chip.

## Parameters

<i>_cfg</i>	1=Bank1 0=Bank0
-------------	-----------------

5.14.3.15 `void SPI_Bank1_Read_Reg ( uint8_t reg, uint8_t * pBuf )`

Read a 32-bit Bank1 register.

## Parameters

<i>reg</i>	A spi register in bank1 to write to <a href="#">BK_SPI_CMD_e</a>
<i>pBuf</i>	A pointer to a 32-bit buffer to be read into

5.14.3.16 `void SPI_Bank1_Write_Reg ( uint8_t reg, const uint8_t * pBuf )`

Write a 32-bit Bank1 register.

## Parameters

<i>reg</i>	A spi register in bank1 to write to <a href="#">BK_SPI_CMD_e</a>
<i>pBuf</i>	A pointer to a 32-bit buffer to be written

5.14.3.17 `uint8_t SPI_Read_Reg ( uint8_t reg )`

Read one uint8\_t from BK2425 register 'reg' via SPI.

## Returns

The register value

## Parameters

<i>reg</i>	The command to write <a href="#">BK_SPI_CMD_e</a>
------------	---

5.14.3.18 void SPI\_Write\_Buf ( uint8\_t *reg*, const uint8\_t \* *pBuf*, uint8\_t *length* )

Writes contents of a buffer to BK2425 via SPI.

## Parameters

<i>reg</i>	The command to write <a href="#">BK_SPI_CMD_e</a>
<i>pBuf</i>	The data to write
<i>length</i>	The length in bytes of the data to write

5.14.3.19 void SPI\_Write\_Cmd ( uint8\_t *reg* )

Write a single byte command to the SPI bus (e.g.

Flush)

## Parameters

<i>reg</i>	The simple command to write <a href="#">BK_SPI_CMD_e</a>
------------	--

5.14.3.20 void SPI\_Write\_Reg ( uint8\_t *reg*, uint8\_t *value* )

Writes value 'value' to register 'reg'.

## Parameters

<i>reg</i>	The command to write <a href="#">BK_SPI_CMD_e</a>
<i>value</i>	The data value to write

## 5.15 printf functions

### Functions

- void [vprintf](#) (const char \*fmt, va\_list ap)  
*Print a string using a va\_list to hold the variable arguments.*
- void [printf](#) (const char \*fmt,...)  
*Small implementation of the standard printf routine.*

### 5.15.1 Detailed Description

### 5.15.2 Function Documentation

#### 5.15.2.1 void printf ( const char \* *fmt*, ... )

Small implementation of the standard printf routine.

##### Parameters

<i>fmt</i>	The format string. <a href="#">format_string</a>
------------	--

#### 5.15.2.2 void vprintf ( const char \* *fmt*, va\_list *ap* )

Print a string using a va\_list to hold the variable arguments.

##### Parameters

<i>fmt</i>	The format string. <a href="#">format_string</a>
<i>ap</i>	All other parameters

## 5.16 Main transmitter code

### Enumerations

- enum `control_mode_t` {  
`STABILIZE` = 0, `ACRO` = 1, `ALT_HOLD` = 2, `AUTO` = 3,  
`GUIDED` = 4, `LOITER` = 5, `RTL` = 6, `CIRCLE` = 7,  
`LAND` = 9, `DRIFT` = 11, `SPORT` = 13, `FLIP` = 14,  
`AUTOTUNE` = 15, `POSHOLD` = 16, `BRAKE` = 17, `THROW` = 18,  
`AVOID_ADSB` = 19, `GUIDED_NOGPS` = 20, `FLOWHOLD` = 21 }

*The current control mode.*

### Functions

- void `main` (void)

*Main entry point for the program.*

#### 5.16.1 Detailed Description

#### 5.16.2 Enumeration Type Documentation

##### 5.16.2.1 enum `control_mode_t`

The current control mode.

#### Enumerator

**STABILIZE** manual airframe angle with manual throttle  
**ACRO** manual body-frame angular rate with manual throttle  
**ALT\_HOLD** manual airframe angle with automatic throttle  
**AUTO** fully automatic waypoint control using mission commands  
**GUIDED** fully automatic fly to coordinate or fly at velocity/direction using GCS immediate commands  
**LOITER** automatic horizontal acceleration with automatic throttle  
**RTL** automatic return to launching point  
**CIRCLE** automatic circular flight with automatic throttle  
**LAND** automatic landing with horizontal position control  
**DRIFT** semi-autonomous position, yaw and throttle control  
**SPORT** manual earth-frame angular rate control with manual throttle  
**FLIP** automatically flip the vehicle on the roll axis  
**AUTOTUNE** automatically tune the vehicle's roll and pitch gains  
**POSHOLD** automatic position hold with manual override, with automatic throttle  
**BRAKE** full-brake using inertial/GPS system, no pilot input  
**THROW** throw to launch mode using inertial/GPS system, no pilot input  
**AVOID\_ADSB** automatic avoidance of obstacles in the macro scale - e.g. full-sized aircraft  
**GUIDED\_NOGPS** guided mode but only accepts attitude and altitude  
**FLOWHOLD** hold with flow sensor





## Chapter 6

# Data Structure Documentation

### 6.1 gpio\_regs Struct Reference

Declaration of how the hardware is laid out on STM8 processors (e.g.

#### Data Fields

- uint8\_t [ODR](#)  
*Output data register.*
- uint8\_t [IDR](#)  
*Input data register.*
- uint8\_t [DDR](#)  
*Data direction register.*
- uint8\_t [CR1](#)  
*Control register one.*
- uint8\_t [CR2](#)  
*Control register two.*

#### 6.1.1 Detailed Description

Declaration of how the hardware is laid out on STM8 processors (e.g. STM85105)

The documentation for this struct was generated from the following file:

- E:/ArduPilot/StreamingGPSTransmitter/lib/gpio.c

### 6.2 packetDataDevice\_s Struct Reference

Data structure for data packet transmitted from device (controller) to host (drone)

#### Data Structures

- union [packetDataDevice\\_u](#)  
< *The variant part of the packets*

## Data Fields

- BK\_PKT\_TYPE [packetType](#)  
*The packet type.*
- uint8\_t [channel](#)  
*Next channel I will broadcast on.*

### 6.2.1 Detailed Description

Data structure for data packet transmitted from device (controller) to host (drone)

The documentation for this struct was generated from the following file:

- E:/ArduPilot/StreamingGPSTransmitter/lib/beken.c

## 6.3 packetDataDevice\_s::packetDataDevice\_u Union Reference

< The variant part of the packets

## Data Fields

- [packetDataDeviceCtrl ctrl](#)  
*Control packets.*
- [packetDataDeviceID id](#)  
*Binding packets.*

### 6.3.1 Detailed Description

< The variant part of the packets

The documentation for this union was generated from the following file:

- E:/ArduPilot/StreamingGPSTransmitter/lib/beken.c

## 6.4 packetDataDeviceCtrl\_s Struct Reference

Data for packets that are not droneid packets Onair order = little-endian.

## Data Fields

- uint8\_t [throttle](#)  
*High 8 bits of the throttle joystick.*
- uint8\_t [roll](#)  
*High 8 bits of the roll joystick.*
- uint8\_t [pitch](#)  
*High 8 bits of the pitch joystick.*
- uint8\_t [yaw](#)  
*High 8 bits of the yaw joystick.*
- uint8\_t [lsb](#)  
*Low 2 bits of throttle, roll, pitch, yaw.*

- uint8\_t [buttons](#)  
*The buttons.*
- uint8\_t [data\\_type](#)  
*Type of extra data being sent.*
- uint8\_t [data\\_value](#)  
*Value of extra data being sent.*

#### 6.4.1 Detailed Description

Data for packets that are not droneid packets Onair order = little-endian.

The documentation for this struct was generated from the following file:

- E:/ArduPilot/StreamingGPSTransmitter/lib/beken.c

## 6.5 packetDataDeviceID\_s Struct Reference

Data for packets that are binding packets Onair order = little-endian.

### Data Fields

- uint8\_t [droneid](#) [SZ\_DRONEID]  
*The UUID of the drone.*
- uint8\_t [reconnectAddress](#) [3]  
*The Address chosen for this connection.*

#### 6.5.1 Detailed Description

Data for packets that are binding packets Onair order = little-endian.

The documentation for this struct was generated from the following file:

- E:/ArduPilot/StreamingGPSTransmitter/lib/beken.c

## 6.6 packetDataDrone\_s Struct Reference

Data structure for data packet transmitted from host (drone) to device (controller)

### Data Fields

- BK\_PKT\_TYPE [packetType](#)  
*The packet type.*
- uint8\_t [channel](#)  
*Next channel I will broadcast on.*
- uint8\_t [data](#) [10]  
*Telemetry data (unspecified so far)*

### 6.6.1 Detailed Description

Data structure for data packet transmitted from host (drone) to device (controller)

The documentation for this struct was generated from the following file:

- E:/ArduPilot/StreamingGPSTransmitter/lib/beken.c

## 6.7 telem\_firmware Struct Reference

Telemetry packet for the command to write to new firmware.

```
#include <telem_structure.h>
```

### 6.7.1 Detailed Description

Telemetry packet for the command to write to new firmware.

This is also used to play a tune.

The documentation for this struct was generated from the following file:

- E:/ArduPilot/StreamingGPSTransmitter/include/telem\_structure.h

## 6.8 telem\_packet Struct Reference

telemetry packet from RX to TX

```
#include <telem_structure.h>
```

### Data Fields

- uint8\_t [crc](#)  
*simple CRC*

### 6.8.1 Detailed Description

telemetry packet from RX to TX

The documentation for this struct was generated from the following file:

- E:/ArduPilot/StreamingGPSTransmitter/include/telem\_structure.h

## 6.9 telem\_play Struct Reference

Telemetry packet for the command to play a tune.

```
#include <telem_structure.h>
```

### 6.9.1 Detailed Description

Telemetry packet for the command to play a tune.

The documentation for this struct was generated from the following file:

- E:/ArduPilot/StreamingGPSTransmitter/include/telem\_structure.h

## 6.10 telem\_status Struct Reference

Telemetry status packet.

```
#include <telem_structure.h>
```

### Data Fields

- uint8\_t [pps](#)  
*packets per second received*
- uint8\_t [rssi](#)  
*lowpass rssi*
- uint8\_t [flags](#)  
*TELEM\_FLAG\_\*.*
- uint8\_t [flight\\_mode](#)  
*flight mode*
- uint8\_t [wifi\\_chan](#)  
*Wi-Fi channel.*
- uint8\_t [tx\\_max](#)  
*tx max*
- uint8\_t [note\\_adjust](#)  
*Note adjustment.*

### 6.10.1 Detailed Description

Telemetry status packet.

The documentation for this struct was generated from the following file:

- E:/ArduPilot/StreamingGPSTransmitter/include/telem\_structure.h

## 6.11 telem\_tx\_status Struct Reference

tx\_status structure sent one byte at a time to RX.

```
#include <telem_structure.h>
```

### Data Fields

- uint8\_t [crc](#)  
*Simple crc.*
- enum [tx\\_telem\\_type](#) [type](#)  
*type of telemetry word*
- uint16\_t [data](#)  
*The telemetry word.*

### 6.11.1 Detailed Description

tx\_status structure sent one byte at a time to RX.

This is packed into channels 8, 9 and 10 (using 32 bits of a possible 33)

The documentation for this struct was generated from the following file:

- E:/ArduPilot/StreamingGPSTransmitter/include/telem\_structure.h

# Index

- ACRO
  - Main transmitter code, [41](#)
- ALT\_HOLD
  - Main transmitter code, [41](#)
- AUTO
  - Main transmitter code, [41](#)
- AUTOTUNE
  - Main transmitter code, [41](#)
- AVOID\_ADSB
  - Main transmitter code, [41](#)
- adc\_channel
  - Analog to Digital Conversion, [9](#)
- adc\_value
  - Analog to Digital Conversion, [9](#)
- Analog to Digital Conversion, [9](#)
  - adc\_channel, [9](#)
  - adc\_value, [9](#)
  - STICK\_PITCH, [9](#)
  - STICK\_ROLL, [9](#)
  - STICK\_THROTTLE, [9](#)
  - STICK\_YAW, [9](#)
- BK2425\_R1\_12
  - Beken BK2425 radio module, [33](#)
- BK2425\_R1\_WHOAMI
  - Beken BK2425 radio module, [33](#)
- BK\_CD
  - Beken BK2425 radio module, [33](#)
- BK\_CHIP\_ID\_BK2425
  - Beken BK2425 radio module, [31](#)
- BK\_CONFIG
  - Beken BK2425 radio module, [33](#)
- BK\_CONFIG\_CRCO
  - Beken BK2425 radio module, [32](#)
- BK\_CONFIG\_EN\_CRC
  - Beken BK2425 radio module, [32](#)
- BK\_CONFIG\_MASK\_MAX\_RT
  - Beken BK2425 radio module, [32](#)
- BK\_CONFIG\_MASK\_RX\_DR
  - Beken BK2425 radio module, [31](#)
- BK\_CONFIG\_MASK\_TX\_DS
  - Beken BK2425 radio module, [31](#)
- BK\_CONFIG\_PRIM\_RX
  - Beken BK2425 radio module, [32](#)
- BK\_CONFIG\_PWR\_UP
  - Beken BK2425 radio module, [32](#)
- BK\_DYNPD
  - Beken BK2425 radio module, [33](#)
- BK\_EN\_AA
  - Beken BK2425 radio module, [33](#)
- BK\_EN\_RXADDR
  - Beken BK2425 radio module, [33](#)
- BK\_FEATURE
  - Beken BK2425 radio module, [33](#)
- BK\_FEATURE\_EN\_DPL
  - Beken BK2425 radio module, [32](#)
- BK\_FIFO\_STATUS
  - Beken BK2425 radio module, [33](#)
- BK\_FIFO\_STATUS\_RX\_EMPTY
  - Beken BK2425 radio module, [32](#)
- BK\_FIFO\_STATUS\_RX\_FULL
  - Beken BK2425 radio module, [32](#)
- BK\_FIFO\_STATUS\_TX\_EMPTY
  - Beken BK2425 radio module, [32](#)
- BK\_FIFO\_STATUS\_TX\_FULL
  - Beken BK2425 radio module, [32](#)
- BK\_FLUSH\_RX
  - Beken BK2425 radio module, [33](#)
- BK\_FLUSH\_TX
  - Beken BK2425 radio module, [33](#)
- BK\_NOP
  - Beken BK2425 radio module, [33](#)
- BK\_OBSERVE\_TX
  - Beken BK2425 radio module, [33](#)
- BK\_PAYLOAD\_WIDTH
  - Beken BK2425 radio module, [33](#)
- BK\_PKT\_TYPE\_AVAILABLE
  - Beken BK2425 radio module, [32](#)
- BK\_PKT\_TYPE\_CTRL
  - Beken BK2425 radio module, [32](#)
- BK\_PKT\_TYPE\_DISCONNECTED
  - Beken BK2425 radio module, [32](#)
- BK\_PKT\_TYPE\_DRONE
  - Beken BK2425 radio module, [32](#)
- BK\_PKT\_TYPE\_INVALID
  - Beken BK2425 radio module, [32](#)
- BK\_PKT\_TYPE\_PAIRING
  - Beken BK2425 radio module, [32](#)
- BK\_RD\_RX\_PLOAD
  - Beken BK2425 radio module, [33](#)
- BK\_READ\_REG
  - Beken BK2425 radio module, [32](#)
- BK\_REG\_MASK
  - Beken BK2425 radio module, [32](#)
- BK\_REUSE\_TX\_PL
  - Beken BK2425 radio module, [33](#)
- BK\_RF\_CH
  - Beken BK2425 radio module, [33](#)
- BK\_RF\_SETUP

- Beken BK2425 radio module, [33](#)
- BK\_RX\_ADDR\_P0
  - Beken BK2425 radio module, [33](#)
- BK\_RX\_ADDR\_P1
  - Beken BK2425 radio module, [33](#)
- BK\_RX\_ADDR\_P2
  - Beken BK2425 radio module, [33](#)
- BK\_RX\_ADDR\_P3
  - Beken BK2425 radio module, [33](#)
- BK\_RX\_ADDR\_P4
  - Beken BK2425 radio module, [33](#)
- BK\_RX\_ADDR\_P5
  - Beken BK2425 radio module, [33](#)
- BK\_RX\_PW\_P0
  - Beken BK2425 radio module, [33](#)
- BK\_RX\_PW\_P1
  - Beken BK2425 radio module, [33](#)
- BK\_RX\_PW\_P2
  - Beken BK2425 radio module, [33](#)
- BK\_RX\_PW\_P3
  - Beken BK2425 radio module, [33](#)
- BK\_RX\_PW\_P4
  - Beken BK2425 radio module, [33](#)
- BK\_RX\_PW\_P5
  - Beken BK2425 radio module, [33](#)
- BK\_SETUP\_AW
  - Beken BK2425 radio module, [33](#)
- BK\_SETUP\_RETR
  - Beken BK2425 radio module, [33](#)
- BK\_STATUS
  - Beken BK2425 radio module, [33](#)
- BK\_STATUS\_MAX\_RT
  - Beken BK2425 radio module, [34](#)
- BK\_STATUS\_RBANK
  - Beken BK2425 radio module, [33](#)
- BK\_STATUS\_RX\_DR
  - Beken BK2425 radio module, [33](#)
- BK\_STATUS\_RX\_MASK
  - Beken BK2425 radio module, [34](#)
- BK\_STATUS\_RX\_P\_0
  - Beken BK2425 radio module, [34](#)
- BK\_STATUS\_RX\_P\_1
  - Beken BK2425 radio module, [34](#)
- BK\_STATUS\_RX\_P\_2
  - Beken BK2425 radio module, [34](#)
- BK\_STATUS\_RX\_P\_3
  - Beken BK2425 radio module, [34](#)
- BK\_STATUS\_RX\_P\_4
  - Beken BK2425 radio module, [34](#)
- BK\_STATUS\_RX\_P\_5
  - Beken BK2425 radio module, [34](#)
- BK\_STATUS\_TX\_DS
  - Beken BK2425 radio module, [33](#)
- BK\_STATUS\_TX\_FULL
  - Beken BK2425 radio module, [34](#)
- BK\_TX\_ADDR
  - Beken BK2425 radio module, [33](#)
- BK\_W\_ACK\_PAYLOAD\_CMD
  - Beken BK2425 radio module, [33](#)
- BK\_WR\_TX\_PLOAD
  - Beken BK2425 radio module, [33](#)
- BK\_WRITE\_REG
  - Beken BK2425 radio module, [32](#)
- BRAKE
  - Main transmitter code, [41](#)
- BUTTON\_LEFT
  - Protocol logical channels, [12](#)
- BUTTON\_LEFT\_SHOULDER
  - Protocol logical channels, [12](#)
- BUTTON\_MODE
  - Protocol logical channels, [12](#)
- BUTTON\_POWER
  - Protocol logical channels, [12](#)
- BUTTON\_RIGHT
  - Protocol logical channels, [12](#)
- BUTTON\_RIGHT\_SHOULDER
  - Protocol logical channels, [12](#)
- BK2425\_Initialize
  - Beken BK2425 radio module, [36](#)
- BK2425\_SetSpeed
  - Beken BK2425 radio module, [36](#)
- BK\_CONFIG\_e
  - Beken BK2425 radio module, [31](#)
- BK\_FEATURE\_e
  - Beken BK2425 radio module, [32](#)
- BK\_FIFO\_STATUS\_e
  - Beken BK2425 radio module, [32](#)
- BK\_PKT\_TYPE\_E
  - Beken BK2425 radio module, [32](#)
- BK\_SPI\_CMD\_e
  - Beken BK2425 radio module, [32](#)
- BK\_STATUS\_e
  - Beken BK2425 radio module, [33](#)
- Beken BK2425 radio module
  - BK2425\_R1\_12, [33](#)
  - BK2425\_R1\_WHOAMI, [33](#)
  - BK\_CD, [33](#)
  - BK\_CHIP\_ID\_BK2425, [31](#)
  - BK\_CONFIG, [33](#)
  - BK\_CONFIG\_CRCO, [32](#)
  - BK\_CONFIG\_EN\_CRC, [32](#)
  - BK\_CONFIG\_MASK\_MAX\_RT, [32](#)
  - BK\_CONFIG\_MASK\_RX\_DR, [31](#)
  - BK\_CONFIG\_MASK\_TX\_DS, [31](#)
  - BK\_CONFIG\_PRIM\_RX, [32](#)
  - BK\_CONFIG\_PWR\_UP, [32](#)
  - BK\_DYNPD, [33](#)
  - BK\_EN\_AA, [33](#)
  - BK\_EN\_RXADDR, [33](#)
  - BK\_FEATURE, [33](#)
  - BK\_FEATURE\_EN\_DPL, [32](#)
  - BK\_FIFO\_STATUS, [33](#)
  - BK\_FIFO\_STATUS\_RX\_EMPTY, [32](#)
  - BK\_FIFO\_STATUS\_RX\_FULL, [32](#)
  - BK\_FIFO\_STATUS\_TX\_EMPTY, [32](#)
  - BK\_FIFO\_STATUS\_TX\_FULL, [32](#)



- BK\_FLUSH\_RX, [33](#)
- BK\_FLUSH\_TX, [33](#)
- BK\_NOP, [33](#)
- BK\_OBSERVE\_TX, [33](#)
- BK\_PAYLOAD\_WIDTH, [33](#)
- BK\_PKT\_TYPE\_AVAILABLE, [32](#)
- BK\_PKT\_TYPE\_CTRL, [32](#)
- BK\_PKT\_TYPE\_DISCONNECTED, [32](#)
- BK\_PKT\_TYPE\_DRONE, [32](#)
- BK\_PKT\_TYPE\_INVALID, [32](#)
- BK\_PKT\_TYPE\_PAIRING, [32](#)
- BK\_RD\_RX\_PLOAD, [33](#)
- BK\_READ\_REG, [32](#)
- BK\_REG\_MASK, [32](#)
- BK\_REUSE\_TX\_PL, [33](#)
- BK\_RF\_CH, [33](#)
- BK\_RF\_SETUP, [33](#)
- BK\_RX\_ADDR\_P0, [33](#)
- BK\_RX\_ADDR\_P1, [33](#)
- BK\_RX\_ADDR\_P2, [33](#)
- BK\_RX\_ADDR\_P3, [33](#)
- BK\_RX\_ADDR\_P4, [33](#)
- BK\_RX\_ADDR\_P5, [33](#)
- BK\_RX\_PW\_P0, [33](#)
- BK\_RX\_PW\_P1, [33](#)
- BK\_RX\_PW\_P2, [33](#)
- BK\_RX\_PW\_P3, [33](#)
- BK\_RX\_PW\_P4, [33](#)
- BK\_RX\_PW\_P5, [33](#)
- BK\_SETUP\_AW, [33](#)
- BK\_SETUP\_RETR, [33](#)
- BK\_STATUS, [33](#)
- BK\_STATUS\_MAX\_RT, [34](#)
- BK\_STATUS\_RBANK, [33](#)
- BK\_STATUS\_RX\_DR, [33](#)
- BK\_STATUS\_RX\_MASK, [34](#)
- BK\_STATUS\_RX\_P\_0, [34](#)
- BK\_STATUS\_RX\_P\_1, [34](#)
- BK\_STATUS\_RX\_P\_2, [34](#)
- BK\_STATUS\_RX\_P\_3, [34](#)
- BK\_STATUS\_RX\_P\_4, [34](#)
- BK\_STATUS\_RX\_P\_5, [34](#)
- BK\_STATUS\_TX\_DS, [33](#)
- BK\_STATUS\_TX\_FULL, [34](#)
- BK\_TX\_ADDR, [33](#)
- BK\_W\_ACK\_PAYLOAD\_CMD, [33](#)
- BK\_WR\_TX\_PLOAD, [33](#)
- BK\_WRITE\_REG, [32](#)
- CHANNEL\_FCC\_HIGH, [34](#)
- CHANNEL\_FCC\_LOW, [34](#)
- CHANNEL\_FCC\_MID, [34](#)
- CHANNEL\_MAX\_PHYSICAL, [34](#)
- CHANNEL\_MIN\_PHYSICAL, [34](#)
- CHANNEL\_TEST\_MODE, [34](#)
- COMMS\_STATE\_DISCONNECTED, [31](#)
- COMMS\_STATE\_PAIRING, [31](#)
- COMMS\_STATE\_PAIRING, [31](#)
- COMMS\_STATE\_UNPAIRED, [31](#)
- ITX\_1000, [34](#)
- ITX\_2000, [34](#)
- ITX\_250, [34](#)
- SPI\_FLAG\_BSY, [34](#)
- SPI\_FLAG\_CRCERR, [34](#)
- SPI\_FLAG\_MODF, [34](#)
- SPI\_FLAG\_OVR, [34](#)
- SPI\_FLAG\_RXNE, [34](#)
- SPI\_FLAG\_TXE, [34](#)
- SPI\_FLAG\_WKUP, [34](#)
- Beken BK2425 radio module, [28](#)
  - BK2425\_Initialize, [36](#)
  - BK2425\_SetSpeed, [36](#)
  - BK\_CONFIG\_e, [31](#)
  - BK\_FEATURE\_e, [32](#)
  - BK\_FIFO\_STATUS\_e, [32](#)
  - BK\_PKT\_TYPE\_E, [32](#)
  - BK\_SPI\_CMD\_e, [32](#)
  - BK\_STATUS\_e, [33](#)
  - beken\_change\_FCC\_channel, [35](#)
  - beken\_set\_CW\_mode, [36](#)
  - beken\_start\_factory\_test, [36](#)
  - CHANNEL\_MHZ\_e, [34](#)
  - ChangeChannel, [36](#)
  - ChangeOutputPower, [36](#)
  - Get\_Chip\_ID, [36](#)
  - ITX\_SPEED\_e, [34](#)
  - LookupChannel, [37](#)
  - NextChannelIndex, [37](#)
  - Receive\_Packet, [37](#)
  - SPI\_Bank1\_Read\_Reg, [38](#)
  - SPI\_Bank1\_Write\_Reg, [38](#)
  - SPI\_Flag\_e, [34](#)
  - SPI\_Read\_Reg, [38](#)
  - SPI\_Write\_Buf, [39](#)
  - SPI\_Write\_Cmd, [39](#)
  - SPI\_Write\_Reg, [39](#)
  - Send\_Packet, [37](#)
  - SetChannelRange, [38](#)
  - SetRBank, [38](#)
- beken\_change\_FCC\_channel
  - Beken BK2425 radio module, [35](#)
- beken\_set\_CW\_mode
  - Beken BK2425 radio module, [36](#)
- beken\_start\_factory\_test
  - Beken BK2425 radio module, [36](#)
- button\_bits
  - Protocol logical channels, [12](#)
- buzzer\_tune
  - Sound buzzer module, [11](#)
- CHANNEL\_FCC\_HIGH
  - Beken BK2425 radio module, [34](#)
- CHANNEL\_FCC\_LOW
  - Beken BK2425 radio module, [34](#)
- CHANNEL\_FCC\_MID
  - Beken BK2425 radio module, [34](#)
- CHANNEL\_MAX\_PHYSICAL
  - Beken BK2425 radio module, [34](#)

- CHANNEL\_MIN\_PHYSICAL
  - Beken BK2425 radio module, [34](#)
- CHANNEL\_TEST\_MODE
  - Beken BK2425 radio module, [34](#)
- CIRCLE
  - Main transmitter code, [41](#)
- COMMS\_STATE\_DISCONNECTED
  - Beken BK2425 radio module, [31](#)
- COMMS\_STATE\_PAIRING
  - Beken BK2425 radio module, [31](#)
- COMMS\_STATE\_PAIRING
  - Beken BK2425 radio module, [31](#)
- COMMS\_STATE\_UNPAIRED
  - Beken BK2425 radio module, [31](#)
- CHANNEL\_MHZ\_e
  - Beken BK2425 radio module, [34](#)
- ChangeChannel
  - Beken BK2425 radio module, [36](#)
- ChangeOutputPower
  - Beken BK2425 radio module, [36](#)
- channel\_value
  - Protocol logical channels, [12](#)
- chip\_init
  - Utility functions, [26](#)
- control\_mode\_t
  - Main transmitter code, [41](#)
- Cyclic Redundancy Check, [15](#)
- DRIFT
  - Main transmitter code, [41](#)
- delay\_ms
  - Utility functions, [26](#)
- delay\_us
  - Utility functions, [27](#)
- EEPROM reading/writing (NOT flash), [16](#)
  - EEPROM\_read, [16](#)
  - EEPROM\_write, [16](#)
- EEPROM\_read
  - EEPROM reading/writing (NOT flash), [16](#)
- EEPROM\_write
  - EEPROM reading/writing (NOT flash), [16](#)
- FLIP
  - Main transmitter code, [41](#)
- FLOWHOLD
  - Main transmitter code, [41](#)
- GPIO\_CLEAR
  - General Purpose Input/Output, [18](#)
- GPIO\_INPUT\_FLOAT
  - General Purpose Input/Output, [17](#)
- GPIO\_INPUT\_FLOAT\_IRQ
  - General Purpose Input/Output, [18](#)
- GPIO\_INPUT\_PULLUP
  - General Purpose Input/Output, [17](#)
- GPIO\_INPUT\_PULLUP\_IRQ
  - General Purpose Input/Output, [18](#)
- GPIO\_OUTPUT\_OPEN\_DRAIN
  - General Purpose Input/Output, [18](#)
- GPIO\_OUTPUT\_OPEN\_DRAIN\_FAST
  - General Purpose Input/Output, [18](#)
- GPIO\_OUTPUT\_PUSHPULL
  - General Purpose Input/Output, [18](#)
- GPIO\_OUTPUT\_PUSHPULL\_FAST
  - General Purpose Input/Output, [18](#)
- GPIO\_PIN0
  - General Purpose Input/Output, [18](#)
- GPIO\_PIN1
  - General Purpose Input/Output, [18](#)
- GPIO\_PIN2
  - General Purpose Input/Output, [18](#)
- GPIO\_PIN3
  - General Purpose Input/Output, [18](#)
- GPIO\_PIN4
  - General Purpose Input/Output, [18](#)
- GPIO\_PIN5
  - General Purpose Input/Output, [18](#)
- GPIO\_PIN6
  - General Purpose Input/Output, [18](#)
- GPIO\_PIN7
  - General Purpose Input/Output, [18](#)
- GPIO\_PORTA
  - General Purpose Input/Output, [18](#)
- GPIO\_PORTB
  - General Purpose Input/Output, [18](#)
- GPIO\_PORTC
  - General Purpose Input/Output, [18](#)
- GPIO\_PORTD
  - General Purpose Input/Output, [18](#)
- GPIO\_PORTE
  - General Purpose Input/Output, [18](#)
- GPIO\_PORTF
  - General Purpose Input/Output, [18](#)
- GPIO\_PORTG
  - General Purpose Input/Output, [18](#)
- GPIO\_PORTH
  - General Purpose Input/Output, [18](#)
- GPIO\_PORTI
  - General Purpose Input/Output, [18](#)
- GPIO\_SET
  - General Purpose Input/Output, [18](#)
- GUIDED
  - Main transmitter code, [41](#)
- GUIDED\_NOGPS
  - Main transmitter code, [41](#)
- General Purpose Input/Output
  - GPIO\_CLEAR, [18](#)
  - GPIO\_INPUT\_FLOAT, [17](#)
  - GPIO\_INPUT\_FLOAT\_IRQ, [18](#)
  - GPIO\_INPUT\_PULLUP, [17](#)
  - GPIO\_INPUT\_PULLUP\_IRQ, [18](#)
  - GPIO\_OUTPUT\_OPEN\_DRAIN, [18](#)
  - GPIO\_OUTPUT\_OPEN\_DRAIN\_FAST, [18](#)
  - GPIO\_OUTPUT\_PUSHPULL, [18](#)
  - GPIO\_OUTPUT\_PUSHPULL\_FAST, [18](#)
  - GPIO\_PIN0, [18](#)

- GPIO\_PIN1, [18](#)
- GPIO\_PIN2, [18](#)
- GPIO\_PIN3, [18](#)
- GPIO\_PIN4, [18](#)
- GPIO\_PIN5, [18](#)
- GPIO\_PIN6, [18](#)
- GPIO\_PIN7, [18](#)
- GPIO\_PORTA, [18](#)
- GPIO\_PORTB, [18](#)
- GPIO\_PORTC, [18](#)
- GPIO\_PORTD, [18](#)
- GPIO\_PORTE, [18](#)
- GPIO\_PORTF, [18](#)
- GPIO\_PORTG, [18](#)
- GPIO\_PORTH, [18](#)
- GPIO\_PORTI, [18](#)
- GPIO\_SET, [18](#)
- General Purpose Input/Output, [17](#)
  - gpio\_clear, [18](#)
  - gpio\_config, [17](#), [18](#)
  - gpio\_get, [19](#)
  - gpio\_pins, [18](#)
  - gpio\_set, [19](#)
  - gpio\_toggle, [19](#)
- Get\_Chip\_ID
  - Beken BK2425 radio module, [36](#)
- get\_buttons
  - Protocol logical channels, [13](#)
- gpio\_clear
  - General Purpose Input/Output, [18](#)
- gpio\_config
  - General Purpose Input/Output, [17](#), [18](#)
- gpio\_get
  - General Purpose Input/Output, [19](#)
- gpio\_pins
  - General Purpose Input/Output, [18](#)
- gpio\_regs, [43](#)
- gpio\_set
  - General Purpose Input/Output, [19](#)
- gpio\_toggle
  - General Purpose Input/Output, [19](#)
- ITX\_1000
  - Beken BK2425 radio module, [34](#)
- ITX\_2000
  - Beken BK2425 radio module, [34](#)
- ITX\_250
  - Beken BK2425 radio module, [34](#)
- ITX\_SPEED\_e
  - Beken BK2425 radio module, [34](#)
- LAND
  - Main transmitter code, [41](#)
- LOITER
  - Main transmitter code, [41](#)
- LookupChannel
  - Beken BK2425 radio module, [37](#)
- Main transmitter code, [41](#)
- ACRO, [41](#)
- ALT\_HOLD, [41](#)
- AUTO, [41](#)
- AUTOTUNE, [41](#)
- AVOID\_ADSB, [41](#)
- BRAKE, [41](#)
- CIRCLE, [41](#)
- control\_mode\_t, [41](#)
- DRIFT, [41](#)
- FLIP, [41](#)
- FLOWHOLD, [41](#)
- GUIDED, [41](#)
- GUIDED\_NOGPS, [41](#)
- LAND, [41](#)
- LOITER, [41](#)
- POSHOLD, [41](#)
- RTL, [41](#)
- SPORT, [41](#)
- STABILIZE, [41](#)
- THROW, [41](#)
- NextChannelIndex
  - Beken BK2425 radio module, [37](#)
- POSHOLD
  - Main transmitter code, [41](#)
- packetDataDevice\_s, [43](#)
- packetDataDevice\_s::packetDataDevice\_u, [44](#)
- packetDataDeviceCtrl\_s, [44](#)
- packetDataDeviceID\_s, [45](#)
- packetDataDrone\_s, [45](#)
- printf
  - printf functions, [40](#)
  - Utility functions, [27](#)
- printf functions, [40](#)
  - printf, [40](#)
  - vprintf, [40](#)
- Product configuration, [14](#)
- Protocol logical channels, [12](#)
  - BUTTON\_LEFT, [12](#)
  - BUTTON\_LEFT\_SHOULDER, [12](#)
  - BUTTON\_MODE, [12](#)
  - BUTTON\_POWER, [12](#)
  - BUTTON\_RIGHT, [12](#)
  - BUTTON\_RIGHT\_SHOULDER, [12](#)
  - button\_bits, [12](#)
  - channel\_value, [12](#)
  - get\_buttons, [13](#)
- RTL
  - Main transmitter code, [41](#)
- Receive\_Packet
  - Beken BK2425 radio module, [37](#)
- SPI\_FLAG\_BSY
  - Beken BK2425 radio module, [34](#)
- SPI\_FLAG\_CRCERR
  - Beken BK2425 radio module, [34](#)
- SPI\_FLAG\_MODF

- Beken BK2425 radio module, [34](#)
- SPI\_FLAG\_OVR
  - Beken BK2425 radio module, [34](#)
- SPI\_FLAG\_RXNE
  - Beken BK2425 radio module, [34](#)
- SPI\_FLAG\_TXE
  - Beken BK2425 radio module, [34](#)
- SPI\_FLAG\_WKUP
  - Beken BK2425 radio module, [34](#)
- SPORT
  - Main transmitter code, [41](#)
- STABILIZE
  - Main transmitter code, [41](#)
- STICK\_PITCH
  - Analog to Digital Conversion, [9](#)
- STICK\_ROLL
  - Analog to Digital Conversion, [9](#)
- STICK\_THROTTLE
  - Analog to Digital Conversion, [9](#)
- STICK\_YAW
  - Analog to Digital Conversion, [9](#)
- SPI interface to radio chip, [20](#)
  - spi\_force\_chip\_select, [20](#)
  - spi\_read1, [20](#)
  - spi\_read\_registers, [20](#)
  - spi\_transfer, [20](#)
  - spi\_write, [21](#)
- SPI\_Bank1\_Read\_Reg
  - Beken BK2425 radio module, [38](#)
- SPI\_Bank1\_Write\_Reg
  - Beken BK2425 radio module, [38](#)
- SPI\_Flag\_e
  - Beken BK2425 radio module, [34](#)
- SPI\_Read\_Reg
  - Beken BK2425 radio module, [38](#)
- SPI\_Write\_Buf
  - Beken BK2425 radio module, [39](#)
- SPI\_Write\_Cmd
  - Beken BK2425 radio module, [39](#)
- SPI\_Write\_Reg
  - Beken BK2425 radio module, [39](#)
- STM8 hardware interface, [22](#)
- Send\_Packet
  - Beken BK2425 radio module, [37](#)
- SetChannelRange
  - Beken BK2425 radio module, [38](#)
- SetRBank
  - Beken BK2425 radio module, [38](#)
- Sound buzzer module, [11](#)
  - buzzer\_tune, [11](#)
  - tune\_index, [11](#)
- spi\_force\_chip\_select
  - SPI interface to radio chip, [20](#)
- spi\_read1
  - SPI interface to radio chip, [20](#)
- spi\_read\_registers
  - SPI interface to radio chip, [20](#)
- spi\_transfer
  - SPI interface to radio chip, [20](#)
- spi\_write
  - SPI interface to radio chip, [21](#)
- TELEM\_FW
  - Telemetry packet interface, [23](#)
- TELEM\_PLAY
  - Telemetry packet interface, [23](#)
- TELEM\_STATUS
  - Telemetry packet interface, [23](#)
- THROW
  - Main transmitter code, [41](#)
- TXTELEM\_CRC1
  - Telemetry packet interface, [23](#)
- TXTELEM\_CRC2
  - Telemetry packet interface, [23](#)
- TXTELEM\_RSSI
  - Telemetry packet interface, [23](#)
- telem\_firmware, [46](#)
- telem\_packet, [46](#)
- telem\_play, [46](#)
- telem\_status, [47](#)
- telem\_tx\_status, [47](#)
- telem\_type
  - Telemetry packet interface, [23](#)
- Telemetry packet interface, [23](#)
  - TELEM\_FW, [23](#)
  - TELEM\_PLAY, [23](#)
  - TELEM\_STATUS, [23](#)
  - TXTELEM\_CRC1, [23](#)
  - TXTELEM\_CRC2, [23](#)
  - TXTELEM\_RSSI, [23](#)
  - telem\_type, [23](#)
  - tx\_telem\_type, [23](#)
- Timer routines, [24](#)
  - timer\_call\_after\_ms, [24](#)
  - timer\_get\_ms, [24](#)
  - timer\_init, [24](#)
  - timer\_irq, [24](#)
- timer\_call\_after\_ms
  - Timer routines, [24](#)
- timer\_get\_ms
  - Timer routines, [24](#)
- timer\_init
  - Timer routines, [24](#)
- timer\_irq
  - Timer routines, [24](#)
- tune\_index
  - Sound buzzer module, [11](#)
- tx\_telem\_type
  - Telemetry packet interface, [23](#)
- UART input/output, [25](#)
- Utility functions, [26](#)
  - chip\_init, [26](#)
  - delay\_ms, [26](#)
  - delay\_us, [27](#)
  - printf, [27](#)

vprintf  
printf functions, [40](#)