

## StreamingTx Libraries

Generated by Doxygen 1.8.5

Fri Feb 9 2018 11:00:18



# Contents

<b>1</b>	<b>PCB layout from schematic.</b>	<b>1</b>
1.1	GPIO pins	1
1.2	layout from 2016 Spiderman Tx.	1
1.3	GPIO pins	1
1.4	GPIO pins	2
<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	<a href="#">get_buttons_held</a>	13
5.3.3.3	<a href="#">get_buttons_toggled</a>	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	<a href="#">eeprom_flash_copy</a>	16
5.6.2.2	<a href="#">eeprom_read</a>	16
5.6.2.3	<a href="#">eeprom_write</a>	16
5.7	General Purpose Input/Output	18
5.7.1	Detailed Description	18
5.7.2	Enumeration Type Documentation	18
5.7.2.1	<a href="#">gpio_config_e</a>	18
5.7.2.2	<a href="#">gpio_pins_e</a>	19
5.7.3	Function Documentation	19
5.7.3.1	<a href="#">gpio_clear</a>	19
5.7.3.2	<a href="#">gpio_config</a>	19
5.7.3.3	<a href="#">gpio_get</a>	20
5.7.3.4	<a href="#">gpio_set</a>	20
5.7.3.5	<a href="#">gpio_toggle</a>	20
5.8	SPI interface to radio chip	21
5.8.1	Detailed Description	21
5.8.2	Function Documentation	21
5.8.2.1	<a href="#">spi_force_chip_select</a>	21
5.8.2.2	<a href="#">spi_read</a>	21
5.8.2.3	<a href="#">spi_read1</a>	21
5.8.2.4	<a href="#">spi_read_registers</a>	21
5.8.2.5	<a href="#">spi_transfer</a>	22
5.8.2.6	<a href="#">spi_write</a>	22
5.9	STM8 hardware interface	23
5.9.1	Detailed Description	23
5.10	Telemetry packet interface	24
5.10.1	Detailed Description	24
5.10.2	Enumeration Type Documentation	24
5.10.2.1	<a href="#">telem_type</a>	24
5.11	Timer routines	25
5.11.1	Detailed Description	25

5.11.2	Function Documentation	25
5.11.2.1	timer_call_after_ms	25
5.11.2.2	timer_get_ms	25
5.11.2.3	timer_init	25
5.11.2.4	timer_irq	25
5.12	UART input/output	26
5.12.1	Detailed Description	26
5.13	Utility functions	27
5.13.1	Detailed Description	27
5.13.2	Function Documentation	27
5.13.2.1	chip_init	27
5.13.2.2	delay_ms	27
5.13.2.3	delay_us	28
5.13.2.4	printf	28
5.14	Beken BK2425 radio module	29
5.14.1	Detailed Description	32
5.14.2	Enumeration Type Documentation	32
5.14.2.1	anonymous enum	32
5.14.2.2	BK_CONFIG_e	32
5.14.2.3	BK_FEATURE_e	33
5.14.2.4	BK_FIFO_STATUS_e	33
5.14.2.5	BK_PKT_TYPE_E	33
5.14.2.6	BK_SPI_CMD_e	33
5.14.2.7	BK_STATUS_e	34
5.14.2.8	CHANNEL_MHZ_e	35
5.14.2.9	ITX_SPEED_e	35
5.14.2.10	SPI_Flag_e	35
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_ChangeChannel	36
5.14.3.5	BK2425_Initialize	36
5.14.3.6	BK2425_SetCarrierMode	36
5.14.3.7	BK2425_SetRBank	36
5.14.3.8	BK2425_SetSpeed	37
5.14.3.9	BK2425_SetTxPower	37
5.14.3.10	ChangeAddressTx	37
5.14.3.11	Get_Chip_ID	37
5.14.3.12	LookupChannel	37

5.14.3.13	NextChannelIndex	37
5.14.3.14	Receive_Packet	38
5.14.3.15	Send_Packet	38
5.14.3.16	SetChannelRange	38
5.14.3.17	SPI_Bank1_Read_Reg	38
5.14.3.18	SPI_Bank1_Write_Reg	39
5.14.3.19	SPI_Read_Reg	39
5.14.3.20	SPI_Write_Buf	39
5.14.3.21	SPI_Write_Cmd	39
5.14.3.22	SPI_Write_Reg	39
5.14.3.23	UpdateTxData	40
5.15	printf functions	41
5.15.1	Detailed Description	41
5.15.2	Function Documentation	41
5.15.2.1	printf	41
5.15.2.2	vprintf	41
5.16	Main transmitter code	42
5.16.1	Detailed Description	42
5.16.2	Enumeration Type Documentation	42
5.16.2.1	control_mode_t	42
<b>6</b>	<b>Data Structure Documentation</b>	<b>43</b>
6.1	FccParams_s Struct Reference	43
6.1.1	Detailed Description	43
6.2	gpio_regs Struct Reference	43
6.2.1	Detailed Description	44
6.3	packetDataDevice_s Struct Reference	44
6.3.1	Detailed Description	44
6.4	packetDataDevice_s::packetDataDevice_u Union Reference	44
6.4.1	Detailed Description	45
6.5	packetDataDeviceBind_s Struct Reference	45
6.5.1	Detailed Description	45
6.6	packetDataDeviceCtrl_s Struct Reference	45
6.6.1	Detailed Description	46
6.7	packetDataDfu_s Struct Reference	46
6.8	packetDataDrone_s Struct Reference	46
6.8.1	Detailed Description	47
6.9	RadiolInfo_s Struct Reference	47
6.10	RadioStats_s Struct Reference	47
6.11	srt_packet Struct Reference	47

6.12 telem_firmware Struct Reference . . . . .	47
6.12.1 Detailed Description . . . . .	47
6.13 telem_packet_cc2500 Struct Reference . . . . .	48
6.14 telem_packet_cypress Struct Reference . . . . .	48
6.14.1 Detailed Description . . . . .	48
6.15 telem_play Struct Reference . . . . .	48
6.15.1 Detailed Description . . . . .	48
6.16 telem_status Struct Reference . . . . .	48
6.16.1 Detailed Description . . . . .	49

**Index****50**





## Chapter 1

# PCB layout from schematic.

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

The schematic PCB1807 "Streaming and Streaming with GPS Drone button board" v0.1 says

### 1.1 GPIO pins

Port	Meaning	Position
A1	BUTTON_STUNT	(SW4) offboard
A2	BUTTON_VIDEO	(SW5) offboard
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 | (SW3) C2 | USER | (SW6) C3 | RADIO\_IRQ | C4 | RADIO\_CS | C5 | RADIO\_SCK | C6 | RADIO\_MOSI | C7 | RADIO\_MISO | D0 | BUTTON\_MODE | (SW1) D1 | SWIM | D2 | RADIO\_CE | D3 | LED\_GPS | D4 | BEEP | D5 | UART\_TX | D6 | UART\_RX | D7 | LED\_MODE | E5 | BUTTON\_LL | (SW2) F4 | VBAT\_SENSE |

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

### 1.2 layout from 2016 Spiderman Tx.

### 1.3 GPIO pins

Port	Meaning
A1	RADIO_PACTL (B5)
A2	RADIO_CE (D2)
B0	CH4 = ROLL (mode2) RightHorizontal
B1	CH3 = PITCH (mode2) RightVertical
B2	CH1 = THROTTLE (mode2) LeftVertical
B3	CH2 = YAW (mode2) LeftHorizontal
B4	AUDIO_Status (removed)
B5	RADIO_IRQ (C3)
C1	ROW1 (replaced)

C2	ROW2 (replaced)
C3	ROW3 (replaced)
C4	ROW4 (replaced)
C5	RADIO_SCK
C6	RADIO_MOSI
C7	RADIO_MISO
D0	COL1 (replaced)
D1	SWIM
D2	COL2 (replaced)
D3	COL3 (replaced)
D4	BEEP/Audio command
D5	UART_TX
D6	PWR (B4)
D7	LED_MODE
E5	RADIO_CS (C4)
F4	SPEED (removed)

## 1.4 GPIO pins

Meaning	Port2018	Port2016
SWIM	D1	D1
UART_TX	D5	D5
UART_RX	D6	none

PWR | B4 | D6 LED\_MODE | D7 | D7 LED\_GPS | D3 | none

USER | C2 | BUTTON\_STUNT | A1 | none BUTTON\_VIDEO | A2 | none BUTTON\_MODE | D0 | none BUTTON\_LL  
| E5 | none BUTTON\_GPS | C1 | none ROW1 | none | C1 ROW2 | none | C2 ROW3 | none | C3 ROW4 | none |  
C4 COL1 | none | D0 COL2 | none | D2 COL3 | none | D3

Analog input CH4/ROLL | B0 | B0 CH3/PITCH | B1 | B1 CH1/THROTTLE | B2 | B2 CH2/YAW | B3 | B3 VBAT\_SE-  
NSE | F5 | none

SPI RADIO\_SCK | C5 | C5 RADIO\_MOSI | C6 | C6 RADIO\_MISO | C7 | C7 RADIO\_IRQ | C3 | B5 RADIO\_CS |  
C4 | E5 RADIO\_CE | C2 | D2 RADIO\_PACTL | B5

## 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 . . . . .	18
SPI interface to radio chip . . . . .	21
STM8 hardware interface . . . . .	23
Telemetry packet interface . . . . .	24
Timer routines . . . . .	25
UART input/output . . . . .	26
Utility functions . . . . .	27
Beken BK2425 radio module . . . . .	29
printf functions . . . . .	41
Main transmitter code . . . . .	42



## Chapter 4

# Data Structure Index

### 4.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">FccParams_s</a>	
Parameters used by the fcc pretests . . . . .	43
<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) . . . . .	44
<a href="#">packetDataDevice_s::packetDataDevice_u</a>	
< The variant part of the packets . . . . .	44
<a href="#">packetDataDeviceBind_s</a>	
Data for packets that are binding packets Onair order = little-endian . . . . .	45
<a href="#">packetDataDeviceCtrl_s</a>	
Data for packets that are not droneid packets Onair order = little-endian . . . . .	45
<a href="#">packetDataDfu_s</a>	
<a href="#">packetDataDrone_s</a>	
Data structure for data packet transmitted from host (drone) to device (controller) . . . . .	46
<a href="#">RadioInfo_s</a>	
<a href="#">RadioStats_s</a>	
<a href="#">srt_packet</a>	
<a href="#">telem_firmware</a>	
Telemetry packet for the command to write to new firmware . . . . .	47
<a href="#">telem_packet_cc2500</a>	
<a href="#">telem_packet_cypress</a>	
Telemetry packet from RX to TX for cypress . . . . .	48
<a href="#">telem_play</a>	
Telemetry packet for the command to play a tune . . . . .	48
<a href="#">telem_status</a>	
Telemetry status packet . . . . .	48





## Chapter 5

# Module Documentation

### 5.1 Analog to Digital Conversion

#### Enumerations

- enum `adc_channel` { `STICK_ROLL` = 1, `STICK_PITCH` = 0, `STICK_THROTTLE` = 2, `STICK_YAW` = 3 }
- 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. Halts the thread until the tune has completed playing!

##### 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_NONE` = 0x00, `BUTTON_RIGHT` = 0x01, `BUTTON_LEFT` = 0x02, `BUTTON_MIDDLE` = 0x04,  
`BUTTON_LEFT_SHOULDER` = 0x08, `BUTTON_RIGHT_SHOULDER` = 0x10, `BUTTON_POWER` = 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_held` (void)  
*Return a byte that contains a bitset of pressed buttons.*
- uint8\_t `get_buttons_toggled` (void)  
*Return a byte that contains a bitset of toggled 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_NONE`** No buttons are held.  
**`BUTTON_RIGHT`** SW1 = The right button (mode)  
**`BUTTON_LEFT`** SW2 = The left button (launch/land)  
**`BUTTON_MIDDLE`** SW3 = The middle button (GPS)  
**`BUTTON_LEFT_SHOULDER`** SW4 = The left shoulder button (stunt)  
**`BUTTON_RIGHT_SHOULDER`** SW5 = The right shoulder button (video)  
**`BUTTON_POWER`** SW6 = The top button (POWER)

#### 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_held ( 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.

5.3.3.3 `uint8_t get_buttons_toggled ( void )`

Return a byte that contains a bitset of toggled 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.*
- void `eeeprom_flash_copy` (uint16\_t offset,const uint8\_t \*data,uint8\_t len)  
*Write to new firmware location, used for OTA update.*

### 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 void eeeprom\_flash\_copy ( uint16\_t offset, const uint8\_t \* data, uint8\_t len )

Write to new firmware location, used for OTA update.

##### Parameters

<i>offset</i>	The offset of the data within EEPROM
<i>data</i>	The data to write
<i>len</i>	The length of the data to write, in bytes

#### 5.6.2.2 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
---------------	--------------------------------------

#### 5.6.2.3 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\\_e](#) {  
[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\\_e](#) {  
[GPIO\\_INPUT\\_FLOAT](#) = 0x0, [GPIO\\_INPUT\\_PULLUP](#) = 0x2, [GPIO\\_INPUT\\_FLOAT\\_IRQ](#) = 0x1, [GPIO\\_INPUL-  
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\\_e](#) 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\\_e](#)

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\_e

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 #gpio_pins
-------------	---

#### 5.7.3.2 void gpio\_config ( uint16\_t pins, enum gpio\_config\_e 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 #gpio_pins
<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 #gpio_pins
------------	--

**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 #gpio_pins
-------------	--

**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 #gpio_pins
-------------	---

## 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_read` (uint8\_t n,uint8\_t \*buf)  
*Read a number of bytes over the SPI interface.*
- 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 void spi\_read ( uint8\_t n, uint8\_t \* buf )

Read a number of bytes over the SPI interface.

##### Parameters

<i>n</i>	The number of bytes to transfer in each direction over the SPI interface.
<i>buf</i>	A buffer array of bytes to store the data read from the SPI interface. Must not be NULL.

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

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

##### Returns

Returns the input byte.

#### 5.8.2.4 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

5.8.2.5 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.

5.8.2.6 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\\_cypress](#)  
*telemetry packet from RX to TX for cypress*
- struct [telem\\_packet\\_cc2500](#)
- struct [srt\\_packet](#)

### Enumerations

- enum [telem\\_type](#) { [TELEM\\_STATUS](#) = 0, [TELEM\\_PLAY](#) = 1, [TELEM\\_FW](#) = 2 }  
*The type of telemetry packet.*

#### 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



## 5.11 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_gps_set` (bool set)  
*Turn the GPS LED green or red as specified.*
- void `led_mode_set` (bool set)  
*Turn the mode LED up or down as specified.*
- void `led_gps_toggle` (void)  
*Toggle the GPS LED green or red.*
- void `led_mode_toggle` (void)  
*Toggle the mode LED up or down.*
- 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 [packetDataDeviceBind\\_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)*
- struct [packetDataDfu\\_s](#)
- struct [RadioStats\\_s](#)
- struct [FccParams\\_s](#)  
*Parameters used by the fcc pretests.*
- struct [RadioInfo\\_s](#)

### Typedefs

- typedef struct  
[packetDataDeviceCtrl\\_s](#) [packetDataDeviceCtrl](#)  
*Data for packets that are not droneid packets Onair order = little-endian.*
- typedef struct  
[packetDataDeviceBind\\_s](#) [packetDataDeviceBind](#)  
*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.*
- typedef struct [FccParams\\_s](#) [FccParams](#)  
*Parameters used by the fcc pretests.*

### Enumerations

- enum [BK\\_PKT\\_TYPE\\_E](#) {  
[BK\\_PKT\\_TYPE\\_INVALID](#) = 0, [BK\\_PKT\\_TYPE\\_CTRL\\_FOUND](#) = 0x10, [BK\\_PKT\\_TYPE\\_CTRL\\_LOST](#) = 0x11, [BK\\_PKT\\_TYPE\\_BIND](#) = 0x12,  
[BK\\_PKT\\_TYPE\\_TELEMETRY](#) = 0x13, [BK\\_PKT\\_TYPE\\_DFU](#) = 0x14 }  
*The type of packets being sent between controller and drone.*
- enum [BK\\_INFO\\_TYPE\\_E](#)  
*The type of info being sent in control packets.*
- 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 hopping parameters.*

- enum `ITX_SPEED_e` { `ITX_250`, `ITX_1000`, `ITX_2000`, `ITX_CARRIER` }

*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_ACTIVATE_CMD` = 0x50 ,  
`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 `BK2425_ChangeChannel` (uint8\_t channelNumber)  
*Change the radio channel.*
- uint8\_t `LookupChannel` (uint8\_t idx)  
*Convert a logical channel index into a physical channel.*
- void `IWDG_Kick` (void)  
*Kick the independant windowed watchdog so that it does not reset the CPU by timing out.*
- 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.*
- `bool SetChannelRange` (uint8\_t min, uint8\_t max)  
*Set the range of the channel indexes we are using.*
- `uint8_t NextChannelIndex` (uint8\_t seq)  
*Channel hopping algorithm implementation.*
- `void BK2425_SwitchToRxMode` (void)  
*Switch the Beken radio to Rx mode.*
- `void BK2425_SwitchToTxMode` (void)  
*Switch the Beken radio to Tx mode.*
- `void BK2425_SwitchToIdleMode` (void)  
*Switch the Beken radio to Idle mode.*
- `void BK2425_SwitchToSleepMode` (void)  
*Switch the Beken radio to Sleep mode.*
- `void BK2425_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 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 ChangeAddressTx` (uint8\_t txch)  
*Change address.*
- `void BK2425_SetTxPower` (uint8\_t power)  
*Change the radio output power of the Beken radio chip.*
- `void BK2425_SetCarrierMode` (uint8\_t cw)  
*Enable/disable the carrier sending mode.*
- `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.*

  - 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 `UpdateTxData` (void)
- bool `CheckUpdateFccParams` (void)

*From the main thread, we must check to see if (slow) parameter changes are needed.*
- 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 (for debug output)*
- 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

Enumerator

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

#### 5.14.2.2 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.

#### 5.14.2.3 enum BK\_FEATURE\_e

Meanings of the BK\_FEATURE register.

Enumerator

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

#### 5.14.2.4 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.

#### 5.14.2.5 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\_FOUND** (Tx->Drone) User control - known receiver

**BK\_PKT\_TYPE\_CTRL\_LOST** (Tx->Drone) User control - unknown receiver

**BK\_PKT\_TYPE\_BIND** (Tx->Drone) Tell drones this tx is broadcasting

**BK\_PKT\_TYPE\_TELEMETRY** (Drone->Tx) Send telemetry to tx

**BK\_PKT\_TYPE\_DFU** (Drone->Tx) Send new firmware to tx

#### 5.14.2.6 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\_ACTIVATE\_CMD** Must NOT have BK\_WRITE\_REG added to it.

**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.

#### 5.14.2.7 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.

#### 5.14.2.8 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.

#### 5.14.2.9 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)  
**ITX\_CARRIER** 0kbps carrier test

#### 5.14.2.10 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\_ChangeChannel ( uint8\_t *channelNumber* )

Change the radio channel.

## Parameters

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

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

BK2425 initialization of radio registers.

## Parameters

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

5.14.3.6 void BK2425\_SetCarrierMode ( uint8\_t *cw* )

Enable/disable the carrier sending mode.

Must be done on main thread since it is slow

## Parameters

<i>cw</i>	carrier mode
-----------	--------------

5.14.3.7 void BK2425\_SetRBank ( char *cfg* )

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

## Parameters

--

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

#### 5.14.3.8 void BK2425\_SetSpeed ( bool *bFast* )

Change between 250kbps and 2000kbps on the fly.

##### Parameters

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

#### 5.14.3.9 void BK2425\_SetTxPower ( uint8\_t *power* )

Change the radio output power of the Beken radio chip.

Must be done on main thread since it is slow

##### Parameters

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

#### 5.14.3.10 void ChangeAddressTx ( uint8\_t *txch* )

Change address.

##### Parameters

<i>txch</i>	0 for data, 1 for binding
-------------	---------------------------

#### 5.14.3.11 uint8\_t Get\_Chip\_ID ( void )

Get the Beken radio chip ID.

##### Returns

BK\_CHIP\_ID\_BK2425

#### 5.14.3.12 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.
------------	--

#### 5.14.3.13 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
------------	--

**5.14.3.14 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
---------------	--------------------

**5.14.3.15 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

**5.14.3.16 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

**5.14.3.17 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

5.14.3.18 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

5.14.3.19 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>
------------	---

5.14.3.20 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

5.14.3.21 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>
------------	--

5.14.3.22 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

5.14.3.23 void UpdateTxData ( void )

< The packet type



## 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, `FLOWHOLD2` = 22 }  
*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  
**FLOWHOLD2** hold with flow sensor

## Chapter 6

# Data Structure Documentation

### 6.1 FccParams\_s Struct Reference

Parameters used by the fcc pretests.

#### Data Fields

- bool [test\\_mode](#)  
*true iff we are sending test signals*
- bool [scan\\_mode](#)  
*true for scanning, false for fixed frequencies*
- bool [CW\\_mode](#)  
*true for carrier wave, false for packets*
- uint8\_t [scan\\_count](#)  
*In scan mode, packet count before incrementing scan.*
- uint8\_t [channel](#)  
*Current frequency 8..70.*
- uint8\_t [power](#)  
*Current power 0..7.*

#### 6.1.1 Detailed Description

Parameters used by the fcc pretests.

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

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

### 6.2 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.*

### 6.2.1 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`

## 6.3 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.*

### 6.3.1 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`

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

< The variant part of the packets

## Data Fields

- [packetDataDeviceCtrl ctrl](#)  
*Control packets.*
- [packetDataDeviceBind bind](#)  
*Binding packets.*

### 6.4.1 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

## 6.5 packetDataDeviceBind\_s Struct Reference

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

## Data Fields

- `uint8_t bind_address` [SZ\_ADDRESS]  
*The address being used by control packets.*
- `uint8_t hopping`  
*The hopping table in use 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 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\\_held](#)  
*The buttons.*
- uint8\_t [buttons\\_toggled](#)  
*The buttons.*
- uint8\_t [data\\_type](#)  
*Type of extra data being sent.*
- uint8\_t [data\\_value\\_lo](#)  
*Value of extra data being sent.*
- uint8\_t [data\\_value\\_hi](#)  
*Value of extra data being sent.*

### 6.6.1 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

## 6.7 packetDataDfu\_s Struct Reference

### Data Fields

- BK\_PKT\_TYPE [packetType](#)  
*0: The packet type*
- uint8\_t [channel](#)  
*1: Next channel I will broadcast on*
- uint8\_t [address\\_lo](#)  
*2:*
- uint8\_t [address\\_hi](#)  
*3:*
- uint8\_t [data](#) [SZ\_DFU]  
*4...19:*

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

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

## 6.8 packetDataDrone\_s Struct Reference

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

### Data Fields

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

- 2:
  - uint8\_t [rssi](#)
- 3:
  - uint8\_t [droneid](#) [SZ\_CRC\_GUID]
- 4...7:
  - uint8\_t [mode](#)
- 8:

### 6.8.1 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

## 6.9 RadiInfo\_s Struct Reference

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

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

## 6.10 RadioStats\_s Struct Reference

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

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

## 6.11 srt\_packet Struct Reference

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

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

## 6.12 telem\_firmware Struct Reference

Telemetry packet for the command to write to new firmware.

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

### 6.12.1 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

## 6.13 telem\_packet\_cc2500 Struct Reference

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

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

## 6.14 telem\_packet\_cypress Struct Reference

telemetry packet from RX to TX for cypress

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

### Data Fields

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

### 6.14.1 Detailed Description

telemetry packet from RX to TX for cypress

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

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

## 6.15 telem\_play Struct Reference

Telemetry packet for the command to play a tune.

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

### 6.15.1 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

## 6.16 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.*

### 6.16.1 Detailed Description

Telemetry status packet.

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

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

# Index

- ACRO
  - Main transmitter code, [42](#)
- ALT\_HOLD
  - Main transmitter code, [42](#)
- AUTO
  - Main transmitter code, [42](#)
- AUTOTUNE
  - Main transmitter code, [42](#)
- AVOID\_ADSB
  - Main transmitter code, [42](#)
- 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, [34](#)
- BK2425\_R1\_WHOAMI
  - Beken BK2425 radio module, [34](#)
- BK\_ACTIVATE\_CMD
  - Beken BK2425 radio module, [33](#)
- BK\_CD
  - Beken BK2425 radio module, [34](#)
- BK\_CHIP\_ID\_BK2425
  - Beken BK2425 radio module, [32](#)
- BK\_CONFIG
  - Beken BK2425 radio module, [34](#)
- BK\_CONFIG\_CRCO
  - Beken BK2425 radio module, [33](#)
- BK\_CONFIG\_EN\_CRC
  - Beken BK2425 radio module, [33](#)
- BK\_CONFIG\_MASK\_MAX\_RT
  - Beken BK2425 radio module, [32](#)
- BK\_CONFIG\_MASK\_RX\_DR
  - Beken BK2425 radio module, [32](#)
- BK\_CONFIG\_MASK\_TX\_DS
  - Beken BK2425 radio module, [32](#)
- BK\_CONFIG\_PRIM\_RX
  - Beken BK2425 radio module, [33](#)
- BK\_CONFIG\_PWR\_UP
  - Beken BK2425 radio module, [33](#)
- BK\_DYNPD
  - Beken BK2425 radio module, [34](#)
- BK\_EN\_AA
  - Beken BK2425 radio module, [34](#)
- BK\_EN\_RXADDR
  - Beken BK2425 radio module, [34](#)
- BK\_FEATURE
  - Beken BK2425 radio module, [34](#)
- BK\_FEATURE\_EN\_DPL
  - Beken BK2425 radio module, [33](#)
- BK\_FIFO\_STATUS
  - Beken BK2425 radio module, [34](#)
- BK\_FIFO\_STATUS\_RX\_EMPTY
  - Beken BK2425 radio module, [33](#)
- BK\_FIFO\_STATUS\_RX\_FULL
  - Beken BK2425 radio module, [33](#)
- BK\_FIFO\_STATUS\_TX\_EMPTY
  - Beken BK2425 radio module, [33](#)
- BK\_FIFO\_STATUS\_TX\_FULL
  - Beken BK2425 radio module, [33](#)
- BK\_FLUSH\_RX
  - Beken BK2425 radio module, [34](#)
- BK\_FLUSH\_TX
  - Beken BK2425 radio module, [34](#)
- BK\_NOP
  - Beken BK2425 radio module, [34](#)
- BK\_OBSERVE\_TX
  - Beken BK2425 radio module, [34](#)
- BK\_PAYLOAD\_WIDTH
  - Beken BK2425 radio module, [34](#)
- BK\_PKT\_TYPE\_BIND
  - Beken BK2425 radio module, [33](#)
- BK\_PKT\_TYPE\_CTRL\_FOUND
  - Beken BK2425 radio module, [33](#)
- BK\_PKT\_TYPE\_CTRL\_LOST
  - Beken BK2425 radio module, [33](#)
- BK\_PKT\_TYPE\_DFU
  - Beken BK2425 radio module, [33](#)
- BK\_PKT\_TYPE\_INVALID
  - Beken BK2425 radio module, [33](#)
- BK\_PKT\_TYPE\_TELEMETRY
  - Beken BK2425 radio module, [33](#)
- BK\_RD\_RX\_PLOAD
  - Beken BK2425 radio module, [33](#)
- BK\_READ\_REG
  - Beken BK2425 radio module, [33](#)
- BK\_REG\_MASK
  - Beken BK2425 radio module, [33](#)
- BK\_REUSE\_TX\_PL
  - Beken BK2425 radio module, [34](#)
- BK\_RF\_CH

- Beken BK2425 radio module, [34](#)
- BK\_RF\_SETUP
  - Beken BK2425 radio module, [34](#)
- BK\_RX\_ADDR\_P0
  - Beken BK2425 radio module, [34](#)
- BK\_RX\_ADDR\_P1
  - Beken BK2425 radio module, [34](#)
- BK\_RX\_ADDR\_P2
  - Beken BK2425 radio module, [34](#)
- BK\_RX\_ADDR\_P3
  - Beken BK2425 radio module, [34](#)
- BK\_RX\_ADDR\_P4
  - Beken BK2425 radio module, [34](#)
- BK\_RX\_ADDR\_P5
  - Beken BK2425 radio module, [34](#)
- BK\_RX\_PW\_P0
  - Beken BK2425 radio module, [34](#)
- BK\_RX\_PW\_P1
  - Beken BK2425 radio module, [34](#)
- BK\_RX\_PW\_P2
  - Beken BK2425 radio module, [34](#)
- BK\_RX\_PW\_P3
  - Beken BK2425 radio module, [34](#)
- BK\_RX\_PW\_P4
  - Beken BK2425 radio module, [34](#)
- BK\_RX\_PW\_P5
  - Beken BK2425 radio module, [34](#)
- BK\_SETUP\_AW
  - Beken BK2425 radio module, [34](#)
- BK\_SETUP\_RETR
  - Beken BK2425 radio module, [34](#)
- BK\_STATUS
  - Beken BK2425 radio module, [34](#)
- BK\_STATUS\_MAX\_RT
  - Beken BK2425 radio module, [34](#)
- BK\_STATUS\_RBANK
  - Beken BK2425 radio module, [34](#)
- BK\_STATUS\_RX\_DR
  - Beken BK2425 radio module, [34](#)
- BK\_STATUS\_RX\_MASK
  - Beken BK2425 radio module, [34](#)
- BK\_STATUS\_RX\_P\_0
  - Beken BK2425 radio module, [35](#)
- BK\_STATUS\_RX\_P\_1
  - Beken BK2425 radio module, [35](#)
- BK\_STATUS\_RX\_P\_2
  - Beken BK2425 radio module, [35](#)
- BK\_STATUS\_RX\_P\_3
  - Beken BK2425 radio module, [35](#)
- BK\_STATUS\_RX\_P\_4
  - Beken BK2425 radio module, [35](#)
- BK\_STATUS\_RX\_P\_5
  - Beken BK2425 radio module, [35](#)
- BK\_STATUS\_TX\_DS
  - Beken BK2425 radio module, [34](#)
- BK\_STATUS\_TX\_FULL
  - Beken BK2425 radio module, [35](#)
- BK\_TX\_ADDR
  - Beken BK2425 radio module, [34](#)
- BK\_W\_ACK\_PAYLOAD\_CMD
  - Beken BK2425 radio module, [34](#)
- BK\_WR\_TX\_PLOAD
  - Beken BK2425 radio module, [33](#)
- BK\_WRITE\_REG
  - Beken BK2425 radio module, [33](#)
- BRAKE
  - Main transmitter code, [42](#)
- BUTTON\_LEFT
  - Protocol logical channels, [12](#)
- BUTTON\_LEFT\_SHOULDER
  - Protocol logical channels, [12](#)
- BUTTON\_MIDDLE
  - Protocol logical channels, [12](#)
- BUTTON\_NONE
  - Protocol logical channels, [12](#)
- BUTTON\_POWER
  - Protocol logical channels, [12](#)
- BUTTON\_RIGHT
  - Protocol logical channels, [12](#)
- BUTTON\_RIGHT\_SHOULDER
  - Protocol logical channels, [12](#)
- BK2425\_ChangeChannel
  - Beken BK2425 radio module, [36](#)
- BK2425\_Initialize
  - Beken BK2425 radio module, [36](#)
- BK2425\_SetCarrierMode
  - Beken BK2425 radio module, [36](#)
- BK2425\_SetRBank
  - Beken BK2425 radio module, [36](#)
- BK2425\_SetSpeed
  - Beken BK2425 radio module, [37](#)
- BK2425\_SetTxPower
  - Beken BK2425 radio module, [37](#)
- BK\_CONFIG\_e
  - Beken BK2425 radio module, [32](#)
- BK\_FEATURE\_e
  - Beken BK2425 radio module, [33](#)
- BK\_FIFO\_STATUS\_e
  - Beken BK2425 radio module, [33](#)
- BK\_PKT\_TYPE\_E
  - Beken BK2425 radio module, [33](#)
- BK\_SPI\_CMD\_e
  - Beken BK2425 radio module, [33](#)
- BK\_STATUS\_e
  - Beken BK2425 radio module, [34](#)
- Beken BK2425 radio module
  - BK2425\_R1\_12, [34](#)
  - BK2425\_R1\_WHOAMI, [34](#)
  - BK\_ACTIVATE\_CMD, [33](#)
  - BK\_CD, [34](#)
  - BK\_CHIP\_ID\_BK2425, [32](#)
  - BK\_CONFIG, [34](#)
  - BK\_CONFIG\_CRCO, [33](#)
  - BK\_CONFIG\_EN\_CRC, [33](#)
  - BK\_CONFIG\_MASK\_MAX\_RT, [32](#)
  - BK\_CONFIG\_MASK\_RX\_DR, [32](#)

BK\_CONFIG\_MASK\_TX\_DS, [32](#)  
 BK\_CONFIG\_PRIM\_RX, [33](#)  
 BK\_CONFIG\_PWR\_UP, [33](#)  
 BK\_DYNPD, [34](#)  
 BK\_EN\_AA, [34](#)  
 BK\_EN\_RXADDR, [34](#)  
 BK\_FEATURE, [34](#)  
 BK\_FEATURE\_EN\_DPL, [33](#)  
 BK\_FIFO\_STATUS, [34](#)  
 BK\_FIFO\_STATUS\_RX\_EMPTY, [33](#)  
 BK\_FIFO\_STATUS\_RX\_FULL, [33](#)  
 BK\_FIFO\_STATUS\_TX\_EMPTY, [33](#)  
 BK\_FIFO\_STATUS\_TX\_FULL, [33](#)  
 BK\_FLUSH\_RX, [34](#)  
 BK\_FLUSH\_TX, [34](#)  
 BK\_NOP, [34](#)  
 BK\_OBSERVE\_TX, [34](#)  
 BK\_PAYLOAD\_WIDTH, [34](#)  
 BK\_PKT\_TYPE\_BIND, [33](#)  
 BK\_PKT\_TYPE\_CTRL\_FOUND, [33](#)  
 BK\_PKT\_TYPE\_CTRL\_LOST, [33](#)  
 BK\_PKT\_TYPE\_DFU, [33](#)  
 BK\_PKT\_TYPE\_INVALID, [33](#)  
 BK\_PKT\_TYPE\_TELEMETRY, [33](#)  
 BK\_RD\_RX\_PLOAD, [33](#)  
 BK\_READ\_REG, [33](#)  
 BK\_REG\_MASK, [33](#)  
 BK\_REUSE\_TX\_PL, [34](#)  
 BK\_RF\_CH, [34](#)  
 BK\_RF\_SETUP, [34](#)  
 BK\_RX\_ADDR\_P0, [34](#)  
 BK\_RX\_ADDR\_P1, [34](#)  
 BK\_RX\_ADDR\_P2, [34](#)  
 BK\_RX\_ADDR\_P3, [34](#)  
 BK\_RX\_ADDR\_P4, [34](#)  
 BK\_RX\_ADDR\_P5, [34](#)  
 BK\_RX\_PW\_P0, [34](#)  
 BK\_RX\_PW\_P1, [34](#)  
 BK\_RX\_PW\_P2, [34](#)  
 BK\_RX\_PW\_P3, [34](#)  
 BK\_RX\_PW\_P4, [34](#)  
 BK\_RX\_PW\_P5, [34](#)  
 BK\_SETUP\_AW, [34](#)  
 BK\_SETUP\_RETR, [34](#)  
 BK\_STATUS, [34](#)  
 BK\_STATUS\_MAX\_RT, [34](#)  
 BK\_STATUS\_RBANK, [34](#)  
 BK\_STATUS\_RX\_DR, [34](#)  
 BK\_STATUS\_RX\_MASK, [34](#)  
 BK\_STATUS\_RX\_P\_0, [35](#)  
 BK\_STATUS\_RX\_P\_1, [35](#)  
 BK\_STATUS\_RX\_P\_2, [35](#)  
 BK\_STATUS\_RX\_P\_3, [35](#)  
 BK\_STATUS\_RX\_P\_4, [35](#)  
 BK\_STATUS\_RX\_P\_5, [35](#)  
 BK\_STATUS\_TX\_DS, [34](#)  
 BK\_STATUS\_TX\_FULL, [35](#)  
 BK\_TX\_ADDR, [34](#)  
 BK\_W\_ACK\_PAYLOAD\_CMD, [34](#)  
 BK\_WR\_TX\_PLOAD, [33](#)  
 BK\_WRITE\_REG, [33](#)  
 CHANNEL\_FCC\_HIGH, [35](#)  
 CHANNEL\_FCC\_LOW, [35](#)  
 CHANNEL\_FCC\_MID, [35](#)  
 CHANNEL\_MAX\_PHYSICAL, [35](#)  
 CHANNEL\_MIN\_PHYSICAL, [35](#)  
 ITX\_1000, [35](#)  
 ITX\_2000, [35](#)  
 ITX\_250, [35](#)  
 ITX\_CARRIER, [35](#)  
 SPI\_FLAG\_BSY, [35](#)  
 SPI\_FLAG\_CRCERR, [35](#)  
 SPI\_FLAG\_MODF, [35](#)  
 SPI\_FLAG\_OVR, [35](#)  
 SPI\_FLAG\_RXNE, [35](#)  
 SPI\_FLAG\_TXE, [35](#)  
 SPI\_FLAG\_WKUP, [35](#)  
 Beken BK2425 radio module, [29](#)  
     BK2425\_ChangeChannel, [36](#)  
     BK2425\_Initialize, [36](#)  
     BK2425\_SetCarrierMode, [36](#)  
     BK2425\_SetRBank, [36](#)  
     BK2425\_SetSpeed, [37](#)  
     BK2425\_SetTxPower, [37](#)  
     BK\_CONFIG\_e, [32](#)  
     BK\_FEATURE\_e, [33](#)  
     BK\_FIFO\_STATUS\_e, [33](#)  
     BK\_PKT\_TYPE\_E, [33](#)  
     BK\_SPI\_CMD\_e, [33](#)  
     BK\_STATUS\_e, [34](#)  
     beken\_change\_FCC\_channel, [35](#)  
     beken\_set\_CW\_mode, [36](#)  
     beken\_start\_factory\_test, [36](#)  
     CHANNEL\_MHZ\_e, [35](#)  
     ChangeAddressTx, [37](#)  
     Get\_Chip\_ID, [37](#)  
     ITX\_SPEED\_e, [35](#)  
     LookupChannel, [37](#)  
     NextChannelIndex, [37](#)  
     Receive\_Packet, [38](#)  
     SPI\_Bank1\_Read\_Reg, [38](#)  
     SPI\_Bank1\_Write\_Reg, [39](#)  
     SPI\_Flag\_e, [35](#)  
     SPI\_Read\_Reg, [39](#)  
     SPI\_Write\_Buf, [39](#)  
     SPI\_Write\_Cmd, [39](#)  
     SPI\_Write\_Reg, [39](#)  
     Send\_Packet, [38](#)  
     SetChannelRange, [38](#)  
     UpdateTxData, [40](#)  
     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, [35](#)
- CHANNEL\_FCC\_LOW
  - Beken BK2425 radio module, [35](#)
- CHANNEL\_FCC\_MID
  - Beken BK2425 radio module, [35](#)
- CHANNEL\_MAX\_PHYSICAL
  - Beken BK2425 radio module, [35](#)
- CHANNEL\_MIN\_PHYSICAL
  - Beken BK2425 radio module, [35](#)
- CIRCLE
  - Main transmitter code, [42](#)
- CHANNEL\_MHZ\_e
  - Beken BK2425 radio module, [35](#)
- ChangeAddressTx
  - Beken BK2425 radio module, [37](#)
- channel\_value
  - Protocol logical channels, [12](#)
- chip\_init
  - Utility functions, [27](#)
- control\_mode\_t
  - Main transmitter code, [42](#)
- Cyclic Redundancy Check, [15](#)
- DRIFT
  - Main transmitter code, [42](#)
- delay\_ms
  - Utility functions, [27](#)
- delay\_us
  - Utility functions, [28](#)
- EEPROM reading/writing (NOT flash), [16](#)
  - EEPROM flash\_copy, [16](#)
  - EEPROM\_read, [16](#)
  - EEPROM\_write, [16](#)
- EEPROM\_flash\_copy
  - EEPROM reading/writing (NOT flash), [16](#)
- EEPROM\_read
  - EEPROM reading/writing (NOT flash), [16](#)
- EEPROM\_write
  - EEPROM reading/writing (NOT flash), [16](#)
- FLIP
  - Main transmitter code, [42](#)
- FLOWHOLD
  - Main transmitter code, [42](#)
- FLOWHOLD2
  - Main transmitter code, [42](#)
- FccParams\_s, [43](#)
- GPIO\_CLEAR
  - General Purpose Input/Output, [19](#)
- GPIO\_INPUT\_FLOAT
  - General Purpose Input/Output, [18](#)
- GPIO\_INPUT\_FLOAT\_IRQ
  - General Purpose Input/Output, [19](#)
- GPIO\_INPUT\_PULLUP
  - General Purpose Input/Output, [18](#)
- GPIO\_INPUT\_PULLUP\_IRQ
  - General Purpose Input/Output, [19](#)
- GPIO\_OUTPUT\_OPEN\_DRAIN
  - General Purpose Input/Output, [19](#)
- GPIO\_OUTPUT\_OPEN\_DRAIN\_FAST
  - General Purpose Input/Output, [19](#)
- GPIO\_OUTPUT\_PUSHPULL
  - General Purpose Input/Output, [19](#)
- GPIO\_OUTPUT\_PUSHPULL\_FAST
  - General Purpose Input/Output, [19](#)
- GPIO\_PIN0
  - General Purpose Input/Output, [19](#)
- GPIO\_PIN1
  - General Purpose Input/Output, [19](#)
- GPIO\_PIN2
  - General Purpose Input/Output, [19](#)
- GPIO\_PIN3
  - General Purpose Input/Output, [19](#)
- GPIO\_PIN4
  - General Purpose Input/Output, [19](#)
- GPIO\_PIN5
  - General Purpose Input/Output, [19](#)
- GPIO\_PIN6
  - General Purpose Input/Output, [19](#)
- GPIO\_PIN7
  - General Purpose Input/Output, [19](#)
- GPIO\_PORTA
  - General Purpose Input/Output, [19](#)
- GPIO\_PORTB
  - General Purpose Input/Output, [19](#)
- GPIO\_PORTC
  - General Purpose Input/Output, [19](#)
- GPIO\_PORTD
  - General Purpose Input/Output, [19](#)
- GPIO\_PORTE
  - General Purpose Input/Output, [19](#)
- GPIO\_PORTF
  - General Purpose Input/Output, [19](#)
- GPIO\_PORTG
  - General Purpose Input/Output, [19](#)
- GPIO\_PORTH
  - General Purpose Input/Output, [19](#)
- GPIO\_PORTI
  - General Purpose Input/Output, [19](#)
- GPIO\_SET
  - General Purpose Input/Output, [19](#)
- GUIDED
  - Main transmitter code, [42](#)
- GUIDED\_NOGPS
  - Main transmitter code, [42](#)
- General Purpose Input/Output
  - GPIO\_CLEAR, [19](#)
  - GPIO\_INPUT\_FLOAT, [18](#)
  - GPIO\_INPUT\_FLOAT\_IRQ, [19](#)

- GPIO\_INPUT\_PULLUP, 18
- GPIO\_INPUT\_PULLUP\_IRQ, 19
- GPIO\_OUTPUT\_OPEN\_DRAIN, 19
- GPIO\_OUTPUT\_OPEN\_DRAIN\_FAST, 19
- GPIO\_OUTPUT\_PUSHPULL, 19
- GPIO\_OUTPUT\_PUSHPULL\_FAST, 19
- GPIO\_PIN0, 19
- GPIO\_PIN1, 19
- GPIO\_PIN2, 19
- GPIO\_PIN3, 19
- GPIO\_PIN4, 19
- GPIO\_PIN5, 19
- GPIO\_PIN6, 19
- GPIO\_PIN7, 19
- GPIO\_PORTA, 19
- GPIO\_PORTB, 19
- GPIO\_PORTC, 19
- GPIO\_PORTD, 19
- GPIO\_PORTE, 19
- GPIO\_PORTF, 19
- GPIO\_PORTG, 19
- GPIO\_PORTH, 19
- GPIO\_PORTI, 19
- GPIO\_SET, 19
- General Purpose Input/Output, 18
  - gpio\_clear, 19
  - gpio\_config, 19
  - gpio\_config\_e, 18
  - gpio\_get, 20
  - gpio\_pins\_e, 19
  - gpio\_set, 20
  - gpio\_toggle, 20
- Get\_Chip\_ID
  - Beken BK2425 radio module, 37
- get\_buttons\_held
  - Protocol logical channels, 13
- get\_buttons\_toggled
  - Protocol logical channels, 13
- gpio\_clear
  - General Purpose Input/Output, 19
- gpio\_config
  - General Purpose Input/Output, 19
- gpio\_config\_e
  - General Purpose Input/Output, 18
- gpio\_get
  - General Purpose Input/Output, 20
- gpio\_pins\_e
  - General Purpose Input/Output, 19
- gpio\_regs, 43
- gpio\_set
  - General Purpose Input/Output, 20
- gpio\_toggle
  - General Purpose Input/Output, 20
- ITX\_1000
  - Beken BK2425 radio module, 35
- ITX\_2000
  - Beken BK2425 radio module, 35
- ITX\_250
  - Beken BK2425 radio module, 35
- ITX\_CARRIER
  - Beken BK2425 radio module, 35
- ITX\_SPEED\_e
  - Beken BK2425 radio module, 35
- LAND
  - Main transmitter code, 42
- LOITER
  - Main transmitter code, 42
- LookupChannel
  - Beken BK2425 radio module, 37
- Main transmitter code, 42
  - ACRO, 42
  - ALT\_HOLD, 42
  - AUTO, 42
  - AUTOTUNE, 42
  - AVOID\_ADSB, 42
  - BRAKE, 42
  - CIRCLE, 42
  - control\_mode\_t, 42
  - DRIFT, 42
  - FLIP, 42
  - FLOWHOLD, 42
  - FLOWHOLD2, 42
  - GUIDED, 42
  - GUIDED\_NOGPS, 42
  - LAND, 42
  - LOITER, 42
  - POSHOLD, 42
  - RTL, 42
  - SPORT, 42
  - STABILIZE, 42
  - THROW, 42
- NextChannelIndex
  - Beken BK2425 radio module, 37
- POSHOLD
  - Main transmitter code, 42
- packetDataDevice\_s, 44
- packetDataDevice\_s::packetDataDevice\_u, 44
- packetDataDeviceBind\_s, 45
- packetDataDeviceCtrl\_s, 45
- packetDataDfu\_s, 46
- packetDataDrone\_s, 46
- printf
  - printf functions, 41
  - Utility functions, 28
- printf functions, 41
  - printf, 41
  - vprintf, 41
- Product configuration, 14
- Protocol logical channels, 12
  - BUTTON\_LEFT, 12
  - BUTTON\_LEFT\_SHOULDER, 12
  - BUTTON\_MIDDLE, 12
  - BUTTON\_NONE, 12

- BUTTON\_POWER, [12](#)
- BUTTON\_RIGHT, [12](#)
- BUTTON\_RIGHT\_SHOULDER, [12](#)
- button\_bits, [12](#)
- channel\_value, [12](#)
- get\_buttons\_held, [13](#)
- get\_buttons\_toggled, [13](#)
- RTL
  - Main transmitter code, [42](#)
- RadiInfo\_s, [47](#)
- RadioStats\_s, [47](#)
- Receive\_Packet
  - Beken BK2425 radio module, [38](#)
- SPI\_FLAG\_BSY
  - Beken BK2425 radio module, [35](#)
- SPI\_FLAG\_CRCERR
  - Beken BK2425 radio module, [35](#)
- SPI\_FLAG\_MODF
  - Beken BK2425 radio module, [35](#)
- SPI\_FLAG\_OVR
  - Beken BK2425 radio module, [35](#)
- SPI\_FLAG\_RXNE
  - Beken BK2425 radio module, [35](#)
- SPI\_FLAG\_TXE
  - Beken BK2425 radio module, [35](#)
- SPI\_FLAG\_WKUP
  - Beken BK2425 radio module, [35](#)
- SPORT
  - Main transmitter code, [42](#)
- STABILIZE
  - Main transmitter code, [42](#)
- 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, [21](#)
  - spi\_force\_chip\_select, [21](#)
  - spi\_read, [21](#)
  - spi\_read1, [21](#)
  - spi\_read\_registers, [21](#)
  - spi\_transfer, [22](#)
  - spi\_write, [22](#)
- SPI\_Bank1\_Read\_Reg
  - Beken BK2425 radio module, [38](#)
- SPI\_Bank1\_Write\_Reg
  - Beken BK2425 radio module, [39](#)
- SPI\_Flag\_e
  - Beken BK2425 radio module, [35](#)
- SPI\_Read\_Reg
  - Beken BK2425 radio module, [39](#)
- 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, [23](#)
- Send\_Packet
  - Beken BK2425 radio module, [38](#)
- SetChannelRange
  - Beken BK2425 radio module, [38](#)
- Sound buzzer module, [11](#)
  - buzzer\_tune, [11](#)
  - tune\_index, [11](#)
- spi\_force\_chip\_select
  - SPI interface to radio chip, [21](#)
- spi\_read
  - SPI interface to radio chip, [21](#)
- spi\_read1
  - SPI interface to radio chip, [21](#)
- spi\_read\_registers
  - SPI interface to radio chip, [21](#)
- spi\_transfer
  - SPI interface to radio chip, [22](#)
- spi\_write
  - SPI interface to radio chip, [22](#)
- srt\_packet, [47](#)
- TELEM\_FW
  - Telemetry packet interface, [24](#)
- TELEM\_PLAY
  - Telemetry packet interface, [24](#)
- TELEM\_STATUS
  - Telemetry packet interface, [24](#)
- THROW
  - Main transmitter code, [42](#)
- telem\_firmware, [47](#)
- telem\_packet\_cc2500, [48](#)
- telem\_packet\_cypress, [48](#)
- telem\_play, [48](#)
- telem\_status, [48](#)
- telem\_type
  - Telemetry packet interface, [24](#)
- Telemetry packet interface, [24](#)
  - TELEM\_FW, [24](#)
  - TELEM\_PLAY, [24](#)
  - TELEM\_STATUS, [24](#)
  - telem\_type, [24](#)
- Timer routines, [25](#)
  - timer\_call\_after\_ms, [25](#)
  - timer\_get\_ms, [25](#)
  - timer\_init, [25](#)
  - timer\_irq, [25](#)
- timer\_call\_after\_ms
  - Timer routines, [25](#)
- timer\_get\_ms
  - Timer routines, [25](#)
- timer\_init
  - Timer routines, [25](#)
- timer\_irq
  - Timer routines, [25](#)
- tune\_index

- Sound buzzer module, [11](#)
- UART input/output, [26](#)
- UpdateTxData
  - Beken BK2425 radio module, [40](#)
- Utility functions, [27](#)
  - chip\_init, [27](#)
  - delay\_ms, [27](#)
  - delay\_us, [28](#)
  - printf, [28](#)
- vprintf
  - printf functions, [41](#)