StreamingTx Libraries

Generated by Doxygen 1.8.5

Wed Nov 29 2017 13:17:36

Contents

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:

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

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

chan	Which channel are we interested in now. See adc_channel

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

t	The tune number. See tune_index

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

chan	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 eeprom_write (uint16_t offset,uint8_t value)

    Write a byte to the EEPROM (must be unlocked)
- uint8_t eeprom_read (uint16_t offset)

    Read a byte from the EEPROM - just uses normal address space.
- void eeprom_unlock (void)

    Unlock the EEPROM memory before writing.
- void progmem_unlock (void)

    Unlock the program memory before writing.
- void eeprom_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 eeprom_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

offset	The offset of the data within EEPROM

5.6.2.2 void eeprom_write (uint16_t *offset,* uint8_t *value*)

Write a byte to the EEPROM (must be unlocked)

Parameters

offset	The offset of the data within EEPROM
value	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_INPUT_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

pins	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 *config*)

Configure one or more pins on a port.

Parameters

pins	One or more pins to configure on a single specified GPIO port. See gpio_pins
config	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

pin	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

pins	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

pins	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)

  Initialse 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

set	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

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

n	The number of bytes to transfer in each direction over the SPI interface.
sendbuf	The array of bytes to write. If NULL then bytes of value 0 are sent.
recvbuf	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

n	The number of bytes to write
buf	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

dt_ms	The time of the requested callback, in milliseconds
callback	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

d	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

d	The number of microseconds to wait

5.13.2.4 void printf (const char ∗ *fmt,* ...)

Small implementation of the standard printf routine.

Parameters

fmt	The format string. format_string

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

change	?

5.14.3.2 void beken_set_CW_mode (bool *cw*)

Go into continuous carrier wave send mode or normal mode.

Parameters

cw	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

test_mode	The type of test to send.

5.14.3.4 void BK2425_Initialize (ITX_SPEED *spd*)

BK2425 initialization of radio registers.

Parameters

spd	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

bFast	false=slow speed, true=fast speed

5.14.3.6 void ChangeChannel (uint8_t *channelNumber*)

Change the radio channel.

Parameters

channelNumber	A physical radio channel. See CHANNEL_MHZ_e

5.14.3.7 void ChangeOutputPower (uint8_t *power*)

Change the radio output power of the Beken radio chip.

Parameters

power	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

idx	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

seq	The current value of the logical channel index

5.14.3.11 uint8_t Receive_Packet (uint8_t *rx_buf[ ]*)

Read FIFO to read a packet.

Returns

0 if no packet, 1 if packet read

Parameters

rx_buf	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

type	WR_TX_PLOAD or W_TX_PAYLOAD_NOACK_CMD
pbuf	a buffer pointer
len	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

min	The minimum logical channel range
max	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

_cfg	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

reg	A spi register in bank1 to write to BK_SPI_CMD_e
pBuf	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

reg	A spi register in bank1 to write to BK_SPI_CMD_e
pBuf	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

reg	The command to write BK_SPI_CMD_e

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

reg	The command to write BK_SPI_CMD_e
pBuf	The data to write
length	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

reg	The simple command to write BK_SPI_CMD_e

5.14.3.20 void SPI_Write_Reg (uint8_t *reg,* uint8_t *value*)

Writes value ’value’ to register ’reg’.

Parameters

reg	The command to write BK_SPI_CMD_e
value	The data value to write

5.15 printf functions

Functions

- void vprintfl (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

fmt	The format string. format_string

5.15.2.2 void vprintfl (const char ∗ *fmt,* va_list *ap*)

Print a string using a va_list to hold the variable arguments.

Parameters

fmt	The format string. format_string
ap	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-automous 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