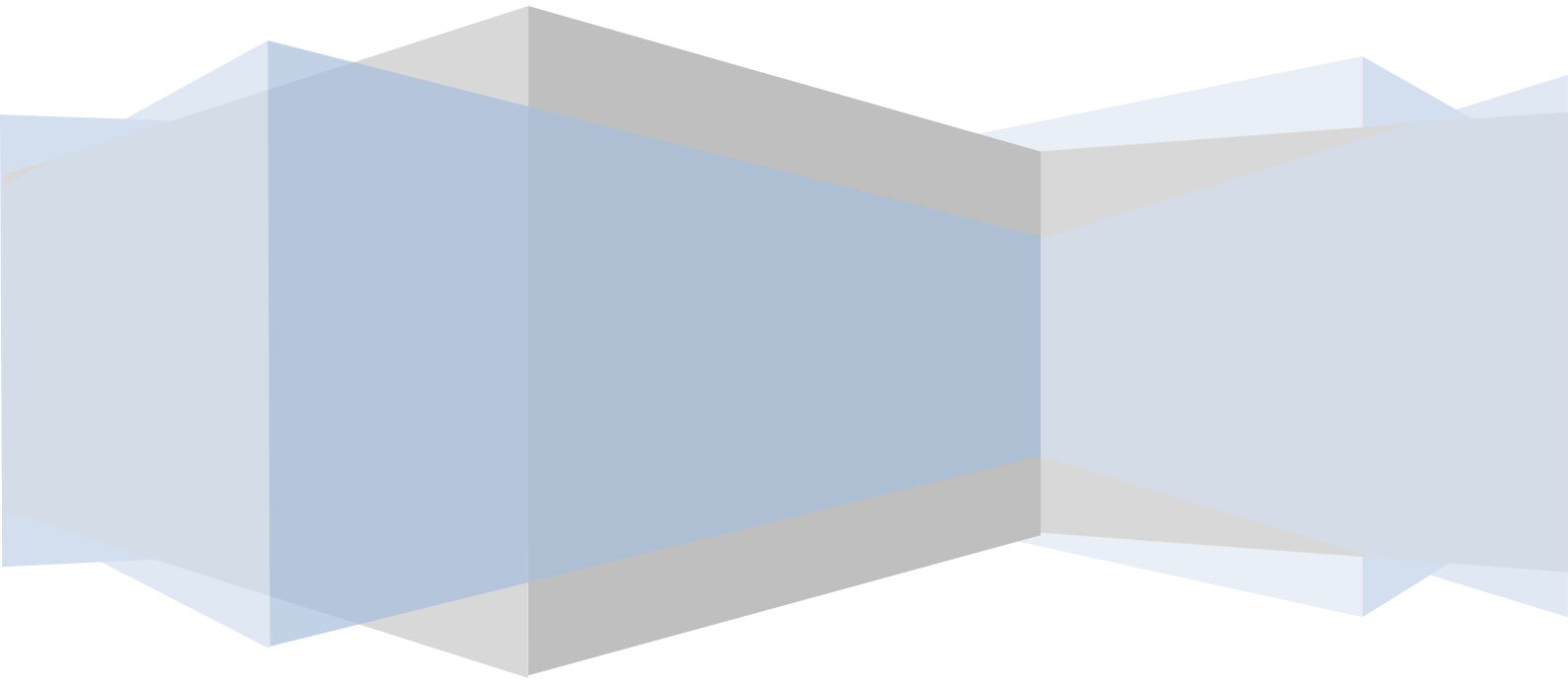
Blox Software Documentation

Dan Cleary, Ankita Kaul, Jesse Tannahill and Zach Wasson



Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Module Documentation

4.1 Drivers

Modules

- **Accelerometer**
- **Counter**
- **Debug**
- **EXTI**
- **FIFO**
- **Filesystem**
- **OLED**
- **Speaker**
- **System**
- **Timer**
- **Touch**
- **USART**
- **USB**
- **VUSART**
- **XBee**

4.2 Accelerometer

- #define **ACCEL_GPIO** GPIOC
- #define **ACCEL_GPIO_CLK** RCC_APB2Periph_GPIOC

- #define **ACCEL_ADC1_CLK** RCC_APB2Periph_ADC1
- #define **ACCEL_DMA_CLK** RCC_AHBPeriph_DMA1
- #define **ACCEL_XOUT_PIN** GPIO_Pin_0
- #define **ACCEL_XOUT_PIN_NUM** 0
- #define **ACCEL_YOUT_PIN** GPIO_Pin_1
- #define **ACCEL_YOUT_PIN_NUM** 1
- #define **ACCEL_ZOUT_PIN** GPIO_Pin_2
- #define **ACCEL_ZOUT_PIN_NUM** 2
- #define **ACCEL_SLEEP_GPIO** GPIOA
- #define **ACCEL_SLEEP_GPIO_CLK** RCC_APB2Periph_GPIOA
- #define **ACCEL_SLEEP_PIN** GPIO_Pin_11
- #define **ACCEL_SLEEP_PIN_NUM** 11
- void **Blox_Accel_Init** (void)

  Initializes the Accelerometer. Initializes the clocks, GPIO pins, DMA location, and ADC configuration. Also sets the sleep pin to high (active low pulse).

- uint16_t **Blox_Accel_GetX** (void)

  Returns the X-value reading of the accelerometer.

- uint16_t **Blox_Accel_GetY** (void)

  Returns the Y-value reading of the accelerometer.

- uint16_t **Blox_Accel_GetZ** (void)

  Returns the Z-value reading of the accelerometer.

- uint8_t **Blox_Accel_GetXTilt** (void)

  Returns the X tilt of the accelerometer.

- uint8_t **Blox_Accel_GetYTilt** (void)

  Returns the Y tilt of the accelerometer.

- uint8_t **Blox_Accel_GetZTilt** (void)

  Returns the Z tilt of the accelerometer.

- #define **ADC1_DR_Address** ((u32)0x4001244C)
- void **Accel_RCC_Configuration** ()

  Initializes the Accelerometer's clocks.

- void **Accel_GPIO_Configuration** ()

Initializes the Accelerometer's GPIO for the ADC and sleep pins.

- void **Accel_DMA_Configuration** ()

  Sets up the DMA for the Accelerometer. The DMA configuration reads to the memory at Accel_Measurements.

- void **Accel_ADC_Configuration** ()

  Initializes the ADC used by the Accelerometer.

- uint16_t **Accel_Measurements** [3]

4.2.1 Detailed Description

The Accelerometer driver

4.2.2 Define Documentation

4.2.2.1 #define ADC1_DR_Address ((u32)0x4001244C)

A Blox contains a MMA7631 3-axis accelerometer. This driver uses an ADC to read its values and return meaningful interpretations to the caller.

Definition at line **18** of file **blox_accel.c**.

4.2.3 Function Documentation

4.2.3.1 void Accel_ADC_Configuration ( )

Initializes the ADC used by the Accelerometer.

Return values

None	

Definition at line **102** of file **blox_accel.c**.

4.2.3.2 void Accel_DMA_Configuration ( )

Sets up the DMA for the Accelerometer. The DMA configuration reads to the memory at Accel_Measurements.

Return values

None	

Definition at line **78** of file **blox_accel.c**.

4.2.3.3 void Accel_GPIO_Configuration ( )

Initializes the Accelerometer's GPIO for the ADC and sleep pins.

Return values

None	

Definition at line **58** of file **blox_accel.c**.

4.2.3.4 void Accel_RCC_Configuration ( )

Initializes the Accelerometer's clocks.

Return values

None	

Definition at line **47** of file **blox_accel.c**.

4.2.3.5 uint16_t Blox_Accel_GetX (void)

Returns the X-value reading of the accelerometer.

Return values

The	X-value of the accelerometer's X axis.

Definition at line **142** of file **blox_accel.c**.

4.2.3.6 uint8_t Blox_Accel_GetXTilt (void)

Returns the X tilt of the accelerometer.

Return values

0	The accelerometer isn't tilted in X.
1	X is tilted in the positive direction.
2	X is tilted in the negative direction.

Definition at line **168** of file **blox_accel.c**.

4.2.3.7 uint16_t Blox_Accel_GetY (void)

Returns the Y-value reading of the accelerometer.

Return values

The	Y-value of the accelerometer's Y axis.

Definition at line **150** of file **blox_accel.c**.

4.2.3.8 uint8_t Blox_Accel_GetYTilt (void)

Returns the Y tilt of the accelerometer.

Return values

0	The accelerometer isn't tilted in Y.
1	Y is tilted in the positive direction.
2	Y is tilted in the negative direction.

Definition at line **185** of file **blox_accel.c**.

4.2.3.9 uint16_t Blox_Accel_GetZ (void)

Returns the Z-value reading of the accelerometer.

Return values

The	Z-value of the accelerometer's Z axis.

Definition at line **158** of file **blox_accel.c**.

4.2.3.10 uint8_t Blox_Accel_GetZTilt (void)

Returns the Z tilt of the accelerometer.

Return values

0	The accelerometer isn't tilted in Z.
1	Z is tilted in the positive direction.
2	Z is tilted in the negative direction.

Definition at line **202** of file **blox_accel.c**.

4.2.3.11 void Blox_Accel_Init (void)

Initializes the Accelerometer. Initializes the clocks, GPIO pins, DMA location, and ADC configuration. Also sets the sleep pin to high (active low pulse).

Return values

None	

Definition at line **34** of file **blox_accel.c**.

4.3 Counter

- void **SysTick_Handler** (void)

  The interrupt handler that updates the global time values.

- void **SysTick_Init** (void)

  Initializes the SysTick driver.

- uint32_t **SysTick_Get_Milliseconds** (void)

  *Returns the number of milliseconds since **SysTick_Init()** (p. 16) was called.*

- uint32_t **SysTick_Get_Seconds** (void)

  *Returns the number of seconds since **SysTick_Init()** (p. 16) was called.*

- uint32_t **SysTick_Get_Minutes** (void)

  *Returns the number of minutes since **SysTick_Init()** (p. 16) was called.*

- void **SysTick_Wait** (uint32_t ms)

  Performs a blocking wait for ms milliseconds.

4.3.1 Detailed Description

The Counter driver for millisecond-resolution time

4.3.2 Function Documentation

4.3.2.1 uint32_t SysTick_Get_Milliseconds (void)

Returns the number of milliseconds since **SysTick_Init()** (p. 16) was called.

Return values

the	number of milliseconds since **SysTick_Init()** (p. 16) was called.

Definition at line **51** of file **blox_counter.c**.

4.3.2.2 uint32_t SysTick_Get_Minutes (void)

Returns the number of minutes since **SysTick_Init()** (p. 16) was called.

Return values

the	number of minutes since **SysTick_Init()** (p. 16) was called.

Definition at line **67** of file **blox_counter.c**.

4.3.2.3 uint32_t SysTick_Get_Seconds (void)

Returns the number of seconds since **SysTick_Init()** (p. 16) was called.

Return values

the	number of seconds since **SysTick_Init()** (p. 16) was called.

Definition at line **59** of file **blox_counter.c**.

4.3.2.4 void SysTick_Handler (void)

The interrupt handler that updates the global time values.

Return values

None.	

Definition at line **85** of file **blox_counter.c**.

4.3.2.5 void SysTick_Init (void)

Initializes the SysTick driver.

Return values

None.	

Definition at line **35** of file **blox_counter.c**.

4.3.2.6 void SysTick_Wait (uint32_t *ms*)

Performs a blocking wait for ms milliseconds.

Parameters

ms	the number of milliseconds to wait

Return values

None.	

Definition at line **76** of file **blox_counter.c**.

4.4 Debug

- #define **BLOX_DEBUG** 0
- #define **Blox_Debug_Init**()
- #define **Blox_Debug**(data)
- #define **Blox_DebugStr**(data)
- #define **Blox_DebugPat**(format,...)

4.5 EXTI

- enum **EXTI_ID** {

EXTI_INVALID_LINE = -2, **EXTI_IRQ_UNAVAILABLE**, **EXTI0**, **EXTI1**,
EXTI2, **EXTI3**, **EXTI4**, **EXTI5**,
EXTI6, **EXTI7**, **EXTI8**, **EXTI9**,
EXTI10, **EXTI11**, **EXTI12**, **EXTI13**,
EXTI14, **EXTI15** }

Enum for possible EXTI ids.

- void **Blox_EXTI_Init** (void)

  Initializes EXTI.

- **EXTI_ID Blox_EXTI_Register_HW_IRQ** (uint8_t GPIO_PortSource, uint8_t line, void(∗EXTI_-Handler)(void))

  Registers an EXTI IRQ that triggers on hardware.

- **EXTI_ID Blox_EXTI_Register_SW_IRQ** (void(∗EXTI_Handler)(void))

  Registers an EXTI IRQ that triggers on software.

- void **Blox_EXTI_Release_IRQ** (**EXTI_ID** id)

  Releases a given EXTI interrupt.

- void **Blox_EXTI_Trigger_SW_IRQ** (**EXTI_ID** id)

  Registers an EXTI IRQ that triggers on software.

- void **Blox_EXTI_Enable_IRQ** (**EXTI_ID** id)

  Enables a given EXTI IRQ.

- void **Blox_EXTI_Disable_IRQ** (**EXTI_ID** id)

  Disables a given EXTI IRQ.

- enum **TOUCH_DIR** { **TOUCH_NORTH_ID** = 1, **TOUCH_SOUTH_ID**, **TOUCH_EAST_ID**, **TOUCH_-WEST_ID** }

  A mapping from touch ids to cardinal directions.

- #define **XBEE_EXTI_LINE** 12
- #define **OLED_EXTI_LINE** 1

- #define **TOUCH1_EXTI_LINE** 10
- #define **TOUCH2_EXTI_LINE** 11
- #define **TOUCH3_EXTI_LINE** 14
- #define **TOUCH4_EXTI_LINE** 7
- void **Blox_EXTI_RCC_Configuration** (void)

  Initializes AFIO clock for the EXTI interface.

- void **Blox_EXTI_NVIC_Configuration** (uint8_t line)

  Initializes NVIC for the EXTI interface.

- uint8_t **isHardwareLine** (uint8_t line)

  Checks whether a line is a designated hardware line.

- void **EXTI0_IRQHandler** (void)

  This function handles External line 0 interrupt request.

- void **EXTI1_IRQHandler** (void)

  This function handles External line 1 interrupt request.

- void **EXTI2_IRQHandler** (void)

  This function handles External line 2 interrupt request.

- void **EXTI3_IRQHandler** (void)

  This function handles External line 3 interrupt request.

- void **EXTI4_IRQHandler** (void)

  This function handles External line 4 interrupt request.

- void **EXTI9_5_IRQHandler** (void)

  This function handles External lines 9 to 5 interrupt request.

- void **EXTI15_10_IRQHandler** (void)

  This function handles External lines 15 to 10 interrupt request.

- void(∗ **EXTIn_Handler** [16])(void) = {NULL}

  Array of pointers for handlers for all 16 EXTI interrupts.

4.5.1 Detailed Description

The EXTI driver for edge interrupts

4.5.2 Function Documentation

4.5.2.1 void Blox_EXTI_Disable_IRQ (EXTI_ID *id*)

Disables a given EXTI IRQ.

Parameters

id	the EXTI_ID for the given EXTI IRQ

Return values

None	

Definition at line **197** of file **blox_exti.c**.

4.5.2.2 void Blox_EXTI_Enable_IRQ (EXTI_ID *id*)

Enables a given EXTI IRQ.

Parameters

id	the EXTI_ID for the given EXTI IRQ

Return values

None	

Definition at line **187** of file **blox_exti.c**.

4.5.2.3 void Blox_EXTI_Init (void)

Initializes EXTI.

Return values

None	

Definition at line **34** of file **blox_exti.c**.

4.5.2.4 void Blox_EXTI_NVIC_Configuration (uint8_t *line*)

Initializes NVIC for the EXTI interface.

Parameters

line,:	specifies the EXTI line to be configured. This parameter can be (0..15).

Return values

None	

Definition at line **140** of file **blox_exti.c**.

4.5.2.5 void Blox_EXTI_RCC_Configuration (void)

Initializes AFIO clock for the EXTI interface.

Return values

None	

Definition at line **130** of file **blox_exti.c**.

4.5.2.6 EXTI_ID Blox_EXTI_Register_HW_IRQ (uint8_t *GPIO_PortSource,* uint8_t *line,* void(∗)(void) *EXTI_Handler*)

Registers an EXTI IRQ that triggers on hardware.

Parameters

GPIO_-PortSource	selects the GPIO port to be used as source for EXTI lines. This parameter can be GPIO_PortSourceGPIOx where x can be (A..G).
line	specifies the EXTI line to be configured. This parameter can be (1,7,10,11,12,14)
EXTI_-Handler	the handler function for the EXTI line.

Return values

EXTI0-EXTI15	on success.
EXTI_INVALID_LINE	if an invalid line is requested.
EXTI_IRQ_-UNAVAILABLE	if an interrupt can't be used.

Definition at line **52** of file **blox_exti.c**.

4.5.2.7 EXTI_ID Blox_EXTI_Register_SW_IRQ (void(*)(void) *EXTI_Handler*)

Registers an EXTI IRQ that triggers on software.

Parameters

EXTI_Handler	the handler function for the EXTI IRQ.

Return values

EXTI0-EXTI15	on success.
EXTI_INVALID_LINE	if an invalid line is requested.
EXTI_IRQ_UNAVAILABLE	if an interrupt can't be used.

Definition at line **75** of file **blox_exti.c**.

4.5.2.8 void Blox_EXTI_Release_IRQ (EXTI_ID *id*)

Releases a given EXTI interrupt.

Parameters

id	specifies the id of the EXTI interrupt

Return values

None	

Definition at line **99** of file **blox_exti.c**.

4.5.2.9 void Blox_EXTI_Trigger_SW_IRQ (EXTI_ID *id*)

Registers an EXTI IRQ that triggers on software.

Parameters

id	the ID of the EXTI IRQ that is to be triggered

Return values

None	

Definition at line **109** of file **blox_exti.c**.

4.5.2.10 void EXTI0_IRQHandler (void)

This function handles External line 0 interrupt request.

Return values

None	

Definition at line **205** of file **blox_exti.c**.

4.5.2.11 void EXTI15_10_IRQHandler (void)

This function handles External lines 15 to 10 interrupt request.

Return values

None	

Definition at line **307** of file **blox_exti.c**.

4.5.2.12 void EXTI1_IRQHandler (void)

This function handles External line 1 interrupt request.

Return values

None	

Definition at line **218** of file **blox_exti.c**.

4.5.2.13 void EXTI2_IRQHandler (void)

This function handles External line 2 interrupt request.

Return values

None	

Definition at line **231** of file **blox_exti.c**.

4.5.2.14 void EXTI3_IRQHandler (void)

This function handles External line 3 interrupt request.

Return values

None	

Definition at line **244** of file **blox_exti.c**.

4.5.2.15 void EXTI4_IRQHandler (void)

This function handles External line 4 interrupt request.

Return values

None	

Definition at line **257** of file **blox_exti.c**.

4.5.2.16 void EXTI9_5_IRQHandler (void)

This function handles External lines 9 to 5 interrupt request.

Return values

None	

Definition at line **270** of file **blox_exti.c**.

4.5.2.17 uint8_t isHardwareLine (uint8_t *line*)

Checks whether a line is a designated hardware line.

Parameters

line	specifies the EXTI line

Return values

1	if the line is a designated hardware line and 0 otherwise

Definition at line **120** of file **blox_exti.c**.

4.6 FIFO

- void **Blox_FIFO_Init** (**FIFO_Type** *fifo)

  *Initializes the **FIFO_Type*** (p. *98*) *structure provided.*

- **FIFO_STATUS Blox_FIFO_Put** (**FIFO_Type** *fifo, uint16_t data)

  Puts data in fifo.

- uint32_t **Blox_FIFO_Get** (**FIFO_Type** *fifo)

  Gets data out of fifo.

- uint8_t **Blox_FIFO_Size** (**FIFO_Type** *fifo)

  Gets size of fifo.

- #define **FIFO_SIZE** 32
- #define **FIFO_BAD** 65536
- enum **FIFO_STATUS** { **FIFO_EMPTY** = -1, **FIFO_OK**, **FIFO_FULL** }

  Enum to return the status of the FIFO driver.

4.6.1 Detailed Description

The FIFO driver.

4.6.2 Function Documentation

4.6.2.1 uint32_t Blox_FIFO_Get (FIFO_Type * *fifo*)

Gets data out of fifo.

Parameters

in	*fifo*	A pointer to the fifo to be initialized

Returns

data or FIFO_BAD if fifo is empty

Definition at line **45** of file **blox_fifo.c**.

4.6.2.2 void Blox_FIFO_Init (FIFO_Type * *fifo*)

Initializes the **FIFO_Type** (p. 98) structure provided.

Parameters

in	*fifo*	A pointer to the fifo to be initialized

Returns

None.

Definition at line **20** of file **blox_fifo.c**.

4.6.2.3 FIFO_STATUS Blox_FIFO_Put (FIFO_Type * *fifo*, uint16_t *data*)

Puts data in fifo.

Parameters

in	*fifo*	A pointer to the fifo to be initialized
in	*data*	The data to be put in the fifo

Returns

status of the FIFO

Definition at line **31** of file **blox_fifo.c**.

4.6.2.4 uint8_t Blox_FIFO_Size (FIFO_Type * *fifo*)

Gets size of fifo.

Parameters

in	*fifo*	A pointer to the fifo to be initialized

Returns

the current size of the fifo

Definition at line **60** of file **blox_fifo.c**.

4.7 Filesystem

- enum **FS_STATUS** {
  FS_CREATE_FAIL = -6, **FS_BAD_WRITE**, **FS_FULL**, **FS_FILE_NOT_INIT**,
  FS_FAT_NOT_INIT, **FS_BAD_FAT**, **FS_OK** }
  Enum containing the status of the filesystem.

- volatile **FS_Table** * **fat** = 0
  The internal pointer to the FAT.

- **FS_STATUS FS_Init** (bool create)
  Initializes the filesystem.

- **FS_STATUS FS_CreateFS** (void)
  Creates a filesystem at the default location. Should only be used once.

- **FS_STATUS FS_ChkValid** (void)
- **FS_File** * **FS_GetFile** (uint8_t id)
  Retrieves a file handle from the filesystem given a specific file id.

- **FS_File** * **FS_GetFileFromName** (char *name)
  Gets a file in the filesystem by looking for the filename.

- uint8_t **FS_GetNumFiles** (void)
  Returns the number of files being managed by the filesystem.

- **FS_STATUS FS_DeleteFile** (uint8_t id)
  Deletes a file from the filesystem. Shifts data up so fragmentation does not occur.

- uint8_t **FS_CreateFile** (char *name, uint8_t numPages)
  Creates a new file of a given size in the filesystem.

- **FS_STATUS FS_WriteFilePage** (uint8_t id, uint32_t *data, uint32_t page_offset)
  Writes data into a file in the filesystem at a given offset.

- void **FS_SwapPage** (uint32_t *src, uint32_t *dst)

Swaps a page in RAM with a page in Flash.

- uint32_t **FS_RoundPageUp** (uint32_t size)

  Rounds a size up to the next multiple of PAGE_SIZE.

- void **FS_RunFile** (uint8_t file_id)

  De-initializes the system and runs the application stored at the file id.

- void **FS_RunStage** (void)

  Runs the application stored in the staging area. Assumes from a system reset.

- uint8_t **FS_GetAppFlag** (void)

  Returns the flag designating if an application should be run.

- void **FS_SetAppFlag** (uint8_t val)

  Sets the flag desinating if an application should be run.

4.7.1 Detailed Description

The Filesystem driver that interacts with Flash

4.7.2 Function Documentation

4.7.2.1 FS_STATUS FS_ChkValid (void)

Determine if the FAT is in a valid state.

Return values

FS_OK	if valid, FS_BAD_FAT on failure

Definition at line **70** of file **blox_filesystem.c**.

4.7.2.2 uint8_t FS_CreateFile (char * *name,* uint8_t *numPages*)

Creates a new file of a given size in the filesystem.

Parameters

name	the name of the new file
numPages	the number of pages the new file needs

Return values

The	unique id of the file within the filesystem. FS_MAX_FILES on error.

Definition at line **187** of file **blox_filesystem.c**.

4.7.2.3 FS_STATUS FS_CreateFS (void)

Creates a filesystem at the default location. Should only be used once.

Return values

FS_OK	if successful, another FS_STATUS if not.

Definition at line **42** of file **blox_filesystem.c**.

4.7.2.4 FS_STATUS FS_DeleteFile (uint8_t *id*)

Deletes a file from the filesystem. Shifts data up so fragmentation does not occur.

Parameters

id	the unique id of the file within the filesystem.

Return values

An	FS_STATUS indicating whether the delete was successful.

Definition at line **145** of file **blox_filesystem.c**.

4.7.2.5 uint8_t FS_GetAppFlag (void)

Returns the flag designating if an application should be run.

Return values

The	flag.

Definition at line **303** of file **blox_filesystem.c**.

4.7.2.6 FS_File* FS_GetFile (uint8_t *id*)

Retrieves a file handle from the filesystem given a specific file id.

Parameters

id	the unique id of the file within the filesystem.

Return values

The	**FS_File** (p. 99) ∗ of the file being requested. 0 on error

Definition at line **108** of file **blox_filesystem.c**.

4.7.2.7 FS_File∗ FS_GetFileFromName (char ∗ *name*)

Gets a file in the filesystem by looking for the filename.

Parameters

name	the name of the application.

Return values

NULL	if the file can't be found, a pointer otherwise.

Definition at line **121** of file **blox_filesystem.c**.

4.7.2.8 uint8_t FS_GetNumFiles (void)

Returns the number of files being managed by the filesystem.

Return values

The	number of files being managed by the filesystem.

Definition at line **137** of file **blox_filesystem.c**.

4.7.2.9 FS_STATUS FS_Init (bool *create*)

Initializes the filesystem.

Return values

FS_OK	if successful, another FS_STATUS if not.

Definition at line **24** of file **blox_filesystem.c**.

4.7.2.10 uint32_t FS_RoundPageUp (uint32_t *size*)

Rounds a size up to the next multiple of PAGE_SIZE.

Parameters

size	the size to be rounded.

Return values

the	rounded size.

Definition at line **253** of file **blox_filesystem.c**.

4.7.2.11 void FS_RunFile (uint8_t *file_id*)

De-initializes the system and runs the application stored at the file id.

Parameters

file_id	the id of the file to be run.

Return values

None.	

Definition at line **265** of file **blox_filesystem.c**.

4.7.2.12 void FS_RunStage (void)

Runs the application stored in the staging area. Assumes from a system reset.

Return values

None.	

Definition at line **282** of file **blox_filesystem.c**.

4.7.2.13 void FS_SetAppFlag (uint8_t *val*)

Sets the flag desinating if an application should be run.

Parameters

val	The flag.

Return values

None.	

Definition at line **312** of file **blox_filesystem.c**.

4.7.2.14 void FS_SwapPage (uint32_t * *src,* uint32_t * *dst*)

Swaps a page in RAM with a page in Flash.

Parameters

src	the address of the start of the page in RAM
dst	the address of the start of the page in Flash

Return values

None.	

Definition at line **240** of file **blox_filesystem.c**.

4.7.2.15 FS_STATUS FS_WriteFilePage (uint8_t *id,* uint32_t * *data,* uint32_t *page_offset*)

Writes data into a file in the filesystem at a given offset.

Parameters

id	the unique id of the file within the filesystem
data	a pointer to the buffer to be copied
page_offset	the offset within the file the buffer should be copied to.

Return values

An	FS_STATUS denoting whether the write was successful.

Definition at line **227** of file **blox_filesystem.c**.

4.8 OLED

- void **OLED_RCC_Configuration** ()

  Initializes clocks for OLED reset pin.

- void **OLED_GPIO_Configuration** ()

  Initializes the gpio for the OLED reset pin.

- void **OLED_ResetDisplay** (void)
- void **OLED_Reset** (void)

  Switches the Reset pin PC3 for the OLED display.

- void **Blox_OLED_Init** (void)

  Initializes the OLED display.

- uint8_t **Blox_OLED_Receive** (void)

  Receive a byte from the OLED. Wrapper around USART.

- void **Blox_OLED_Send** (uint8_t data)

  Send a byte to the OLED. Wrapper around USART.

- void **Blox_OLED_Clear** (void)

  Clear OLED display.

- void **Blox_OLED_DrawCircle** (uint8_t x, uint8_t y, uint8_t radius, uint16_t color)

  Draw circle on OLED display.

- void **Blox_OLED_DrawLine** (uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, uint16_t color)

  Draw line on OLED display.

- void **Blox_OLED_DrawPixel** (uint8_t x, uint8_t y, uint16_t color)

  Draw pixel on OLED display.

- void **Blox_OLED_DrawRectangle** (uint8_t x, uint8_t y, uint8_t width, uint8_t height, uint16_t color)

  Draw line on OLED display.

- void **Blox_OLED_SetFont** (uint8_t font)

  Sets font type for OLED characters.

- void **Blox_OLED_SetOpaque** (void)

  Sets the font to have an opaque background.

- void **Blox_OLED_DrawStringGraphics** (uint8_t x, uint8_t y, uint8_t font, uint16_t color, uint8_t width, uint8_t height, uint8_t ∗string)

  Draw graphics formatted string on OLED display.

- void **Blox_OLED_DrawCharText** (uint8_t character, uint8_t column, uint8_t row, uint16_t color)

  Draw text formatted character on OLED display.

- void **Blox_OLED_DrawStringText** (uint8_t column, uint8_t row, uint8_t font_size, uint16_t color, uint8_t ∗string)

  Draw text formatted string on OLED display.

- void **Blox_OLED_DrawCharGraphics** (uint8_t character, uint8_t x, uint8_t y, uint16_t color, uint8_t width, uint8_t height)

  Draw graphics formatted character on OLED display.

- void **Blox_OLED_SD_DisplayIcon** (uint8_t x, uint8_t y, uint8_t width, uint8_t height, uint32_t sector)

  Displays bitmap image icon stored in SD card onto OLED screen.

- #define **OLED_USART_ID** 2
- #define **OLED_RESET_GPIO** GPIOC
- #define **OLED_RESET_GPIO_CLK** RCC_APB2Periph_GPIOC
- #define **OLED_RESET_PIN** GPIO_Pin_3
- #define **OLED_RESET_PIN_NUM** 3
- #define **DELAY_2N** 0
- #define **OLED_BAUDRATE** 9600
- #define **OLED_RESETPIN** 8
- #define **OLED_INITDELAYMS** 5000
- #define **OLED_ACK** 0x06
- #define **OLED_NAK** 0x15
- #define **OLED_AUTOBAUD** 0x55
- #define **OLED_DEVICE_INFO** 0x56
- #define **OLED_BKG_COLOR** 0x42
- #define **OLED_CLEAR** 0x45
- #define **OLED_DISPLAY_CONTROL** 0x59
- #define **OLED_COMMAND_DISPLAY** 0x01
- #define **OLED_COMMAND_CONTRAST** 0x02
- #define **OLED_COMMAND_POWER** 0x03
- #define **OLED_SLEEP** 0x5A
- #define **OLED_ADD_USER_CHAR** 0x41
- #define **OLED_DRAW_CIRCLE** 0x43
- #define **OLED_DRAW_USER_CHAR** 0x44

- #define **OLED_DRAW_TRIANGLE** 0x47
- #define **OLED_DRAW_ICON** 0x49
- #define **OLED_OPAQUE_BKG_COLOR** 0x4B
- #define **OLED_DRAW_LINE** 0x4C
- #define **OLED_DRAW_PIXEL** 0x50
- #define **OLED_READ_PIXEL** 0x52
- #define **OLED_SCREEN_COPY_PASTE** 0x63
- #define **OLED_DRAW_POLYGON** 0x67
- #define **OLED_REPLACE_COLOR** 0x6B
- #define **OLED_SET_PIN_SIZE** 0x70
- #define **OLED_DRAW_RECT** 0x72
- #define **OLED_SET_FONT** 0x46
- #define **OLED_FONT_5X7** 0x00
- #define **OLED_FONT_8X8** 0x01
- #define **OLED_FONT_8X12** 0x02
- #define **OLED_SET_VIS** 0x4F
- #define **OLED_SET_VIS_TRANS** 0x00
- #define **OLED_SET_VIS_OPAQ** 0x01
- #define **OLED_DRAW_STRING_GRAPHICS** 0x53
- #define **OLED_DRAW_CHAR_TEXT** 0x54
- #define **OLED_DRAW_TEXT_BUTTON** 0x62
- #define **OLED_DRAW_STRING_TEXT** 0x73
- #define **OLED_DRAW_CHAR_GRAPHICS** 0x74
- #define **OLED_SD_CMD_EXT** 0x40
- #define **OLED_SD_CMODE_256** 0x08
- #define **OLED_SD_CMODE_65K** 0x10
- #define **OLED_SD_SET_ADDRESS_PNT** 0x41
- #define **OLED_SD_SCREEN_SAVE** 0x43
- #define **OLED_SD_DISPLAY_ICON** 0x49
- #define **OLED_SD_DISPLAY_OBJECT** 0x4F
- #define **OLED_SD_RUN_SCRIPT** 0x50
- #define **OLED_SD_READ_SECTOR** 0x52
- #define **OLED_SD_DISPLAY_VID** 0x56
- #define **OLED_SD_WRITE_SECTOR** 0x57
- #define **OLED_SD_INIT_CARD** 0x69
- #define **OLED_SD_READ_BYTE** 0x72
- #define **OLED_SD_WRITE_BYTE** 0x77
- #define **OLED_IMG_0** 0x001000
- #define **OLED_IMG_1** 0x001010
- #define **OLED_IMG_2** 0x001010
- #define **OLED_IMG_3** 0x001010
- #define **OLED_IMG_4** 0x001010
- #define **OLED_IMG_5** 0x001010

- #define **OLED_IMG_6** 0x001010
- #define **OLED_IMG_7** 0x001010
- #define **OLED_IMG_8** 0x001010
- #define **OLED_IMG_9** 0x001010
- #define **OLED_IMG_10** 0x001010
- #define **OLED_IMG_11** 0x001010
- #define **OLED_IMG_12** 0x001010
- #define **OLED_IMG_13** 0x001010
- #define **OLED_IMG_14** 0x001010
- #define **COLOR_BLACK** 0x0000
- #define **COLOR_BLUE** 0x001F
- #define **COLOR_WHITE** 0xFFFF
- #define **COLOR_YELLOW** 0xFFE0
- #define **COLOR_GRAPE** 0x8210
- #define **COLOR_LIGHT_BROWN** 0xBAA7
- #define **COLOR_DARK_BROWN** 0x7000

4.8.1 Detailed Description

The OLED display for ouputting to the display

4.8.2 Function Documentation

4.8.2.1 void Blox_OLED_Clear (void)

Clear OLED display.

Return values

none.	

Definition at line **100** of file **blox_oled.c**.

4.8.2.2 void Blox_OLED_DrawCharGraphics (uint8_t *character,* uint8_t *x,* uint8_t *y,* uint16_t *color,* uint8_t *width,* uint8_t *height*)

Draw graphics formatted character on OLED display.

Parameters

character	Character to display
x	X-coordinate of top left corner of character

y	Y-coordinate of top left corner of character
color	2 byte RGB color of pixel.
width	width multiplier of default width
height	height multiplier of default height

Return values

None	

Definition at line **318** of file **blox_oled.c**.

4.8.2.3 void Blox_OLED_DrawCharText (uint8_t *character,* uint8_t *column,* uint8_t *row,* uint16_t *color*)

Draw text formatted character on OLED display.

Parameters

character,:	Character to display
column	Horizontal start position of text. range: 0 - 20 for 5x7 range: 0 - 15 for 8x8 and 8x12
row	Vertical start position of text. range: 0 - 15 for 5x7 range: 0 - 9 for 8x8 and 8x12
color	2 byte RGB color of pixel.

Return values

None	

Definition at line **261** of file **blox_oled.c**.

4.8.2.4 void Blox_OLED_DrawCircle (uint8_t *x,* uint8_t *y,* uint8_t *radius,* uint16_t *color*)

Draw circle on OLED display.

Parameters

x	X-coordinate of circle's center.
y	Y-coordinate of circle's center.
radius	Radius of circle.
color	2 byte RGB color of circle.

Return values

None	

Definition at line **117** of file **blox_oled.c**.

4.8.2.5 void Blox_OLED_DrawLine (uint8_t *x1,* uint8_t *y1,* uint8_t *x2,* uint8_t *y2,* uint16_t *color*)

Draw line on OLED display.

Parameters

x1	X-coordinate of line's start position.
y1	Y-coordinate of line's start position.
x2	X-coordinate of line's end position.
y2	Y-coordinate of line's end position.
color	2 byte RGB color of pixel.

Return values

None	

Definition at line **136** of file **blox_oled.c**.

4.8.2.6 void Blox_OLED_DrawPixel (uint8_t *x,* uint8_t *y,* uint16_t *color*)

Draw pixel on OLED display.

Parameters

x	X-coordinate of pixel.
y	Y-coordinate of pixel.
color	2 byte RGB color of pixel.

Return values

None	

Definition at line **154** of file **blox_oled.c**.

4.8.2.7 void Blox_OLED_DrawRectangle (uint8_t *x1,* uint8_t *y1,* uint8_t *x2,* uint8_t *y2,* uint16_t *color*)

Draw line on OLED display.

Parameters

x1	X-coordinate of top-left corner of rectangle.

y1	Y-coordinate of top-left corner of rectangle.
x2	X-coordinate of bottom-right corner of rectangle.
y2	Y-coordinate of bottom-right corner of rectangle.
color	2 byte RGB color of pixel.

Return values

None	

Definition at line **172** of file **blox_oled.c**.

4.8.2.8 void Blox_OLED_DrawStringGraphics (uint8_t *x,* uint8_t *y,* uint8_t *font,* uint16_t *color,* uint8_t *width,* uint8_t *height,* uint8_t * *string*)

Draw graphics formatted string on OLED display.

Parameters

x	X-coordinate of top left corner of character
y	Y-coordinate of top left corner of character range: 0 - 9 for 8x8 and 8x12
font	Font type. • OLED_FONT_5X7 • OLED_FONT_8X8 • OLED_FONT_8x12
color	2 byte RGB color of pixel.
width	width multiplier of default width
height	height multiplier of default height
string	Pointer to string of text to be displayed.

Return values

None	

Definition at line **226** of file **blox_oled.c**.

4.8.2.9 void Blox_OLED_DrawStringText (uint8_t *column,* uint8_t *row,* uint8_t *font,* uint16_t *color,* uint8_t * *string*)

Draw text formatted string on OLED display.

Parameters

column	Horizontal start position of text. range: 0 - 20 for 5x7 range: 0 - 15 for 8x8 and 8x12
row	Vertical start position of text. range: 0 - 15 for 5x7 range: 0 - 9 for 8x8 and 8x12
font	Font type. • OLED_FONT_5X7 • OLED_FONT_8X8 • OLED_FONT_8x12
color	2 byte RGB color of pixel.
string	Pointer to string of text to be displayed.

Return values

None	

Definition at line **287** of file **blox_oled.c**.

4.8.2.10 void Blox_OLED_Init (void)

Initializes the OLED display.

Return values

None	

Definition at line **37** of file **blox_oled.c**.

4.8.2.11 uint8_t Blox_OLED_Receive (void)

Receive a byte from the OLED. Wrapper around USART.

Return values

The	received command or 0 on error.

Definition at line **77** of file **blox_oled.c**.

4.8.2.12 void Blox_OLED_SD_DisplayIcon (uint8_t *x,* uint8_t *y,* uint8_t *width,* uint8_t *height,* uint32_t *sector*)

Displays bitmap image icon stored in SD card onto OLED screen.

Parameters

x	X-coordinate of top left corner of rectangle.
y	Y-coordinate of top left corner of rectangle.
width	Width of rectangle.
height	Height of rectangle.
sector	the sector where the icon is located

Return values

None	

Definition at line **343** of file **blox_oled.c**.

4.8.2.13 void Blox_OLED_Send (uint8_t *data*)

Send a byte to the OLED. Wrapper around USART.

Parameters

data	The byte to send.

Return values

None	

Definition at line **88** of file **blox_oled.c**.

4.8.2.14 void Blox_OLED_SetFont (uint8_t *font*)

Sets font type for OLED characters.

Parameters

font,:	Font type. • OLED_FONT_5X7 • OLED_FONT_8X8 • OLED_FONT_8x12

Return values

None	

Definition at line **195** of file **blox_oled.c**.

4.8.2.15 void Blox_OLED_SetOpaque (void)

Sets the font to have an opaque background.

Return values

None	

Definition at line **205** of file **blox_oled.c**.

4.8.2.16 void OLED_GPIO_Configuration ( )

Initializes the gpio for the OLED reset pin.

Return values

None	

Definition at line **63** of file **blox_oled.c**.

4.8.2.17 void OLED_RCC_Configuration ( )

Initializes clocks for OLED reset pin.

Return values

None	

Definition at line **55** of file **blox_oled.c**.

4.8.2.18 void OLED_Reset (void)

Switches the Reset pin PC3 for the OLED display.

Return values

None	

Definition at line **25** of file **blox_oled.c**.

4.9 Speaker

- void **Sin_gen** (void)

  Generates the sine wave for the treble clef.

- void **Blox_Speaker_Init** (void)

  Initializes the speaker.

- void **PlayMusic** (void)

  Plays the music included in blox_music.h.

- void **StopMusic** (void)

  Stops the music.

- #define **SpkClock** 52400
- #define **SINE_POINTS** 32
- #define **C7** SpkClock/2093/SINE_POINTS
- #define **B6** SpkClock/1975/SINE_POINTS
- #define **Bb6** SpkClock/1865/SINE_POINTS
- #define **A6** SpkClock/1760/SINE_POINTS
- #define **Ab6** SpkClock/1661/SINE_POINTS
- #define **G6** SpkClock/1567/SINE_POINTS
- #define **Gb6** SpkClock/1480/SINE_POINTS
- #define **F6** SpkClock/1396/SINE_POINTS
- #define **E6** SpkClock/1319/SINE_POINTS
- #define **Eb6** SpkClock/1245/SINE_POINTS
- #define **D6** SpkClock/1175/SINE_POINTS
- #define **Db6** SpkClock/1108/SINE_POINTS
- #define **C6** SpkClock/1046/SINE_POINTS
- #define **B5** SpkClock/988/SINE_POINTS
- #define **Bb5** SpkClock/932/SINE_POINTS
- #define **A5** SpkClock/880/SINE_POINTS
- #define **Ab5** SpkClock/830/SINE_POINTS
- #define **G5** SpkClock/784/SINE_POINTS
- #define **Gb5** SpkClock/740/SINE_POINTS
- #define **F5** SpkClock/698/SINE_POINTS
- #define **E5** SpkClock/659/SINE_POINTS

- #define **Eb5** SpkClock/622/SINE_POINTS
- #define **D5** SpkClock/587/SINE_POINTS
- #define **Db5** SpkClock/554/SINE_POINTS
- #define **C5** SpkClock/523/SINE_POINTS
- #define **B4** SpkClock/494/SINE_POINTS
- #define **Bb4** SpkClock/466/SINE_POINTS
- #define **A4** SpkClock/440/SINE_POINTS
- #define **Ab4** SpkClock/415/SINE_POINTS
- #define **G4** SpkClock/392/SINE_POINTS
- #define **Gb4** SpkClock/370/SINE_POINTS
- #define **F4** SpkClock/349/SINE_POINTS
- #define **E4** SpkClock/330/SINE_POINTS
- #define **Eb4** SpkClock/311/SINE_POINTS
- #define **D4** SpkClock/294/SINE_POINTS
- #define **Db4** SpkClock/277/SINE_POINTS
- #define **C4** SpkClock/262/SINE_POINTS
- #define **B3** SpkClock/247/SINE_POINTS
- #define **Bb3** SpkClock/233/SINE_POINTS
- #define **A3** SpkClock/220/SINE_POINTS
- #define **Ab3** SpkClock/208/SINE_POINTS
- #define **G3** SpkClock/196/SINE_POINTS
- #define **Gb3** SpkClock/185/SINE_POINTS
- #define **F3** SpkClock/175/SINE_POINTS
- #define **E3** SpkClock/165/SINE_POINTS
- #define **Eb3** SpkClock/155/SINE_POINTS
- #define **D3** SpkClock/147/SINE_POINTS
- #define **Db3** SpkClock/139/SINE_POINTS
- #define **C3** SpkClock/131/SINE_POINTS
- #define **B2** SpkClock/123/SINE_POINTS
- #define **Bb2** SpkClock/117/SINE_POINTS
- #define **A2** SpkClock/110/SINE_POINTS
- #define **Ab2** SpkClock/104/SINE_POINTS
- #define **G2** SpkClock/98/SINE_POINTS
- #define **Gb2** SpkClock/93/SINE_POINTS
- #define **F2** SpkClock/87/SINE_POINTS
- #define **E2** SpkClock/82/SINE_POINTS
- #define **Eb2** SpkClock/78/SINE_POINTS
- #define **D2** SpkClock/73/SINE_POINTS
- #define **Db2** SpkClock/69/SINE_POINTS
- #define **C2** SpkClock/65/SINE_POINTS
- #define **BEAT** (SpkClock/16)-1
- typedef struct **Note NoteType**
- typedef **NoteType** ∗ **NotePtr**

4.9.1 Detailed Description

The Speaker driver

4.9.2 Function Documentation

4.9.2.1 void Blox_Speaker_Init (void)

Initializes the speaker.

Return values

None	

Definition at line **51** of file **blox_speaker.c**.

4.9.2.2 void PlayMusic (void)

Plays the music included in blox_music.h.

Return values

None	

Definition at line **73** of file **blox_speaker.c**.

4.9.2.3 void Sin_gen (void)

Generates the sine wave for the treble clef.

Return values

None	

Definition at line **107** of file **blox_speaker.c**.

4.9.2.4 void StopMusic (void)

Stops the music.

Return values

None	

Definition at line **94** of file **blox_speaker.c**.

4.10 System

- **SysVar** ∗ **sys** = (**SysVar** ∗)MEM_SYS_VAR_START

- #define **TRUE** 1
- #define **FALSE** 0
- #define **NULL** 0
- #define **PAGE_SIZE** 0x800
- #define **WORD_SIZE** 0x4
- #define **MEM_MAP_START** 0x08000000
- #define **MEM_MAP_SIZE** 0x08080000
- #define **MEM_BASE_PROG_START** 0x08000000
- #define **MEM_BASE_PROG_SIZE** PAGE_SIZE∗32
- #define **MEM_SYS_VAR_START** 0x08010000
- #define **MEM_SYS_VAR_SIZE** PAGE_SIZE
- #define **MEM_STAGE_START** 0x08010800
- #define **MEM_START_SIZE** PAGE_SIZE∗32
- #define **MEM_FAT_START** 0x08020800
- #define **MEM_FAT_SIZE** PAGE_SIZE
- #define **MEM_STORE_START** 0x08021000
- #define **MEM_STORE_SIZE** PAGE_SIZE∗190
- #define **MAX_DEINIT_FN** 128
- #define **SYS_MAGIC** 0xCAFEBABE
- #define **SYS_INV_ID** 0xFFFFFFFF
- typedef void(∗ **ptrVoidFn**)(void)
- void **Blox_System_Init** (void)

  Initializes the pointer to the system variables array.

- void **Blox_System_Create** (void)

  *Creates an initial **SysVar** (*p. *110)* *struct.*

- void **Blox_System_DeInit** (void)

  De-initializes all the peripherals in the system.

- void **Blox_System_Register_DeInit** (ptrVoidFn fn)

Registers a new function to be called when the system deInits. Only adds if it isn't already there.

- uint32_t **Blox_System_GetId** (void)

  *Returns the current **SysVar** (p. 110) struct.*

- void **Blox_System_GetVars** (**SysVar** *retSys)

  *Returns the current **SysVar** (p. 110) struct.*

- void **Blox_System_WriteVars** (**SysVar** *newVars)

  *Writes over the **SysVar** (p. 110) in flash.*

4.10.1 Detailed Description

The system driver for accessing system variables

4.10.2 Define Documentation

4.10.2.1 #define MEM_MAP_START 0x08000000

Memory Map

Definition at line **40** of file **blox_system.h**.

4.10.3 Function Documentation

4.10.3.1 void Blox_System_Create (void)

Creates an initial **SysVar** (p. 110) struct.

Return values

None.	

Definition at line **72** of file **blox_system.c**.

4.10.3.2 void Blox_System_DeInit (void)

De-initializes all the peripherals in the system.

Return values

None	

Definition at line **40** of file **blox_system.c**.

4.10.3.3 uint32_t Blox_System_GetId (void)

Returns the current **SysVar** (p. 110) struct.

Return values

the	blox's id

Definition at line **97** of file **blox_system.c**.

4.10.3.4 void Blox_System_GetVars (SysVar * *retSys*)

Returns the current **SysVar** (p. 110) struct.

Return values

the	current **SysVar** (p. 110) struct

Definition at line **105** of file **blox_system.c**.

4.10.3.5 void Blox_System_Init (void)

Initializes the pointer to the system variables array.

Return values

None	

Definition at line **25** of file **blox_system.c**.

4.10.3.6 void Blox_System_Register_DeInit (ptrVoidFn *fn*)

Registers a new function to be called when the system deInits. Only adds if it isn't already there.

Parameters

fn,:	a fn pointer to a module's deinit function

Return values

None	

Definition at line **52** of file **blox_system.c**.

4.10.3.7 void Blox_System_WriteVars (SysVar * *newVars*)

Writes over the **SysVar** (p. 110) in flash.

Parameters

newVars,:	a **SysVar** (p. 110) with the new contents to be stored in flash.

Return values

None.	

Definition at line **114** of file **blox_system.c**.

4.11 Timer

- #define **TIM1_CLK** RCC_APB2Periph_TIM1
- #define **TIM2_CLK** RCC_APB1Periph_TIM2
- #define **TIM3_CLK** RCC_APB1Periph_TIM3
- #define **TIM4_CLK** RCC_APB1Periph_TIM4
- #define **TIM5_CLK** RCC_APB1Periph_TIM5
- #define **TIM6_CLK** RCC_APB1Periph_TIM6
- #define **TIM7_CLK** RCC_APB1Periph_TIM7
- #define **TIM8_CLK** RCC_APB2Periph_TIM8
- enum **TIMER_ID** {

  INVALID_TIMER = -2, **IRQ_UNAVAILABLE**, **TIM1UP**, **TIM1CH1**,

  TIM1CH2, **TIM1CH3**, **TIM1CH4**, **TIM2CH1**,

  TIM2CH2, **TIM2CH3**, **TIM2CH4**, **TIM3CH1**,

  TIM3CH2, **TIM3CH3**, **TIM3CH4**, **TIM4CH1**,

  TIM4CH2, **TIM4CH3**, **TIM4CH4**, **TIM5CH1**,

  TIM5CH2, **TIM5CH3**, **TIM5CH4**, **TIM6UP**,

  TIM7UP, **TIM8UP**, **TIM8CH1**, **TIM8CH2**,

  TIM8CH3, **TIM8CH4** }

Enum for possible Timer ids.

- void **Blox_Timer_Init** (uint8_t TIMx, uint32_t TIM_CLK)

  Initializes Timer.

- **TIMER_ID Blox_Timer_Register_IRQ** (uint8_t TIMx, uint16_t period, void(∗Timer_-Handler)(void), FunctionalState NewState)

  Registers a timer interrupt for a given timer.

- void **Blox_Timer_Release_IRQ** (**TIMER_ID** id)

  Releases a given timer interrupt.

- void **Blox_Timer_Modify_IRQ** (**TIMER_ID** id, uint16_t period)

  Modifies the period for a given output compare interrupt.

- void **Blox_Timer_Enable_IRQ** (**TIMER_ID** id)

  Enables interrupts for a given interrupt.

- void **Blox_Timer_Disable_IRQ** (**TIMER_ID** id)

  Disables interrupts for a given interrupt.

4.11.1 Detailed Description

The Timer driver for clocks

4.11.2 Function Documentation

4.11.2.1 void Blox_Timer_Disable_IRQ (TIMER_ID *id*)

Disables interrupts for a given interrupt.

Parameters

id	specifies the id of the timer interrupt

Return values

None	

Definition at line **686** of file **blox_tim.c**.

4.11.2.2 void Blox_Timer_Enable_IRQ (TIMER_ID *id*)

Enables interrupts for a given interrupt.

Parameters

id	specifies the id of the timer interrupt

Return values

None	

Definition at line **537** of file **blox_tim.c**.

4.11.2.3 void Blox_Timer_Init (uint8_t *TIMx,* uint32_t *TIM_CLK*)

Initializes Timer.

Parameters

TIMx	where x can be (1...8) to select the timer.
TIM_CLK	specifies the TIM_CLK to set for the given timer where TIM_CLK can be in the range of 1.1kHz to 72MHz TIM_CLK = SYS_CLK / (PSC + 1)

Return values

None	

Definition at line **42** of file **blox_tim.c**.

4.11.2.4 void Blox_Timer_Modify_IRQ (TIMER_ID *id,* uint16_t *period*)

Modifies the period for a given output compare interrupt.

Parameters

id	specifies the id of the timer interrupt
period	specifies the new period between timer interrupts.

Return values

None	

Definition at line **781** of file **blox_tim.c**.

4.11.2.5 TIMER_ID Blox_Timer_Register_IRQ (uint8_t *TIMx,* uint16_t *period,* void(∗)(void) *Timer_Handler,* FunctionalState *NewState*)

Registers a timer interrupt for a given timer.

Parameters

TIMx	where x can be (1...8) to select the timer.
period	specifies the period between timer interrupts.
Timer_Handler	the handler function for the timer interrupt.
NewState	new state of the timer interrupt. ENABLE or DISABLE

Return values

the	id for the given interrupt or error

Definition at line **183** of file **blox_tim.c**.

4.11.2.6 void Blox_Timer_Release_IRQ (TIMER_ID *id*)

Releases a given timer interrupt.

Parameters

id	specifies the id of the timer interrupt

Return values

None	

Definition at line **526** of file **blox_tim.c**.

4.12 Touch

- void **Touch_RCC_Init** ()

  Initializes clocks for SPI and the GPIO the SPI & BUSY Pins are on.

- void **Touch_GPIO_Init** ()

  Initializes the gpio for the SPI pins, and BUSY.

- void **Touch_GPIO_DeInit** ()

De-initializes the gpio for the SPI pins, and BUSY.

- void **Touch_SPI_Init** ()

  Initializes the SPI for the Touchpanel.

- void **Touch_SPI_DeInit** ()

  De-initializes the SPI for the Touchpanel.

- void **Blox_Touch_Init** (void)

  Initializes the IR module. Basically a wrapper on USART.

- uint16_t **Blox_Touch_GetX** (int numTouch)

  Get the X-value of a press on the touchpanel.

- uint16_t **Blox_Touch_GetY** (int numTouch)

  Get the Y-value of a press on the touchpanel./.

- uint16_t **Blox_Touch_GetZ1** (int numTouch)

  Get the Z1-value of a press on the touchpanel.

- uint16_t **Blox_Touch_GetZ2** (int numTouch)

  Get the Z2-value of a press on the touchpanel.

- void **Touch_SPI_Send** (uint16_t data)

  Sends a byte out on SPI to the touchpanel.

- uint16_t **Touch_SPI_Receive** (void)

  Receive a byte from the touchpanel.

4.12.1 Detailed Description

The Touchpanel driver

4.12.2 Function Documentation

4.12.2.1 uint16_t Blox_Touch_GetX (int *numTouch*)

Get the X-value of a press on the touchpanel.

Parameters

numTouch	ID of the touchpanel to retrieve from

Return values

The	12-bit return value

Definition at line **124** of file **blox_touch.c**.

4.12.2.2 uint16_t Blox_Touch_GetY (int *numTouch*)

Get the Y-value of a press on the touchpanel./.

Parameters

numTouch	touchpanel id to retrieve from

Return values

The	12-bit return value

Definition at line **167** of file **blox_touch.c**.

4.12.2.3 uint16_t Blox_Touch_GetZ1 (int *numTouch*)

Get the Z1-value of a press on the touchpanel.

Parameters

numTouch	touchpanel id to retrieve from

Return values

The	12-bit return value

Definition at line **209** of file **blox_touch.c**.

4.12.2.4 uint16_t Blox_Touch_GetZ2 (int *numTouch*)

Get the Z2-value of a press on the touchpanel.

Parameters

numTouch	touchpanel id to retrieve from

Return values

The	12-bit return value

Definition at line **253** of file **blox_touch.c**.

4.12.2.5 void Blox_Touch_Init (void)

Initializes the IR module. Basically a wrapper on USART.

Return values

None	

Definition at line **27** of file **blox_touch.c**.

4.12.2.6 void Touch_GPIO_DeInit ( )

De-initializes the gpio for the SPI pins, and BUSY.

Return values

None	

Definition at line **86** of file **blox_touch.c**.

4.12.2.7 void Touch_GPIO_Init ( )

Initializes the gpio for the SPI pins, and BUSY.

Return values

None	

Definition at line **54** of file **blox_touch.c**.

4.12.2.8 void Touch_RCC_Init ( )

Initializes clocks for SPI and the GPIO the SPI & BUSY Pins are on.

Return values

None	

Definition at line **41** of file **blox_touch.c**.

4.12.2.9 void Touch_SPI_DeInit ( )

De-initializes the SPI for the Touchpanel.

Return values

None	

Definition at line **115** of file **blox_touch.c**.

4.12.2.10 void Touch_SPI_Init ( )

Initializes the SPI for the Touchpanel.

Return values

None	

Definition at line **94** of file **blox_touch.c**.

4.12.2.11 uint16_t Touch_SPI_Receive (void)

Receive a byte from the touchpanel.

Return values

the	received byte.

Definition at line **308** of file **blox_touch.c**.

4.12.2.12 void Touch_SPI_Send (uint16_t *data*)

Sends a byte out on SPI to the touchpanel.

Parameters

data,:	the byte to send

Return values

None.	

Definition at line **297** of file **blox_touch.c**.

4.13 USART

- void **Blox_USART_DeInit_USART** (void)

  De-initializes all the USART interfaces.

- void **Blox_USART_Init** (uint8_t)

  Initializes the USART module.

- uint8_t **Blox_USART_Receive** (uint8_t id)

  Receive a byte on the given USART.

- int16_t **Blox_USART_TryReceive** (uint8_t id)

  Receive a byte on the given USART.

- void **Blox_USART_Send** (uint8_t id, uint8_t data)

  Sends a byte out on the given USART.

- void **Blox_USART_Register_RXNE_IRQ** (uint8_t id, void(*RXNE_Handler)(void))

  Registers a USART Interrupt on RXNE.

- void **Blox_USART_Enable_RXNE_IRQ** (uint8_t id)

  Disables the USART Interrupt on RXNE.

- void **Blox_USART_Disable_RXNE_IRQ** (uint8_t id)

  Disables the USART Interrupt on RXNE.

- #define **USART1_GPIO** GPIOA
- #define **USART1_CLK** RCC_APB2Periph_USART1
- #define **USART1_GPIO_CLK** RCC_APB2Periph_GPIOA
- #define **USART1_RxPin** GPIO_Pin_10
- #define **USART1_TxPin** GPIO_Pin_9
- #define **USART2_GPIO** GPIOA
- #define **USART2_CLK** RCC_APB1Periph_USART2
- #define **USART2_GPIO_CLK** RCC_APB2Periph_GPIOA
- #define **USART2_RxPin** GPIO_Pin_3
- #define **USART2_TxPin** GPIO_Pin_2
- #define **USART3_GPIO** GPIOB

- #define **USART3_CLK** RCC_APB1Periph_USART3
- #define **USART3_GPIO_CLK** RCC_APB2Periph_GPIOB
- #define **USART3_RxPin** GPIO_Pin_11
- #define **USART3_TxPin** GPIO_Pin_10
- #define **UART4_GPIO** GPIOC
- #define **UART4_CLK** RCC_APB1Periph_UART4
- #define **UART4_GPIO_CLK** RCC_APB2Periph_GPIOC
- #define **UART4_RxPin** GPIO_Pin_11
- #define **UART4_TxPin** GPIO_Pin_10
- #define **UART5_GPIO_TX** GPIOC
- #define **UART5_GPIO_RX** GPIOD
- #define **UART5_CLK** RCC_APB1Periph_UART5
- #define **UART5_GPIO_TX_CLK** RCC_APB2Periph_GPIOC
- #define **UART5_GPIO_RX_CLK** RCC_APB2Periph_GPIOD
- #define **UART5_RxPin** GPIO_Pin_2
- #define **UART5_TxPin** GPIO_Pin_12

4.13.1 Detailed Description

The hardware USART driver

4.13.2 Function Documentation

4.13.2.1 void Blox_USART_DeInit_USART (void)

De-initializes all the USART interfaces.

Return values

None	

Definition at line **88** of file **blox_usart.c**.

4.13.2.2 void Blox_USART_Disable_RXNE_IRQ (uint8_t *id*)

Disables the USART Interrupt on RXNE.

Parameters

id	the USART id to use.

Return values

The	current status of the USART.

Definition at line **377** of file **blox_usart.c**.

4.13.2.3 void Blox_USART_Enable_RXNE_IRQ (uint8_t *id*)

Disables the USART Interrupt on RXNE.

Parameters

id	the USART id to use.

Return values

The	current status of the USART.

Definition at line **352** of file **blox_usart.c**.

4.13.2.4 void Blox_USART_Init (uint8_t *id*)

Initializes the USART module.

Parameters

id,:	the id of the USART interface.

Return values

None	

Definition at line **32** of file **blox_usart.c**.

4.13.2.5 uint8_t Blox_USART_Receive (uint8_t *id*)

Receive a byte on the given USART.

Parameters

id	the USART id to use.

Return values

The	received byte.

Definition at line **233** of file **blox_usart.c**.

4.13.2.6 void Blox_USART_Register_RXNE_IRQ (uint8_t *id,* void(∗)(void) *RXNE_Handler*)

Registers a USART Interrupt on RXNE.

Parameters

id	the USART id to use.
RXNE_Handler	the handler function for the USART RXNE IRQ.

Return values

The	current status of the USART.

Definition at line **327** of file **blox_usart.c**.

4.13.2.7 void Blox_USART_Send (uint8_t *id,* uint8_t *data*)

Sends a byte out on the given USART.

Parameters

id	the USART id to use
data	the byte to send

Return values

None.	

Definition at line **296** of file **blox_usart.c**.

4.13.2.8 int16_t Blox_USART_TryReceive (uint8_t *id*)

Receive a byte on the given USART.

Parameters

id	the USART id to use.

Return values

The	received byte.

Definition at line **259** of file **blox_usart.c**.

4.14 USB

- #define **USB_USART_ID** 1
- void **USB_Init** (void)

  Initializes the USB module. Basically a wrapper on USART.

- uint8_t **USB_Receive** (void)

  Blocking receive of a byte over USB. A wrapper around USART.

- int16_t **USB_TryReceive** (void)

  Non-blocking receive of a byte over USB. A wrapper around USART.

- void **USB_Send** (uint8_t data)

  Sends a byte over USB. A wrapper around USART.

- void **USB_SendData** (uint8_t *data, uint32_t len)

  Sends len bytes over USB. A wrapper around USART.

- void **USB_SendPat** (char *format,...)

  Sends a string based on pattern passed over USB. A wrapper around USART.

4.14.1 Detailed Description

The USB driver

4.14.2 Function Documentation

4.14.2.1 void USB_Init (void)

Initializes the USB module. Basically a wrapper on USART.

Return values

None	

Definition at line **24** of file **blox_usb.c**.

4.14.2.2 uint8_t USB_Receive (void)

Blocking receive of a byte over USB. A wrapper around USART.

Return values

The	received byte.

Definition at line **36** of file **blox_usb.c**.

4.14.2.3 void USB_Send (uint8_t *data*)

Sends a byte over USB. A wrapper around USART.

Parameters

data	the byte to send

Return values

None.	

Definition at line **54** of file **blox_usb.c**.

4.14.2.4 void USB_SendData (uint8_t * *data,* uint32_t *len*)

Sends len bytes over USB. A wrapper around USART.

Parameters

data	a pointer to the buffer of data to
len	number of bytes to send from the buffer

Return values

None.	

Definition at line **64** of file **blox_usb.c**.

4.14.2.5 void USB_SendPat (char * *format, ...*)

Sends a string based on pattern passed over USB. A wrapper around USART.

Parameters

format	a format string
...	values to replace patterns in the string

Return values

None.	

Definition at line **76** of file **blox_usb.c**.

4.14.2.6 int16_t USB_TryReceive (void)

Non-blocking receive of a byte over USB. A wrapper around USART.

Return values

The	received byte or -1 on failure.

Definition at line **45** of file **blox_usb.c**.

4.15 VUSART

- void **Blox_VUSART_RCC_Configuration** (uint8_t id)

  Initializes clocks for the given the virtual USART interface.

- void **Blox_VUSART_Init** (uint8_t id)

  Initializes the virtual USART module.

- void **Blox_VUSART_SetBaudrate** (uint8_t id, uint16_t baudrate)

  Sets the baudrate of the given ID.

- **VUSART_STATUS Blox_VUSART_TryReceive** (uint8_t id, uint8_t *data)

  Tries to receive a byte on the given virtual USART.

- **VUSART_STATUS Blox_VUSART_TrySend** (uint8_t id, uint8_t data)

  Tries to send a byte out on the given virtual USART.

- **VUSART_STATUS Blox_VUSART_Receive** (uint8_t id, uint8_t ∗data)

  Receives a blocking byte on the given virtual USART.

- **VUSART_STATUS Blox_VUSART_Send** (uint8_t id, uint8_t data)

  Sends a blocking byte out on the given virtual USART.

- **VUSART_STATUS Blox_VUSART_SendData** (uint8_t id, uint8_t ∗data, uint32_t len)

  Sends a blocking byte out on the given virtual USART.

- **VUSART_STATUS Blox_VUSART_Register_RXNE_IRQ** (uint8_t id, void(∗RXNE_Handler)(void))

  Registers a function to be called in the SWInterrupt that occurs when a receive happens.

- **VUSART_STATUS Blox_VUSART_Enable_RXNE_IRQ** (uint8_t id)

  Enables the SW Interrupt on RXNE.

- **VUSART_STATUS Blox_VUSART_Disable_RXNE_IRQ** (uint8_t id)

  Disables the SW Interrupt on RXNE.

- #define **VUSART_TIMx** 2
- #define **VUSART_TIM_IRQn** TIM2_IRQn
- #define **VUSART_TIM_CLK** 72000000
- #define **_9600bps** (uint16_t)(VUSART_TIM_CLK / 9600)
- #define **_19200bps** (uint16_t)(VUSART_TIM_CLK / 19200)
- #define **_38400bps** (uint16_t)(VUSART_TIM_CLK / 38400)
- #define **_57600bps** (uint16_t)(VUSART_TIM_CLK / 57600)
- #define **_115200bps** (uint16_t)(VUSART_TIM_CLK / 115200)
- #define **VUSART1_GPIO** GPIOB
- #define **VUSART1_GPIO_CLK** RCC_APB2Periph_GPIOB
- #define **VUSART1_CLK** TIM_CLK
- #define **VUSART1_RxPin** GPIO_Pin_12
- #define **VUSART1_TxPin** GPIO_Pin_13
- #define **VUSART1_RxPinSource** 12
- #define **VUSART1_RxPortSource** GPIO_PortSourceGPIOB
- #define **VUSART2_GPIO** GPIOA
- #define **VUSART2_GPIO_CLK** RCC_APB2Periph_GPIOA
- #define **VUSART2_CLK** TIM_CLK
- #define **VUSART2_RxPin** GPIO_Pin_1
- #define **VUSART2_TxPin** GPIO_Pin_0
- #define **VUSART2_RxPinSource** 1
- #define **VUSART2_RxPortSource** GPIO_PortSourceGPIOA

- enum **VUSART_STATUS** {
  VUSART_SUCCESS = 0, **INVALID_ID**, **TX_BUSY**, **RX_EMPTY**,
  RXNE_IRQ_UNAVAILABLE }
  Status to return on VUSART commands.

4.15.1 Detailed Description

The virtual USART driver for OLED and XBee

4.15.2 Function Documentation

4.15.2.1 VUSART_STATUS Blox_VUSART_Disable_RXNE_IRQ (uint8_t *id*)

Disables the SW Interrupt on RXNE.

Parameters

id	the virtual USART id to use.

Return values

The	current status of the VUSART.

Definition at line **479** of file **blox_vusart.c**.

4.15.2.2 VUSART_STATUS Blox_VUSART_Enable_RXNE_IRQ (uint8_t *id*)

Enables the SW Interrupt on RXNE.

Parameters

id	the virtual USART id to use.

Return values

The	current status of the VUSART.

Definition at line **461** of file **blox_vusart.c**.

4.15.2.3 void Blox_VUSART_Init (uint8_t *id*)

Initializes the virtual USART module.

Parameters

id	the id of the virtual USART interface.

Return values

None	

Definition at line **61** of file **blox_vusart.c**.

4.15.2.4 void Blox_VUSART_RCC_Configuration (uint8_t *id*)

Initializes clocks for the given the virtual USART interface.

Parameters

id	the id of the virtual USART interface.

Return values

None	

Definition at line **276** of file **blox_vusart.c**.

4.15.2.5 VUSART_STATUS Blox_VUSART_Receive (uint8_t *id,* uint8_t * *data*)

Receives a blocking byte on the given virtual USART.

Parameters

id	the virtual USART id to use.
data	a pointer to the location the data will be returned

Return values

The	current status of the VUSART.

Definition at line **400** of file **blox_vusart.c**.

4.15.2.6 VUSART_STATUS Blox_VUSART_Register_RXNE_IRQ (uint8_t *id,* void(*)(void) *RXNE_Handler*)

Registers a function to be called in the SWInterrupt that occurs when a receive happens.

Parameters

id	the virtual USART id to use.
RXNE_Handler	the function to be called.

Return values

The	current status of the VUSART.

Definition at line **439** of file **blox_vusart.c**.

4.15.2.7 VUSART_STATUS Blox_VUSART_Send (uint8_t *id,* uint8_t *data*)

Sends a blocking byte out on the given virtual USART.

Parameters

id	the virtual USART id to use
data	the byte to send

Return values

The	current status of the VUSART.

Definition at line **411** of file **blox_vusart.c**.

4.15.2.8 VUSART_STATUS Blox_VUSART_SendData (uint8_t *id,* uint8_t * *data,* uint32_t *len*)

Sends a blocking byte out on the given virtual USART.

Parameters

id	the virtual USART id to use
data	the byte to send
len	the length of the data

Return values

The	current status of the VUSART.

Definition at line **423** of file **blox_vusart.c**.

4.15.2.9 void Blox_VUSART_SetBaudrate (uint8_t *id,* uint16_t *baudrate*)

Sets the baudrate of the given ID.

Parameters

id	the id of the virtual USART interface./
baudrate	the baudrate to set to (_9600, etc)

Return values

None	

Definition at line **102** of file **blox_vusart.c**.

4.15.2.10 VUSART_STATUS Blox_VUSART_TryReceive (uint8_t *id,* uint8_t * *data*)

Tries to receive a byte on the given virtual USART.

Parameters

id	the virtual USART id to use.
data	a pointer to the location the data will be returned

Return values

The	current status of the VUSART.

Definition at line **329** of file **blox_vusart.c**.

4.15.2.11 VUSART_STATUS Blox_VUSART_TrySend (uint8_t *id,* uint8_t *data*)

Tries to send a byte out on the given virtual USART.

Parameters

id	the virtual USART id to use
data	the byte to send

Return values

The	current status of the VUSART.

Definition at line **363** of file **blox_vusart.c**.

4.16 XBee

- void(∗ **XBee_RX_Handler**)(**BloxFrame** ∗)

  The function pointer called when an interrupt occurs.

- uint8_t **XBee_RX_Enable** = FALSE

  Flag if the RX interrupt will call the user registered function.

- uint8_t **XBee_TxStatus_Flag** = XBEE_TXSTATUS_NORMAL

  Flag used to communicate information between interrupt and TxStatus.

- void **XBee_RCC_Configuration** (void)

  Initializes clocks for XBee sleep and reset pins.

- void **XBee_GPIO_Configuration** ()

  Initializes the gpio for the XBee reset and sleep pins.

- uint8_t **XBee_CheckOkResponse** (void)

  Checks if "OK<CR>" is waiting in the buffer.

- **XBEE_STATUS XBee_SendTxFrame** (**XBeeTxFrame** ∗frame)

  *Sends a **XBeeTxFrame** (p. 115).*

- **XBEE_STATUS XBee_TxStatus** (void)
- void **Blox_XBee_VUSART_RXNE_IRQ** (void)

  The function that XBee registers with VUSART to execute on byte received.

- void **Blox_XBee_Register_RX_IRQ** (void(∗RX_Handler)(**BloxFrame** ∗frame))

  Registers a function to execute when a complete XBee frame is received.

- #define **XBEE_VUSART_ID** 1
- #define **XBEE_RESET_GPIO** GPIOB
- #define **XBEE_RESET_GPIO_CLK** RCC_APB2Periph_GPIOB
- #define **XBEE_RESET_PIN** GPIO_Pin_14
- #define **XBEE_RESET_PIN_NUM** 14
- #define **XBEE_SLEEP_GPIO** GPIOB
- #define **XBEE_SLEEP_GPIO_CLK** RCC_APB2Periph_GPIOB

- #define **XBEE_SLEEP_PIN** GPIO_Pin_15
- #define **XBEE_SLEEP_PIN_NUM** 15
- #define **MESSAGE_SIZE** 100
- #define **CR** 0x0D
- #define **LF** 0x0A
- #define **BS** 0x08
- #define **ESC** 0x1B
- #define **SP** 0x20
- #define **DEL** 0x7F
- #define **API_TX_STATUS** 0x89
- #define **API_RX_FRAME** 0x81
- #define **BLOX_FRAME_DATA_LEN** 75
- #define **XBEE_BLOX_BROADCAST_ID** 0xFFFFFFFF
- #define **XBEE_HOLD_PERIOD** 1000
- enum **XBEE_STATUS** { **XBEE_TX_STATUS_FAIL** = -3, **XBEE_INIT_FAIL**, **XBEE_TX_FAIL**, **XBEE_OK** }

  Enum for possible XBee statuses.

- enum **BloxFrameType** { **FRAME_TYPE_BASE**, **FRAME_TYPE_ROLE**, **FRAME_TYPE_USER** }

  Enum for possible XBee frame types.

- enum **XBEE_TXSTATUS** { **XBEE_TXSTATUS_ERROR** = 0, **XBEE_TXSTATUS_NORMAL**, **XBEE_TXSTATUS_SUCCESS** }

  Enum for possible XBee TX statuses.

- **XBEE_STATUS Blox_XBee_Config** (void)

  Configures the XBee and writes the configuration to non-volatile mem.

- **XBEE_STATUS Blox_XBee_Print** (void)

  Prints out the configuration options of the XBee.

- **XBEE_STATUS Blox_XBee_Init** (void)

  Initializes the XBees sleep and reset pins, and then resets the XBee.

- **XBEE_STATUS Blox_XBee_Send** (uint8_t *data, uint32_t len, **BloxFrameType** type, uint32_t dst_id)

  Sends data out of a specific type on the XBee.

- void **Blox_XBee_Send_Period** (uint8_t *data, uint32_t len, **BloxFrameType** type, uint32_t dst_id, uint32_t millis)

  Sends data out of a specific type on the XBee for a period of time.

- void **Blox_XBee_Register_Read** (void(∗Read_Handler)(void))
- **BloxFrame** ∗ **Blox_XBee_Receive** (void)

  *Receives a **BloxFrame** (p. 97) from the XBee.*

- void **Blox_XBee_Register_RX_IRQ** (void(∗RX_Handler)(**BloxFrame** ∗))
- void **Blox_XBee_Enable_RX_IRQ** (void)

  Enables the sw interrupt that occurs when a XBee reads a byte.

- void **Blox_XBee_Disable_RX_IRQ** (void)

  Disables the sw interrupt that occurs when a XBee reads a byte.

4.16.1 Detailed Description

The XBee driver.

4.16.2 Function Documentation

4.16.2.1 XBEE_STATUS Blox_XBee_Config (void)

Configures the XBee and writes the configuration to non-volatile mem.

Return values

XBEE_OK	if successfull, XBEE_INIT_FAIL on failure.

Definition at line **41** of file **blox_xbee.c**.

4.16.2.2 void Blox_XBee_Disable_RX_IRQ (void)

Disables the sw interrupt that occurs when a XBee reads a byte.

Return values

None.	

Definition at line **333** of file **blox_xbee.c**.

4.16.2.3 void Blox_XBee_Enable_RX_IRQ (void)

Enables the sw interrupt that occurs when a XBee reads a byte.

Return values

None.	

Definition at line **325** of file **blox_xbee.c**.

4.16.2.4 XBEE_STATUS Blox_XBee_Init (void)

Initializes the XBees sleep and reset pins, and then resets the XBee.

Return values

None	

Definition at line **165** of file **blox_xbee.c**.

4.16.2.5 XBEE_STATUS Blox_XBee_Print (void)

Prints out the configuration options of the XBee.

Return values

XBEE_OK	if successfull, XBEE_INIT_FAIL on failure.

Definition at line **112** of file **blox_xbee.c**.

4.16.2.6 BloxFrame* Blox_XBee_Receive (void)

Receives a **BloxFrame** (p. 97) from the XBee.

Return values

The	received frame or NULL on error.

Definition at line **341** of file **blox_xbee.c**.

4.16.2.7 void Blox_XBee_Register_RX_IRQ (void(*)(BloxFrame *frame) *RX_Handler*)

Registers a function to execute when a complete XBee frame is received.

Return values

None.	

Definition at line **317** of file **blox_xbee.c**.

4.16.2.8 XBEE_STATUS Blox_XBee_Send (uint8_t * *data,* uint32_t *len,* BloxFrameType *type,* uint32_t *dst_id*)

Sends data out of a specific type on the XBee.

Parameters

data,:	the data to be sent.
len,:	the amount of data being sent.
type,:	the type of the **BloxFrame** (p. 97).
dst_id,:	the dest xbee id to send to.

Return values

XBEE_OK	if successful, XBEE_INIT_FAIL on failure.

Definition at line **400** of file **blox_xbee.c**.

4.16.2.9 void Blox_XBee_Send_Period (uint8_t * *data,* uint32_t *len,* BloxFrameType *type,* uint32_t *dst_id,* uint32_t *millis*)

Sends data out of a specific type on the XBee for a period of time.

Parameters

data,:	the data to be sent.
len,:	the amount of data being sent.
type,:	the type of the **BloxFrame** (p. 97).
dst_id,:	the dest xbee id to send to.
millis,:	the amount to continue transmission.

Return values

None.	

Definition at line **441** of file **blox_xbee.c**.

4.16.2.10 void Blox_XBee_VUSART_RXNE_IRQ (void)

The function that XBee registers with VUSART to execute on byte received.

Return values

None.	

Definition at line **247** of file **blox_xbee.c**.

4.16.2.11 uint8_t XBee_CheckOkResponse (void)

Checks if "OK<CR>" is waiting in the buffer.

Return values

TRUE	if "OK<CR>" is received, FALSE otherwise.

Definition at line **222** of file **blox_xbee.c**.

4.16.2.12 void XBee_GPIO_Configuration ( )

Initializes the gpio for the XBee reset and sleep pins.

Return values

None	

Definition at line **202** of file **blox_xbee.c**.

4.16.2.13 void XBee_RCC_Configuration (void)

Initializes clocks for XBee sleep and reset pins.

Return values

None	

Definition at line **194** of file **blox_xbee.c**.

4.16.2.14 XBEE_STATUS XBee_SendTxFrame (XBeeTxFrame * *frame*)

Sends a **XBeeTxFrame** (p. 115).

Parameters

frame,:	the frame to be sent.

Return values

XBEE_OK	if successful, XBEE_INIT_FAIL on failure.

Definition at line **453** of file **blox_xbee.c**.

4.17 Feature modules for advanced functionaltiy

Modules

- **Role Management**
- **Gesture Detection**
- **Power Management**

4.18 Role Management

- #define **ROLE_FLAG_LOC** 0x20006000
- #define **ROLE_FN_LOC** 0x20006004
- enum **State** { **STATE_EMPTY**, **STATE_PARENT**, **STATE_CHILD** }
- void **Blox_Role_RX** (**BloxFrame** ∗frame)

  *Executes when XBee receives a **BloxFrame** (p. 97).*

- uint8_t **Role_NextID** (void)

  Returns the next role_id to be allocated All the role have their minimum number required filled first in order. Then the maxiumum number are filled in a fair manner by adding one to each role in order.

- **ROLE_STATUS Blox_Role_Init** (char ∗name, uint8_t len)

  Initializes the role driver's data structures.

- **ROLE_STATUS Blox_Role_Add** (ptrVoidFn fn, uint8_t min, uint8_t max)

  Adds a new role to the role driver.

- **ROLE_STATUS Blox_Role_Run** (void)

  Runs the role, which is a multiple-step procedure. First the Blox should determine if it is the original starter of this application. if so, then it will query Blox running base programs to see if they can run the requested application. It then requests those base programs begin running the application and designates a role to the application once it requests a role.

- #define **ROLE_MAX** 64
- #define **ROLE_INF** 255
- enum **ROLE_STATUS** { **ROLE_RUN_FAIL** = -3, **ROLE_ADD_FAIL**, **ROLE_OOM**, **ROLE_OK** }

  *Status to return on **Role** (p. 106) functions.*

- enum **RoleFrameOps** {

  EMPTY, **PROG_QUERY**, **PROG_ACK**, **PROG_START**,

  PARENT_QUERY, **PARENT_ACK** }

  *An enum of the possible **RoleFrame** (p. 107) opcodes.*

4.18.1 Detailed Description

Role (p. 106) Management facilitates the creation of a distributed application by allowing a program to be broken up into roles and allocated on the fly at application start.

4.18.2 Enumeration Type Documentation

4.18.2.1 enum State

An ENUM for possible parent/child states

Definition at line **32** of file **blox_role.c**.

4.18.3 Function Documentation

4.18.3.1 ROLE_STATUS Blox_Role_Add (ptrVoidFn *fn,* uint8_t *min,* uint8_t *max*)

Adds a new role to the role driver.

Parameters

fn,:	the function to be run for the new role.
min,:	the minimum number of Blox necessary of this role.
max,:	the maxiumum number of Blox wanted of this role.

Return values

ROLE_OK	if the add is successful.
ROLE_OOM	if the maximum number of roles have been added.
ROLE_ADD_FAIL	if min is negative or larger than max

Definition at line **138** of file **blox_role.c**.

4.18.3.2 ROLE_STATUS Blox_Role_Init (char * *name,* uint8_t *len*)

Initializes the role driver's data structures.

Parameters

name	The name of the role
len	The length of the name

Return values

ROLE_OK	if everything initializes correctly.

Definition at line **57** of file **blox_role.c**.

4.18.3.3 ROLE_STATUS Blox_Role_Run (void)

Runs the role, which is a multiple-step procedure. First the Blox should determine if it is the original starter of this application. if so, then it will query Blox running base programs to see if they can run the requested application. It then requests those base programs begin running the application and designates a role to the application once it requests a role.

Return values

ROLE_RUN_FAIL	if there are not enough Blox in the area to run. otherwise this method never returns.

Definition at line **168** of file **blox_role.c**.

4.18.3.4 void Blox_Role_RX (BloxFrame * *frame*)

Executes when XBee receives a **BloxFrame** (p. 97).

Parameters

frame	The **BloxFrame** (p. 97) parsed out of a received XBee frame.

Return values

None.	

Definition at line **83** of file **blox_role.c**.

4.18.3.5 uint8_t Role_NextID (void)

Returns the next role_id to be allocated All the role have their minimum number required filled first in order. Then the maxiumum number are filled in a fair manner by adding one to each role in order.

Return values

The	role_id to allocate

Definition at line **220** of file **blox_role.c**.

4.19 Gesture Detection

- **TIMER_ID touch1ID**
- **TIMER_ID touch2ID**
- **TIMER_ID touch3ID**
- **TIMER_ID touch4ID**
- uint16_t **val** [4] = {0,0,0,0}
- uint16_t **XVals** [4][50]
- uint16_t **YVals** [4][50]
- **GestureRecord LastGesture** [4]
- void **Blox_touch1_isTouched** (void)

  Determines if user is touching Blox, interrupts every 0.02seconds.

- void **Blox_touch2_isTouched** (void)

  Determines if user is touching Blox, interrupts every 0.02seconds.

- void **Blox_touch3_isTouched** (void)

  Determines if user is touching Blox, interrupts every 0.02seconds.

- void **Blox_touch4_isTouched** (void)

  Determines if user is touching Blox, interrupts every 0.02seconds.

- void **Blox_gestureHandler** (int touchNumber)

  Determines gesture movement from pre-populated tracking gesture array.

- void **Blox_Gesture_Init** (void)

  Initializes timer interrupt and array that hold touch values.

- void **Blox_Gesture_DeInit** (void)

  Clears timer interrupt and array that holds touch value.

- void **Blox_touch1_tracker** (void)

  Populated array tracking gesture movement.

- void **Blox_touch2_tracker** (void)

  Populated array tracking gesture movement.

- void **Blox_touch3_tracker** (void)

  Populated array tracking gesture movement.

- void **Blox_touch4_tracker** (void)

  Populated array tracking gesture movement.

- int **Blox_Gesture_GetGesture** (int touchNumber)

  Returns most recent gesture id for a specified touch panel.

- int **Blox_Gesture_GetGestureTime** (int touchNumber)

  Returns most recent gesture timestamp for a specified touch panel.

- #define **TOUCH_TIMx** 3
- #define **TOUCH_CLK** 180000
- #define **TOUCH_DETECT_FREQ** 3600
- #define **PRESSURE_THRESHOLD** 5
- #define **XTHRESH** 25
- #define **YTHRESH** 25
- #define **XNOMOV** 7
- #define **YNOMOV** 7
- #define **TOUCH_X_STABLE** -1
- #define **TOUCH_Y_STABLE** -1
- #define **TOUCH_GESTURE_LR** 1
- #define **TOUCH_GESTURE_RL** 2
- #define **TOUCH_GESTURE_DU** 3
- #define **TOUCH_GESTURE_UD** 4
- #define **TOUCH_DIAG_DLUR** 5
- #define **TOUCH_DIAG_DRUL** 6
- #define **TOUCH_DIAG_ULDR** 7
- #define **TOUCH_DIAG_URDL** 8
- #define **TOUCH_GESTURE_TAP** 9

4.19.1 Detailed Description

Gesture Detection detects advanced touchpanel input such as swipes.

4.19.2 Function Documentation

4.19.2.1 void Blox_Gesture_DeInit (void)

Clears timer interrupt and array that holds touch value.

Return values

None	

Definition at line **60** of file **blox_gesture.c**.

4.19.2.2 int Blox_Gesture_GetGesture (int *touchNumber*)

Returns most recent gesture id for a specified touch panel.

Return values

The	gesture id

Definition at line **300** of file **blox_gesture.c**.

4.19.2.3 int Blox_Gesture_GetGestureTime (int *touchNumber*)

Returns most recent gesture timestamp for a specified touch panel.

Return values

The	gesture timestamp

Definition at line **308** of file **blox_gesture.c**.

4.19.2.4 void Blox_Gesture_Init (void)

Initializes timer interrupt and array that hold touch values.

Return values

None	

Definition at line **34** of file **blox_gesture.c**.

4.19.2.5 void Blox_gestureHandler (int *touchNumber*)

Determines gesture movement from pre-populated tracking gesture array.

Return values

None	

Definition at line **224** of file **blox_gesture.c**.

4.19.2.6 void Blox_touch1_isTouched (void)

Determines if user is touching Blox, interrupts every 0.02seconds.

Return values

none	

Definition at line **78** of file **blox_gesture.c**.

4.19.2.7 void Blox_touch1_tracker (void)

Populated array tracking gesture movement.

Return values

None	

Definition at line **146** of file **blox_gesture.c**.

4.19.2.8 void Blox_touch2_isTouched (void)

Determines if user is touching Blox, interrupts every 0.02seconds.

Return values

none	

Definition at line **94** of file **blox_gesture.c**.

4.19.2.9 void Blox_touch2_tracker (void)

Populated array tracking gesture movement.

Return values

None	

Definition at line **165** of file **blox_gesture.c**.

4.19.2.10 void Blox_touch3_isTouched (void)

Determines if user is touching Blox, interrupts every 0.02seconds.

Return values

none	

Definition at line **114** of file **blox_gesture.c**.

4.19.2.11 void Blox_touch3_tracker (void)

Populated array tracking gesture movement.

Return values

None	

Definition at line **186** of file **blox_gesture.c**.

4.19.2.12 void Blox_touch4_isTouched (void)

Determines if user is touching Blox, interrupts every 0.02seconds.

Return values

none	

Definition at line **130** of file **blox_gesture.c**.

4.19.2.13 void Blox_touch4_tracker (void)

Populated array tracking gesture movement.

Return values

None	

Definition at line **205** of file **blox_gesture.c**.

4.20 Power Management

- static uint32_t **numPowerSleep**
- void **Blox_Power_Register_Power** (void(∗Power_Wake)(void), void(∗Power_Sleep)(void))

  Registers a new function to be called when the system wakes or sleeps. Only adds if it isn't already there.

- void **Blox_Power_Sleep** (void)

  Puts the system to sleep.

- void **Blox_Power_Wake** (void)

  Wakes the system up.

- #define **MAX_POWER_WAKE_FN** 32
- #define **MAX_POWER_SLEEP_FN** 32

4.20.1 Detailed Description

Power Management allows developers to easily put a Blox into a low power state and wake it up on various input events.

4.20.2 Function Documentation

4.20.2.1 void Blox_Power_Register_Power (void(∗)(void) *Power_Wake,* void(∗)(void) *Power_Sleep*)

Registers a new function to be called when the system wakes or sleeps. Only adds if it isn't already there.

Parameters

Power_Wake	a fn pointer to a module's wake function
Power_Sleep	a fn pointer to a module's sleep function

Return values

None	

Definition at line **25** of file **power.c**.

4.20.2.2 void Blox_Power_Sleep (void)

Puts the system to sleep.

Return values

None	

Definition at line **58** of file **power.c**.

4.20.2.3 void Blox_Power_Wake (void)

Wakes the system up.

Return values

None	

Definition at line **70** of file **power.c**.

4.21 System Programs

Modules

- **Base Program**
- **Blox Setup**

4.22 Base Program

Modules

- **Base Program UI**
- **Transfer**

- #define **MAIN_MENU_NUM_ENTRIES** 4
- #define **TEXT_START_LOCATION** 43
- #define **TEXT_LOCATION_LEFT_ALIGNED** 16
- #define **TEXT_LOCATION_OFFSET** 14
- void **Base_RX** (**BloxFrame** *frame)

  Listens for other Blox asking for new participants.

- void **Base_UI_MainMenu** (void)

  Draws the main menu.

- void **Base_UI_ApplicationsMenu** (void)

  Draws the applications menu.

- void **Base_UI_CalibrationMenu** (void)

  Draws the calibration menu.

- void **Base_UI_SysInfoMenu** (void)

  Draws the system info menu. The system info menu displays the ID of the Blox, the current number of programs, and the amount of free and used space.

- void **Base_UI_USBMenu** (void)

  Draws the USB menu and starts listening for USB packets.

- void **Base_UI_Loading** (void)

  Draws a loading screen.

- int **main** (void)

  Provides a GUI for the base program. The base program allows users to select applications to run, calibrate system variables, receive USB commands, and view system info.

- void **Application_Handler** (void)

  Handler for calling applications using blox_base_ui.

4.22.1 Function Documentation

4.22.1.1 void Application_Handler (void)

Handler for calling applications using blox_base_ui.

Return values

None	

Definition at line **132** of file **base_program.c**.

4.22.1.2 void Base_RX (BloxFrame * *frame*)

Listens for other Blox asking for new participants.

Return values

None	

Definition at line **255** of file **base_program.c**.

4.22.1.3 void Base_UI_ApplicationsMenu (void)

Draws the applications menu.

Return values

None	

Definition at line **141** of file **base_program.c**.

4.22.1.4 void Base_UI_CalibrationMenu (void)

Draws the calibration menu.

Return values

None	

Definition at line **169** of file **base_program.c**.

4.22.1.5 void Base_UI_Loading (void)

Draws a loading screen.

Return values

None	

Definition at line **243** of file **base_program.c**.

4.22.1.6 void Base_UI_MainMenu (void)

Draws the main menu.

Return values

None	

Definition at line **106** of file **base_program.c**.

4.22.1.7 void Base_UI_SysInfoMenu (void)

Draws the system info menu. The system info menu displays the ID of the Blox, the current number of programs, and the amount of free and used space.

Return values

None	

Definition at line **184** of file **base_program.c**.

4.22.1.8 void Base_UI_USBMenu (void)

Draws the USB menu and starts listening for USB packets.

Note

The Blox must be reset after entering USB mode

Return values

None	

Definition at line **225** of file **base_program.c**.

4.22.1.9 int main (void)

Provides a GUI for the base program. The base program allows users to select applications to run, calibrate system variables, receive USB commands, and view system info.

Return values

None	

Definition at line **53** of file **base_program.c**.

4.23 Base Program UI

- void **Blox_UI_DrawEntry** (char *entry, uint8_t index, uint16_t color)

  Draws a single entry.

- void **Blox_UI_DrawHeader** (void)

  Draws the header for the UI.

- void **Blox_UI_DrawFooter** (void)

  Draws the footer for the UI.

- void **Blox_UI_DrawTitle** (char *title)

  Draws the title for the UI.

- void **Blox_UI_SetEntries** (char **entries, ptrVoidFn *entries_handler, ptrVoidFn BackHandler, uint8_t numEntries)

  Sets the entries for the UI page.

- void **Blox_UI_DrawEntries** (void)

  Draws the entries for the UI page.

- void **Blox_UI_SelectEntry** (char *entry)

  Selects an entry for the UI.

- void **Blox_UI_SelectEntryAbove** (void)

Selects the entry above the currently selected one.

- void **Blox_UI_SelectEntryBelow** (void)

  Selects the entry below the currently selected one.

- void **Blox_UI_RunEntry** (void)

  Runs the currently selected entry.

- uint8_t **Blox_UI_GetEntryID** (void)

  Gets the id of the selected entry.

- void **Blox_UI_Back** (void)

  Calls the Back function for the UI page.

- void **Blox_UI_ClearScreen** (void)

  Clears the current UI page.

- #define **CENTER_TEXT**(text, width) (64-width∗TITLE_SIZE∗strlen(text)/2)
- #define **HEADER_BACKGROUND_COLOR** COLOR_BLUE
- #define **HEADER_TEXT_COLOR** COLOR_WHITE
- #define **HEADER_HEIGHT** 12
- #define **HEADER_TITLE_LOCATION_X** 3
- #define **HEADER_TITLE_LOCATION_Y** 2
- #define **HEADER_VERSION_LOCATION_X** 96
- #define **HEADER_VERSION_LOCATION_Y** 2
- #define **TITLE_SIZE** 1
- #define **TITLE_LOCATION_Y** 19
- #define **TITLE_COLOR** COLOR_WHITE
- #define **TITLE_LINE_X_START** 15
- #define **TITLE_LINE_X_END** 127-(TITLE_LINE_X_START+1)
- #define **TITLE_LINE_Y** 35
- #define **TITLE_LINE_COLOR** COLOR_BLUE
- #define **ENTRY_START_LOCATION** 43
- #define **ENTRY_LOCATION_OFFSET** 14
- #define **MAX_ENTRIES** 5
- #define **ENTRY_COLOR** COLOR_WHITE
- #define **ENTRY_SELECTED_COLOR** COLOR_YELLOW
- #define **MAX_ENTRY_LENGTH** 20
- #define **FOOTER_TEXT** "Project Blox"
- #define **FOOTER_BACKGROUND_COLOR** COLOR_BLUE
- #define **FOOTER_TEXT_COLOR** COLOR_WHITE
- #define **FOOTER_HEIGHT** 12
- #define **FOOTER_COPYRIGHT_LOCATION_X** CENTER_TEXT(FOOTER_TEXT, 6)
- #define **FOOTER_COPYRIGHT_LOCATION_Y** 117

4.23.1 Function Documentation

4.23.1.1 void Blox_UI_Back (void)

Calls the Back function for the UI page.

Return values

None	

Definition at line **179** of file **blox_base_ui.c**.

4.23.1.2 void Blox_UI_ClearScreen (void)

Clears the current UI page.

Return values

None	

Definition at line **189** of file **blox_base_ui.c**.

4.23.1.3 void Blox_UI_DrawEntries (void)

Draws the entries for the UI page.

Return values

None	

Definition at line **80** of file **blox_base_ui.c**.

4.23.1.4 void Blox_UI_DrawEntry (char * *entry,* uint8_t *index,* uint16_t *color*)

Draws a single entry.

Parameters

entry	The entry to be displayed
index	The index of the entry in order to offset the text
color	The color of the entry text

Return values

None	

Definition at line **95** of file **blox_base_ui.c**.

4.23.1.5 void Blox_UI_DrawFooter (void)

Draws the footer for the UI.

Return values

None	

Definition at line **40** of file **blox_base_ui.c**.

4.23.1.6 void Blox_UI_DrawHeader (void)

Draws the header for the UI.

Return values

None	

Definition at line **26** of file **blox_base_ui.c**.

4.23.1.7 void Blox_UI_DrawTitle (char * *title*)

Draws the title for the UI.

Parameters

title	The title text to be displayed

Return values

None	

Definition at line **51** of file **blox_base_ui.c**.

4.23.1.8 uint8_t Blox_UI_GetEntryID (void)

Gets the id of the selected entry.

Return values

The	id of the currently selected entry

Definition at line **171** of file **blox_base_ui.c**.

4.23.1.9 void Blox_UI_RunEntry (void)

Runs the currently selected entry.

Return values

None	

Definition at line **161** of file **blox_base_ui.c**.

4.23.1.10 void Blox_UI_SelectEntry (char * *entry*)

Selects an entry for the UI.

Parameters

entry	The entry to be selected

Return values

None	

Definition at line **116** of file **blox_base_ui.c**.

4.23.1.11 void Blox_UI_SelectEntryAbove (void)

Selects the entry above the currently selected one.

Return values

None	

Definition at line **131** of file **blox_base_ui.c**.

4.23.1.12 void Blox_UI_SelectEntryBelow (void)

Selects the entry below the currently selected one.

Return values

None	

Definition at line **146** of file **blox_base_ui.c**.

4.23.1.13 void Blox_UI_SetEntries (char ** *entries,* ptrVoidFn * *entries_handler,* ptrVoidFn *BackHandler,* uint8_t *numEntries*)

Sets the entries for the UI page.

Parameters

entries	The names of entries for the UI page
entries_handler	The handler functions for the entries
BackHandler	The handler function for going back a page
numEntries	The number of entries for the UI page

Return values

None	

Definition at line **65** of file **blox_base_ui.c**.

4.24 Transfer

- **TRANSFER_STATUS Cmd_RCV_APP** (void)

  Receives an application from a sender and stores it in the fs.

- **TRANSFER_STATUS Cmd_DEL_APP** (void)

  Receives an app. id from a sender and removes it from the fs.

- **TRANSFER_STATUS Cmd_LST_APPS** (void)

  Sends a command with headers for all the applications in the fs.

- **TRANSFER_STATUS Cmd_RUN_APP** (void)

  Receives an app. id, retrieves it from the fs, and runs the app.

- void **Transfer_Init** (void)

  Initializes the transfer module.

- void **Transfer_Slave** (void)

  Run in slave mode accepting and processing commands.

- #define **TRANSFER_ACK** 0x79
- #define **TRANSFER_NAK** 0x1F
- enum **TRANSFER_STATUS** { **TRANSFER_OK** = 0, **TRANSFER_CMD_FAIL**, **TRANSFER_INV_OPCODE** }

  Enum of the possible transfer statuses.

- enum **TRANSFER_OPCODE** {

  OP_BOT = 0, **RCV_APP**, **DEL_APP**, **LST_APPS**,

  RUN_APP, **OP_TOP** }

  Enum of the available opcodes.

4.24.1 Function Documentation

4.24.1.1 TRANSFER_STATUS Cmd_DEL_APP (void)

Receives an app. id from a sender and removes it from the fs.

Return values

TRANSFER_OK	if the receive and delete succeed. -TRANSFER_CMD_FAIL if the receive or delete fail.

Definition at line **151** of file **blox_transfer.c**.

4.24.1.2 TRANSFER_STATUS Cmd_LST_APPS (void)

Sends a command with headers for all the applications in the fs.

Return values

TRANSFER_OK	if the send succeeds. -TRANSFER_CMD_FAIL if the send fails.

Definition at line **173** of file **blox_transfer.c**.

4.24.1.3 TRANSFER_STATUS Cmd_RCV_APP (void)

Receives an application from a sender and stores it in the fs.

Return values

TRANSFER_OK	if the receive and application store succeed. -TRANSFER_CMD_-FAIL if the receive or store fail.

Definition at line **69** of file **blox_transfer.c**.

4.24.1.4 TRANSFER_STATUS Cmd_RUN_APP (void)

Receives an app. id, retrieves it from the fs, and runs the app.

Return values

Doesn't	return on success. -TRANSFER_CMD_FAIL on failure.

Definition at line **208** of file **blox_transfer.c**.

4.24.1.5 void Transfer_Init (void)

Initializes the transfer module.

Return values

TRANSFER_OK	if successful, another TRANSFER_STATUS if not.

Definition at line **26** of file **blox_transfer.c**.

4.25 Blox Setup

- #define **BLOX_ID** 3
- #define **DEF_ACCEL_X** 128
- #define **DEF_ACCEL_Y** 128
- #define **DEF_ACCEL_Z** 128
- #define **DEF_TOUCH_X** 64
- #define **DEF_TOUCH_Y** 64
- void **RCC_Configuration** (void)

  Configures the clocks.

- void **GPIO_Configuration** (void)

  Configures the GPIOs.

- int **main** (void)

  Sets up the the blox system.

4.25.1 Function Documentation

4.25.1.1 void GPIO_Configuration (void)

Configures the GPIOs.

Return values

None	

Definition at line **92** of file **blox_setup.c**.

4.25.1.2 int main (void)

Sets up the the blox system.

Return values

None	

Definition at line **34** of file **blox_setup.c**.

4.25.1.3 void RCC_Configuration (void)

Configures the clocks.

Return values

None	

Definition at line **83** of file **blox_setup.c**.

4.26 User Applications

Modules

- **Memory Maze**

- Countdown

4.27 Memory Maze

4.28 Countdown

4.29 Example programs

Chapter 5

Data Structure Documentation

5.1 BloxFrame Struct Reference

App-level frame that is parsed from a **XBeeFrame** (p. 113).

`#include <blox_xbee.h>`

Data Fields

- uint32_t **src_id**
- uint32_t **dst_id**
- uint8_t **len**
- **BloxFrameType type**
- uint8_t **data** [BLOX_FRAME_DATA_LEN]

5.1.1 Detailed Description

App-level frame that is parsed from a **XBeeFrame** (p. 113).

Definition at line **85** of file **blox_xbee.h**.

5.1.2 Field Documentation

5.1.2.1 uint8_t data[BLOX_FRAME_DATA_LEN]

the array of raw data

Definition at line **90** of file **blox_xbee.h**.

5.1.2.2 uint32_t dst_id

the id of the receiving Blox

Definition at line **87** of file **blox_xbee.h**.

5.1.2.3 uint8_t len

the number of bytes of good data

Definition at line **88** of file **blox_xbee.h**.

5.1.2.4 uint32_t src_id

the id of the sending Blox

Definition at line **86** of file **blox_xbee.h**.

5.1.2.5 BloxFrameType type

the type of **BloxFrame** (p. 97)

Definition at line **89** of file **blox_xbee.h**.

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

- drivers/inc/**blox_xbee.h**

5.2 FIFO_Type Struct Reference

Basic struct for each fifo.

`#include <blox_fifo.h>`

Data Fields

- uint16_t **data** [FIFO_SIZE]
- uint8_t **read**
- uint8_t **write**

5.2.1 Detailed Description

Basic struct for each fifo.

Definition at line **24** of file **blox_fifo.h**.

5.2.2 Field Documentation

5.2.2.1 uint16_t data[FIFO_SIZE]

Buffer for reading

Definition at line **25** of file **blox_fifo.h**.

5.2.2.2 uint8_t read

Next character to read

Definition at line **26** of file **blox_fifo.h**.

5.2.2.3 uint8_t write

Next character to write

Definition at line **27** of file **blox_fifo.h**.

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

- drivers/inc/**blox_fifo.h**

5.3 FS_File Struct Reference

Defines a file's header. Contains the id of the file within the FAT, the size in bytes of the file, and a pointer to the data.

`#include <blox_filesystem.h>`

Data Fields

- uint8_t **id**
- char **name** [FS_FILE_MAX_NAME_LEN]
- uint8_t **numPages**
- uint32_t * **data**

5.3.1 Detailed Description

Defines a file's header. Contains the id of the file within the FAT, the size in bytes of the file, and a pointer to the data.

Definition at line **50** of file **blox_filesystem.h**.

5.3.2 Field Documentation

5.3.2.1 uint32_t* data

a pointer to the data of the file

Definition at line **54** of file **blox_filesystem.h**.

5.3.2.2 uint8_t id

The unique identifer of the file in the fs

Definition at line **51** of file **blox_filesystem.h**.

5.3.2.3 char name[FS_FILE_MAX_NAME_LEN]

a string name for the file

Definition at line **52** of file **blox_filesystem.h**.

5.3.2.4 uint8_t numPages

the number of pages the application takes up

Definition at line **53** of file **blox_filesystem.h**.

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

- drivers/inc/**blox_filesystem.h**

5.4 FS_Table Struct Reference

The table of all FS_Files.

`#include <blox_filesystem.h>`

Data Fields

- volatile uint32_t **magic**
- volatile uint32_t **numFiles**
- uint32_t ∗ **free_top**
- uint32_t **free_numPages**
- **FS_File table** [FS_MAX_FILES]

5.4.1 Detailed Description

The table of all FS_Files.

Definition at line **60** of file **blox_filesystem.h**.

5.4.2 Field Documentation

5.4.2.1 uint32_t free_numPages

the number of free pages left

Definition at line **64** of file **blox_filesystem.h**.

5.4.2.2 uint32_t∗ free_top

the top of the free memory area

Definition at line **63** of file **blox_filesystem.h**.

5.4.2.3 volatile uint32_t magic

a constant to validate the FS

Definition at line **61** of file **blox_filesystem.h**.

5.4.2.4 volatile uint32_t numFiles

the number of files in the FS

Definition at line **62** of file **blox_filesystem.h**.

5.4.2.5 FS_File table[FS_MAX_FILES]

the table of FS_Files

Definition at line **65** of file **blox_filesystem.h**.

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

- drivers/inc/**blox_filesystem.h**

5.5 GestureRecord Struct Reference

Contains gesture data.

`#include <blox_gesture.h>`

Data Fields

- uint32_t **timestamp**
- int **gesture**

5.5.1 Detailed Description

Contains gesture data.

Definition at line **51** of file **blox_gesture.h**.

5.5.2 Field Documentation

5.5.2.1 int gesture

The gesture id

Definition at line **53** of file **blox_gesture.h**.

5.5.2.2 uint32_t timestamp

The timestamp of the gesture

Definition at line **52** of file **blox_gesture.h**.

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

- feature_modules/**blox_gesture.h**

5.6 IRFrame Struct Reference

Defines data an **IRFrame** (p. 103) sends.

`#include <blox_ir.h>`

Data Fields

- uint8_t **src_id**
- uint8_t **src_face_id**
- **IRFrameType type**
- uint8_t **len**
- uint8_t * **data**
- uint8_t **checksum**

5.6.1 Detailed Description

Defines data an **IRFrame** (p. 103) sends.

Definition at line **52** of file **blox_ir.h**.

5.6.2 Field Documentation

5.6.2.1 uint8_t checksum

a checksum of the data

Definition at line **58** of file **blox_ir.h**.

5.6.2.2 uint8_t* data

a pointer to the data being sent

Definition at line **57** of file **blox_ir.h**.

5.6.2.3 uint8_t len

the amount of data being sent

Definition at line **56** of file **blox_ir.h**.

5.6.2.4 uint8_t src_face_id

the ID of the sending face

Definition at line **54** of file **blox_ir.h**.

5.6.2.5 uint8_t src_id

the BLOX_ID of this Blox

Definition at line **53** of file **blox_ir.h**.

5.6.2.6 IRFrameType type

Type to differentiate uses of IR

Definition at line **55** of file **blox_ir.h**.

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

- drivers/inc/**blox_ir.h**

5.7 Note Struct Reference

Defines data a note contains.

`#include <blox_speaker.h>`

Data Fields

- unsigned short **noteName**
- unsigned short **duration**

5.7.1 Detailed Description

Defines data a note contains.

Definition at line **91** of file **blox_speaker.h**.

5.7.2 Field Documentation

5.7.2.1 unsigned short duration

the duration the note is held

Definition at line **94** of file **blox_speaker.h**.

5.7.2.2 unsigned short noteName

the name of the note

Definition at line **93** of file **blox_speaker.h**.

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

- drivers/inc/**blox_speaker.h**

5.8 ParentAckFrame Struct Reference

App-layer frame for a PARENT_ACK.

`#include <blox_role.h>`

Data Fields

- uint8_t **role_id**

5.8.1 Detailed Description

App-layer frame for a PARENT_ACK.

Definition at line **90** of file **blox_role.h**.

5.8.2 Field Documentation

5.8.2.1 uint8_t role_id

The role_id this child should run as

Definition at line **91** of file **blox_role.h**.

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

- feature_modules/blox_role.h

5.9 QueryFrame Struct Reference

App-layer frame for a [PARENT|PROG]_QUERY.

`#include <blox_role.h>`

Data Fields

- char **name** [FS_FILE_MAX_NAME_LEN]

5.9.1 Detailed Description

App-layer frame for a [PARENT|PROG]_QUERY.

Definition at line **83** of file **blox_role.h**.

5.9.2 Field Documentation

5.9.2.1 char name[FS_FILE_MAX_NAME_LEN]

The name of the program being queried about

Definition at line **84** of file **blox_role.h**.

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

- feature_modules/blox_role.h

5.10 Role Struct Reference

Struct that represents a role.

`#include <blox_role.h>`

Data Fields

- uint8_t **min**
- uint8_t **max**
- ptrVoidFn **fn**
- uint8_t **num_allocated**

5.10.1 Detailed Description

Struct that represents a role.

Definition at line **38** of file **blox_role.h**.

5.10.2 Field Documentation

5.10.2.1 ptrVoidFn fn

The function to call for this role

Definition at line **41** of file **blox_role.h**.

5.10.2.2 uint8_t max

The maximum number of Blox desired of this role

Definition at line **40** of file **blox_role.h**.

5.10.2.3 uint8_t min

The minimum number of Blox needed of this role

Definition at line **39** of file **blox_role.h**.

5.10.2.4 uint8_t num_allocated

The number of Blox allocated so far of this role

Definition at line **42** of file **blox_role.h**.

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

- feature_modules/blox_role.h

5.11 RoleFrame Struct Reference

Role-level frame passed into XBee.

`#include <blox_role.h>`

Data Fields

- uint8_t **opcode**
- uint8_t **data** [BLOX_FRAME_DATA_LEN-1]

5.11.1 Detailed Description

Role-level frame passed into XBee.

Definition at line **75** of file **blox_role.h**.

5.11.2 Field Documentation

5.11.2.1 uint8_t data[BLOX_FRAME_DATA_LEN-1]

the raw data being sent

Definition at line **77** of file **blox_role.h**.

5.11.2.2 uint8_t opcode

the RoleFrameOps value for the desired opcode

Definition at line **76** of file **blox_role.h**.

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

- feature_modules/blox_role.h

5.12 RoleInfo Struct Reference

Struct with all role information.

`#include <blox_role.h>`

Data Fields

- char **name** [FS_FILE_MAX_NAME_LEN]
- uint8_t **name_len**
- uint8_t **num_roles**
- **Role roles** [ROLE_MAX]
- uint8_t **num_blox_found**
- uint8_t **num_blox_started**

- uint8_t **num_needed**
- uint8_t **num_wanted**
- uint32_t **blox_found** [ROLE_MAX]

5.12.1 Detailed Description

Struct with all role information.

Definition at line **48** of file **blox_role.h**.

5.12.2 Field Documentation

5.12.2.1 uint32_t blox_found[ROLE_MAX]

An array of the Blox_Ids found

Definition at line **57** of file **blox_role.h**.

5.12.2.2 char name[FS_FILE_MAX_NAME_LEN]

The name of the program as a string

Definition at line **49** of file **blox_role.h**.

5.12.2.3 uint8_t name_len

The length of the name

Definition at line **50** of file **blox_role.h**.

5.12.2.4 uint8_t num_blox_found

The number of blox with the application foun so far

Definition at line **53** of file **blox_role.h**.

5.12.2.5 uint8_t num_blox_started

The number of blox that have stated this application so far

Definition at line **54** of file **blox_role.h**.

5.12.2.6 uint8_t num_needed

The total number of Blox needed

Definition at line **55** of file **blox_role.h**.

5.12.2.7 uint8_t num_roles

The number of Roles

Definition at line **51** of file **blox_role.h**.

5.12.2.8 uint8_t num_wanted

The total number of Blox wanted

Definition at line **56** of file **blox_role.h**.

5.12.2.9 Role roles[ROLE_MAX]

An array of Roles

Definition at line **52** of file **blox_role.h**.

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

- feature_modules/blox_role.h

5.13 SysVar Struct Reference

Defines a system variable.

`#include <blox_system.h>`

Data Fields

- uint32_t **magic**
- uint32_t **id**
- int32_t **ACCEL_X**
- int32_t **ACCEL_Y**
- int32_t **ACCEL_Z**
- int32_t **TOUCH_1_X**
- int32_t **TOUCH_1_Y**

- int32_t **TOUCH_2_X**
- int32_t **TOUCH_2_Y**
- int32_t **TOUCH_3_X**
- int32_t **TOUCH_3_Y**
- int32_t **TOUCH_4_X**
- int32_t **TOUCH_4_Y**

5.13.1 Detailed Description

Defines a system variable.

Definition at line **61** of file **blox_system.h**.

5.13.2 Field Documentation

5.13.2.1 int32_t ACCEL_X

calibration data for X-axis of accelerometer

Definition at line **64** of file **blox_system.h**.

5.13.2.2 int32_t ACCEL_Y

calibration data for Y-axis of accelerometer

Definition at line **65** of file **blox_system.h**.

5.13.2.3 int32_t ACCEL_Z

calibration data for Z-axis of accelerometer

Definition at line **66** of file **blox_system.h**.

5.13.2.4 uint32_t id

the id of the Blox

Definition at line **63** of file **blox_system.h**.

5.13.2.5 uint32_t magic

a constant to validate the system variable

Definition at line **62** of file **blox_system.h**.

5.13.2.6 int32_t TOUCH_1_X

calibration data for X-coordinate of touch1

Definition at line **67** of file **blox_system.h**.

5.13.2.7 int32_t TOUCH_1_Y

calibration data for Y-coordinate of touch1

Definition at line **68** of file **blox_system.h**.

5.13.2.8 int32_t TOUCH_2_X

calibration data for X-coordinate of touch2

Definition at line **69** of file **blox_system.h**.

5.13.2.9 int32_t TOUCH_2_Y

calibration data for Y-coordinate of touch2

Definition at line **70** of file **blox_system.h**.

5.13.2.10 int32_t TOUCH_3_X

calibration data for X-coordinate of touch3

Definition at line **71** of file **blox_system.h**.

5.13.2.11 int32_t TOUCH_3_Y

calibration data for Y-coordinate of touch3

Definition at line **72** of file **blox_system.h**.

5.13.2.12 int32_t TOUCH_4_X

calibration data for X-coordinate of touch4

Definition at line **73** of file **blox_system.h**.

5.13.2.13 int32_t TOUCH_4_Y

calibration data for Y-coordinate of touch4

Definition at line **74** of file **blox_system.h**.

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

- drivers/inc/**blox_system.h**

5.14 XBeeFrame Struct Reference

A general xbee frame to parse in any command.

`#include <blox_xbee.h>`

Data Fields

- uint16_t **length**
- uint8_t **data** [125]

5.14.1 Detailed Description

A general xbee frame to parse in any command.

Definition at line **110** of file **blox_xbee.h**.

5.14.2 Field Documentation

5.14.2.1 uint8_t data[125]

the data of the xbee frame

Definition at line **112** of file **blox_xbee.h**.

5.14.2.2 uint16_t length

the length of the xbee frame

Definition at line **111** of file **blox_xbee.h**.

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

- drivers/inc/**blox_xbee.h**

5.15 XBeeRxFrame Struct Reference

Struct for reading in the Rx frame format.

`#include <blox_xbee.h>`

Data Fields

- uint16_t **length**
- uint8_t **api**
- uint16_t **source**
- uint8_t **rssi**
- uint8_t **options**
- **BloxFrame blox_frame**
- uint8_t **checksum**

5.15.1 Detailed Description

Struct for reading in the Rx frame format.

Definition at line **128** of file **blox_xbee.h**.

5.15.2 Field Documentation

5.15.2.1 uint8_t api

the api of the command

Definition at line **130** of file **blox_xbee.h**.

5.15.2.2 BloxFrame blox_frame

the frame the sender originally sent

Definition at line **134** of file **blox_xbee.h**.

5.15.2.3 uint8_t checksum

checksum a checksum to detect errors

Definition at line **135** of file **blox_xbee.h**.

5.15.2.4 uint16_t length

the number of bytes of cmdData

Definition at line **129** of file **blox_xbee.h**.

5.15.2.5 uint8_t options

Bytes the sender can set to block ACKs

Definition at line **133** of file **blox_xbee.h**.

5.15.2.6 uint8_t rssi

the signal strength in dB

Definition at line **132** of file **blox_xbee.h**.

5.15.2.7 uint16_t source

the source xbee id of the sender

Definition at line **131** of file **blox_xbee.h**.

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

- drivers/inc/**blox_xbee.h**

5.16 XBeeTxFrame Struct Reference

the XBee Transmission Frame struct

`#include <blox_xbee.h>`

Data Fields

- uint8_t **start**
- uint16_t **length**
- uint8_t **api**
- uint8_t **id**
- uint16_t **dest_addr**
- uint8_t **options**
- **BloxFrame blox_frame**
- uint8_t **checksum**

5.16.1 Detailed Description

the XBee Transmission Frame struct

Definition at line **96** of file **blox_xbee.h**.

5.16.2 Field Documentation

5.16.2.1 uint8_t api

the api code for this command

Definition at line **99** of file **blox_xbee.h**.

5.16.2.2 BloxFrame blox_frame

the blox_frame to send inside the xbee frame

Definition at line **103** of file **blox_xbee.h**.

5.16.2.3 uint8_t checksum

the checksum to verify the sent data

Definition at line **104** of file **blox_xbee.h**.

5.16.2.4 uint16_t dest_addr

the xbee dest_addr to send to

Definition at line **101** of file **blox_xbee.h**.

5.16.2.5 uint8_t id

the id of the packet

Definition at line **100** of file **blox_xbee.h**.

5.16.2.6 uint16_t length

the number of bytes in cmdData

Definition at line **98** of file **blox_xbee.h**.

5.16.2.7 uint8_t options

byte to set ACK/etc

Definition at line **102** of file **blox_xbee.h**.

5.16.2.8 uint8_t start

the start delimiter

Definition at line **97** of file **blox_xbee.h**.

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

- drivers/inc/**blox_xbee.h**

5.17 XBeeTxStatusFrame Struct Reference

Struct for reading in the TxStatus frame format.

`#include <blox_xbee.h>`

Data Fields

- uint16_t **length**
- uint8_t **api**
- uint8_t **frame_id**
- uint8_t **status**

5.17.1 Detailed Description

Struct for reading in the TxStatus frame format.

Definition at line **118** of file **blox_xbee.h**.

5.17.2 Field Documentation

5.17.2.1 uint8_t api

the number of bytes of cmdData

Definition at line **120** of file **blox_xbee.h**.

5.17.2.2 uint8_t frame_id

the api of the command

Definition at line **121** of file **blox_xbee.h**.

5.17.2.3 uint16_t length

Definition at line **119** of file **blox_xbee.h**.

5.17.2.4 uint8_t status

the id of the packet

Definition at line **122** of file **blox_xbee.h**.

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

- drivers/inc/**blox_xbee.h**

Chapter 6

File Documentation

6.1 drivers/inc/blox_accel.h File Reference

Driver for Blox Accelerometer.

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_adc.h"
#include "stm32f10x_dma.h"
```

- #define **ACCEL_GPIO** GPIOC
- #define **ACCEL_GPIO_CLK** RCC_APB2Periph_GPIOC
- #define **ACCEL_ADC1_CLK** RCC_APB2Periph_ADC1
- #define **ACCEL_DMA_CLK** RCC_AHBPeriph_DMA1
- #define **ACCEL_XOUT_PIN** GPIO_Pin_0
- #define **ACCEL_XOUT_PIN_NUM** 0
- #define **ACCEL_YOUT_PIN** GPIO_Pin_1
- #define **ACCEL_YOUT_PIN_NUM** 1
- #define **ACCEL_ZOUT_PIN** GPIO_Pin_2
- #define **ACCEL_ZOUT_PIN_NUM** 2
- #define **ACCEL_SLEEP_GPIO** GPIOA
- #define **ACCEL_SLEEP_GPIO_CLK** RCC_APB2Periph_GPIOA
- #define **ACCEL_SLEEP_PIN** GPIO_Pin_11
- #define **ACCEL_SLEEP_PIN_NUM** 11

- void **Blox_Accel_Init** (void)

  Initializes the Accelerometer. Initializes the clocks, GPIO pins, DMA location, and ADC configuration. Also sets the sleep pin to high (active low pulse).

- uint16_t **Blox_Accel_GetX** (void)

  Returns the X-value reading of the accelerometer.

- uint16_t **Blox_Accel_GetY** (void)

  Returns the Y-value reading of the accelerometer.

- uint16_t **Blox_Accel_GetZ** (void)

  Returns the Z-value reading of the accelerometer.

- uint8_t **Blox_Accel_GetXTilt** (void)

  Returns the X tilt of the accelerometer.

- uint8_t **Blox_Accel_GetYTilt** (void)

  Returns the Y tilt of the accelerometer.

- uint8_t **Blox_Accel_GetZTilt** (void)

  Returns the Z tilt of the accelerometer.

6.1.1 Detailed Description

Driver for Blox Accelerometer.

Author

Dan Cleary

Version

V0.1

Date

10/20/2010

Definition in file **blox_accel.h**.

6.2 drivers/inc/blox_accel.h

```
00001
00008 #ifndef __BLOX_ACCEL_H
00009 #define __BLOX_ACCEL_H
00010
00011 #include "stm32f10x.h"
00012 #include "stm32f10x_gpio.h"
00013 #include "stm32f10x_rcc.h"
00014 #include "stm32f10x_adc.h"
00015 #include "stm32f10x_dma.h"
00016
00022 #define ACCEL_GPIO                    GPIOC
00023 #define ACCEL_GPIO_CLK        RCC_APB2Periph_GPIOC
00024
00025 #define ACCEL_ADC1_CLK        RCC_APB2Periph_ADC1
00026 #define ACCEL_DMA_CLK         RCC_AHBPeriph_DMA1
00027
00028 #define ACCEL_XOUT_PIN                GPIO_Pin_0
00029 #define ACCEL_XOUT_PIN_NUM        0
00030
00031 #define ACCEL_YOUT_PIN                GPIO_Pin_1
00032 #define ACCEL_YOUT_PIN_NUM        1
00033
00034 #define ACCEL_ZOUT_PIN                GPIO_Pin_2
00035 #define ACCEL_ZOUT_PIN_NUM        2
00036
00037 #define ACCEL_SLEEP_GPIO              GPIOA
00038 #define ACCEL_SLEEP_GPIO_CLK    RCC_APB2Periph_GPIOA
00039 #define ACCEL_SLEEP_PIN               GPIO_Pin_11
00040 #define ACCEL_SLEEP_PIN_NUM           11
00041
00042 void Blox_Accel_Init(void);
00043 uint16_t Blox_Accel_GetX(void);
00044 uint16_t Blox_Accel_GetY(void);
00045 uint16_t Blox_Accel_GetZ(void);
00046
00047 uint8_t Blox_Accel_GetXTilt(void);
00048 uint8_t Blox_Accel_GetYTilt(void);
00049 uint8_t Blox_Accel_GetZTilt(void);
00051 #endif
```

6.3 drivers/inc/blox_counter.h File Reference

Contains function prototypes for the counter.

```
#include "stm32f10x.h"
```

Functions

- void **SysTick_Init** (void)

Initializes the SysTick driver.

- uint32_t **SysTick_Get_Milliseconds** (void)

  *Returns the number of milliseconds since **SysTick_Init()** (p. 16) was called.*

- uint32_t **SysTick_Get_Seconds** (void)

  *Returns the number of seconds since **SysTick_Init()** (p. 16) was called.*

- uint32_t **SysTick_Get_Minutes** (void)

  *Returns the number of minutes since **SysTick_Init()** (p. 16) was called.*

- void **SysTick_Wait** (uint32_t ms)

  Performs a blocking wait for ms milliseconds.

6.3.1 Detailed Description

Contains function prototypes for the counter.

Author

Zach Wasson

Version

V0.1

Date

10/30/2010

Definition in file **blox_counter.h**.

6.4 drivers/inc/blox_counter.h

```
00001
00008 #ifndef __BLOX_COUNTER_H
00009 #define __BLOX_COUNTER_H
00010
00011 #include "stm32f10x.h"
00016 void SysTick_Init(void);
00017 uint32_t SysTick_Get_Milliseconds(void);
00018 uint32_t SysTick_Get_Seconds(void);
00019 uint32_t SysTick_Get_Minutes(void);
00020 void SysTick_Wait(uint32_t ms);
00022 #endif
00023
```

6.5 drivers/inc/blox_debug.h File Reference

Contains global debugging information for different modules to use.

Defines

- #define **BLOX_DEBUG** 0
- #define **Blox_Debug_Init**()
- #define **Blox_Debug**(data)
- #define **Blox_DebugStr**(data)
- #define **Blox_DebugPat**(format,...)

6.5.1 Detailed Description

Contains global debugging information for different modules to use.

Author

Jesse Tannahill

Version

V1.0

Date

10/27/2010

Definition in file **blox_debug.h**.

6.6 drivers/inc/blox_debug.h

```
00001
00009 #ifndef __BLOX_DEBUG_H
00010 #define __BLOX_DEBUG_H
00011
00016 #define BLOX_DEBUG 0
00017 #if BLOX_DEBUG
00018   #include "string.h"
00019   #include "blox_usb.h"
00020
00021   #define Blox_Debug_Init() USB_Init();
00022   #define Blox_Debug(data) USB_Send(data);
00023   #define Blox_DebugStr(data) { USB_SendData((uint8_t *)data, strlen(data)); }
00024   #define Blox_DebugPat(format, ...) { USB_SendPat(format, __VA_ARGS__); }
00025
```

```
00026 #else
00027   #define Blox_Debug_Init()
00028   #define Blox_Debug(data)
00029   #define Blox_DebugStr(data)
00030   #define Blox_DebugPat(format, ...)
00031 #endif
00032
00034 #endif
```

6.7 drivers/inc/blox_exti.h File Reference

Contains function prototypes for the EXTI interface.

```
#include "blox_system.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "misc.h"
```

- enum **EXTI_ID** {
  EXTI_INVALID_LINE = -2, **EXTI_IRQ_UNAVAILABLE**, **EXTI0**, **EXTI1**,
  EXTI2, **EXTI3**, **EXTI4**, **EXTI5**,
  EXTI6, **EXTI7**, **EXTI8**, **EXTI9**,
  EXTI10, **EXTI11**, **EXTI12**, **EXTI13**,
  EXTI14, **EXTI15** }
  Enum for possible EXTI ids.
- void **Blox_EXTI_Init** (void)
  Initializes EXTI.
- **EXTI_ID Blox_EXTI_Register_HW_IRQ** (uint8_t GPIO_PortSource, uint8_t line, void(∗EXTI_-Handler)(void))
  Registers an EXTI IRQ that triggers on hardware.
- **EXTI_ID Blox_EXTI_Register_SW_IRQ** (void(∗EXTI_Handler)(void))
  Registers an EXTI IRQ that triggers on software.
- void **Blox_EXTI_Release_IRQ** (**EXTI_ID** id)
  Releases a given EXTI interrupt.

- void **Blox_EXTI_Trigger_SW_IRQ** (**EXTI_ID** id)

  Registers an EXTI IRQ that triggers on software.

- void **Blox_EXTI_Enable_IRQ** (**EXTI_ID** id)

  Enables a given EXTI IRQ.

- void **Blox_EXTI_Disable_IRQ** (**EXTI_ID** id)

  Disables a given EXTI IRQ.

6.7.1 Detailed Description

Contains function prototypes for the EXTI interface.

Author

Zach Wasson

Version

V0.1

Date

10/20/2010

Definition in file **blox_exti.h**.

6.8 drivers/inc/blox_exti.h

```
00001
00008 #ifndef __BLOX_EXTI_H
00009 #define __BLOX_EXTI_H
00010
00011 #include "blox_system.h"
00012 #include "stm32f10x_exti.h"
00013 #include "stm32f10x_gpio.h"
00014 #include "stm32f10x_rcc.h"
00015 #include "misc.h"
00016
00023 typedef enum {
00024   EXTI_INVALID_LINE = -2,
00025   EXTI_IRQ_UNAVAILABLE,
00026   EXTI0,
00027   EXTI1,
00028   EXTI2,
```

```
00029   EXTI3,
00030   EXTI4,
00031   EXTI5,
00032   EXTI6,
00033   EXTI7,
00034   EXTI8,
00035   EXTI9,
00036   EXTI10,
00037   EXTI11,
00038   EXTI12,
00039   EXTI13,
00040   EXTI14,
00041   EXTI15
00042 } EXTI_ID;
00043
00044 void Blox_EXTI_Init(void);
00045 EXTI_ID Blox_EXTI_Register_HW_IRQ(uint8_t GPIO_PortSource, uint8_t line, void (*E
      XTI_Handler)(void));
00046 EXTI_ID Blox_EXTI_Register_SW_IRQ(void (*EXTI_Handler)(void));
00047 void Blox_EXTI_Release_IRQ(EXTI_ID id);
00048 void Blox_EXTI_Trigger_SW_IRQ(EXTI_ID id);
00049 void Blox_EXTI_Enable_IRQ(EXTI_ID id);
00050 void Blox_EXTI_Disable_IRQ(EXTI_ID id);
00052 #endif
```

6.9 drivers/inc/blox_fifo.h File Reference

Contains FIFO definition and function prototypes for FIFO interaction.

```
#include "stm32f10x.h"
```

Data Structures

- struct **FIFO_Type**

  Basic struct for each fifo.

- #define **FIFO_SIZE** 32
- #define **FIFO_BAD** 65536
- enum **FIFO_STATUS** { **FIFO_EMPTY** = -1, **FIFO_OK**, **FIFO_FULL** }

  Enum to return the status of the FIFO driver.

- void **Blox_FIFO_Init** (**FIFO_Type** *fifo)

  *Initializes the **FIFO_Type** (p. 98) structure provided.*

- **FIFO_STATUS Blox_FIFO_Put** (**FIFO_Type** *fifo, uint16_t data)

  Puts data in fifo.

- uint32_t **Blox_FIFO_Get** (**FIFO_Type** *fifo)

  Gets data out of fifo.

- uint8_t **Blox_FIFO_Size** (**FIFO_Type** *fifo)

  Gets size of fifo.

6.9.1 Detailed Description

Contains FIFO definition and function prototypes for FIFO interaction.

Author

Zach Wasson

Version

V0.1

Date

10/20/2010

Definition in file **blox_fifo.h**.

6.10 drivers/inc/blox_fifo.h

```
00001
00008 #ifndef __BLOX_FIFO_H
00009 #define __BLOX_FIFO_H
00010
00011 #include "stm32f10x.h"
00012
00017 /* FIFO_SIZE must be a power of two and less than 256*/
00018 #define FIFO_SIZE 32
00019 #define FIFO_BAD  65536
00020
00024 typedef struct {
00025   uint16_t data[FIFO_SIZE];
00026   uint8_t read;
00027   uint8_t write;
00028 } FIFO_Type;
00029
00033 typedef enum {
00034   FIFO_EMPTY = -1,
```

```
00035   FIFO_OK,
00036   FIFO_FULL
00037 } FIFO_STATUS;
00038
00039 void Blox_FIFO_Init(FIFO_Type *fifo);
00040 FIFO_STATUS Blox_FIFO_Put(FIFO_Type *fifo, uint16_t data);
00041 uint32_t Blox_FIFO_Get(FIFO_Type *fifo);
00042 uint8_t Blox_FIFO_Size(FIFO_Type *fifo);
00044 #endif
```

6.11 drivers/inc/blox_filesystem.h File Reference

Contains function prototypes for the filesystem interface.

```
#include "blox_system.h"
#include "stm32f10x_flash.h"
#include "misc.h"
#include "string.h"
```

Data Structures

- struct **FS_File**

  Defines a file's header. Contains the id of the file within the FAT, the size in bytes of the file, and a pointer to the data.

- struct **FS_Table**

  The table of all FS_Files.

- #define **FS_MAX_FILES** 16
- #define **FS_MAGIC** 0xdeadbeef
- #define **FS_FILE_MAX_NAME_LEN** 32
- #define **FS_APP_FLAG_LOC** 0x20005000
- enum **FS_STATUS** {

  FS_CREATE_FAIL = -6, **FS_BAD_WRITE**, **FS_FULL**, **FS_FILE_NOT_INIT**,

  FS_FAT_NOT_INIT, **FS_BAD_FAT**, **FS_OK** }

  Enum containing the status of the filesystem.

- **FS_STATUS FS_Init** (uint8_t create)
- **FS_STATUS FS_ChkValid** (void)

- **FS_File** ∗ **FS_GetFile** (uint8_t id)

  Retrieves a file handle from the filesystem given a specific file id.

- **FS_File** ∗ **FS_GetFileFromName** (char ∗name)

  Gets a file in the filesystem by looking for the filename.

- uint8_t **FS_GetNumFiles** (void)

  Returns the number of files being managed by the filesystem.

- **FS_STATUS FS_DeleteFile** (uint8_t id)

  Deletes a file from the filesystem. Shifts data up so fragmentation does not occur.

- uint8_t **FS_CreateFile** (char ∗name, uint8_t numPages)

  Creates a new file of a given size in the filesystem.

- **FS_STATUS FS_WriteFilePage** (uint8_t id, uint32_t ∗data, uint32_t page_offset)

  Writes data into a file in the filesystem at a given offset.

- **FS_STATUS FS_CreateFS** (void)

  Creates a filesystem at the default location. Should only be used once.

- void **FS_SwapPage** (uint32_t ∗src, uint32_t ∗dst)

  Swaps a page in RAM with a page in Flash.

- uint32_t **FS_RoundPageUp** (uint32_t size)

  Rounds a size up to the next multiple of PAGE_SIZE.

- void **FS_RunFile** (uint8_t file_id)

  De-initializes the system and runs the application stored at the file id.

- void **FS_RunStage** (void)

  Runs the application stored in the staging area. Assumes from a system reset.

- uint8_t **FS_GetAppFlag** (void)

  Returns the flag designating if an application should be run.

- void **FS_SetAppFlag** (uint8_t val)

  Sets the flag desinating if an application should be run.

6.11.1 Detailed Description

Contains function prototypes for the filesystem interface.

Author

Jesse Tannahill

Version

V0.1

Date

10/18/2010

Definition in file **blox_filesystem.h**.

6.11.2 Define Documentation

6.11.2.1 #define FS_MAX_FILES 16

For our processor, Flash goes from 0x08000000 - 0x08080000, 512K 512K total, 2K page, 4K sector size

Definition at line **40** of file **blox_filesystem.h**.

6.11.3 Function Documentation

6.11.3.1 FS_STATUS FS_ChkValid (void)

Determine if the FAT is in a valid state.

Return values

FS_OK	if valid, FS_BAD_FAT on failure

Definition at line **70** of file **blox_filesystem.c**.

6.11.3.2 uint8_t FS_CreateFile (char * *name,* uint8_t *numPages*)

Creates a new file of a given size in the filesystem.

Parameters

name	the name of the new file
numPages	the number of pages the new file needs

Return values

The	unique id of the file within the filesystem. FS_MAX_FILES on error.

Definition at line **187** of file **blox_filesystem.c**.

6.11.3.3 FS_STATUS FS_CreateFS (void)

Creates a filesystem at the default location. Should only be used once.

Return values

FS_OK	if successful, another FS_STATUS if not.

Definition at line **42** of file **blox_filesystem.c**.

6.11.3.4 FS_STATUS FS_DeleteFile (uint8_t *id*)

Deletes a file from the filesystem. Shifts data up so fragmentation does not occur.

Parameters

id	the unique id of the file within the filesystem.

Return values

An	FS_STATUS indicating whether the delete was successful.

Definition at line **145** of file **blox_filesystem.c**.

6.11.3.5 uint8_t FS_GetAppFlag (void)

Returns the flag designating if an application should be run.

Return values

The	flag.

Definition at line **303** of file **blox_filesystem.c**.

6.11.3.6 FS_File* FS_GetFile (uint8_t *id*)

Retrieves a file handle from the filesystem given a specific file id.

Parameters

id	the unique id of the file within the filesystem.

Return values

The	**FS_File** (p. 99) ∗ of the file being requested. 0 on error

Definition at line **108** of file **blox_filesystem.c**.

6.11.3.7 FS_File∗ FS_GetFileFromName (char ∗ *name*)

Gets a file in the filesystem by looking for the filename.

Parameters

name	the name of the application.

Return values

NULL	if the file can't be found, a pointer otherwise.

Definition at line **121** of file **blox_filesystem.c**.

6.11.3.8 uint8_t FS_GetNumFiles (void)

Returns the number of files being managed by the filesystem.

Return values

The	number of files being managed by the filesystem.

Definition at line **137** of file **blox_filesystem.c**.

6.11.3.9 uint32_t FS_RoundPageUp (uint32_t *size*)

Rounds a size up to the next multiple of PAGE_SIZE.

Parameters

size	the size to be rounded.

Return values

the	rounded size.

Definition at line **253** of file **blox_filesystem.c**.

6.11.3.10 void FS_RunFile (uint8_t *file_id*)

De-initializes the system and runs the application stored at the file id.

Parameters

file_id	the id of the file to be run.

Return values

None.	

Definition at line **265** of file **blox_filesystem.c**.

6.11.3.11 void FS_RunStage (void)

Runs the application stored in the staging area. Assumes from a system reset.

Return values

None.	

Definition at line **282** of file **blox_filesystem.c**.

6.11.3.12 void FS_SetAppFlag (uint8_t *val*)

Sets the flag desinating if an application should be run.

Parameters

val	The flag.

Return values

None.	

Definition at line **312** of file **blox_filesystem.c**.

6.11.3.13 void FS_SwapPage (uint32_t * *src,* uint32_t * *dst*)

Swaps a page in RAM with a page in Flash.

Parameters

src	the address of the start of the page in RAM
dst	the address of the start of the page in Flash

Return values

None.	

Definition at line **240** of file **blox_filesystem.c**.

6.11.3.14 FS_STATUS FS_WriteFilePage (uint8_t *id*, uint32_t * *data*, uint32_t *page_offset*)

Writes data into a file in the filesystem at a given offset.

Parameters

id	the unique id of the file within the filesystem
data	a pointer to the buffer to be copied
page_offset	the offset within the file the buffer should be copied to.

Return values

An	FS_STATUS denoting whether the write was successful.

Definition at line **227** of file **blox_filesystem.c**.

6.12 drivers/inc/blox_filesystem.h

```
00001
00009 #ifndef __BLOX_FILESYSTEM_H
00010 #define __BLOX_FILESYSTEM_H
00011
00012 #include "blox_system.h"
00013 #include "stm32f10x_flash.h"
00014 #include "misc.h"
00015
00016 #include "string.h"
00017
00026 typedef enum {
00027   FS_CREATE_FAIL = -6,
00028   FS_BAD_WRITE,
00029         FS_FULL,
00030         FS_FILE_NOT_INIT,
00031         FS_FAT_NOT_INIT,
00032         FS_BAD_FAT,
00033   FS_OK
00034 } FS_STATUS;
00035
```

```
00040 #define FS_MAX_FILES 16
00041 #define FS_MAGIC 0xdeadbeef
00042 #define FS_FILE_MAX_NAME_LEN 32
00043 #define FS_APP_FLAG_LOC 0x20005000
00044
00050 typedef struct {
00051         uint8_t id;
00052   char name[FS_FILE_MAX_NAME_LEN];
00053         uint8_t numPages;
00054         uint32_t *data;
00055 } FS_File;
00056
00060 typedef struct {
00061   volatile uint32_t magic;
00062         volatile uint32_t numFiles;
00063         uint32_t *free_top;
00064   uint32_t free_numPages;
00065         FS_File table[FS_MAX_FILES];
00066 } FS_Table;
00067
00068
00069 FS_STATUS FS_Init(uint8_t create);
00070 FS_STATUS FS_ChkValid(void);
00071 FS_File * FS_GetFile(uint8_t id);
00072 FS_File * FS_GetFileFromName(char *name);
00073 uint8_t FS_GetNumFiles(void);
00074 FS_STATUS FS_DeleteFile(uint8_t id);
00075 uint8_t FS_CreateFile(char *name, uint8_t numPages);
00076 FS_STATUS FS_WriteFilePage(uint8_t id, uint32_t *data, uint32_t page_offset);
00077 FS_STATUS FS_CreateFS(void);
00078 void FS_SwapPage(uint32_t *src, uint32_t *dst);
00079 uint32_t FS_RoundPageUp(uint32_t size);
00080 void FS_RunFile(uint8_t file_id);
00081 void FS_RunStage(void);
00082 uint8_t FS_GetAppFlag(void);
00083 void FS_SetAppFlag(uint8_t val);
00085 #endif
```

6.13 drivers/inc/blox_ir.h File Reference

Contains function prototypes for the IR interface.

```
#include "stm32f10x.h"
#include "blox_usart.h"
#include "stdio.h"
#include "string.h"
```

Data Structures

- struct **IRFrame**

*Defines data an **IRFrame** (p. 103) sends.*

- #define **IR_1_USART_ID** 4
- #define **IR_2_USART_ID** 5
- #define **IR_3_USART_ID** 2
- #define **IR_4_USART_ID** 3
- #define **IR_SHUTDOWN_GPIO** GPIOD
- #define **IR_SHUTDOWN_GPIO_CLK** RCC_APB2Periph_GPIOD
- #define **IR_SHUTDOWN_PIN** GPIO_Pin_13
- #define **IR_SHUTDOWN_PIN_NUM** 13
- #define **IR_MAX_FRAME_LEN** 100
- enum **IR_DIR** { **IR_EAST_ID** = 1, **IR_SOUTH_ID**, **IR_NORTH_ID**, **IR_WEST_ID** }

  A mapping from IR ID to cardinal directions.

- enum **IRFrameType** { **IR_FRAME_TYPE_NEIGHBOR**, **IR_FRAME_TYPE_USER** }

  Mapping for types of IR frames.

- void **IR_Init** (uint8_t id)

  Initializes the IR module. Basically a wrapper on USART.

- uint8_t **IR_Receive** (uint8_t id)

  Receive a byte on the given IR. A wrapper around USART.

- uint8_t **IR_TryReceive** (uint8_t id)

  Receive a byte on the given IR. A wrapper around USART.

- void **IR_Send** (uint8_t id, uint8_t data)

  Sends a byte out on the given IR. Wrapper around USART.

- void **IR_Sleep** (void)

  Put all the IRs to sleep.

- void **IR_Wake** (void)

  Wakes all the IRs from sleep.

- void **Blox_IR_Register_RX_IRQ** (uint8_t id, void(∗RX_Handler)(**IRFrame** ∗frame))

  Registers a function to execute when a complete IR frame is received.

- void **Blox_IR_Enable_RX_IRQ** (uint8_t id)

Enables the sw interrupt that occurs when a XBee reads a byte.

- void **Blox_IR_Disable_RX_IRQ** (uint8_t id)

  Disables the sw interrupt that occurs when a XBee reads a byte.

- void **IR_SendFrame** (uint8_t id, **IRFrameType** type, uint8_t *data, uint8_t len)

  *Sends a **IRFrame*** (p. *103)* *out on the given IR.*

6.13.1 Detailed Description

Contains function prototypes for the IR interface.

Author

Jesse Tannahill

Version

V0.1

Date

10/19/2010

Definition in file **blox_ir.h**.

6.13.2 Function Documentation

6.13.2.1 void Blox_IR_Disable_RX_IRQ (uint8_t *id*)

Disables the sw interrupt that occurs when a XBee reads a byte.

Parameters

id	the IR id

Return values

None.	

Definition at line **409** of file **blox_ir.c**.

6.13.2.2 void Blox_IR_Enable_RX_IRQ (uint8_t *id*)

Enables the sw interrupt that occurs when a XBee reads a byte.

Parameters

id	the IR id

Return values

None.	

Definition at line **383** of file **blox_ir.c**.

6.13.2.3 void Blox_IR_Register_RX_IRQ (uint8_t *id,* void(∗)(IRFrame ∗frame) *RX_Handler*)

Registers a function to execute when a complete IR frame is received.

Parameters

id	the IR id
RX_Handler	the user function to call on interrupt

Return values

None.	

Definition at line **361** of file **blox_ir.c**.

6.13.2.4 void IR_Init (uint8_t *id*)

Initializes the IR module. Basically a wrapper on USART.

Parameters

id	the id of the USART interface to initialize.

Return values

None	

Definition at line **65** of file **blox_ir.c**.

6.13.2.5 uint8_t IR_Receive (uint8_t *id*)

Receive a byte on the given IR. A wrapper around USART.

Parameters

id	the IR id to use.

Return values

The	received command or 0 on error.

Definition at line **435** of file **blox_ir.c**.

6.13.2.6 void IR_Send (uint8_t *id,* uint8_t *data*)

Sends a byte out on the given IR. Wrapper around USART.

Parameters

id	the IR id to use
data	the byte to send

Return values

None.	

Definition at line **474** of file **blox_ir.c**.

6.13.2.7 void IR_SendFrame (uint8_t *id,* IRFrameType *type,* uint8_t * *data,* uint8_t *len*)

Sends a **IRFrame** (p. 103) out on the given IR.

Parameters

id	the IR id to use
type	the type of the **IRFrame** (p. 103)
data	a pointer to the data to be sent
len	the length of the data to be sent

Return values

None.	

Definition at line **499** of file **blox_ir.c**.

6.13.2.8 void IR_Sleep (void)

Put all the IRs to sleep.

Return values

None.	

Definition at line **566** of file **blox_ir.c**.

6.13.2.9 uint8_t IR_TryReceive (uint8_t *id*)

Receive a byte on the given IR. A wrapper around USART.

Parameters

id	the IR id to use.

Return values

The	received command or 0 on error.

Definition at line **454** of file **blox_ir.c**.

6.13.2.10 void IR_Wake (void)

Wakes all the IRs from sleep.

Return values

None.	

Definition at line **558** of file **blox_ir.c**.

6.14 drivers/inc/blox_ir.h

```
00001
00008 #ifndef __BLOX_IR_H
00009 #define __BLOX_IR_H
00010
00011 #include "stm32f10x.h"
00012 #include "blox_usart.h"
00013
00014 #include "stdio.h"
00015 #include "string.h"
00016
00023 typedef enum {
00024   IR_EAST_ID = 1,
00025   IR_SOUTH_ID,
00026   IR_NORTH_ID,
00027   IR_WEST_ID
```

```
00028 } IR_DIR;
00029
00030 #define IR_1_USART_ID 4
00031 #define IR_2_USART_ID 5
00032 #define IR_3_USART_ID 2
00033 #define IR_4_USART_ID 3
00034
00035 #define IR_SHUTDOWN_GPIO                GPIOD
00036 #define IR_SHUTDOWN_GPIO_CLK    RCC_APB2Periph_GPIOD
00037 #define IR_SHUTDOWN_PIN                 GPIO_Pin_13
00038 #define IR_SHUTDOWN_PIN_NUM             13
00039
00043 typedef enum {
00044   IR_FRAME_TYPE_NEIGHBOR,
00045   IR_FRAME_TYPE_USER
00046 } IRFrameType;
00047
00048 #define IR_MAX_FRAME_LEN 100
00049
00052 typedef struct {
00053   uint8_t src_id;
00054   uint8_t src_face_id;
00055   IRFrameType type;
00056   uint8_t len;
00057   uint8_t *data;
00058   uint8_t checksum;
00059 } IRFrame;
00060
00061 void IR_Init(uint8_t id);
00062 uint8_t IR_Receive(uint8_t id);
00063 uint8_t IR_TryReceive(uint8_t id);
00064 void IR_Send(uint8_t id, uint8_t data);
00065 void IR_Sleep(void);
00066 void IR_Wake(void);
00067 void Blox_IR_Register_RX_IRQ(uint8_t id, void (*RX_Handler)(IRFrame *frame));
00068 void Blox_IR_Enable_RX_IRQ(uint8_t id);
00069 void Blox_IR_Disable_RX_IRQ(uint8_t id);
00070 void IR_SendFrame(uint8_t id, IRFrameType type, uint8_t *data, uint8_t len);
00072 #endif
```

6.15 drivers/inc/blox_led.h File Reference

Contains function prototypes for the LEDs.

```
#include "blox_system.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
```

- #define **LED_GPIO** GPIOC

- #define **LED_CLK** RCC_APB2Periph_GPIOC
- #define **LED1_GPIO_Pin** GPIO_Pin_6
- #define **LED2_GPIO_Pin** GPIO_Pin_7
- #define **LED3_GPIO_Pin** GPIO_Pin_8
- #define **LED4_GPIO_Pin** GPIO_Pin_9
- #define **LED_EAST** LED1
- #define **LED_SOUTH** LED2
- #define **LED_WEST** LED3
- #define **LED_NORTH** LED4
- enum **LED_ID** { **LED1** = 1, **LED2**, **LED3**, **LED4** }

  A mapping from LED ID to cardinal directions.

- void **Blox_LED_Init** (void)

  Initializes the LEDs.

- void **Blox_LED_On** (**LED_ID** id)

  Turns an LED on.

- void **Blox_LED_Off** (**LED_ID** id)

  Turns an LED off.

- void **Blox_LED_Toggle** (**LED_ID** id)

  Toggles an LED.

6.15.1 Detailed Description

Contains function prototypes for the LEDs.

Author

Zach Wasson

Version

V0.1

Date

10/30/2010

Definition in file **blox_led.h**.

6.15.2 Function Documentation

6.15.2.1 void Blox_LED_Init (void)

Initializes the LEDs.

Return values

None	

Definition at line **22** of file **blox_led.c**.

6.15.2.2 void Blox_LED_Off (LED_ID *id*)

Turns an LED off.

Parameters

id	the id of the LED

Return values

None	

Definition at line **60** of file **blox_led.c**.

6.15.2.3 void Blox_LED_On (LED_ID *id*)

Turns an LED on.

Parameters

id	the id of the LED

Return values

None	

Definition at line **43** of file **blox_led.c**.

6.15.2.4 void Blox_LED_Toggle (LED_ID *id*)

Toggles an LED.

Parameters

id	the id of the LED

Return values

None	

Definition at line **77** of file **blox_led.c**.

6.16 drivers/inc/blox_led.h

```
00001
00008 #ifndef __BLOX_LED_H
00009 #define __BLOX_LED_H
00010
00011 #include "blox_system.h"
00012 #include "stm32f10x_gpio.h"
00013 #include "stm32f10x_rcc.h"
00014
00019 #define LED_GPIO        GPIOC
00020 #define LED_CLK         RCC_APB2Periph_GPIOC
00021 #define LED1_GPIO_Pin   GPIO_Pin_6
00022 #define LED2_GPIO_Pin   GPIO_Pin_7
00023 #define LED3_GPIO_Pin   GPIO_Pin_8
00024 #define LED4_GPIO_Pin   GPIO_Pin_9
00025
00029 typedef enum {
00030   LED1 = 1,
00031   LED2,
00032   LED3,
00033   LED4
00034 } LED_ID;
00035
00036 #define LED_EAST  LED1
00037 #define LED_SOUTH LED2
00038 #define LED_WEST  LED3
00039 #define LED_NORTH LED4
00040
00041 void Blox_LED_Init(void);
00042 void Blox_LED_On(LED_ID id);
00043 void Blox_LED_Off(LED_ID id);
00044 void Blox_LED_Toggle(LED_ID id);
00046 #endif
```

6.17 drivers/inc/blox_oled.h File Reference

Contains function prototypes for the OLED interface.

```
#include "blox_system.h"
```

```
#include "blox_counter.h"
#include "blox_vusart.h"
```

- #define **OLED_USART_ID** 2
- #define **OLED_RESET_GPIO** GPIOC
- #define **OLED_RESET_GPIO_CLK** RCC_APB2Periph_GPIOC
- #define **OLED_RESET_PIN** GPIO_Pin_3
- #define **OLED_RESET_PIN_NUM** 3
- #define **DELAY_2N** 0
- #define **OLED_BAUDRATE** 9600
- #define **OLED_RESETPIN** 8
- #define **OLED_INITDELAYMS** 5000
- #define **OLED_ACK** 0x06
- #define **OLED_NAK** 0x15
- #define **OLED_AUTOBAUD** 0x55
- #define **OLED_DEVICE_INFO** 0x56
- #define **OLED_BKG_COLOR** 0x42
- #define **OLED_CLEAR** 0x45
- #define **OLED_DISPLAY_CONTROL** 0x59
- #define **OLED_COMMAND_DISPLAY** 0x01
- #define **OLED_COMMAND_CONTRAST** 0x02
- #define **OLED_COMMAND_POWER** 0x03
- #define **OLED_SLEEP** 0x5A
- #define **OLED_ADD_USER_CHAR** 0x41
- #define **OLED_DRAW_CIRCLE** 0x43
- #define **OLED_DRAW_USER_CHAR** 0x44
- #define **OLED_DRAW_TRIANGLE** 0x47
- #define **OLED_DRAW_ICON** 0x49
- #define **OLED_OPAQUE_BKG_COLOR** 0x4B
- #define **OLED_DRAW_LINE** 0x4C
- #define **OLED_DRAW_PIXEL** 0x50
- #define **OLED_READ_PIXEL** 0x52
- #define **OLED_SCREEN_COPY_PASTE** 0x63
- #define **OLED_DRAW_POLYGON** 0x67
- #define **OLED_REPLACE_COLOR** 0x6B
- #define **OLED_SET_PIN_SIZE** 0x70
- #define **OLED_DRAW_RECT** 0x72
- #define **OLED_SET_FONT** 0x46
- #define **OLED_FONT_5X7** 0x00
- #define **OLED_FONT_8X8** 0x01

- #define **OLED_FONT_8X12** 0x02
- #define **OLED_SET_VIS** 0x4F
- #define **OLED_SET_VIS_TRANS** 0x00
- #define **OLED_SET_VIS_OPAQ** 0x01
- #define **OLED_DRAW_STRING_GRAPHICS** 0x53
- #define **OLED_DRAW_CHAR_TEXT** 0x54
- #define **OLED_DRAW_TEXT_BUTTON** 0x62
- #define **OLED_DRAW_STRING_TEXT** 0x73
- #define **OLED_DRAW_CHAR_GRAPHICS** 0x74
- #define **OLED_SD_CMD_EXT** 0x40
- #define **OLED_SD_CMODE_256** 0x08
- #define **OLED_SD_CMODE_65K** 0x10
- #define **OLED_SD_SET_ADDRESS_PNT** 0x41
- #define **OLED_SD_SCREEN_SAVE** 0x43
- #define **OLED_SD_DISPLAY_ICON** 0x49
- #define **OLED_SD_DISPLAY_OBJECT** 0x4F
- #define **OLED_SD_RUN_SCRIPT** 0x50
- #define **OLED_SD_READ_SECTOR** 0x52
- #define **OLED_SD_DISPLAY_VID** 0x56
- #define **OLED_SD_WRITE_SECTOR** 0x57
- #define **OLED_SD_INIT_CARD** 0x69
- #define **OLED_SD_READ_BYTE** 0x72
- #define **OLED_SD_WRITE_BYTE** 0x77
- #define **OLED_IMG_0** 0x001000
- #define **OLED_IMG_1** 0x001010
- #define **OLED_IMG_2** 0x001010
- #define **OLED_IMG_3** 0x001010
- #define **OLED_IMG_4** 0x001010
- #define **OLED_IMG_5** 0x001010
- #define **OLED_IMG_6** 0x001010
- #define **OLED_IMG_7** 0x001010
- #define **OLED_IMG_8** 0x001010
- #define **OLED_IMG_9** 0x001010
- #define **OLED_IMG_10** 0x001010
- #define **OLED_IMG_11** 0x001010
- #define **OLED_IMG_12** 0x001010
- #define **OLED_IMG_13** 0x001010
- #define **OLED_IMG_14** 0x001010
- #define **COLOR_BLACK** 0x0000
- #define **COLOR_BLUE** 0x001F
- #define **COLOR_WHITE** 0xFFFF
- #define **COLOR_YELLOW** 0xFFE0
- #define **COLOR_GRAPE** 0x8210

- #define **COLOR_LIGHT_BROWN** 0xBAA7
- #define **COLOR_DARK_BROWN** 0x7000
- void **Blox_OLED_Init** (void)

  Initializes the OLED display.

- uint8_t **Blox_OLED_Receive** (void)

  Receive a byte from the OLED. Wrapper around USART.

- void **Blox_OLED_Send** (uint8_t data)

  Send a byte to the OLED. Wrapper around USART.

- void **Blox_OLED_Clear** (void)

  Clear OLED display.

- void **Blox_OLED_DrawCircle** (uint8_t x, uint8_t y, uint8_t radius, uint16_t color)

  Draw circle on OLED display.

- void **Blox_OLED_DrawLine** (uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, uint16_t color)

  Draw line on OLED display.

- void **Blox_OLED_DrawPixel** (uint8_t x, uint8_t y, uint16_t color)

  Draw pixel on OLED display.

- void **Blox_OLED_DrawRectangle** (uint8_t x, uint8_t y, uint8_t width, uint8_t height, uint16_t color)

  Draw line on OLED display.

- void **Blox_OLED_SetFont** (uint8_t font)

  Sets font type for OLED characters.

- void **Blox_OLED_SetOpaque** (void)

  Sets the font to have an opaque background.

- void **Blox_OLED_DrawStringGraphics** (uint8_t x, uint8_t y, uint8_t font, uint16_t color, uint8_t width, uint8_t height, uint8_t ∗string)

  Draw graphics formatted string on OLED display.

- void **Blox_OLED_DrawCharText** (uint8_t character, uint8_t column, uint8_t row, uint16_t color)

  Draw text formatted character on OLED display.

- void **Blox_OLED_DrawStringText** (uint8_t column, uint8_t row, uint8_t font_size, uint16_t color, uint8_t *string)

  Draw text formatted string on OLED display.

- void **Blox_OLED_DrawCharGraphics** (uint8_t character, uint8_t x, uint8_t y, uint16_t color, uint8_t width, uint8_t height)

  Draw graphics formatted character on OLED display.

- void **Blox_OLED_SD_DisplayIcon** (uint8_t x, uint8_t y, uint8_t width, uint8_t height, uint32_t sector)

  Displays bitmap image icon stored in SD card onto OLED screen.

6.17.1 Detailed Description

Contains function prototypes for the OLED interface.

Author

Dan Cleary

Version

V0.1

Date

10/20/2010

Definition in file **blox_oled.h**.

6.18 drivers/inc/blox_oled.h

```
00001
00009 #ifndef __BLOX_OLED_H
00010 #define __BLOX_OLED_H
00011
00012 #include "blox_system.h"
00013 #include "blox_counter.h"
00014 #include "blox_vusart.h"
00015
00020 #define OLED_USART_ID 2
00021
00022 #define OLED_RESET_GPIO                 GPIOC
00023 #define OLED_RESET_GPIO_CLK     RCC_APB2Periph_GPIOC
00024 #define OLED_RESET_PIN                  GPIO_Pin_3
00025 #define OLED_RESET_PIN_NUM              3
```

```
00026
00027
00028 /* If processor works on high frequency delay has to be increased, it can be
00029    increased by factor 2^N by this constant                                   */
00030 #define DELAY_2N     0
00031
00032 #define OLED_BAUDRATE                 9600//57600
00033 #define OLED_RESETPIN                 8      // PIN of reset
00034 #define OLED_INITDELAYMS              5000
00035
00036 #define OLED_ACK  0x06  // Ok
00037 #define OLED_NAK  0x15  // Error
00038
00039 /*****************************************/
00040 /*            General Commands            */
00041 /*****************************************/
00042
00043 #define OLED_AUTOBAUD                  0x55
00044 #define OLED_DEVICE_INFO               0x56
00045 #define OLED_BKG_COLOR                 0x42
00046 #define OLED_CLEAR                     0x45
00047 #define OLED_DISPLAY_CONTROL           0x59
00048 #define OLED_COMMAND_DISPLAY                                    0x01
00049 #define OLED_COMMAND_CONTRAST                                   0x02
00050 #define OLED_COMMAND_POWER                                        0x03
00051 #define OLED_SLEEP                     0x5A
00052
00053 /*****************************************/
00054 /*            Graphics Commands           */
00055 /*****************************************/
00056 #define OLED_ADD_USER_CHAR             0x41
00057 #define OLED_DRAW_CIRCLE               0x43
00058 #define OLED_DRAW_USER_CHAR            0x44
00059 #define OLED_DRAW_TRIANGLE             0x47
00060 #define OLED_DRAW_ICON                 0x49
00061 #define OLED_OPAQUE_BKG_COLOR          0x4B
00062 #define OLED_DRAW_LINE                 0x4C
00063 #define OLED_DRAW_PIXEL                0x50
00064 #define OLED_READ_PIXEL                0x52
00065 #define OLED_SCREEN_COPY_PASTE         0x63
00066 #define OLED_DRAW_POLYGON              0x67
00067 #define OLED_REPLACE_COLOR             0x6B
00068 #define OLED_SET_PIN_SIZE              0x70
00069 #define OLED_DRAW_RECT                 0x72
00070
00071 /*****************************************/
00072 /*            Text Commands               */
00073 /*****************************************/
00074 #define OLED_SET_FONT                  0x46
00075   #define       OLED_FONT_5X7                0x00
00076   #define       OLED_FONT_8X8                0x01
00077   #define       OLED_FONT_8X12               0x02
00078
00079 #define OLED_SET_VIS                   0x4F
00080   #define OLED_SET_VIS_TRANS           0x00
00081   #define OLED_SET_VIS_OPAQ            0x01
00082 #define OLED_DRAW_STRING_GRAPHICS      0x53
```

```
00083 #define OLED_DRAW_CHAR_TEXT             0x54
00084 #define OLED_DRAW_TEXT_BUTTON           0x62
00085 #define OLED_DRAW_STRING_TEXT           0x73
00086 #define OLED_DRAW_CHAR_GRAPHICS         0x74
00087
00088
00089 /*****************************************/
00090 /*            SD Commands                */
00091 /*****************************************/
00092 #define OLED_SD_CMD_EXT                 0x40
00093 #define OLED_SD_CMODE_256               0x08
00094 #define OLED_SD_CMODE_65K               0x10
00095
00096 #define OLED_SD_SET_ADDRESS_PNT         0x41
00097 #define OLED_SD_SCREEN_SAVE             0x43
00098 #define OLED_SD_DISPLAY_ICON            0x49
00099 #define OLED_SD_DISPLAY_OBJECT          0x4F
00100 #define OLED_SD_RUN_SCRIPT              0x50
00101 #define OLED_SD_READ_SECTOR             0x52
00102 #define OLED_SD_DISPLAY_VID             0x56
00103 #define OLED_SD_WRITE_SECTOR            0x57
00104 #define OLED_SD_INIT_CARD               0x69
00105 #define OLED_SD_READ_BYTE               0x72
00106 #define OLED_SD_WRITE_BYTE              0x77
00107
00108 /*****************************************/
00109 /*            SD Sectors                 */
00110 /*****************************************/
00111 #define OLED_IMG_0                    0x001000
00112 #define OLED_IMG_1                    0x001010
00113 #define OLED_IMG_2                    0x001010
00114 #define OLED_IMG_3                    0x001010
00115 #define OLED_IMG_4                    0x001010
00116 #define OLED_IMG_5                    0x001010
00117 #define OLED_IMG_6                    0x001010
00118 #define OLED_IMG_7                    0x001010
00119 #define OLED_IMG_8                    0x001010
00120 #define OLED_IMG_9                    0x001010
00121 #define OLED_IMG_10                   0x001010
00122 #define OLED_IMG_11                   0x001010
00123 #define OLED_IMG_12                   0x001010
00124 #define OLED_IMG_13                   0x001010
00125 #define OLED_IMG_14                   0x001010
00126
00127 /*****************************************/
00128 /*                Colors                 */
00129 /*****************************************/
00130 #define COLOR_BLACK                   0x0000
00131 #define COLOR_BLUE                    0x001F
00132 #define COLOR_WHITE                   0xFFFF
00133 #define COLOR_YELLOW                  0xFFE0
00134 #define COLOR_GRAPE                   0x8210
00135 #define COLOR_LIGHT_BROWN             0xBAA7
00136 #define COLOR_DARK_BROWN              0x7000
00137
00138 //void OLED_Reset (void);
00139 void Blox_OLED_Init (void);
```

```
00140 uint8_t Blox_OLED_Receive (void);
00141 void Blox_OLED_Send (uint8_t data);
00142
00143 /* General Commands */
00144 void Blox_OLED_Clear(void);
00145
00146 /* Graphics Commands */
00147 void Blox_OLED_DrawCircle(uint8_t x, uint8_t y, uint8_t radius, uint16_t color);
00148 void Blox_OLED_DrawLine(uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, uint16_t
      color);
00149 void Blox_OLED_DrawPixel(uint8_t x, uint8_t y, uint16_t color);
00150 void Blox_OLED_DrawRectangle(uint8_t x, uint8_t y, uint8_t width, uint8_t height,
       uint16_t color);
00151
00152 /* Text Commands */
00153 void Blox_OLED_SetFont(uint8_t font);
00154 void Blox_OLED_SetOpaque(void);
00155 void Blox_OLED_DrawStringGraphics(uint8_t x, uint8_t y, uint8_t font, uint16_t co
      lor, uint8_t width, uint8_t height, uint8_t *string);
00156 void Blox_OLED_DrawCharText(uint8_t character, uint8_t column, uint8_t row, uint1
      6_t color);
00157 void Blox_OLED_DrawStringText(uint8_t column, uint8_t row, uint8_t font_size, uin
      t16_t color, uint8_t *string);
00158 void Blox_OLED_DrawCharGraphics(uint8_t character, uint8_t x, uint8_t y, uint16_t
       color, uint8_t width, uint8_t height);
00159
00160 /* SD Commands */
00161 void Blox_OLED_SD_DisplayIcon(uint8_t x, uint8_t y, uint8_t width, uint8_t height
      , uint32_t sector);
00163 #endif
```

6.19 drivers/inc/blox_speaker.h File Reference

Basic device driver header for Blox speaker.

Data Structures

- struct **Note**

  Defines data a note contains.

- #define **SpkClock** 52400
- #define **SINE_POINTS** 32
- #define **C7** SpkClock/2093/SINE_POINTS
- #define **B6** SpkClock/1975/SINE_POINTS
- #define **Bb6** SpkClock/1865/SINE_POINTS

- #define **A6** SpkClock/1760/SINE_POINTS
- #define **Ab6** SpkClock/1661/SINE_POINTS
- #define **G6** SpkClock/1567/SINE_POINTS
- #define **Gb6** SpkClock/1480/SINE_POINTS
- #define **F6** SpkClock/1396/SINE_POINTS
- #define **E6** SpkClock/1319/SINE_POINTS
- #define **Eb6** SpkClock/1245/SINE_POINTS
- #define **D6** SpkClock/1175/SINE_POINTS
- #define **Db6** SpkClock/1108/SINE_POINTS
- #define **C6** SpkClock/1046/SINE_POINTS
- #define **B5** SpkClock/988/SINE_POINTS
- #define **Bb5** SpkClock/932/SINE_POINTS
- #define **A5** SpkClock/880/SINE_POINTS
- #define **Ab5** SpkClock/830/SINE_POINTS
- #define **G5** SpkClock/784/SINE_POINTS
- #define **Gb5** SpkClock/740/SINE_POINTS
- #define **F5** SpkClock/698/SINE_POINTS
- #define **E5** SpkClock/659/SINE_POINTS
- #define **Eb5** SpkClock/622/SINE_POINTS
- #define **D5** SpkClock/587/SINE_POINTS
- #define **Db5** SpkClock/554/SINE_POINTS
- #define **C5** SpkClock/523/SINE_POINTS
- #define **B4** SpkClock/494/SINE_POINTS
- #define **Bb4** SpkClock/466/SINE_POINTS
- #define **A4** SpkClock/440/SINE_POINTS
- #define **Ab4** SpkClock/415/SINE_POINTS
- #define **G4** SpkClock/392/SINE_POINTS
- #define **Gb4** SpkClock/370/SINE_POINTS
- #define **F4** SpkClock/349/SINE_POINTS
- #define **E4** SpkClock/330/SINE_POINTS
- #define **Eb4** SpkClock/311/SINE_POINTS
- #define **D4** SpkClock/294/SINE_POINTS
- #define **Db4** SpkClock/277/SINE_POINTS
- #define **C4** SpkClock/262/SINE_POINTS
- #define **B3** SpkClock/247/SINE_POINTS
- #define **Bb3** SpkClock/233/SINE_POINTS
- #define **A3** SpkClock/220/SINE_POINTS
- #define **Ab3** SpkClock/208/SINE_POINTS
- #define **G3** SpkClock/196/SINE_POINTS
- #define **Gb3** SpkClock/185/SINE_POINTS
- #define **F3** SpkClock/175/SINE_POINTS
- #define **E3** SpkClock/165/SINE_POINTS
- #define **Eb3** SpkClock/155/SINE_POINTS

- #define **D3** SpkClock/147/SINE_POINTS
- #define **Db3** SpkClock/139/SINE_POINTS
- #define **C3** SpkClock/131/SINE_POINTS
- #define **B2** SpkClock/123/SINE_POINTS
- #define **Bb2** SpkClock/117/SINE_POINTS
- #define **A2** SpkClock/110/SINE_POINTS
- #define **Ab2** SpkClock/104/SINE_POINTS
- #define **G2** SpkClock/98/SINE_POINTS
- #define **Gb2** SpkClock/93/SINE_POINTS
- #define **F2** SpkClock/87/SINE_POINTS
- #define **E2** SpkClock/82/SINE_POINTS
- #define **Eb2** SpkClock/78/SINE_POINTS
- #define **D2** SpkClock/73/SINE_POINTS
- #define **Db2** SpkClock/69/SINE_POINTS
- #define **C2** SpkClock/65/SINE_POINTS
- #define **BEAT** (SpkClock/16)-1
- typedef struct **Note NoteType**
- typedef **NoteType** * **NotePtr**
- void **Blox_Speaker_Init** (void)

  Initializes the speaker.

- void **PlayMusic** (void)

  Plays the music included in blox_music.h.

- void **StopMusic** (void)

  Stops the music.

6.19.1 Detailed Description

Basic device driver header for Blox speaker.

Author

Ankita kaul

Version

V0.1

Date

10/25/2010

Definition in file **blox_speaker.h**.

6.20 drivers/inc/blox_speaker.h

```
00001
00009 #ifndef __BLOX_SPEAKER_H
00010 #define __BLOX_SPEAKER_H
00011
00016 //#define SpkClock 104800 //(SpkClock = 6550*Beats/Sec, at 16bps
00017 #define SpkClock 52400
00018 //#define SpkClock 26200 //(SpkClock = 6550*Beats/Sec, at 4bps
00019 #define SINE_POINTS 32
00020
00021 //Definitions of notes: = clock frequency/pitch frequency/# of table elements
00022 #define C7              SpkClock/2093/SINE_POINTS
00023 #define B6              SpkClock/1975/SINE_POINTS
00024 #define Bb6             SpkClock/1865/SINE_POINTS
00025 #define A6              SpkClock/1760/SINE_POINTS
00026 #define Ab6             SpkClock/1661/SINE_POINTS
00027 #define G6              SpkClock/1567/SINE_POINTS
00028 #define Gb6             SpkClock/1480/SINE_POINTS
00029 #define F6              SpkClock/1396/SINE_POINTS
00030 #define E6              SpkClock/1319/SINE_POINTS
00031 #define Eb6             SpkClock/1245/SINE_POINTS
00032 #define D6              SpkClock/1175/SINE_POINTS
00033 #define Db6             SpkClock/1108/SINE_POINTS
00034 #define C6              SpkClock/1046/SINE_POINTS
00035 #define B5              SpkClock/988/SINE_POINTS
00036 #define Bb5             SpkClock/932/SINE_POINTS
00037 #define A5              SpkClock/880/SINE_POINTS
00038 #define Ab5             SpkClock/830/SINE_POINTS
00039 #define G5              SpkClock/784/SINE_POINTS
00040 #define Gb5             SpkClock/740/SINE_POINTS
00041 #define F5              SpkClock/698/SINE_POINTS
00042 #define E5              SpkClock/659/SINE_POINTS
00043 #define Eb5             SpkClock/622/SINE_POINTS
00044 #define D5              SpkClock/587/SINE_POINTS
00045 #define Db5             SpkClock/554/SINE_POINTS
00046 #define C5              SpkClock/523/SINE_POINTS
00047 #define B4              SpkClock/494/SINE_POINTS
00048 #define Bb4             SpkClock/466/SINE_POINTS
00049 #define A4              SpkClock/440/SINE_POINTS     //middle A = 440Hz
00050 #define Ab4             SpkClock/415/SINE_POINTS
00051 #define G4              SpkClock/392/SINE_POINTS
00052 #define Gb4             SpkClock/370/SINE_POINTS
00053 #define F4              SpkClock/349/SINE_POINTS
00054 #define E4              SpkClock/330/SINE_POINTS
00055 #define Eb4             SpkClock/311/SINE_POINTS
00056 #define D4              SpkClock/294/SINE_POINTS
00057 #define Db4             SpkClock/277/SINE_POINTS
00058 #define C4              SpkClock/262/SINE_POINTS
00059 #define B3              SpkClock/247/SINE_POINTS
00060 #define Bb3             SpkClock/233/SINE_POINTS
00061 #define A3              SpkClock/220/SINE_POINTS
00062 #define Ab3             SpkClock/208/SINE_POINTS
00063 #define G3              SpkClock/196/SINE_POINTS
00064 #define Gb3     SpkClock/185/SINE_POINTS
00065 #define F3      SpkClock/175/SINE_POINTS
00066 #define E3      SpkClock/165/SINE_POINTS
```

```
00067 #define Eb3     SpkClock/155/SINE_POINTS
00068 #define D3      SpkClock/147/SINE_POINTS
00069 #define Db3     SpkClock/139/SINE_POINTS
00070 #define C3              SpkClock/131/SINE_POINTS
00071 #define B2              SpkClock/123/SINE_POINTS
00072 #define Bb2             SpkClock/117/SINE_POINTS
00073 #define A2              SpkClock/110/SINE_POINTS
00074 #define Ab2             SpkClock/104/SINE_POINTS
00075 #define G2              SpkClock/98/SINE_POINTS
00076 #define Gb2     SpkClock/93/SINE_POINTS
00077 #define F2      SpkClock/87/SINE_POINTS
00078 #define E2      SpkClock/82/SINE_POINTS
00079 #define Eb2     SpkClock/78/SINE_POINTS
00080 #define D2      SpkClock/73/SINE_POINTS
00081 #define Db2     SpkClock/69/SINE_POINTS
00082 #define C2              SpkClock/65/SINE_POINTS
00083
00084 #define BEAT    (SpkClock/16)-1 //beat duration (16 beats per sec)
00085 //#define BEAT  (SpkClock/8)-1  //beat duration (8 beats per sec)
00086 //#define BEAT  (SpkClock/4)-1  //beat duration (4 beats per sec)
00087
00091 struct Note
00092 {
00093   unsigned short noteName;
00094   unsigned short duration;
00095 };
00096
00097 typedef const struct Note NoteType;
00098 typedef NoteType * NotePtr;
00099
00100 void Blox_Speaker_Init(void);
00101 void PlayMusic(void);
00102 void StopMusic(void);
00104 #endif
```

6.21 drivers/inc/blox_system.h File Reference

Prototypes for system-wide functions and definitions.

```
#include "stm32f10x.h"
#include "stdlib.h"
#include "blox_debug.h"
```

Data Structures

- struct **SysVar**

  Defines a system variable.

- #define **TRUE** 1
- #define **FALSE** 0
- #define **NULL** 0
- #define **PAGE_SIZE** 0x800
- #define **WORD_SIZE** 0x4
- #define **MEM_MAP_START** 0x08000000
- #define **MEM_MAP_SIZE** 0x08080000
- #define **MEM_BASE_PROG_START** 0x08000000
- #define **MEM_BASE_PROG_SIZE** PAGE_SIZE∗32
- #define **MEM_SYS_VAR_START** 0x08010000
- #define **MEM_SYS_VAR_SIZE** PAGE_SIZE
- #define **MEM_STAGE_START** 0x08010800
- #define **MEM_START_SIZE** PAGE_SIZE∗32
- #define **MEM_FAT_START** 0x08020800
- #define **MEM_FAT_SIZE** PAGE_SIZE
- #define **MEM_STORE_START** 0x08021000
- #define **MEM_STORE_SIZE** PAGE_SIZE∗190
- #define **MAX_DEINIT_FN** 128
- #define **SYS_MAGIC** 0xCAFEBABE
- #define **SYS_INV_ID** 0xFFFFFFFF
- typedef void(∗ **ptrVoidFn**)(void)
- void **Blox_System_Init** (void)

  Initializes the pointer to the system variables array.

- void **Blox_System_Create** (void)

  *Creates an initial **SysVar** (*p. *110) struct.*

- void **Blox_System_DeInit** (void)

  De-initializes all the peripherals in the system.

- void **Blox_System_Register_DeInit** (ptrVoidFn fn)

  Registers a new function to be called when the system deInits. Only adds if it isn't already there.

- uint32_t **Blox_System_GetId** (void)

  *Returns the current **SysVar** (*p. *110) struct.*

- void **Blox_System_GetVars** (**SysVar** ∗retSys)

  *Returns the current **SysVar** (*p. *110) struct.*

- void **Blox_System_WriteVars** (**SysVar** ∗newVars)

  *Writes over the **SysVar** (*p. *110) in flash.*

6.21.1 Detailed Description

Prototypes for system-wide functions and definitions. Defines system-wide concepts including the memory map, and deinitialization.

Author

Jesse Tannahill

Version

V0.1

Date

10/31/2010

Author

Jesse Tannahill

Version

V0.1

Date

11/03/2010

Definition in file **blox_system.h**.

6.22 drivers/inc/blox_system.h

```
00001
00008 #ifndef __BLOX_SYSTEM_H
00009 #define __BLOX_SYSTEM_H
00010
00011 #include "stm32f10x.h"
00012 #include "stdlib.h"
00013
00014 #include "blox_debug.h"
00015
00020 #define TRUE 1
00021 #define FALSE 0
00022
00023 #define NULL 0
00024
00025 #define PAGE_SIZE 0x800
00026 #define WORD_SIZE 0x4
00027
00029 /* On STM32F103VE Flash : 0x08000000-0x08080000, 2K page size
```

```
00030  * 0x08000000
00031  *      .            Base Program
00032  * 0x08008000
00033  *      .       System Variables
00034  * 0x08008800
00035  *      .            FS FAT
00036  * 0x08009000
00037  *      .            FS File Store
00038  * 0x08080000
00039  */
00040 #define MEM_MAP_START 0x08000000
00041 #define MEM_MAP_SIZE 0x08080000
00042 #define MEM_BASE_PROG_START     0x08000000
00043 #define MEM_BASE_PROG_SIZE      PAGE_SIZE*32
00044 #define MEM_SYS_VAR_START       0x08010000
00045 #define MEM_SYS_VAR_SIZE        PAGE_SIZE
00046 #define MEM_STAGE_START         0x08010800
00047 #define MEM_START_SIZE          PAGE_SIZE*32
00048 #define MEM_FAT_START           0x08020800
00049 #define MEM_FAT_SIZE            PAGE_SIZE
00050 #define MEM_STORE_START         0x08021000
00051 #define MEM_STORE_SIZE          PAGE_SIZE*190
00052
00053 #define MAX_DEINIT_FN 128
00054 typedef void (*ptrVoidFn)(void);
00055
00056 #define SYS_MAGIC 0xCAFEBABE
00057 #define SYS_INV_ID 0xFFFFFFFF
00058
00061 typedef struct {
00062   uint32_t magic;
00063         uint32_t id;
00064         int32_t ACCEL_X;
00065   int32_t ACCEL_Y;
00066   int32_t ACCEL_Z;
00067         int32_t TOUCH_1_X;
00068   int32_t TOUCH_1_Y;
00069   int32_t TOUCH_2_X;
00070   int32_t TOUCH_2_Y;
00071   int32_t TOUCH_3_X;
00072   int32_t TOUCH_3_Y;
00073   int32_t TOUCH_4_X;
00074   int32_t TOUCH_4_Y;
00075 } SysVar;
00076
00077 void Blox_System_Init(void);
00078 void Blox_System_Create(void);
00079 void Blox_System_DeInit(void);
00080 void Blox_System_Register_DeInit(ptrVoidFn fn);
00081 uint32_t Blox_System_GetId(void);
00082 void Blox_System_GetVars(SysVar *retSys);
00083 void Blox_System_WriteVars(SysVar *newVars);
00085 #endif
```

6.23 drivers/inc/blox_tim.h File Reference

Contains function prototypes for the timers.

```
#include "blox_system.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_tim.h"
#include "misc.h"
```

- #define **TIM1_CLK** RCC_APB2Periph_TIM1
- #define **TIM2_CLK** RCC_APB1Periph_TIM2
- #define **TIM3_CLK** RCC_APB1Periph_TIM3
- #define **TIM4_CLK** RCC_APB1Periph_TIM4
- #define **TIM5_CLK** RCC_APB1Periph_TIM5
- #define **TIM6_CLK** RCC_APB1Periph_TIM6
- #define **TIM7_CLK** RCC_APB1Periph_TIM7
- #define **TIM8_CLK** RCC_APB2Periph_TIM8
- enum **TIMER_ID** {

  INVALID_TIMER = -2, **IRQ_UNAVAILABLE**, **TIM1UP**, **TIM1CH1**,

  TIM1CH2, **TIM1CH3**, **TIM1CH4**, **TIM2CH1**,

  TIM2CH2, **TIM2CH3**, **TIM2CH4**, **TIM3CH1**,

  TIM3CH2, **TIM3CH3**, **TIM3CH4**, **TIM4CH1**,

  TIM4CH2, **TIM4CH3**, **TIM4CH4**, **TIM5CH1**,

  TIM5CH2, **TIM5CH3**, **TIM5CH4**, **TIM6UP**,

  TIM7UP, **TIM8UP**, **TIM8CH1**, **TIM8CH2**,

  TIM8CH3, **TIM8CH4** }

  Enum for possible Timer ids.

- void **Blox_Timer_Init** (uint8_t TIMx, uint32_t TIM_CLK)

  Initializes Timer.

- **TIMER_ID Blox_Timer_Register_IRQ** (uint8_t TIMx, uint16_t period, void(∗Timer_-Handler)(void), FunctionalState NewState)

  Registers a timer interrupt for a given timer.

- void **Blox_Timer_Release_IRQ** (**TIMER_ID** id)

  Releases a given timer interrupt.

- void **Blox_Timer_Modify_IRQ** (**TIMER_ID** id, uint16_t period)

  Modifies the period for a given output compare interrupt.

- void **Blox_Timer_Enable_IRQ** (**TIMER_ID** id)

  Enables interrupts for a given interrupt.

- void **Blox_Timer_Disable_IRQ** (**TIMER_ID** id)

  Disables interrupts for a given interrupt.

6.23.1 Detailed Description

Contains function prototypes for the timers.

Author

Zach Wasson

Version

V0.1

Date

10/20/2010

Definition in file **blox_tim.h**.

6.24 drivers/inc/blox_tim.h

```
00001
00008 #ifndef __BLOX_TIM_H
00009 #define __BLOX_TIM_H
00010
00011 #include "blox_system.h"
00012 #include "stm32f10x_rcc.h"
00013 #include "stm32f10x_tim.h"
00014 #include "misc.h"
00015
00020 #define TIM1_CLK    RCC_APB2Periph_TIM1
00021 #define TIM2_CLK    RCC_APB1Periph_TIM2
00022 #define TIM3_CLK    RCC_APB1Periph_TIM3
00023 #define TIM4_CLK    RCC_APB1Periph_TIM4
00024 #define TIM5_CLK    RCC_APB1Periph_TIM5
00025 #define TIM6_CLK    RCC_APB1Periph_TIM6
00026 #define TIM7_CLK    RCC_APB1Periph_TIM7
```

```
00027 #define TIM8_CLK    RCC_APB2Periph_TIM8
00028
00032 typedef enum {
00033   INVALID_TIMER = -2,
00034   IRQ_UNAVAILABLE,
00035   TIM1UP,
00036   TIM1CH1,
00037   TIM1CH2,
00038   TIM1CH3,
00039   TIM1CH4,
00040   TIM2CH1,
00041   TIM2CH2,
00042   TIM2CH3,
00043   TIM2CH4,
00044   TIM3CH1,
00045   TIM3CH2,
00046   TIM3CH3,
00047   TIM3CH4,
00048   TIM4CH1,
00049   TIM4CH2,
00050   TIM4CH3,
00051   TIM4CH4,
00052   TIM5CH1,
00053   TIM5CH2,
00054   TIM5CH3,
00055   TIM5CH4,
00056   TIM6UP,
00057   TIM7UP,
00058   TIM8UP,
00059   TIM8CH1,
00060   TIM8CH2,
00061   TIM8CH3,
00062   TIM8CH4
00063 } TIMER_ID;
00064
00065 void Blox_Timer_Init(uint8_t TIMx, uint32_t TIM_CLK);
00066 TIMER_ID Blox_Timer_Register_IRQ(uint8_t TIMx, uint16_t period, void (*Timer_Hand
      ler)(void), FunctionalState NewState);
00067 void Blox_Timer_Release_IRQ(TIMER_ID id);
00068 void Blox_Timer_Modify_IRQ(TIMER_ID id, uint16_t period);
00069 void Blox_Timer_Enable_IRQ(TIMER_ID id);
00070 void Blox_Timer_Disable_IRQ(TIMER_ID id);
00072 #endif
```

6.25 drivers/inc/blox_touch.h File Reference

Contains function prototypes for the Touchpanel interface.

```
#include "stm32f10x_rcc.h"

#include "stm32f10x_gpio.h"

#include "stm32f10x_spi.h"

#include "blox_filesystem.h"
```

```
#include "blox_system.h"
#include "blox_counter.h"
#include "blox_exti.h"
```

- #define **TOUCH_SPI** SPI1
- #define **TOUCH_SPI_CLK** RCC_APB2Periph_SPI1
- #define **TOUCH_SPI_GPIO** GPIOA
- #define **TOUCH_SPI_GPIO_CLK** RCC_APB2Periph_GPIOA
- #define **TOUCH_SPI_SCK_PIN** GPIO_Pin_5
- #define **TOUCH_SPI_MISO_PIN** GPIO_Pin_6
- #define **TOUCH_SPI_MOSI_PIN** GPIO_Pin_7
- #define **TOUCH_CS_GPIO** GPIOE
- #define **TOUCH_CS_GPIO_CLK** RCC_APB2Periph_GPIOE
- #define **TOUCH3_CS_PIN** GPIO_Pin_12
- #define **TOUCH1_CS_PIN** GPIO_Pin_8
- #define **TOUCH2_CS_PIN** GPIO_Pin_13
- #define **TOUCH4_CS_PIN** GPIO_Pin_9
- #define **TOUCH_PENIRQ_GPIO** GPIOE
- #define **TOUCH1_PENIRQ_PIN** GPIO_Pin_10
- #define **TOUCH2_PENIRQ_PIN** GPIO_Pin_11
- #define **TOUCH3_PENIRQ_PIN** GPIO_Pin_14
- #define **TOUCH4_PENIRQ_PIN** GPIO_Pin_7
- #define **TOUCH_CTL_X** 0xDA

  1101 1011

- #define **TOUCH_CTL_Y** 0x9A
- #define **TOUCH_CTL_Z1** 0xBA
- #define **TOUCH_CTL_Z2** 0xCA
- enum **TOUCH_DIR** { **TOUCH_NORTH_ID** = 1, **TOUCH_SOUTH_ID**, **TOUCH_EAST_ID**, **TOUCH_-WEST_ID** }

  A mapping from touch ids to cardinal directions.

- void **Blox_Touch_Init** (void)

  Initializes the IR module. Basically a wrapper on USART.

- void **Blox_Touch_SPI_Send** (uint16_t data)
- uint16_t **Blox_Touch_GetX** (int numTouch)

  Get the X-value of a press on the touchpanel.

- uint16_t **Blox_Touch_GetY** (int numTouch)

  Get the Y-value of a press on the touchpanel./.

- uint16_t **Blox_Touch_GetZ1** (int numTouch)

  Get the Z1-value of a press on the touchpanel.

- uint16_t **Blox_Touch_GetZ2** (int numTouch)

  Get the Z2-value of a press on the touchpanel.

- void **Touch_SPI_Send** (uint16_t data)

  Sends a byte out on SPI to the touchpanel.

- uint16_t **Touch_SPI_Receive** (void)

  Receive a byte from the touchpanel.

6.25.1 Detailed Description

Contains function prototypes for the Touchpanel interface.

Author

Ankita Kaul & Jesse Tannahill

Version

V0.1

Date

11/01/10

Definition in file **blox_touch.h**.

6.25.2 Define Documentation

6.25.2.1 #define TOUCH_CTL_X 0xDA

1101 1011

Touchpanel Controller Control Byte 7 - Start Bit [6:4] - Channel Select Bits 3 - 12-Bit/8-Bit Conversion Mode 2 - Single-Ended/Differential [1:0] - Power-Down Select Bits

Definition at line **71** of file **blox_touch.h**.

6.25.3 Function Documentation

6.25.3.1 uint16_t Blox_Touch_GetX (int *numTouch*)

Get the X-value of a press on the touchpanel.

Parameters

numTouch	ID of the touchpanel to retrieve from

Return values

The	12-bit return value

Definition at line **124** of file **blox_touch.c**.

6.25.3.2 uint16_t Blox_Touch_GetY (int *numTouch*)

Get the Y-value of a press on the touchpanel./.

Parameters

numTouch	touchpanel id to retrieve from

Return values

The	12-bit return value

Definition at line **167** of file **blox_touch.c**.

6.25.3.3 uint16_t Blox_Touch_GetZ1 (int *numTouch*)

Get the Z1-value of a press on the touchpanel.

Parameters

numTouch	touchpanel id to retrieve from

Return values

The	12-bit return value

Definition at line **209** of file **blox_touch.c**.

6.25.3.4 uint16_t Blox_Touch_GetZ2 (int *numTouch*)

Get the Z2-value of a press on the touchpanel.

Parameters

numTouch	touchpanel id to retrieve from

Return values

The	12-bit return value

Definition at line **253** of file **blox_touch.c**.

6.25.3.5 void Blox_Touch_Init (void)

Initializes the IR module. Basically a wrapper on USART.

Return values

None	

Definition at line **27** of file **blox_touch.c**.

6.25.3.6 uint16_t Touch_SPI_Receive (void)

Receive a byte from the touchpanel.

Return values

the	received byte.

Definition at line **308** of file **blox_touch.c**.

6.25.3.7 void Touch_SPI_Send (uint16_t *data*)

Sends a byte out on SPI to the touchpanel.

Parameters

data,:	the byte to send

Return values

None.	

Definition at line **297** of file **blox_touch.c**.

6.26 drivers/inc/blox_touch.h

```
00001
00008 #ifndef __BLOX_TOUCH_H
00009 #define __BLOX_TOUCH_H
00010
00011 #include "stm32f10x_rcc.h"
00012 #include "stm32f10x_gpio.h"
00013 #include "stm32f10x_spi.h"
00014 #include "blox_filesystem.h"
00015 #include "blox_system.h"
00016 #include "blox_counter.h"
00017 #include "blox_exti.h"
00018
00025 typedef enum {
00026   TOUCH_NORTH_ID = 1,
00027   TOUCH_SOUTH_ID,
00028   TOUCH_EAST_ID,
00029   TOUCH_WEST_ID
00030 } TOUCH_DIR;
00031
00032 #define TOUCH_SPI                    SPI1
00033 #define TOUCH_SPI_CLK              RCC_APB2Periph_SPI1
00034 #define TOUCH_SPI_GPIO             GPIOA
00035 #define TOUCH_SPI_GPIO_CLK       RCC_APB2Periph_GPIOA
00036 //#define TOUCH_SPI_NSS_PIN        GPIO_Pin_12
00037 #define TOUCH_SPI_SCK_PIN          GPIO_Pin_5
00038 #define TOUCH_SPI_MISO_PIN       GPIO_Pin_6
00039 #define TOUCH_SPI_MOSI_PIN       GPIO_Pin_7
00040 //#define TOUCH_BUSY_GPIO      GPIOB
00041 //#define TOUCH_BUSY_PIN                 GPIO_Pin_10
00042
00043 #define TOUCH_CS_GPIO         GPIOE
00044 #define TOUCH_CS_GPIO_CLK     RCC_APB2Periph_GPIOE
00045 #define TOUCH3_CS_PIN         GPIO_Pin_12             //TOUCH3_CS on schematic
00046 #define TOUCH1_CS_PIN         GPIO_Pin_8
00047 #define TOUCH2_CS_PIN         GPIO_Pin_13
00048 #define TOUCH4_CS_PIN         GPIO_Pin_9
00049
00050 #define TOUCH_PENIRQ_GPIO     GPIOE
00051 #define TOUCH1_PENIRQ_PIN     GPIO_Pin_10
00052 #define TOUCH2_PENIRQ_PIN     GPIO_Pin_11
00053 #define TOUCH3_PENIRQ_PIN     GPIO_Pin_14
00054 #define TOUCH4_PENIRQ_PIN     GPIO_Pin_7
00055
00064
00065 //#define TOUCH_CTL_X   0xDB
00066 //#define TOUCH_CTL_Y   0x9B
00067 //#define TOUCH_CTL_Z1  0xBB
00068 //#define TOUCH_CTL_Z2  0xCB
00069
00070 //1101 1010
```

```
00071 #define TOUCH_CTL_X   0xDA
00072 #define TOUCH_CTL_Y   0x9A
00073 #define TOUCH_CTL_Z1  0xBA
00074 #define TOUCH_CTL_Z2  0xCA
00075
00076 void Blox_Touch_Init(void);
00077 void Blox_Touch_SPI_Send(uint16_t data);
00078 uint16_t Blox_Touch_GetX(int numTouch);
00079 uint16_t Blox_Touch_GetY(int numTouch);
00080 uint16_t Blox_Touch_GetZ1(int numTouch);
00081 uint16_t Blox_Touch_GetZ2(int numTouch);
00082 void Touch_SPI_Send(uint16_t data);
00083 uint16_t Touch_SPI_Receive(void);
00085 #endif
```

6.27 drivers/inc/blox_usart.h File Reference

Contains function prototypes for the usart interface.

```
#include "blox_system.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_usart.h"
#include "misc.h"
```

- #define **USART1_GPIO** GPIOA
- #define **USART1_CLK** RCC_APB2Periph_USART1
- #define **USART1_GPIO_CLK** RCC_APB2Periph_GPIOA
- #define **USART1_RxPin** GPIO_Pin_10
- #define **USART1_TxPin** GPIO_Pin_9
- #define **USART2_GPIO** GPIOA
- #define **USART2_CLK** RCC_APB1Periph_USART2
- #define **USART2_GPIO_CLK** RCC_APB2Periph_GPIOA
- #define **USART2_RxPin** GPIO_Pin_3
- #define **USART2_TxPin** GPIO_Pin_2
- #define **USART3_GPIO** GPIOB
- #define **USART3_CLK** RCC_APB1Periph_USART3
- #define **USART3_GPIO_CLK** RCC_APB2Periph_GPIOB
- #define **USART3_RxPin** GPIO_Pin_11
- #define **USART3_TxPin** GPIO_Pin_10
- #define **UART4_GPIO** GPIOC

- #define **UART4_CLK** RCC_APB1Periph_UART4
- #define **UART4_GPIO_CLK** RCC_APB2Periph_GPIOC
- #define **UART4_RxPin** GPIO_Pin_11
- #define **UART4_TxPin** GPIO_Pin_10
- #define **UART5_GPIO_TX** GPIOC
- #define **UART5_GPIO_RX** GPIOD
- #define **UART5_CLK** RCC_APB1Periph_UART5
- #define **UART5_GPIO_TX_CLK** RCC_APB2Periph_GPIOC
- #define **UART5_GPIO_RX_CLK** RCC_APB2Periph_GPIOD
- #define **UART5_RxPin** GPIO_Pin_2
- #define **UART5_TxPin** GPIO_Pin_12
- void **Blox_USART_Init** (uint8_t)

  Initializes the USART module.

- uint8_t **Blox_USART_Receive** (uint8_t id)

  Receive a byte on the given USART.

- int16_t **Blox_USART_TryReceive** (uint8_t id)

  Receive a byte on the given USART.

- void **Blox_USART_Send** (uint8_t id, uint8_t data)

  Sends a byte out on the given USART.

- void **Blox_USART_Register_RXNE_IRQ** (uint8_t id, void(∗RXNE_Handler)(void))

  Registers a USART Interrupt on RXNE.

- void **Blox_USART_Enable_RXNE_IRQ** (uint8_t id)

  Disables the USART Interrupt on RXNE.

- void **Blox_USART_Disable_RXNE_IRQ** (uint8_t id)

  Disables the USART Interrupt on RXNE.

6.27.1 Detailed Description

Contains function prototypes for the usart interface.

Author

Jesse Tannahill

Version

V0.1

Date

10/18/2010

Definition in file **blox_usart.h**.

6.28 drivers/inc/blox_usart.h

```
00001
00008 #ifndef __BLOX_USART_H
00009 #define __BLOX_USART_H
00010
00011 #include "blox_system.h"
00012 #include "stm32f10x_gpio.h"
00013 #include "stm32f10x_rcc.h"
00014 #include "stm32f10x_usart.h"
00015 #include "misc.h"
00016
00022 #define USART1_GPIO             GPIOA
00023 #define USART1_CLK              RCC_APB2Periph_USART1
00024 #define USART1_GPIO_CLK         RCC_APB2Periph_GPIOA
00025 #define USART1_RxPin            GPIO_Pin_10
00026 #define USART1_TxPin            GPIO_Pin_9
00027
00028 #define USART2_GPIO             GPIOA
00029 #define USART2_CLK              RCC_APB1Periph_USART2
00030 #define USART2_GPIO_CLK         RCC_APB2Periph_GPIOA
00031 #define USART2_RxPin            GPIO_Pin_3
00032 #define USART2_TxPin            GPIO_Pin_2
00033
00034 #define USART3_GPIO             GPIOB
00035 #define USART3_CLK              RCC_APB1Periph_USART3
00036 #define USART3_GPIO_CLK         RCC_APB2Periph_GPIOB
00037 #define USART3_RxPin            GPIO_Pin_11
00038 #define USART3_TxPin            GPIO_Pin_10
00039
00040 #define UART4_GPIO              GPIOC
00041 #define UART4_CLK               RCC_APB1Periph_UART4
00042 #define UART4_GPIO_CLK          RCC_APB2Periph_GPIOC
00043 #define UART4_RxPin             GPIO_Pin_11
00044 #define UART4_TxPin             GPIO_Pin_10
00045
00046 #define UART5_GPIO_TX           GPIOC
00047 #define UART5_GPIO_RX             GPIOD
00048 #define UART5_CLK               RCC_APB1Periph_UART5
00049 #define UART5_GPIO_TX_CLK RCC_APB2Periph_GPIOC
00050 #define UART5_GPIO_RX_CLK RCC_APB2Periph_GPIOD
00051 #define UART5_RxPin             GPIO_Pin_2
00052 #define UART5_TxPin             GPIO_Pin_12
00053
00054 void Blox_USART_Init(uint8_t);
00055 uint8_t Blox_USART_Receive(uint8_t id);
00056 int16_t Blox_USART_TryReceive(uint8_t id);
00057 void Blox_USART_Send(uint8_t id, uint8_t data);
```

```
00058 void Blox_USART_Register_RXNE_IRQ(uint8_t id, void (*RXNE_Handler)(void));
00059 void Blox_USART_Enable_RXNE_IRQ(uint8_t id);
00060 void Blox_USART_Disable_RXNE_IRQ(uint8_t id);
00062 #endif
```

6.29 drivers/inc/blox_usb.h File Reference

Contains function prototypes for the USB interface.

```
#include "stm32f10x.h"
#include "blox_usart.h"
#include "stdio.h"
#include "stdarg.h"
#include "string.h"
```

- #define **USB_USART_ID** 1
- void **USB_Init** (void)

  Initializes the USB module. Basically a wrapper on USART.

- uint8_t **USB_Receive** (void)

  Blocking receive of a byte over USB. A wrapper around USART.

- int16_t **USB_TryReceive** (void)

  Non-blocking receive of a byte over USB. A wrapper around USART.

- void **USB_Send** (uint8_t data)

  Sends a byte over USB. A wrapper around USART.

- void **USB_SendData** (uint8_t *data, uint32_t len)

  Sends len bytes over USB. A wrapper around USART.

- void **USB_SendPat** (char *format,...)

  Sends a string based on pattern passed over USB. A wrapper around USART.

6.29.1 Detailed Description

Contains function prototypes for the USB interface.

Author

Jesse Tannahill

Version

V0.1

Date

10/19/2010

Definition in file **blox_usb.h**.

6.30 drivers/inc/blox_usb.h

```
00001
00008 #ifndef __BLOX_USB_H
00009 #define __BLOX_USB_H
00010
00011 #include "stm32f10x.h"
00012 #include "blox_usart.h"
00013 #include "stdio.h"
00014 #include "stdarg.h"
00015 #include "string.h"
00016
00021 #define USB_USART_ID 1
00022
00023 void USB_Init(void);
00024 uint8_t USB_Receive(void);
00025 int16_t USB_TryReceive(void);
00026 void USB_Send(uint8_t data);
00027 void USB_SendData(uint8_t *data, uint32_t len);
00028 void USB_SendPat(char *format, ...);
00030 #endif
```

6.31 drivers/inc/blox_vusart.h File Reference

Contains function prototypes for the virtual USART interface.

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_exti.h"
#include "misc.h"
#include "blox_tim.h"
#include "blox_exti.h"
```

- #define **VUSART_TIMx** 2
- #define **VUSART_TIM_IRQn** TIM2_IRQn
- #define **VUSART_TIM_CLK** 72000000
- #define **_9600bps** (uint16_t)(VUSART_TIM_CLK / 9600)
- #define **_19200bps** (uint16_t)(VUSART_TIM_CLK / 19200)
- #define **_38400bps** (uint16_t)(VUSART_TIM_CLK / 38400)
- #define **_57600bps** (uint16_t)(VUSART_TIM_CLK / 57600)
- #define **_115200bps** (uint16_t)(VUSART_TIM_CLK / 115200)
- #define **VUSART1_GPIO** GPIOB
- #define **VUSART1_GPIO_CLK** RCC_APB2Periph_GPIOB
- #define **VUSART1_CLK** TIM_CLK
- #define **VUSART1_RxPin** GPIO_Pin_12
- #define **VUSART1_TxPin** GPIO_Pin_13
- #define **VUSART1_RxPinSource** 12
- #define **VUSART1_RxPortSource** GPIO_PortSourceGPIOB
- #define **VUSART2_GPIO** GPIOA
- #define **VUSART2_GPIO_CLK** RCC_APB2Periph_GPIOA
- #define **VUSART2_CLK** TIM_CLK
- #define **VUSART2_RxPin** GPIO_Pin_1
- #define **VUSART2_TxPin** GPIO_Pin_0
- #define **VUSART2_RxPinSource** 1
- #define **VUSART2_RxPortSource** GPIO_PortSourceGPIOA
- enum **VUSART_STATUS** {

  VUSART_SUCCESS = 0, **INVALID_ID**, **TX_BUSY**, **RX_EMPTY**,

  RXNE_IRQ_UNAVAILABLE }

  Status to return on VUSART commands.

- void **Blox_VUSART_Init** (uint8_t id)

  Initializes the virtual USART module.

- void **Blox_VUSART_SetBaudrate** (uint8_t id, uint16_t baudrate)

  Sets the baudrate of the given ID.

- **VUSART_STATUS Blox_VUSART_TryReceive** (uint8_t id, uint8_t *data)

  Tries to receive a byte on the given virtual USART.

- **VUSART_STATUS Blox_VUSART_TrySend** (uint8_t id, uint8_t data)

  Tries to send a byte out on the given virtual USART.

- **VUSART_STATUS Blox_VUSART_Receive** (uint8_t id, uint8_t *data)

Receives a blocking byte on the given virtual USART.

- **VUSART_STATUS Blox_VUSART_Send** (uint8_t id, uint8_t data)

  Sends a blocking byte out on the given virtual USART.

- **VUSART_STATUS Blox_VUSART_SendData** (uint8_t id, uint8_t *data, uint32_t len)

  Sends a blocking byte out on the given virtual USART.

- **VUSART_STATUS Blox_VUSART_Register_RXNE_IRQ** (uint8_t id, void(*RXNE_Handler)(void))

  Registers a function to be called in the SWInterrupt that occurs when a receive happens.

- **VUSART_STATUS Blox_VUSART_Enable_RXNE_IRQ** (uint8_t id)

  Enables the SW Interrupt on RXNE.

- **VUSART_STATUS Blox_VUSART_Disable_RXNE_IRQ** (uint8_t id)

  Disables the SW Interrupt on RXNE.

6.31.1 Detailed Description

Contains function prototypes for the virtual USART interface.

Author

Zach Wasson

Version

V0.1

Date

11/02/2010

Definition in file **blox_vusart.h**.

6.32 drivers/inc/blox_vusart.h

```
00001
00008 #ifndef __BLOX_VUSART_H
00009 #define __BLOX_VUSART_H
00010
00011 #include "stm32f10x.h"
```

```
00012 #include "stm32f10x_gpio.h"
00013 #include "stm32f10x_rcc.h"
00014 #include "stm32f10x_exti.h"
00015 #include "misc.h"
00016 #include "blox_tim.h"
00017 #include "blox_exti.h"
00018
00023 #define VUSART_TIMx              2
00024 #define VUSART_TIM_IRQn   TIM2_IRQn
00025 #define VUSART_TIM_CLK           72000000
00026 #define _9600bps          (uint16_t)(VUSART_TIM_CLK / 9600)
00027 #define _19200bps         (uint16_t)(VUSART_TIM_CLK / 19200)
00028 #define _38400bps         (uint16_t)(VUSART_TIM_CLK / 38400)
00029 #define _57600bps         (uint16_t)(VUSART_TIM_CLK / 57600)
00030 #define _115200bps        (uint16_t)(VUSART_TIM_CLK / 115200)
00031
00032 /* virtual USART for XBee */
00033 #define VUSART1_GPIO                 GPIOB
00034 #define VUSART1_GPIO_CLK             RCC_APB2Periph_GPIOB
00035 #define VUSART1_CLK             TIM_CLK
00036 #define VUSART1_RxPin                GPIO_Pin_12
00037 #define VUSART1_TxPin                GPIO_Pin_13
00038 #define VUSART1_RxPinSource     12
00039 #define VUSART1_RxPortSource    GPIO_PortSourceGPIOB
00040
00041 /* virtual USART for OLED Display */
00042 #define VUSART2_GPIO                 GPIOA
00043 #define VUSART2_GPIO_CLK             RCC_APB2Periph_GPIOA
00044 #define VUSART2_CLK             TIM_CLK
00045 #define VUSART2_RxPin                GPIO_Pin_1
00046 #define VUSART2_TxPin                GPIO_Pin_0
00047 #define VUSART2_RxPinSource     1
00048 #define VUSART2_RxPortSource    GPIO_PortSourceGPIOA
00049
00053 typedef enum {
00054   VUSART_SUCCESS = 0,
00055   INVALID_ID,
00056   TX_BUSY,
00057   RX_EMPTY,
00058   RXNE_IRQ_UNAVAILABLE
00059 } VUSART_STATUS;
00060
00061 void Blox_VUSART_Init(uint8_t id);
00062 void Blox_VUSART_SetBaudrate(uint8_t id, uint16_t baudrate);
00063 VUSART_STATUS Blox_VUSART_TryReceive(uint8_t id, uint8_t *data);
00064 VUSART_STATUS Blox_VUSART_TrySend(uint8_t id, uint8_t data);
00065 VUSART_STATUS Blox_VUSART_Receive(uint8_t id, uint8_t *data);
00066 VUSART_STATUS Blox_VUSART_Send(uint8_t id, uint8_t data);
00067 VUSART_STATUS Blox_VUSART_SendData(uint8_t id, uint8_t *data, uint32_t len);
00068 VUSART_STATUS Blox_VUSART_Register_RXNE_IRQ(uint8_t id, void (*RXNE_Handler)(void
      ));
00069 VUSART_STATUS Blox_VUSART_Enable_RXNE_IRQ(uint8_t id);
00070 VUSART_STATUS Blox_VUSART_Disable_RXNE_IRQ(uint8_t id);
00072 #endif
```

6.33 drivers/inc/blox_xbee.h File Reference

Contains function prototypes for the XBee interface.

```
#include "blox_system.h"
#include "blox_vusart.h"
#include "blox_counter.h"
#include "stdio.h"
#include "string.h"
```

Data Structures

- struct **BloxFrame**

  *App-level frame that is parsed from a **XBeeFrame*** (p. *113*).

- struct **XBeeTxFrame**

  the XBee Transmission Frame struct

- struct **XBeeFrame**

  A general xbee frame to parse in any command.

- struct **XBeeTxStatusFrame**

  Struct for reading in the TxStatus frame format.

- struct **XBeeRxFrame**

  Struct for reading in the Rx frame format.

- #define **XBEE_VUSART_ID** 1
- #define **XBEE_RESET_GPIO** GPIOB
- #define **XBEE_RESET_GPIO_CLK** RCC_APB2Periph_GPIOB
- #define **XBEE_RESET_PIN** GPIO_Pin_14
- #define **XBEE_RESET_PIN_NUM** 14
- #define **XBEE_SLEEP_GPIO** GPIOB
- #define **XBEE_SLEEP_GPIO_CLK** RCC_APB2Periph_GPIOB
- #define **XBEE_SLEEP_PIN** GPIO_Pin_15
- #define **XBEE_SLEEP_PIN_NUM** 15
- #define **MESSAGE_SIZE** 100

- #define **CR** 0x0D
- #define **LF** 0x0A
- #define **BS** 0x08
- #define **ESC** 0x1B
- #define **SP** 0x20
- #define **DEL** 0x7F
- #define **API_TX_STATUS** 0x89
- #define **API_RX_FRAME** 0x81
- #define **BLOX_FRAME_DATA_LEN** 75
- #define **XBEE_BLOX_BROADCAST_ID** 0xFFFFFFFF
- #define **XBEE_HOLD_PERIOD** 1000
- enum **XBEE_STATUS** { **XBEE_TX_STATUS_FAIL** = -3, **XBEE_INIT_FAIL**, **XBEE_TX_FAIL**, **XBEE_OK** }

  Enum for possible XBee statuses.

- enum **BloxFrameType** { **FRAME_TYPE_BASE**, **FRAME_TYPE_ROLE**, **FRAME_TYPE_USER** }

  Enum for possible XBee frame types.

- enum **XBEE_TXSTATUS** { **XBEE_TXSTATUS_ERROR** = 0, **XBEE_TXSTATUS_NORMAL**, **XBEE_TXSTATUS_SUCCESS** }

  Enum for possible XBee TX statuses.

- **XBEE_STATUS Blox_XBee_Config** (void)

  Configures the XBee and writes the configuration to non-volatile mem.

- **XBEE_STATUS Blox_XBee_Print** (void)

  Prints out the configuration options of the XBee.

- **XBEE_STATUS Blox_XBee_Init** (void)

  Initializes the XBees sleep and reset pins, and then resets the XBee.

- **XBEE_STATUS Blox_XBee_Send** (uint8_t *data, uint32_t len, **BloxFrameType** type, uint32_t dst_id)

  Sends data out of a specific type on the XBee.

- void **Blox_XBee_Send_Period** (uint8_t *data, uint32_t len, **BloxFrameType** type, uint32_t dst_id, uint32_t millis)

  Sends data out of a specific type on the XBee for a period of time.

- void **Blox_XBee_Register_Read** (void(*Read_Handler)(void))
- **BloxFrame** * **Blox_XBee_Receive** (void)

  *Receives a **BloxFrame** (p. 97) from the XBee.*

- void **Blox_XBee_Register_RX_IRQ** (void(∗RX_Handler)(**BloxFrame** ∗))
- void **Blox_XBee_Enable_RX_IRQ** (void)

  Enables the sw interrupt that occurs when a XBee reads a byte.

- void **Blox_XBee_Disable_RX_IRQ** (void)

  Disables the sw interrupt that occurs when a XBee reads a byte.

6.33.1 Detailed Description

Contains function prototypes for the XBee interface.

Author

Dan Cleary

Version

V0.1

Date

10/27/2010

Definition in file **blox_xbee.h**.

6.34 drivers/inc/blox_xbee.h

```
00001
00009 #ifndef __BLOX_XBEE_H
00010 #define __BLOX_XBEE_H
00011
00012 #include "blox_system.h"
00013 #include "blox_vusart.h"
00014 #include "blox_counter.h"
00015
00016 #include "stdio.h"
00017 #include "string.h"
00018
00024 #define XBEE_VUSART_ID 1
00025
00026 #define XBEE_RESET_GPIO                 GPIOB
00027 #define XBEE_RESET_GPIO_CLK     RCC_APB2Periph_GPIOB
00028 #define XBEE_RESET_PIN                  GPIO_Pin_14
00029 #define XBEE_RESET_PIN_NUM              14
00030
00031 #define XBEE_SLEEP_GPIO                 GPIOB
```

```
00032 #define XBEE_SLEEP_GPIO_CLK     RCC_APB2Periph_GPIOB
00033 #define XBEE_SLEEP_PIN                   GPIO_Pin_15
00034 #define XBEE_SLEEP_PIN_NUM              15
00035
00036 #define MESSAGE_SIZE 100
00037
00038 // standard ASCII symbols
00039 #define CR   0x0D
00040 #define LF   0x0A
00041 #define BS   0x08
00042 #define ESC  0x1B
00043 #define SP   0x20
00044 #define DEL  0x7F
00045
00046 //XBee API IDs
00047 #define API_TX_STATUS 0x89
00048 #define API_RX_FRAME  0x81
00049
00053 typedef enum {
00054   XBEE_TX_STATUS_FAIL = -3,
00055   XBEE_INIT_FAIL,
00056   XBEE_TX_FAIL,
00057   XBEE_OK
00058 } XBEE_STATUS;
00059
00063 typedef enum {
00064   FRAME_TYPE_BASE,
00065   FRAME_TYPE_ROLE,
00066   FRAME_TYPE_USER
00067 } BloxFrameType;
00068
00072 typedef enum {
00073   XBEE_TXSTATUS_ERROR = 0,
00074   XBEE_TXSTATUS_NORMAL,
00075   XBEE_TXSTATUS_SUCCESS
00076 } XBEE_TXSTATUS;
00077
00078 #define BLOX_FRAME_DATA_LEN 75
00079 #define XBEE_BLOX_BROADCAST_ID 0xFFFFFFFF
00080 #define XBEE_HOLD_PERIOD 1000
00081
00085 typedef struct {
00086   uint32_t src_id;
00087   uint32_t dst_id;
00088   uint8_t len;
00089   BloxFrameType type;
00090   uint8_t data[BLOX_FRAME_DATA_LEN];
00091 } BloxFrame;
00092
00096 typedef struct {
00097   uint8_t start;
00098   uint16_t length;
00099   uint8_t api;
00100   uint8_t id;
00101   uint16_t dest_addr;
00102   uint8_t options;
00103   BloxFrame blox_frame;
```

```
00104   uint8_t checksum;
00105 } XBeeTxFrame;
00106
00110 typedef struct {
00111   uint16_t length;
00112   uint8_t data[125];
00113 } XBeeFrame;
00114
00118 typedef struct {
00119   uint16_t length;
00120   uint8_t api;
00121   uint8_t frame_id;
00122   uint8_t status;
00123 } XBeeTxStatusFrame;
00128 typedef struct {
00129   uint16_t length;
00130   uint8_t api;
00131   uint16_t source;
00132   uint8_t rssi;
00133   uint8_t options;
00134   BloxFrame blox_frame;
00135   uint8_t checksum;
00136 } XBeeRxFrame;
00137
00138 XBEE_STATUS Blox_XBee_Config(void);
00139 XBEE_STATUS Blox_XBee_Print(void);
00140 XBEE_STATUS Blox_XBee_Init (void);
00141 XBEE_STATUS Blox_XBee_Send (uint8_t *data, uint32_t len, BloxFrameType type, uint
      32_t dst_id);
00142 void Blox_XBee_Send_Period(uint8_t *data, uint32_t len, BloxFrameType type, uint3
      2_t dst_id, uint32_t millis);
00143 void Blox_XBee_Register_Read(void (*Read_Handler)(void));
00144 BloxFrame *Blox_XBee_Receive(void);
00145 void Blox_XBee_Register_RX_IRQ(void (*RX_Handler)(BloxFrame *));
00146 void Blox_XBee_Enable_RX_IRQ(void);
00147 void Blox_XBee_Disable_RX_IRQ(void);
00149 #endif
```

6.35 drivers/inc/example.h File Reference

Contains function prototypes for the example interface.

```
#include "blox_system.h"
```

Functions

- void **Blox_MyModule_MyFunc** (type1 Arg1, type2 Arg2, type3 Arg3)

  A brief description of the function's purpose.

6.35.1 Detailed Description

Contains function prototypes for the example interface.

Author

Jesse Tannahill

Version

V0.1

Date

10/18/2010

Definition in file **example.h**.

6.35.2 Function Documentation

6.35.2.1 void Blox_MyModule_MyFunc (type1 *Arg1,* type2 *Arg2,* type3 *Arg3*)

A brief description of the function's purpose.

Parameters

Arg1,:	Arg1 is used for ... .
Arg2,:	Arg2 is used for ... . This parameter can be any combination of the following values: • val1: Indicates ... • val2: Indicates ... • val3: Indicates ...
Arg3,:	Arg3 is used for ... . This parameter can be: ENABLE or DISABLE.

Return values

None	

Definition at line **21** of file **example.c**.

6.36 drivers/inc/example.h

```
00001
00008 #ifndef __BLOX_EXAMPLE_H
```

```
00009 #define __BLOX_EXAMPLE_H
00010
00011 #include "blox_system.h"
00012
00013 void Blox_MyModule_MyFunc(type1 Arg1, type2 Arg2, type3 Arg3);
00014 #endif
```

6.37 drivers/src/blox_accel.c File Reference

Drivers for Blox accelerometer.

```
#include "blox_accel.h"
```

- #define **ADC1_DR_Address** ((u32)0x4001244C)
- uint16_t **Accel_Measurements** [3]
- void **Accel_RCC_Configuration** ()

  Initializes the Accelerometer's clocks.

- void **Accel_GPIO_Configuration** ()

  Initializes the Accelerometer's GPIO for the ADC and sleep pins.

- void **Accel_DMA_Configuration** ()

  Sets up the DMA for the Accelerometer. The DMA configuration reads to the memory at Accel_Measurements.

- void **Accel_ADC_Configuration** ()

  Initializes the ADC used by the Accelerometer.

- void **Blox_Accel_Init** (void)

  Initializes the Accelerometer. Initializes the clocks, GPIO pins, DMA location, and ADC configuration. Also sets the sleep pin to high (active low pulse).

- uint16_t **Blox_Accel_GetX** (void)

  Returns the X-value reading of the accelerometer.

- uint16_t **Blox_Accel_GetY** (void)

  Returns the Y-value reading of the accelerometer.

- uint16_t **Blox_Accel_GetZ** (void)

  Returns the Z-value reading of the accelerometer.

- uint8_t **Blox_Accel_GetXTilt** (void)

  Returns the X tilt of the accelerometer.

- uint8_t **Blox_Accel_GetYTilt** (void)

  Returns the Y tilt of the accelerometer.

- uint8_t **Blox_Accel_GetZTilt** (void)

  Returns the Z tilt of the accelerometer.

6.37.1 Detailed Description

Drivers for Blox accelerometer.

Author

Dan Cleary

Version

V0.1

Date

10/20/2010

Definition in file **blox_accel.c**.

6.38 drivers/src/blox_accel.c

```
00001
00009 #include "blox_accel.h"
00010
00018 #define ADC1_DR_Address    ((u32)0x4001244C)
00019
00020 uint16_t Accel_Measurements[3];
00021
00022 /* Private function prototypes */
00023 void Accel_RCC_Configuration(void);
00024 void Accel_GPIO_Configuration(void);
00025 void Accel_DMA_Configuration(void);
00026 void Accel_ADC_Configuration(void);
00027
00034 void Blox_Accel_Init(void) {
00035   Accel_RCC_Configuration();
00036   Accel_GPIO_Configuration();
00037   Accel_DMA_Configuration();
00038   Accel_ADC_Configuration();
```

```
00039
00040   ACCEL_SLEEP_GPIO->ODR |= ACCEL_SLEEP_PIN;
00041 }
00042
00047 void Accel_RCC_Configuration() {
00048     RCC_APB2PeriphClockCmd(ACCEL_GPIO_CLK, ENABLE);
00049     RCC_APB2PeriphClockCmd(ACCEL_SLEEP_GPIO_CLK, ENABLE);
00050     RCC_AHBPeriphClockCmd(ACCEL_DMA_CLK, ENABLE);
00051     RCC_APB2PeriphClockCmd(ACCEL_ADC1_CLK, ENABLE);
00052 }
00053
00058 void Accel_GPIO_Configuration() {
00059   GPIO_InitTypeDef GPIO_InitStructure;
00060
00061   //Accelerometer X-out, Y-out, and Z-out pins
00062   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
00063   GPIO_InitStructure.GPIO_Pin = ACCEL_XOUT_PIN | ACCEL_YOUT_PIN | ACCEL_ZOUT_PIN;

00064   GPIO_Init(ACCEL_GPIO, &GPIO_InitStructure);
00065
00066   //Accelerometer Sleep Pin
00067   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
00068   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
00069   GPIO_InitStructure.GPIO_Pin = ACCEL_SLEEP_PIN;
00070   GPIO_Init(ACCEL_SLEEP_GPIO, &GPIO_InitStructure);
00071 }
00072
00078 void Accel_DMA_Configuration() {
00079   DMA_InitTypeDef DMA_InitStructure;
00080
00081   DMA_InitStructure.DMA_PeripheralBaseAddr = ADC1_DR_Address;
00082   DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&Accel_Measurements;
00083   DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
00084   DMA_InitStructure.DMA_BufferSize = 3;
00085   DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
00086   DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
00087   DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
00088   DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
00089   DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
00090   DMA_InitStructure.DMA_Priority = DMA_Priority_High;
00091   DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
00092   DMA_Init(DMA1_Channel1, &DMA_InitStructure);
00093
00094   /* Enable DMA1 channel1 */
00095   DMA_Cmd(DMA1_Channel1, ENABLE);
00096 }
00097
00102 void Accel_ADC_Configuration() {
00103   ADC_InitTypeDef ADC_InitStructure;
00104
00105   ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
00106   ADC_InitStructure.ADC_ScanConvMode = ENABLE;
00107   ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
00108   ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
00109   ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
00110   ADC_InitStructure.ADC_NbrOfChannel = 3;
00111   ADC_Init(ADC1, &ADC_InitStructure);
```

```
00112
00113   /* ADC1 regular channel 10, 11, 12 configuration */
00114   ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 1, ADC_SampleTime_55Cycles5);
00115   ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 2, ADC_SampleTime_55Cycles5);
00116   ADC_RegularChannelConfig(ADC1, ADC_Channel_12, 3, ADC_SampleTime_55Cycles5);
00117
00118   /* Enable ADC1 DMA */
00119   ADC_DMACmd(ADC1, ENABLE);
00120
00121   /* Enable ADC1 */
00122   ADC_Cmd(ADC1, ENABLE);
00123
00124   /* Enable ADC1 reset calibaration register */
00125   ADC_ResetCalibration(ADC1);
00126   /* Check the end of ADC1 reset calibration register */
00127   while(ADC_GetResetCalibrationStatus(ADC1));
00128
00129   /* Start ADC1 calibaration */
00130   ADC_StartCalibration(ADC1);
00131   /* Check the end of ADC1 calibration */
00132   while(ADC_GetCalibrationStatus(ADC1));
00133
00134   /* Start ADC1 Software Conversion */
00135   ADC_SoftwareStartConvCmd(ADC1, ENABLE);
00136 }
00137
00142 uint16_t Blox_Accel_GetX(void) {
00143   return Accel_Measurements[0];
00144 }
00145
00150 uint16_t Blox_Accel_GetY(void) {
00151   return Accel_Measurements[1];
00152 }
00153
00158 uint16_t Blox_Accel_GetZ(void) {
00159   return Accel_Measurements[2];
00160 }
00161
00168 uint8_t Blox_Accel_GetXTilt(void) {
00169   float meas = (float)(Accel_Measurements[0] * 3.3/0xFFF);
00170   if (meas < 1.4)
00171     return 2;
00172   else if (meas < 2)
00173     return 1;
00174   else
00175     return 0;
00176   return 1;
00177 }
00178
00185 uint8_t Blox_Accel_GetYTilt(void) {
00186   float meas = (float)(Accel_Measurements[1] * 3.3/0xFFF);
00187   if (meas < 1.3)
00188     return 0;
00189   else if (meas < 2.0)
00190     return 1;
00191   else
00192     return 2;
```

```
00193   return 1;
00194 }
00195
00202 uint8_t Blox_Accel_GetZTilt(void) {
00203   float meas = (float)(Accel_Measurements[2] * 3.3/0xFFF);
00204   if (meas < 1.3)
00205     return 2;
00206   else if (meas < 2.0)
00207     return 1;
00208   else
00209     return 0;
00210   return 1;
00211 }
```

6.39 drivers/src/blox_counter.c File Reference

Basic millisecond-resolution counter. Wraps after ~500 days.

```
#include "blox_counter.h"
```

- void **SysTick_Init** (void)

  Initializes the SysTick driver.

- uint32_t **SysTick_Get_Milliseconds** (void)

  *Returns the number of milliseconds since **SysTick_Init()** (p. 16) was called.*

- uint32_t **SysTick_Get_Seconds** (void)

  *Returns the number of seconds since **SysTick_Init()** (p. 16) was called.*

- uint32_t **SysTick_Get_Minutes** (void)

  *Returns the number of minutes since **SysTick_Init()** (p. 16) was called.*

- void **SysTick_Wait** (uint32_t ms)

  Performs a blocking wait for ms milliseconds.

- void **SysTick_Handler** (void)

  The interrupt handler that updates the global time values.

6.39.1 Detailed Description

Basic millisecond-resolution counter. Wraps after ~500 days.

Author

Zach Wasson

Version

V0.1

Date

10/30/2010

Definition in file **blox_counter.c**.

6.40 drivers/src/blox_counter.c

```
00001
00009 #include "blox_counter.h"
00010
00019 static uint32_t milliseconds;
00020
00024 static uint32_t seconds;
00025
00029 static uint32_t minutes;
00030
00035 void SysTick_Init(void) {
00036   milliseconds = 0;
00037   seconds = 0;
00038   minutes = 0;
00039   if (SysTick_Config(SystemCoreClock / 1000))
00040   {
00041     /* Capture error */
00042     while (1);
00043   }
00044   NVIC_SetPriority(SysTick_IRQn, 0);
00045 }
00046
00051 uint32_t SysTick_Get_Milliseconds(void) {
00052   return milliseconds;
00053 }
00054
00059 uint32_t SysTick_Get_Seconds(void) {
00060   return seconds;
00061 }
00062
00067 uint32_t SysTick_Get_Minutes(void) {
00068   return minutes;
00069 }
00070
00076 void SysTick_Wait(uint32_t ms) {
00077   uint32_t currentTime = SysTick_Get_Milliseconds();
00078   while(SysTick_Get_Milliseconds() < currentTime + ms);
00079 }
00080
```

```
00085 void SysTick_Handler(void) {
00086   milliseconds++;
00087   if((milliseconds % 1000) == 0) {
00088     seconds++;
00089           if((seconds % 60) == 0) {
00090       minutes++;
00091         }
00092   }
00093 }
```

6.41 drivers/src/blox_exti.c File Reference

A wrapper around EXTI for the STM32F103.

```
#include "blox_exti.h"
```

- #define **XBEE_EXTI_LINE** 12
- #define **OLED_EXTI_LINE** 1
- #define **TOUCH1_EXTI_LINE** 10
- #define **TOUCH2_EXTI_LINE** 11
- #define **TOUCH3_EXTI_LINE** 14
- #define **TOUCH4_EXTI_LINE** 7
- void(∗ **EXTIn_Handler** [16])(void) = {NULL}

  Array of pointers for handlers for all 16 EXTI interrupts.

- void **Blox_EXTI_RCC_Configuration** (void)

  Initializes AFIO clock for the EXTI interface.

- void **Blox_EXTI_NVIC_Configuration** (uint8_t line)

  Initializes NVIC for the EXTI interface.

- uint8_t **isHardwareLine** (uint8_t line)

  Checks whether a line is a designated hardware line.

- void **Blox_EXTI_Init** (void)

  Initializes EXTI.

- **EXTI_ID Blox_EXTI_Register_HW_IRQ** (uint8_t GPIO_PortSource, uint8_t line, void(∗EXTI_-Handler)(void))

  Registers an EXTI IRQ that triggers on hardware.

- **EXTI_ID Blox_EXTI_Register_SW_IRQ** (void(∗EXTI_Handler)(void))

Registers an EXTI IRQ that triggers on software.

- void **Blox_EXTI_Release_IRQ** (**EXTI_ID** id)

  Releases a given EXTI interrupt.

- void **Blox_EXTI_Trigger_SW_IRQ** (**EXTI_ID** id)

  Registers an EXTI IRQ that triggers on software.

- void **Blox_EXTI_Enable_IRQ** (**EXTI_ID** id)

  Enables a given EXTI IRQ.

- void **Blox_EXTI_Disable_IRQ** (**EXTI_ID** id)

  Disables a given EXTI IRQ.

- void **EXTI0_IRQHandler** (void)

  This function handles External line 0 interrupt request.

- void **EXTI1_IRQHandler** (void)

  This function handles External line 1 interrupt request.

- void **EXTI2_IRQHandler** (void)

  This function handles External line 2 interrupt request.

- void **EXTI3_IRQHandler** (void)

  This function handles External line 3 interrupt request.

- void **EXTI4_IRQHandler** (void)

  This function handles External line 4 interrupt request.

- void **EXTI9_5_IRQHandler** (void)

  This function handles External lines 9 to 5 interrupt request.

- void **EXTI15_10_IRQHandler** (void)

  This function handles External lines 15 to 10 interrupt request.

6.41.1 Detailed Description

A wrapper around EXTI for the STM32F103.

Author

Zach Wasson

Version

V0.1

Date

10/20/2010

Definition in file **blox_exti.c**.

6.42 drivers/src/blox_exti.c

```
00001
00009 #include "blox_exti.h"
00010
00014 #define XBEE_EXTI_LINE    12
00015 #define OLED_EXTI_LINE    1
00016 #define TOUCH1_EXTI_LINE  10
00017 #define TOUCH2_EXTI_LINE  11
00018 #define TOUCH3_EXTI_LINE  14
00019 #define TOUCH4_EXTI_LINE  7
00020
00024 void (*EXTIn_Handler[16])(void) = {NULL};
00025
00026 void Blox_EXTI_RCC_Configuration(void);
00027 void Blox_EXTI_NVIC_Configuration (uint8_t line);
00028 uint8_t isHardwareLine(uint8_t line);
00029
00034 void Blox_EXTI_Init(void) {
00035   Blox_EXTI_RCC_Configuration();
00036
00037   Blox_System_Register_DeInit(&RCC_DeInit);
00038   Blox_System_Register_DeInit(&EXTI_DeInit);
00039 }
00040
00052 EXTI_ID Blox_EXTI_Register_HW_IRQ(uint8_t GPIO_PortSource, uint8_t line, void (*E
      XTI_Handler)(void)) {
00053   if(isHardwareLine(line) == TRUE) {
00054     EXTI_InitTypeDef EXTI_InitStructure;
00055     Blox_EXTI_NVIC_Configuration(line);
00056     GPIO_EXTILineConfig(GPIO_PortSource, line);
00057     EXTI_InitStructure.EXTI_Line = (1<<line);
00058     EXTI_InitStructure.EXTI_LineCmd = ENABLE;
00059     EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
00060     EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
00061     EXTI_Init(&EXTI_InitStructure);
00062     EXTIn_Handler[line] = EXTI_Handler;
00063     return (EXTI_ID)(line);
00064   }
00065   return EXTI_INVALID_LINE;
00066 }
00067
00075 EXTI_ID Blox_EXTI_Register_SW_IRQ(void (*EXTI_Handler)(void)) {
00076   uint8_t line;
```

```
00077   EXTI_InitTypeDef EXTI_InitStructure;
00078   for(line = 0; line <= 15; line++) {
00079     if(isHardwareLine(line) == FALSE && EXTIn_Handler[line] == NULL)
00080       break;
00081   }
00082   if (line > 15)
00083     return EXTI_IRQ_UNAVAILABLE;
00084   Blox_EXTI_NVIC_Configuration(line);
00085   EXTI_InitStructure.EXTI_Line = (1<<line);
00086   EXTI_InitStructure.EXTI_LineCmd = ENABLE;
00087   EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
00088   EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
00089   EXTI_Init(&EXTI_InitStructure);
00090   EXTIn_Handler[line] = EXTI_Handler;
00091   return (EXTI_ID)(line);
00092 }
00093
00099 void Blox_EXTI_Release_IRQ(EXTI_ID id) {
00100   Blox_EXTI_Disable_IRQ(id);
00101   EXTIn_Handler[id] = NULL;
00102 }
00103
00109 void Blox_EXTI_Trigger_SW_IRQ(EXTI_ID id) {
00110   if(isHardwareLine(id) == FALSE) {
00111     EXTI_GenerateSWInterrupt(1<<id);
00112   }
00113 }
00114
00120 uint8_t isHardwareLine(uint8_t line) {
00121   return (line == XBEE_EXTI_LINE || line == OLED_EXTI_LINE ||
00122       line == TOUCH1_EXTI_LINE || line == TOUCH2_EXTI_LINE ||
00123       line == TOUCH3_EXTI_LINE || line == TOUCH4_EXTI_LINE);
00124 }
00125
00130 void Blox_EXTI_RCC_Configuration(void) {
00131   RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
00132 }
00133
00140 void Blox_EXTI_NVIC_Configuration (uint8_t line) {
00141   NVIC_InitTypeDef NVIC_InitStructure;
00142   NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);
00143   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 3;
00144   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
00145   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
00146   switch(line) {
00147     case 0:
00148       NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 13;
00149       NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;
00150       break;
00151     case 1:
00152       NVIC_InitStructure.NVIC_IRQChannel = EXTI1_IRQn;
00153       break;
00154     case 2:
00155       NVIC_InitStructure.NVIC_IRQChannel = EXTI2_IRQn;
00156       break;
00157     case 3:
00158       NVIC_InitStructure.NVIC_IRQChannel = EXTI3_IRQn;
```

```
00159       break;
00160     case 4:
00161       NVIC_InitStructure.NVIC_IRQChannel = EXTI4_IRQn;
00162       break;
00163     case 5:
00164     case 6:
00165     case 7:
00166     case 8:
00167     case 9:
00168       NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;
00169       break;
00170     case 10:
00171     case 11:
00172     case 13:
00173     case 12:
00174     case 14:
00175     case 15:
00176       NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;
00177       break;
00178   }
00179   NVIC_Init(&NVIC_InitStructure);
00180 }
00181
00187 void Blox_EXTI_Enable_IRQ(EXTI_ID id) {
00188   EXTI_ClearITPendingBit(1<<id);
00189   EXTI->IMR |= (1<<id);
00190 }
00191
00197 void Blox_EXTI_Disable_IRQ(EXTI_ID id) {
00198   EXTI->IMR &= ~(1<<id);
00199 }
00200
00205 void EXTI0_IRQHandler(void) {
00206   if(EXTI_GetITStatus(EXTI_Line0) != RESET) {
00207     if(EXTIn_Handler[0] != NULL) {
00208       (*EXTIn_Handler[0])();
00209     }
00210     EXTI_ClearITPendingBit(EXTI_Line0);
00211   }
00212 }
00213
00218 void EXTI1_IRQHandler(void) {
00219   if(EXTI_GetITStatus(EXTI_Line1) != RESET) {
00220     if(EXTIn_Handler[1] != NULL) {
00221       (*EXTIn_Handler[1])();
00222     }
00223     EXTI_ClearITPendingBit(EXTI_Line1);
00224   }
00225 }
00226
00231 void EXTI2_IRQHandler(void) {
00232   if(EXTI_GetITStatus(EXTI_Line2) != RESET) {
00233     if(EXTIn_Handler[2] != NULL) {
00234       (*EXTIn_Handler[2])();
00235     }
00236     EXTI_ClearITPendingBit(EXTI_Line2);
00237   }
```

```
00238 }
00239
00244 void EXTI3_IRQHandler(void) {
00245   if(EXTI_GetITStatus(EXTI_Line3) != RESET) {
00246     if(EXTIn_Handler[3] != NULL) {
00247       (*EXTIn_Handler[3])();
00248     }
00249     EXTI_ClearITPendingBit(EXTI_Line3);
00250   }
00251 }
00252
00257 void EXTI4_IRQHandler(void) {
00258   if(EXTI_GetITStatus(EXTI_Line4) != RESET) {
00259     if(EXTIn_Handler[4] != NULL) {
00260       (*EXTIn_Handler[4])();
00261     }
00262     EXTI_ClearITPendingBit(EXTI_Line4);
00263   }
00264 }
00265
00270 void EXTI9_5_IRQHandler(void) {
00271   if(EXTI_GetITStatus(EXTI_Line5) != RESET) {
00272     if(EXTIn_Handler[5] != NULL) {
00273       (*EXTIn_Handler[5])();
00274     }
00275     EXTI_ClearITPendingBit(EXTI_Line5);
00276   }
00277   else if(EXTI_GetITStatus(EXTI_Line6) != RESET) {
00278     if(EXTIn_Handler[6] != NULL) {
00279       (*EXTIn_Handler[6])();
00280     }
00281     EXTI_ClearITPendingBit(EXTI_Line6);
00282   }
00283   else if(EXTI_GetITStatus(EXTI_Line7) != RESET) {
00284     if(EXTIn_Handler[7] != NULL) {
00285       (*EXTIn_Handler[7])();
00286     }
00287     EXTI_ClearITPendingBit(EXTI_Line7);
00288   }
00289   else if(EXTI_GetITStatus(EXTI_Line8) != RESET) {
00290     if(EXTIn_Handler[8] != NULL) {
00291       (*EXTIn_Handler[8])();
00292     }
00293     EXTI_ClearITPendingBit(EXTI_Line8);
00294   }
00295   else if(EXTI_GetITStatus(EXTI_Line9) != RESET) {
00296     if(EXTIn_Handler[9] != NULL) {
00297       (*EXTIn_Handler[9])();
00298     }
00299     EXTI_ClearITPendingBit(EXTI_Line9);
00300   }
00301 }
00302
00307 void EXTI15_10_IRQHandler(void) {
00308   if(EXTI_GetITStatus(EXTI_Line10) != RESET) {
00309     if(EXTIn_Handler[10] != NULL) {
00310       (*EXTIn_Handler[10])();
```

```
00311     }
00312     EXTI_ClearITPendingBit(EXTI_Line10);
00313   }
00314   else if(EXTI_GetITStatus(EXTI_Line11) != RESET) {
00315     if(EXTIn_Handler[11] != NULL) {
00316       (*EXTIn_Handler[11])();
00317     }
00318     EXTI_ClearITPendingBit(EXTI_Line11);
00319   }
00320   else if(EXTI_GetITStatus(EXTI_Line12) != RESET) {
00321     if(EXTIn_Handler[12] != NULL) {
00322       (*EXTIn_Handler[12])();
00323     }
00324     EXTI_ClearITPendingBit(EXTI_Line12);
00325   }
00326   else if(EXTI_GetITStatus(EXTI_Line13) != RESET) {
00327     if(EXTIn_Handler[13] != NULL) {
00328       (*EXTIn_Handler[13])();
00329     }
00330     EXTI_ClearITPendingBit(EXTI_Line13);
00331   }
00332   else if(EXTI_GetITStatus(EXTI_Line14) != RESET) {
00333     if(EXTIn_Handler[14] != NULL) {
00334       (*EXTIn_Handler[14])();
00335     }
00336     EXTI_ClearITPendingBit(EXTI_Line14);
00337   }
00338   else if(EXTI_GetITStatus(EXTI_Line15) != RESET) {
00339     if(EXTIn_Handler[15] != NULL) {
00340       (*EXTIn_Handler[15])();
00341     }
00342     EXTI_ClearITPendingBit(EXTI_Line15);
00343   }
00344 }
```

6.43 drivers/src/blox_fifo.c File Reference

A FIFO implemenation.

```
#include "blox_fifo.h"
```

Functions

- void **Blox_FIFO_Init** (**FIFO_Type** *fifo)

  *Initializes the **FIFO_Type** (p. 98) structure provided.*

- **FIFO_STATUS Blox_FIFO_Put** (**FIFO_Type** *fifo, uint16_t data)

  Puts data in fifo.

- uint32_t **Blox_FIFO_Get** (**FIFO_Type** *fifo)

Gets data out of fifo.

- uint8_t **Blox_FIFO_Size** (**FIFO_Type** *fifo)

  Gets size of fifo.

6.43.1 Detailed Description

A FIFO implemenation.

Author

Zach Wasson

Version

V0.1

Date

10/20/2010

Definition in file **blox_fifo.c**.

6.44 drivers/src/blox_fifo.c

```
00001
00009 #include "blox_fifo.h"
00020 void Blox_FIFO_Init(FIFO_Type *fifo) {
00021   fifo->read = 0;
00022   fifo->write = 0;
00023 }
00024
00031 FIFO_STATUS Blox_FIFO_Put(FIFO_Type *fifo, uint16_t data) {
00032   if(Blox_FIFO_Size(fifo) < FIFO_SIZE) {
00033     fifo->data[fifo->write & (FIFO_SIZE - 1)] = data;
00034     fifo->write++;
00035     return FIFO_OK;
00036   }
00037   return -FIFO_FULL;
00038 }
00039
00045 uint32_t Blox_FIFO_Get(FIFO_Type *fifo) {
00046   if(Blox_FIFO_Size(fifo) > 0) {
00047     uint16_t data;
00048     data = fifo->data[fifo->read & (FIFO_SIZE - 1)];
00049     fifo->read++;
00050     return data;
00051   }
```

```
00052   return FIFO_BAD;
00053 }
00054
00060 uint8_t Blox_FIFO_Size(FIFO_Type *fifo) {
00061   return (fifo->write - fifo->read);
00062 }
00063
```

6.45 drivers/src/blox_filesystem.c File Reference

Provides a filesystem interface to a section of flash memory.

```
#include "blox_filesystem.h"
```

- volatile **FS_Table** * **fat** = 0

  The internal pointer to the FAT.

- **FS_STATUS FS_Init** (bool create)

  Initializes the filesystem.

- **FS_STATUS FS_CreateFS** (void)

  Creates a filesystem at the default location. Should only be used once.

- **FS_STATUS FS_ChkValid** (void)
- **FS_File** * **FS_GetFile** (uint8_t id)

  Retrieves a file handle from the filesystem given a specific file id.

- **FS_File** * **FS_GetFileFromName** (char *name)

  Gets a file in the filesystem by looking for the filename.

- uint8_t **FS_GetNumFiles** (void)

  Returns the number of files being managed by the filesystem.

- **FS_STATUS FS_DeleteFile** (uint8_t id)

  Deletes a file from the filesystem. Shifts data up so fragmentation does not occur.

- uint8_t **FS_CreateFile** (char *name, uint8_t numPages)

  Creates a new file of a given size in the filesystem.

- **FS_STATUS FS_WriteFilePage** (uint8_t id, uint32_t *data, uint32_t page_offset)

  Writes data into a file in the filesystem at a given offset.

- void **FS_SwapPage** (uint32_t *src, uint32_t *dst)

  Swaps a page in RAM with a page in Flash.

- uint32_t **FS_RoundPageUp** (uint32_t size)

  Rounds a size up to the next multiple of PAGE_SIZE.

- void **FS_RunFile** (uint8_t file_id)

  De-initializes the system and runs the application stored at the file id.

- void **FS_RunStage** (void)

  Runs the application stored in the staging area. Assumes from a system reset.

- uint8_t **FS_GetAppFlag** (void)

  Returns the flag designating if an application should be run.

- void **FS_SetAppFlag** (uint8_t val)

  Sets the flag desinating if an application should be run.

6.45.1 Detailed Description

Provides a filesystem interface to a section of flash memory.

Author

Jesse Tannahill

Version

V0.1

Date

10/18/2010

Definition in file **blox_filesystem.c**.

6.46 drivers/src/blox_filesystem.c

```
00001
00009 #include "blox_filesystem.h"
00018 volatile FS_Table *fat = 0;
00019
```

```
00024 FS_STATUS FS_Init(bool create) {
00025   if(fat != 0)
00026     return FS_OK;
00027   fat = (FS_Table *)MEM_FAT_START;
00028   //Create the file system if desired
00029   if(fat->magic != FS_MAGIC && create)
00030     FS_CreateFS();
00031   else
00032     return FS_BAD_FAT;
00033
00034   return FS_ChkValid();
00035 }
00036
00042 FS_STATUS FS_CreateFS(void) {
00043   uint32_t i;
00044   FS_Table *fat_new = (FS_Table *)malloc(PAGE_SIZE);
00045
00046   if(fat_new == NULL)
00047     return FS_CREATE_FAIL;
00048
00049   fat_new->magic = FS_MAGIC;
00050   fat_new->numFiles = 0;
00051   fat_new->free_top = (uint32_t *)(MEM_STORE_START);
00052   fat_new->free_numPages = (MEM_STORE_SIZE/PAGE_SIZE);
00053   for(i = 0; i < FS_MAX_FILES; i++) {
00054     fat_new->table[i].id = FS_MAX_FILES;
00055     fat_new->table[i].numPages = 0;
00056     fat_new->table[i].data = 0;
00057   }
00058
00059   fat = (FS_Table *)MEM_FAT_START;
00060
00061   FS_SwapPage((uint32_t *)fat_new, (uint32_t *)fat);
00062   free(fat_new);
00063   return FS_OK;
00064 }
00065
00070 FS_STATUS FS_ChkValid(void) {
00071   uint32_t i, real_free_top, real_free_numPages;
00072   if (fat->magic != FS_MAGIC) {
00073     Blox_DebugStr("FS_ChkValid failed! Bad magic.\r\n");
00074           return FS_BAD_FAT;
00075   }
00076
00077   for(i = 0; i < fat->numFiles; i++) {
00078           if(fat->table[i].id == FS_MAX_FILES) {
00079       Blox_DebugPat("FS_ChkValid failed! Bad uninitialized id: %u\r\n", fat->
      table[i].id);
00080       return FS_BAD_FAT;
00081     }
00082   }
00083
00084   if(fat->numFiles == 0) {
00085     if ((uint32_t)fat->free_top != MEM_STORE_START
00086           || fat->free_numPages != MEM_STORE_SIZE/PAGE_SIZE) {
00087       Blox_DebugStr("FS_ChkValid failed! Bad empty free_top or free_size.\r\n");
00088       return FS_BAD_FAT;
```

```
00089     }
00090   } else {
00091     real_free_top = (uint32_t)fat->table[fat->numFiles-1].data
00092                   + fat->table[fat->numFiles-1].numPages * PAGE_SIZE;
00093     real_free_numPages = (MEM_STORE_START + MEM_STORE_SIZE - real_free_top)/PAGE_
      SIZE;
00094     if ((uint32_t)fat->free_top != real_free_top
00095             || fat->free_numPages != real_free_numPages) {
00096       Blox_DebugStr("FS_ChkValid failed! Bad real free_top or free_numPages.\r\n"
      );
00097       return FS_BAD_FAT;
00098     }
00099   }
00100   return FS_OK;
00101 }
00102
00108 FS_File * FS_GetFile(uint8_t id) {
00109   if (fat == 0)
00110           return 0;
00111   if (id > (fat->numFiles-1))
00112           return 0;
00113   return (FS_File *)(fat->table+id);
00114 }
00115
00121 FS_File * FS_GetFileFromName(char *name) {
00122   uint8_t num_files = FS_GetNumFiles();
00123   uint8_t i;
00124   for (i = 0; i < num_files; i++) {
00125     FS_File *file;
00126     file = FS_GetFile(i);
00127     if (memcmp(name, file->name, FS_FILE_MAX_NAME_LEN) == 0)
00128       return file;
00129   }
00130   return NULL;
00131 }
00132
00137 uint8_t FS_GetNumFiles(void) { return fat->numFiles; }
00138
00145 FS_STATUS FS_DeleteFile(uint8_t id) {
00146   uint32_t i, j;
00147   FS_Table *fat_new;
00148   if (fat == 0)
00149           return FS_FAT_NOT_INIT;
00150   if (id > (fat->numFiles-1))
00151           return FS_FILE_NOT_INIT;
00152   fat_new = (FS_Table *)malloc(PAGE_SIZE);
00153   memmove(fat_new, (const void *)fat, PAGE_SIZE);
00154
00155   if (fat_new->numFiles-1 == id) {
00156     fat_new->free_top = (uint32_t *)((uint32_t)(fat_new->free_top) - fat_new->
      table[id].numPages * PAGE_SIZE);
00157   } else {
00158     fat_new->free_top = fat_new->table[id].data;
00159     for (i = id; i < fat->numFiles-1; i++) {
00160       fat_new->table[i].numPages = fat_new->table[i+1].numPages;
00161       strcpy(fat_new->table[i].name, fat_new->table[i+1].name);
00162       fat_new->table[i].data = fat_new->free_top;
```

```
00163       for (j = 0; j < fat_new->table[i].numPages; j++) {
00164         FS_SwapPage((uint32_t *)((uint32_t)(fat_new->table[i+1].data)+ j*PAGE_SIZ
      E),
00165                     (uint32_t *)((uint32_t)(fat_new->table[i].data)  + j*PAGE_SIZ
      E));
00166       }
00167       fat_new->free_top = (uint32_t *)((uint32_t)(fat_new->table[i].data) + fat_n
      ew->table[i].numPages * PAGE_SIZE);
00168     }
00169   }
00170
00171   fat_new->free_numPages = (MEM_STORE_START+MEM_STORE_SIZE-(uint32_t)(fat_new->
      free_top))/PAGE_SIZE;
00172   fat_new->numFiles--;
00173   fat_new->table[fat_new->numFiles].id = FS_MAX_FILES;
00174
00175   FS_SwapPage((uint32_t *)fat_new, (uint32_t *)fat);
00176   free(fat_new);
00177
00178   return FS_ChkValid();
00179 }
00180
00187 uint8_t FS_CreateFile(char *name, uint8_t numPages) {
00188 uint8_t new_id;
00189 FS_Table *new_fat;
00190   Blox_DebugStr("Creating a file!\r\n");
00191   if (fat == 0)
00192     return FS_MAX_FILES;
00193   if (fat->numFiles == FS_MAX_FILES || strlen(name) > FS_FILE_MAX_NAME_LEN-1)
00194     return FS_MAX_FILES;
00195   if (fat->free_numPages < numPages || numPages == 0)
00196     return FS_MAX_FILES;
00197   new_fat = (FS_Table *)malloc(PAGE_SIZE);
00198   memmove(new_fat, (const void *)fat, PAGE_SIZE);
00199   new_id = new_fat->numFiles++;
00200   new_fat->table[new_id].id = new_id;
00201   strcpy(new_fat->table[new_id].name, name);
00202   new_fat->table[new_id].numPages = numPages;
00203   new_fat->table[new_id].data = new_fat->free_top;
00204   new_fat->free_top = (uint32_t *)((uint32_t)new_fat->free_top + new_fat->table[n
      ew_id].numPages * PAGE_SIZE);
00205   new_fat->free_numPages -= new_fat->table[new_id].numPages;
00206
00207   FS_SwapPage((uint32_t *)new_fat, (uint32_t *)fat);
00208   free(new_fat);
00209
00210   Blox_DebugStr("Created a new file!\r\n");
00211
00212   if(FS_ChkValid() != FS_OK) {
00213     Blox_DebugStr("FS_CreateFile ChkValid failed!\r\n");
00214     return FS_MAX_FILES;
00215   }
00216
00217   return new_id;
00218 }
00219
00227 FS_STATUS FS_WriteFilePage(uint8_t id, uint32_t *data, uint32_t page_offset) {
```

```
00228   if(page_offset > fat->table[id].numPages-1)
00229     return FS_BAD_WRITE;
00230   FS_SwapPage(data, (uint32_t *)((uint32_t)(fat->table[id].data) + page_offset*PA
      GE_SIZE));
00231   return FS_OK;
00232 }
00233
00240 void FS_SwapPage(uint32_t *src, uint32_t *dst) {
00241 int i;
00242   FLASH_Unlock();
00243   FLASH_ErasePage((uint32_t)dst);
00244   for(i = 0; i < PAGE_SIZE/WORD_SIZE; i++)
00245     FLASH_ProgramWord((uint32_t)(dst++), src[i]);
00246 }
00247
00253 uint32_t FS_RoundPageUp(uint32_t size) {
00254   if((uint32_t)size % PAGE_SIZE)
00255     return (uint32_t)size / PAGE_SIZE + 1;
00256   else
00257     return (uint32_t)size / PAGE_SIZE;
00258 }
00259
00265 void FS_RunFile(uint8_t file_id) {
00266   uint32_t i;
00267   FS_File *file = FS_GetFile(file_id);
00268   //Swap to staging area
00269   for (i = 0; i < file->numPages; i++) {
00270     FS_SwapPage((uint32_t *)((char *)(file->data)+i*PAGE_SIZE),
00271                 (uint32_t *)((char *)MEM_STAGE_START+i*PAGE_SIZE));
00272   }
00273
00274   FS_SetAppFlag(1);
00275   NVIC_SystemReset();
00276 }
00277
00282 void FS_RunStage(void) {
00283   uint32_t app_addr;
00284   volatile uint32_t jump_addr;
00285   void (*app_fn)(void);
00286   //Deinit everything, set up vector/stack, and jump
00287   app_addr = MEM_STAGE_START;
00288   jump_addr = *(__IO uint32_t *) (app_addr+4);
00289   app_fn = (void (*)(void))jump_addr;
00290
00291   FS_SetAppFlag(0);
00292   NVIC_SetVectorTable(NVIC_VectTab_FLASH, app_addr);
00293   __set_MSP(*(__IO uint32_t*)app_addr);
00294   app_fn();
00295
00296   while (1) ; //Shouldn't get here
00297 }
00298
00303 uint8_t FS_GetAppFlag(void) {
00304   return *(uint8_t *)FS_APP_FLAG_LOC;
00305 }
00306
00312 void FS_SetAppFlag(uint8_t val) {
```

```
00313   *(uint8_t *)FS_APP_FLAG_LOC = val;
00314 }
```

6.47 drivers/src/blox_ir.c File Reference

A wrapper class for the IR sensors that use a USART interface.

```
#include "blox_ir.h"
```

- void(∗ **IR1_RX_Handler**)(**IRFrame** ∗frame) = NULL
  Handler for users to register a function with in IR1.
- void(∗ **IR2_RX_Handler**)(**IRFrame** ∗frame) = NULL
  Handler for users to register a function with in IR2.
- void(∗ **IR3_RX_Handler**)(**IRFrame** ∗frame) = NULL
  Handler for users to register a function with in IR3.
- void(∗ **IR4_RX_Handler**)(**IRFrame** ∗frame) = NULL
  Handler for users to register a function with in IR4.
- uint8_t **IR1_RX_Enable** = FALSE
  Flag to determine if a user register gets called in an interrupt for IR1.
- uint8_t **IR2_RX_Enable** = FALSE
  Flag to determine if a user register gets called in an interrupt for IR2.
- uint8_t **IR3_RX_Enable** = FALSE
  Flag to determine if a user register gets called in an interrupt for IR3.
- uint8_t **IR4_RX_Enable** = FALSE
  Flag to determine if a user register gets called in an interrupt for IR4.
- void **IR_RCC_Configuration** ()
  Initializes clocks for IR shutdown pin.
- void **IR_GPIO_Configuration** ()
  Initializes the gpio for the IR shutdown pin.
- void **Blox_IR1_USART_RXNE_IRQ** (void)

The function that IR1 registers with USART to execute on byte received.

- void **Blox_IR2_USART_RXNE_IRQ** (void)

  The function that IR2 registers with USART to execute on byte received.

- void **Blox_IR3_USART_RXNE_IRQ** (void)

  The function that IR3 registers with USART to execute on byte received.

- void **Blox_IR4_USART_RXNE_IRQ** (void)

  The function that IR4 registers with USART to execute on byte received.

- void **IR_Init** (uint8_t id)

  Initializes the IR module. Basically a wrapper on USART.

- void **Blox_IR_Register_RX_IRQ** (uint8_t id, void(∗RX_Handler)(**IRFrame** ∗frame))

  Registers a function to execute when a complete IR frame is received.

- void **Blox_IR_Enable_RX_IRQ** (uint8_t id)

  Enables the sw interrupt that occurs when a XBee reads a byte.

- void **Blox_IR_Disable_RX_IRQ** (uint8_t id)

  Disables the sw interrupt that occurs when a XBee reads a byte.

- uint8_t **IR_Receive** (uint8_t id)

  Receive a byte on the given IR. A wrapper around USART.

- uint8_t **IR_TryReceive** (uint8_t id)

  Receive a byte on the given IR. A wrapper around USART.

- void **IR_Send** (uint8_t id, uint8_t data)

  Sends a byte out on the given IR. Wrapper around USART.

- void **IR_SendFrame** (uint8_t id, **IRFrameType** type, uint8_t ∗data, uint8_t len)

  *Sends a **IRFrame** (*p. *103)* *out on the given IR.*

- void **IR_Wake** (void)

  Wakes all the IRs from sleep.

- void **IR_Sleep** (void)

  Put all the IRs to sleep.

6.47.1 Detailed Description

A wrapper class for the IR sensors that use a USART interface.

Author

Jesse Tannahill

Version

V0.1

Date

10/19/2010

Definition in file **blox_ir.c**.

6.47.2 Function Documentation

6.47.2.1 void Blox_IR1_USART_RXNE_IRQ (void)

The function that IR1 registers with USART to execute on byte received.

Return values

None.	

Definition at line **102** of file **blox_ir.c**.

6.47.2.2 void Blox_IR2_USART_RXNE_IRQ (void)

The function that IR2 registers with USART to execute on byte received.

Return values

None.	

Definition at line **163** of file **blox_ir.c**.

6.47.2.3 void Blox_IR3_USART_RXNE_IRQ (void)

The function that IR3 registers with USART to execute on byte received.

Return values

None.	

Definition at line **224** of file **blox_ir.c**.

6.47.2.4 void Blox_IR4_USART_RXNE_IRQ (void)

The function that IR4 registers with USART to execute on byte received.

Return values

None.	

Definition at line **285** of file **blox_ir.c**.

6.47.2.5 void Blox_IR_Disable_RX_IRQ (uint8_t *id*)

Disables the sw interrupt that occurs when a XBee reads a byte.

Parameters

id	the IR id

Return values

None.	

Definition at line **409** of file **blox_ir.c**.

6.47.2.6 void Blox_IR_Enable_RX_IRQ (uint8_t *id*)

Enables the sw interrupt that occurs when a XBee reads a byte.

Parameters

id	the IR id

Return values

None.	

Definition at line **383** of file **blox_ir.c**.

6.47.2.7 void Blox_IR_Register_RX_IRQ (uint8_t *id,* void(*)(IRFrame *frame) *RX_Handler*)

Registers a function to execute when a complete IR frame is received.

Parameters

id	the IR id
RX_Handler	the user function to call on interrupt

Return values

None.	

Definition at line **361** of file **blox_ir.c**.

6.47.2.8 void IR_GPIO_Configuration ( )

Initializes the gpio for the IR shutdown pin.

Return values

None	

Definition at line **346** of file **blox_ir.c**.

6.47.2.9 void IR_Init (uint8_t *id*)

Initializes the IR module. Basically a wrapper on USART.

Parameters

id	the id of the USART interface to initialize.

Return values

None	

Definition at line **65** of file **blox_ir.c**.

6.47.2.10 void IR_RCC_Configuration ( )

Initializes clocks for IR shutdown pin.

Return values

None	

Definition at line **94** of file **blox_ir.c**.

6.47.2.11 uint8_t IR_Receive (uint8_t *id*)

Receive a byte on the given IR. A wrapper around USART.

Parameters

id	the IR id to use.

Return values

The	received command or 0 on error.

Definition at line **435** of file **blox_ir.c**.

6.47.2.12 void IR_Send (uint8_t *id,* uint8_t *data*)

Sends a byte out on the given IR. Wrapper around USART.

Parameters

id	the IR id to use
data	the byte to send

Return values

None.	

Definition at line **474** of file **blox_ir.c**.

6.47.2.13 void IR_SendFrame (uint8_t *id,* IRFrameType *type,* uint8_t * *data,* uint8_t *len*)

Sends a **IRFrame** (p. 103) out on the given IR.

Parameters

id	the IR id to use
type	the type of the **IRFrame** (p. 103)
data	a pointer to the data to be sent
len	the length of the data to be sent

Return values

None.	

Definition at line **499** of file **blox_ir.c**.

6.47.2.14 void IR_Sleep (void)

Put all the IRs to sleep.

Return values

None.	

Definition at line **566** of file **blox_ir.c**.

6.47.2.15 uint8_t IR_TryReceive (uint8_t *id*)

Receive a byte on the given IR. A wrapper around USART.

Parameters

id	the IR id to use.

Return values

The	received command or 0 on error.

Definition at line **454** of file **blox_ir.c**.

6.47.2.16 void IR_Wake (void)

Wakes all the IRs from sleep.

Return values

None.	

Definition at line **558** of file **blox_ir.c**.

6.48 drivers/src/blox_ir.c

```
00001
00009 #include "blox_ir.h"
00010
00016 /* Private function prototypes */
00017 void IR_RCC_Configuration(void);
00018 void IR_GPIO_Configuration(void);
```

```
00019
00020 void Blox_IR1_USART_RXNE_IRQ(void);
00021 void Blox_IR2_USART_RXNE_IRQ(void);
00022 void Blox_IR3_USART_RXNE_IRQ(void);
00023 void Blox_IR4_USART_RXNE_IRQ(void);
00024
00028 void (*IR1_RX_Handler)(IRFrame *frame) = NULL;
00032 void (*IR2_RX_Handler)(IRFrame *frame) = NULL;
00036 void (*IR3_RX_Handler)(IRFrame *frame) = NULL;
00040 void (*IR4_RX_Handler)(IRFrame *frame) = NULL;
00041
00042
00046 uint8_t IR1_RX_Enable = FALSE;
00050 uint8_t IR2_RX_Enable = FALSE;
00054 uint8_t IR3_RX_Enable = FALSE;
00058 uint8_t IR4_RX_Enable = FALSE;
00059
00065 void IR_Init(uint8_t id) {
00066   IR_RCC_Configuration();
00067   IR_GPIO_Configuration();
00068   IR_Wake();
00069
00070   switch(id) {
00071   case 1:
00072     Blox_USART_Init(IR_1_USART_ID);
00073     Blox_USART_Register_RXNE_IRQ(IR_1_USART_ID, &Blox_IR1_USART_RXNE_IRQ);
00074     break;
00075   case 2:
00076     Blox_USART_Init(IR_2_USART_ID);
00077     Blox_USART_Register_RXNE_IRQ(IR_2_USART_ID, &Blox_IR2_USART_RXNE_IRQ);
00078     break;
00079   case 3:
00080     Blox_USART_Init(IR_3_USART_ID);
00081     Blox_USART_Register_RXNE_IRQ(IR_3_USART_ID, &Blox_IR3_USART_RXNE_IRQ);
00082     break;
00083   case 4:
00084     Blox_USART_Init(IR_4_USART_ID);
00085     Blox_USART_Register_RXNE_IRQ(IR_4_USART_ID, &Blox_IR4_USART_RXNE_IRQ);
00086     break;
00087   }
00088 }
00089
00094 void IR_RCC_Configuration() {
00095   RCC_APB2PeriphClockCmd(IR_SHUTDOWN_GPIO_CLK, ENABLE);
00096 }
00097
00102 void Blox_IR1_USART_RXNE_IRQ(void) {
00103   static uint8_t num = 0;
00104   static uint8_t checksum = 0;
00105   static IRFrame frame;
00106   uint8_t data;
00107   int16_t temp;
00108
00109   temp = Blox_USART_TryReceive(IR_1_USART_ID);
00110   if (temp == -1)
00111     return;
00112
```

```
00113   data = (uint8_t) temp;
00114   if (num == 0) {
00115     if (data == 0x7E) {
00116       num = 1; //Start of a frame
00117       checksum = 0;
00118     }
00119   } else if (num == 1) {
00120     frame.src_id = data;
00121     num++;
00122   } else if (num == 2) {
00123     frame.src_face_id = data;
00124     num++;
00125   } else if (num == 3) {
00126     frame.type = (IRFrameType)(data & 0xFF);
00127     num++;
00128   } else if (num == 4) {
00129     frame.len = data;
00130     if (frame.len > IR_MAX_FRAME_LEN) {
00131       num = 0;
00132     } else {
00133       frame.data = (uint8_t *)malloc(frame.len*sizeof(uint8_t));
00134       num++;
00135     }
00136   } else if (num > 4) {
00137     if (num == frame.len+5) {
00138       if(checksum == data) {
00139         if (IR1_RX_Handler != NULL && IR1_RX_Enable == TRUE) {
00140           IRFrame *retFrame;
00141           retFrame = (IRFrame *)malloc(sizeof(IRFrame));
00142           memcpy(retFrame, &(frame), sizeof(IRFrame));
00143           (*IR1_RX_Handler)(retFrame);
00144         } else {
00145           free(frame.data);
00146         }
00147       } else {
00148         free(frame.data);
00149       }
00150       num = 0;
00151     } else {
00152       frame.data[num-5] = data;
00153       checksum ^= data;
00154       num++;
00155     }
00156   }
00157 }
00158
00163 void Blox_IR2_USART_RXNE_IRQ(void) {
00164   static uint8_t num = 0;
00165   static uint8_t checksum = 0;
00166   static IRFrame frame;
00167   uint8_t data;
00168   int16_t temp;
00169
00170   temp = Blox_USART_TryReceive(IR_2_USART_ID);
00171   if (temp == -1)
00172     return;
00173
```

```
00174   data = (uint8_t) temp;
00175   if (num == 0) {
00176     if (data == 0x7E) {
00177       num = 1; //Start of a frame
00178       checksum = 0;
00179     }
00180   } else if (num == 1) {
00181     frame.src_id = data;
00182     num++;
00183   } else if (num == 2) {
00184     frame.src_face_id = data;
00185     num++;
00186   } else if (num == 3) {
00187     frame.type = (IRFrameType)(data & 0xFF);
00188     num++;
00189   } else if (num == 4) {
00190     frame.len = data;
00191     if (frame.len > IR_MAX_FRAME_LEN) {
00192       num = 0;
00193     } else {
00194       frame.data = (uint8_t *)malloc(frame.len*sizeof(uint8_t));
00195       num++;
00196     }
00197   } else if (num > 4) {
00198     if (num == frame.len+5) {
00199       if(checksum == data) {
00200         if (IR2_RX_Handler != NULL && IR2_RX_Enable == TRUE) {
00201           IRFrame *retFrame;
00202           retFrame = (IRFrame *)malloc(sizeof(IRFrame));
00203           memcpy(retFrame, &(frame), sizeof(IRFrame));
00204           (*IR2_RX_Handler)(retFrame);
00205         } else {
00206           free(frame.data);
00207         }
00208       } else {
00209         free(frame.data);
00210       }
00211       num = 0;
00212     } else {
00213       frame.data[num-5] = data;
00214       checksum ^= data;
00215       num++;
00216     }
00217   }
00218 }
00219
00224 void Blox_IR3_USART_RXNE_IRQ(void) {
00225   static uint8_t num = 0;
00226   static uint8_t checksum = 0;
00227   static IRFrame frame;
00228   uint8_t data;
00229   int16_t temp;
00230
00231   temp = Blox_USART_TryReceive(IR_3_USART_ID);
00232   if (temp == -1)
00233     return;
00234
```

```
00235   data = (uint8_t) temp;
00236   if (num == 0) {
00237     if (data == 0x7E) {
00238       num = 1; //Start of a frame
00239       checksum = 0;
00240     }
00241   } else if (num == 1) {
00242     frame.src_id = data;
00243     num++;
00244   } else if (num == 2) {
00245     frame.src_face_id = data;
00246     num++;
00247   } else if (num == 3) {
00248     frame.type = (IRFrameType)(data & 0xFF);
00249     num++;
00250   } else if (num == 4) {
00251     frame.len = data;
00252     if (frame.len > IR_MAX_FRAME_LEN) {
00253       num = 0;
00254     } else {
00255       frame.data = (uint8_t *)malloc(frame.len*sizeof(uint8_t));
00256       num++;
00257     }
00258   } else if (num > 4) {
00259     if (num == frame.len+5) {
00260       if(checksum == data) {
00261         if (IR3_RX_Handler != NULL && IR3_RX_Enable == TRUE) {
00262           IRFrame *retFrame;
00263           retFrame = (IRFrame *)malloc(sizeof(IRFrame));
00264           memcpy(retFrame, &(frame), sizeof(IRFrame));
00265           (*IR3_RX_Handler)(retFrame);
00266         } else {
00267           free(frame.data);
00268         }
00269       } else {
00270         free(frame.data);
00271       }
00272       num = 0;
00273     } else {
00274       frame.data[num-5] = data;
00275       checksum ^= data;
00276       num++;
00277     }
00278   }
00279 }
00280
00285 void Blox_IR4_USART_RXNE_IRQ(void) {
00286   static uint8_t num = 0;
00287   static uint8_t checksum = 0;
00288   static IRFrame frame;
00289   uint8_t data;
00290   int16_t temp;
00291
00292   temp = Blox_USART_TryReceive(IR_4_USART_ID);
00293   if (temp == -1)
00294     return;
00295
```

```
00296   data = (uint8_t) temp;
00297   if (num == 0) {
00298     if (data == 0x7E) {
00299       num = 1; //Start of a frame
00300       checksum = 0;
00301     }
00302   } else if (num == 1) {
00303     frame.src_id = data;
00304     num++;
00305   } else if (num == 2) {
00306     frame.src_face_id = data;
00307     num++;
00308   } else if (num == 3) {
00309     frame.type = (IRFrameType)(data & 0xFF);
00310     num++;
00311   } else if (num == 4) {
00312     frame.len = data;
00313     if (frame.len > IR_MAX_FRAME_LEN) {
00314       num = 0;
00315     } else {
00316       frame.data = (uint8_t *)malloc(frame.len*sizeof(uint8_t));
00317       num++;
00318     }
00319   } else if (num > 4) {
00320     if (num == frame.len+5) {
00321       if(checksum == data) {
00322         if (IR4_RX_Handler != NULL && IR4_RX_Enable == TRUE) {
00323           IRFrame *retFrame;
00324           retFrame = (IRFrame *)malloc(sizeof(IRFrame));
00325           memcpy(retFrame, &(frame), sizeof(IRFrame));
00326           (*IR4_RX_Handler)(retFrame);
00327         } else {
00328           free(frame.data);
00329         }
00330       } else {
00331         free(frame.data);
00332       }
00333       num = 0;
00334     } else {
00335       frame.data[num-5] = data;
00336       checksum ^= data;
00337       num++;
00338     }
00339   }
00340 }
00341
00346 void IR_GPIO_Configuration() {
00347 GPIO_InitTypeDef GPIO_InitStructure;
00348   //Set up Rx as Floating
00349   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
00350   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
00351   GPIO_InitStructure.GPIO_Pin = IR_SHUTDOWN_PIN;
00352   GPIO_Init(IR_SHUTDOWN_GPIO, &GPIO_InitStructure);
00353 }
00354
00361 void Blox_IR_Register_RX_IRQ(uint8_t id, void (*RX_Handler)(IRFrame *frame)) {
00362   switch(id) {
```

```
00363     case 1:
00364       IR1_RX_Handler = RX_Handler;
00365       break;
00366     case 2:
00367       IR2_RX_Handler = RX_Handler;
00368       break;
00369     case 3:
00370       IR3_RX_Handler = RX_Handler;
00371       break;
00372     case 4:
00373       IR4_RX_Handler = RX_Handler;
00374       break;
00375   }
00376 }
00377
00383 void Blox_IR_Enable_RX_IRQ(uint8_t id) {
00384   switch(id) {
00385     case 1:
00386       Blox_USART_Enable_RXNE_IRQ(IR_1_USART_ID);
00387       IR1_RX_Enable = TRUE;
00388       break;
00389     case 2:
00390       Blox_USART_Enable_RXNE_IRQ(IR_2_USART_ID);
00391       IR2_RX_Enable = TRUE;
00392       break;
00393     case 3:
00394       Blox_USART_Enable_RXNE_IRQ(IR_3_USART_ID);
00395       IR3_RX_Enable = TRUE;
00396       break;
00397     case 4:
00398       Blox_USART_Enable_RXNE_IRQ(IR_4_USART_ID);
00399       IR4_RX_Enable = TRUE;
00400       break;
00401   }
00402 }
00403
00409 void Blox_IR_Disable_RX_IRQ(uint8_t id) {
00410   switch(id) {
00411     case 1:
00412       Blox_USART_Disable_RXNE_IRQ(IR_1_USART_ID);
00413       IR1_RX_Enable = FALSE;
00414       break;
00415     case 2:
00416       Blox_USART_Disable_RXNE_IRQ(IR_2_USART_ID);
00417       IR2_RX_Enable = FALSE;
00418       break;
00419     case 3:
00420       Blox_USART_Disable_RXNE_IRQ(IR_3_USART_ID);
00421       IR3_RX_Enable = FALSE;
00422       break;
00423     case 4:
00424       Blox_USART_Disable_RXNE_IRQ(IR_4_USART_ID);
00425       IR4_RX_Enable = FALSE;
00426       break;
00427   }
00428 }
00429
```

```
00435 uint8_t IR_Receive(uint8_t id) {
00436   switch(id) {
00437   case 1:
00438     return Blox_USART_Receive(IR_1_USART_ID);
00439   case 2:
00440     return Blox_USART_Receive(IR_2_USART_ID);
00441   case 3:
00442     return Blox_USART_Receive(IR_3_USART_ID);
00443   case 4:
00444     return Blox_USART_Receive(IR_4_USART_ID);
00445   }
00446   return 0;
00447 }
00448
00454 uint8_t IR_TryReceive(uint8_t id) {
00455   switch(id) {
00456   case 1:
00457     return Blox_USART_TryReceive(IR_1_USART_ID);
00458   case 2:
00459     return Blox_USART_TryReceive(IR_2_USART_ID);
00460   case 3:
00461     return Blox_USART_TryReceive(IR_3_USART_ID);
00462   case 4:
00463     return Blox_USART_TryReceive(IR_4_USART_ID);
00464   }
00465   return 0;
00466 }
00467
00474 void IR_Send(uint8_t id, uint8_t data) {
00475   switch(id) {
00476   case 1:
00477     Blox_USART_Send(IR_1_USART_ID, data);
00478     break;
00479   case 2:
00480     Blox_USART_Send(IR_2_USART_ID, data);
00481     break;
00482   case 3:
00483     Blox_USART_Send(IR_3_USART_ID, data);
00484     break;
00485   case 4:
00486     Blox_USART_Send(IR_4_USART_ID, data);
00487     break;
00488   }
00489 }
00490
00499 void IR_SendFrame(uint8_t id, IRFrameType type, uint8_t *data, uint8_t len) {
00500   uint8_t i;
00501   uint8_t checksum = 0;
00502   switch(id) {
00503     case 1:
00504       Blox_USART_Send(IR_1_USART_ID, 0x7E);
00505       Blox_USART_Send(IR_1_USART_ID, (uint8_t)Blox_System_GetId());
00506       Blox_USART_Send(IR_1_USART_ID, 1);
00507       Blox_USART_Send(IR_1_USART_ID, type);
00508       Blox_USART_Send(IR_1_USART_ID, len);
00509       for(i = 0; i < len; i++) {
00510         Blox_USART_Send(IR_1_USART_ID, data[i]);
```

```
00511         checksum ^= data[i];
00512       }
00513       Blox_USART_Send(IR_1_USART_ID, checksum);
00514       break;
00515     case 2:
00516       Blox_USART_Send(IR_2_USART_ID, 0x7E);
00517       Blox_USART_Send(IR_2_USART_ID, (uint8_t)Blox_System_GetId());
00518       Blox_USART_Send(IR_2_USART_ID, 2);
00519       Blox_USART_Send(IR_2_USART_ID, type);
00520       Blox_USART_Send(IR_2_USART_ID, len);
00521       for(i = 0; i < len; i++) {
00522         Blox_USART_Send(IR_2_USART_ID, data[i]);
00523         checksum ^= data[i];
00524       }
00525       Blox_USART_Send(IR_2_USART_ID, checksum);
00526       break;
00527     case 3:
00528       Blox_USART_Send(IR_3_USART_ID, 0x7E);
00529       Blox_USART_Send(IR_3_USART_ID, (uint8_t)Blox_System_GetId());
00530       Blox_USART_Send(IR_3_USART_ID, 3);
00531       Blox_USART_Send(IR_3_USART_ID, type);
00532       Blox_USART_Send(IR_3_USART_ID, len);
00533       for(i = 0; i < len; i++) {
00534         Blox_USART_Send(IR_3_USART_ID, data[i]);
00535         checksum ^= data[i];
00536       }
00537       Blox_USART_Send(IR_3_USART_ID, checksum);
00538       break;
00539     case 4:
00540       Blox_USART_Send(IR_4_USART_ID, 0x7E);
00541       Blox_USART_Send(IR_4_USART_ID, (uint8_t)Blox_System_GetId());
00542       Blox_USART_Send(IR_4_USART_ID, 4);
00543       Blox_USART_Send(IR_4_USART_ID, type);
00544       Blox_USART_Send(IR_4_USART_ID, len);
00545       for(i = 0; i < len; i++) {
00546         Blox_USART_Send(IR_4_USART_ID, data[i]);
00547         checksum ^= data[i];
00548       }
00549       Blox_USART_Send(IR_4_USART_ID, checksum);
00550       break;
00551   }
00552 }
00553
00558 void IR_Wake(void) {
00559   IR_SHUTDOWN_GPIO->ODR &= ~(1 << IR_SHUTDOWN_PIN_NUM);
00560 }
00561
00566 void IR_Sleep(void) {
00567   IR_SHUTDOWN_GPIO->ODR |= (1 << IR_SHUTDOWN_PIN_NUM);
00568 }
00569
```

6.49 drivers/src/blox_led.c File Reference

A driver for the Blox LEDs.

```
#include "blox_led.h"
```

Functions

- void **Blox_LED_DeInit_GPIO** (void)
  Deinitializes the LEDs.
- void **Blox_LED_RCC_Configuration** (void)
  Initializes the clocks for the LEDs.
- void **Blox_LED_GPIO_Configuration** (void)
  Initializes the GPIOs for the LEDs.
- void **Blox_LED_Init** (void)
  Initializes the LEDs.
- void **Blox_LED_On** (**LED_ID** id)
  Turns an LED on.
- void **Blox_LED_Off** (**LED_ID** id)
  Turns an LED off.
- void **Blox_LED_Toggle** (**LED_ID** id)
  Toggles an LED.

6.49.1 Detailed Description

A driver for the Blox LEDs.

Author

Zach Wasson

Version

V0.1

Date

10/30/2010

Definition in file **blox_led.c**.

6.49.2 Function Documentation

6.49.2.1 void Blox_LED_DeInit_GPIO (void)

Deinitializes the LEDs.

Return values

None	

Definition at line **34** of file **blox_led.c**.

6.49.2.2 void Blox_LED_GPIO_Configuration (void)

Initializes the GPIOs for the LEDs.

Return values

None	

Definition at line **101** of file **blox_led.c**.

6.49.2.3 void Blox_LED_Init (void)

Initializes the LEDs.

Return values

None	

Definition at line **22** of file **blox_led.c**.

6.49.2.4 void Blox_LED_Off (LED_ID *id*)

Turns an LED off.

Parameters

id	the id of the LED

Return values

None	

Definition at line **60** of file **blox_led.c**.

6.49.2.5 void Blox_LED_On (LED_ID *id*)

Turns an LED on.

Parameters

id	the id of the LED

Return values

None	

Definition at line **43** of file **blox_led.c**.

6.49.2.6 void Blox_LED_RCC_Configuration (void)

Initializes the clocks for the LEDs.

Return values

None	

Definition at line **93** of file **blox_led.c**.

6.49.2.7 void Blox_LED_Toggle (LED_ID *id*)

Toggles an LED.

Parameters

id	the id of the LED

Return values

None	

Definition at line **77** of file **blox_led.c**.

6.50 drivers/src/blox_led.c

```
00001
00008 #include "blox_led.h"
00009
00010 void Blox_LED_DeInit_GPIO(void);
00011 void Blox_LED_RCC_Configuration(void);
```

```
00012 void Blox_LED_GPIO_Configuration(void);
00013
00022 void Blox_LED_Init(void) {
00023   Blox_LED_RCC_Configuration();
00024   Blox_LED_GPIO_Configuration();
00025
00026   Blox_System_Register_DeInit(&RCC_DeInit);
00027   Blox_System_Register_DeInit(&Blox_LED_DeInit_GPIO);
00028 }
00029
00034 void Blox_LED_DeInit_GPIO(void) {
00035   GPIO_DeInit(LED_GPIO);
00036 }
00037
00043 void Blox_LED_On(LED_ID id) {
00044   if(id == 1) {
00045     LED_GPIO->ODR |= LED1_GPIO_Pin;
00046   } else if(id == 2) {
00047     LED_GPIO->ODR |= LED2_GPIO_Pin;
00048   } else if(id == 3) {
00049     LED_GPIO->ODR |= LED3_GPIO_Pin;
00050   } else if(id == 4) {
00051     LED_GPIO->ODR |= LED4_GPIO_Pin;
00052   }
00053 }
00054
00060 void Blox_LED_Off(LED_ID id) {
00061   if(id == 1) {
00062     LED_GPIO->ODR &= ~LED1_GPIO_Pin;
00063   } else if(id == 2) {
00064     LED_GPIO->ODR &= ~LED2_GPIO_Pin;
00065   } else if(id == 3) {
00066     LED_GPIO->ODR &= ~LED3_GPIO_Pin;
00067   } else if(id == 4) {
00068     LED_GPIO->ODR &= ~LED4_GPIO_Pin;
00069   }
00070 }
00071
00077 void Blox_LED_Toggle(LED_ID id) {
00078   if(id == 1) {
00079     LED_GPIO->ODR ^= LED1_GPIO_Pin;
00080   } else if(id == 2) {
00081     LED_GPIO->ODR ^= LED2_GPIO_Pin;
00082   } else if(id == 3) {
00083     LED_GPIO->ODR ^= LED3_GPIO_Pin;
00084   } else if(id == 4) {
00085     LED_GPIO->ODR ^= LED4_GPIO_Pin;
00086   }
00087 }
00088
00093 void Blox_LED_RCC_Configuration(void) {
00094   RCC_APB2PeriphClockCmd(LED_CLK, ENABLE);
00095 }
00096
00101 void Blox_LED_GPIO_Configuration(void) {
00102   GPIO_InitTypeDef GPIO_InitStructure;
00103   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
```

```
00104   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
00105   GPIO_InitStructure.GPIO_Pin = LED1_GPIO_Pin | LED2_GPIO_Pin | LED3_GPIO_Pin | L
      ED4_GPIO_Pin;
00106   GPIO_Init(LED_GPIO, &GPIO_InitStructure);
00107 }
```

6.51 drivers/src/blox_oled.c File Reference

Driver for Blox OLED screen.

```
#include "blox_oled.h"
```

Functions

- void **OLED_RCC_Configuration** ()

  Initializes clocks for OLED reset pin.

- void **OLED_GPIO_Configuration** ()

  Initializes the gpio for the OLED reset pin.

- void **OLED_ResetDisplay** (void)
- void **OLED_Reset** (void)

  Switches the Reset pin PC3 for the OLED display.

- void **Blox_OLED_Init** (void)

  Initializes the OLED display.

- uint8_t **Blox_OLED_Receive** (void)

  Receive a byte from the OLED. Wrapper around USART.

- void **Blox_OLED_Send** (uint8_t data)

  Send a byte to the OLED. Wrapper around USART.

- void **Blox_OLED_Clear** (void)

  Clear OLED display.

- void **Blox_OLED_DrawCircle** (uint8_t x, uint8_t y, uint8_t radius, uint16_t color)

  Draw circle on OLED display.

- void **Blox_OLED_DrawLine** (uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, uint16_t color)

  Draw line on OLED display.

- void **Blox_OLED_DrawPixel** (uint8_t x, uint8_t y, uint16_t color)

  Draw pixel on OLED display.

- void **Blox_OLED_DrawRectangle** (uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, uint16_t color)

  Draw line on OLED display.

- void **Blox_OLED_SetFont** (uint8_t font)

  Sets font type for OLED characters.

- void **Blox_OLED_SetOpaque** (void)

  Sets the font to have an opaque background.

- void **Blox_OLED_DrawStringGraphics** (uint8_t x, uint8_t y, uint8_t font, uint16_t color, uint8_t width, uint8_t height, uint8_t *string)

  Draw graphics formatted string on OLED display.

- void **Blox_OLED_DrawCharText** (uint8_t character, uint8_t column, uint8_t row, uint16_t color)

  Draw text formatted character on OLED display.

- void **Blox_OLED_DrawStringText** (uint8_t column, uint8_t row, uint8_t font, uint16_t color, uint8_t *string)

  Draw text formatted string on OLED display.

- void **Blox_OLED_DrawCharGraphics** (uint8_t character, uint8_t x, uint8_t y, uint16_t color, uint8_t width, uint8_t height)

  Draw graphics formatted character on OLED display.

- void **Blox_OLED_SD_DisplayIcon** (uint8_t x, uint8_t y, uint8_t width, uint8_t height, uint32_t sector)

  Displays bitmap image icon stored in SD card onto OLED screen.

6.51.1 Detailed Description

Driver for Blox OLED screen.

Author

Dan Cleary

Version

V0.1

Date

10/20/2010

Definition in file **blox_oled.c**.

6.52 drivers/src/blox_oled.c

```
00001
00009 #include "blox_oled.h"
00010
00016 /* Private function prototypes */
00017 void OLED_RCC_Configuration(void);
00018 void OLED_GPIO_Configuration(void);
00019 void OLED_ResetDisplay(void);
00020
00025 void OLED_Reset(void) {
00026   GPIOC->ODR &= ~(1<<3);
00027   //TODO: insert timer wait 200ms
00028   SysTick_Wait(20);
00029   GPIOC->ODR |= (1<<3);
00030   SysTick_Wait(20);
00031 }
00032
00037 void Blox_OLED_Init(void) {
00038   uint8_t garbage;
00039   OLED_RCC_Configuration();
00040   OLED_GPIO_Configuration();
00041   SysTick_Init();
00042   Blox_VUSART_Init(OLED_USART_ID);
00043
00044   OLED_Reset();
00045   SysTick_Wait(2000);
00046   Blox_VUSART_TryReceive(2, &garbage);  //receive any garbage data
00047   Blox_OLED_Send(OLED_AUTOBAUD);
00048   Blox_OLED_Receive();
00049 }
00050
00055 void OLED_RCC_Configuration() {
00056     RCC_APB2PeriphClockCmd(OLED_RESET_GPIO_CLK, ENABLE);
00057 }
00058
00063 void OLED_GPIO_Configuration() {
00064   GPIO_InitTypeDef GPIO_InitStructure;
00065
00066   //OLED Reset is push-pull, 50Mz
00067   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
00068   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
00069   GPIO_InitStructure.GPIO_Pin = OLED_RESET_PIN;
00070   GPIO_Init(OLED_RESET_GPIO, &GPIO_InitStructure);
```

```
00071 }
00072
00077 uint8_t Blox_OLED_Receive(void) {
00078   uint8_t data;
00079   while(Blox_VUSART_TryReceive(2, &data) == RX_EMPTY);
00080   return data;
00081 }
00082
00088 void Blox_OLED_Send(uint8_t data) {
00089   while(Blox_VUSART_TrySend(2, data) == TX_BUSY);
00090 }
00091
00092 /*****************************************/
00093 /*            General Commands           */
00094 /*****************************************/
00095
00100 void Blox_OLED_Clear(void){
00101   Blox_OLED_Send(OLED_CLEAR); // Pixel write
00102   Blox_OLED_Receive();
00103 }
00104
00105 /*****************************************/
00106 /*            Graphics Commands          */
00107 /*****************************************/
00108
00117 void Blox_OLED_DrawCircle(uint8_t x, uint8_t y, uint8_t radius, uint16_t color){
00118   Blox_OLED_Send(OLED_DRAW_CIRCLE);
00119         Blox_OLED_Send(x);
00120         Blox_OLED_Send(y);
00121         Blox_OLED_Send(radius);
00122         Blox_OLED_Send(color >> 8);
00123         Blox_OLED_Send(color & 0xFF);
00124   Blox_OLED_Receive();
00125 }
00126
00136 void Blox_OLED_DrawLine(uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, uint16_t
      color){
00137   Blox_OLED_Send(OLED_DRAW_LINE);
00138         Blox_OLED_Send(x1);
00139         Blox_OLED_Send(y1);
00140         Blox_OLED_Send(x2);
00141         Blox_OLED_Send(y2);
00142         Blox_OLED_Send(color >> 8);
00143         Blox_OLED_Send(color & 0xFF);
00144   Blox_OLED_Receive();
00145 }
00146
00154 void Blox_OLED_DrawPixel(uint8_t x, uint8_t y, uint16_t color){
00155   Blox_OLED_Send(OLED_DRAW_PIXEL);
00156         Blox_OLED_Send(x);
00157         Blox_OLED_Send(y);
00158         Blox_OLED_Send(color >> 8);
00159         Blox_OLED_Send(color & 0xFF);
00160   Blox_OLED_Receive();
00161 }
00162
00172 void Blox_OLED_DrawRectangle(uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, uint
```

```
      16_t color){
00173   Blox_OLED_Send(OLED_DRAW_RECT);
00174         Blox_OLED_Send(x1);
00175         Blox_OLED_Send(y1);
00176         Blox_OLED_Send(x2);
00177         Blox_OLED_Send(y2);
00178         Blox_OLED_Send(color >> 8);
00179         Blox_OLED_Send(color & 0xFF);
00180   Blox_OLED_Receive();
00181 }
00182
00183 /*****************************************/
00184 /*            Text Commands              */
00185 /*****************************************/
00186
00195 void Blox_OLED_SetFont(uint8_t font){
00196   Blox_OLED_Send(OLED_SET_FONT);
00197         Blox_OLED_Send(font);
00198   Blox_OLED_Receive();
00199 }
00200
00205 void Blox_OLED_SetOpaque(void){
00206   Blox_OLED_Send(OLED_SET_VIS);
00207   Blox_OLED_Send(OLED_SET_VIS_OPAQ);
00208   Blox_OLED_Receive();
00209 }
00210
00226 void Blox_OLED_DrawStringGraphics(uint8_t x, uint8_t y, uint8_t font, uint16_t co
      lor, uint8_t width, uint8_t height, uint8_t *string){
00227   uint8_t *pt;
00228
00229   Blox_OLED_Send(OLED_DRAW_STRING_GRAPHICS);
00230         Blox_OLED_Send(x);
00231         Blox_OLED_Send(y);
00232         Blox_OLED_Send(font);
00233         Blox_OLED_Send(color >> 8);
00234         Blox_OLED_Send(color & 0xFF);
00235         Blox_OLED_Send(width);
00236         Blox_OLED_Send(height);
00237
00238   pt = string;
00239   while (*pt)
00240         {
00241                 Blox_OLED_Send(*pt);
00242     pt++;
00243         }
00244
00245         Blox_OLED_Send(0x00); //string terminator
00246   Blox_OLED_Receive();
00247 }
00248
00261 void Blox_OLED_DrawCharText(uint8_t character, uint8_t column, uint8_t row, uint1
      6_t color){
00262   Blox_OLED_Send(OLED_DRAW_CHAR_TEXT);
00263         Blox_OLED_Send(character);
00264         Blox_OLED_Send(column);
00265         Blox_OLED_Send(row);
```

```
00266         Blox_OLED_Send(color >> 8);
00267         Blox_OLED_Send(color & 0xFF);
00268   Blox_OLED_Receive();
00269 }
00270
00287 void Blox_OLED_DrawStringText(uint8_t column, uint8_t row, uint8_t font, uint16_t
       color, uint8_t *string){
00288   uint8_t *pt;
00289
00290   Blox_OLED_Send(OLED_DRAW_STRING_TEXT);
00291         Blox_OLED_Send(column);
00292         Blox_OLED_Send(row);
00293         Blox_OLED_Send(font);
00294         Blox_OLED_Send(color >> 8);
00295         Blox_OLED_Send(color & 0xFF);
00296
00297   pt = string;
00298   while (*pt)
00299         {
00300                 Blox_OLED_Send(*pt);
00301     pt++;
00302         }
00303
00304         Blox_OLED_Send(0x00); //string terminator
00305   Blox_OLED_Receive();
00306 }
00307
00318 void Blox_OLED_DrawCharGraphics(uint8_t character, uint8_t x, uint8_t y, uint16_t
       color, uint8_t width, uint8_t height){
00319   Blox_OLED_Send(OLED_DRAW_CHAR_GRAPHICS);
00320         Blox_OLED_Send(character);
00321         Blox_OLED_Send(x);
00322         Blox_OLED_Send(y);
00323         Blox_OLED_Send(color >> 8);
00324         Blox_OLED_Send(color & 0xFF);
00325         Blox_OLED_Send(width);
00326         Blox_OLED_Send(height);
00327   Blox_OLED_Receive();
00328 }
00329
00330 /*****************************************/
00331 /*            SD Commands                */
00332 /*****************************************/
00333
00343 void Blox_OLED_SD_DisplayIcon(uint8_t x, uint8_t y, uint8_t width, uint8_t height
      , uint32_t sector){
00344   Blox_OLED_Send(0x40);
00345   Blox_OLED_Send(0x49);
00346   Blox_OLED_Send(x);
00347   Blox_OLED_Send(y);
00348   Blox_OLED_Send(width);
00349   Blox_OLED_Send(height);
00350   Blox_OLED_Send(8);  //color mode of all gifs are 65k
00351
00352   Blox_OLED_Send((sector >> 16) & 0x0FF);
00353   Blox_OLED_Send((sector >> 8) & 0x0FF);
00354   Blox_OLED_Send(sector & 0x0FF);
```

```
00355   Blox_OLED_Receive();
00356 }
```

6.53 drivers/src/blox_speaker.c File Reference

Device driver for the speaker.

```
#include <stdio.h>
#include <blox_system.h>
#include <stm32f10x_dac.h>
#include <stm32f10x_tim.h>
#include <stm32f10x_gpio.h>
#include <misc.h>
#include "blox_tim.h"
#include "blox_speaker.h"
#include "blox_music.h"
```

- DAC_InitTypeDef **DAC_InitStructure**
- uint16_t **CCR1_Val** = 0
- uint16_t **CCR2_Val** = 0
- uint16_t **attack1**
- uint16_t **attack2**
- uint16_t **decay1**
- uint16_t **decay2** = 0
- **TIMER_ID score1ID**
- **TIMER_ID score2ID**
- **TIMER_ID note1ID**
- **TIMER_ID note2ID**
- **NotePtr score1note**
- **NotePtr score2note**
- static unsigned short **durCount2** = 0
- unsigned short **I0**
- unsigned short **I1**
- unsigned short **Out**
- unsigned short **Out2** = 0
- const unsigned short **wave** [32]
- void **Sin_gen** (void)

Generates the sine wave for the treble clef.

- void **Sin_gen2** (void)

  Generates the sine wave for the bass clef.

- void **NoteHandler** (void)

  Plays each note for its specified duration.

- void **EnvelopeGen** (void)

  Creates an envelope around the output waveform.

- void **Blox_Speaker_Init** (void)

  Initializes the speaker.

- void **PlayMusic** (void)

  Plays the music included in blox_music.h.

- void **StopMusic** (void)

  Stops the music.

6.53.1 Detailed Description

Device driver for the speaker.

Author

Ankita Kaul

Version

V0.1

Date

10/19/2010

Definition in file **blox_speaker.c**.

6.53.2 Function Documentation

6.53.2.1 void Blox_Speaker_Init (void)

Initializes the speaker.

Return values

None	

Definition at line **51** of file **blox_speaker.c**.

6.53.2.2 void EnvelopeGen (void)

Creates an envelope around the output waveform.

Return values

None	

Definition at line **139** of file **blox_speaker.c**.

6.53.2.3 void NoteHandler (void)

Plays each note for its specified duration.

Return values

None	

Definition at line **170** of file **blox_speaker.c**.

6.53.2.4 void PlayMusic (void)

Plays the music included in blox_music.h.

Return values

None	

Definition at line **73** of file **blox_speaker.c**.

6.53.2.5 void Sin_gen2 (void)

Generates the sine wave for the bass clef.

Return values

None	

Definition at line **123** of file **blox_speaker.c**.

6.53.2.6 void StopMusic (void)

Stops the music.

Return values

None	

Definition at line **94** of file **blox_speaker.c**.

6.53.3 Variable Documentation

6.53.3.1 const unsigned short wave[32]

Initial value:

```
 {
  1024,1200,1368,1524,1660,1772,1855,1907,1924,1907,1855,
  1772,1660,1524,1368,1200,1024,848,680,524,388,276,
  193,141,124,141,193,276,388,524,680,848
}
```

Definition at line **41** of file **blox_speaker.c**.

6.54 drivers/src/blox_speaker.c

```
00001
00009 #include <stdio.h>
00010 #include <blox_system.h>
00011 #include <stm32f10x_dac.h>
00012 #include <stm32f10x_tim.h>
00013 #include <stm32f10x_gpio.h>
00014 #include <misc.h>
00015 #include "blox_tim.h"
00016 #include "blox_speaker.h"
00017 #include "blox_music.h"
00018
00023 //private functions//
00024 void Sin_gen(void);
00025 void Sin_gen2(void);
00026 void NoteHandler(void);
00027 void EnvelopeGen(void);
00028
00029 DAC_InitTypeDef DAC_InitStructure;
00030 uint16_t CCR1_Val = 0;  //interrupt dependent on freq of note to be played
```

```
00031 uint16_t CCR2_Val = 0; // "  "
00032 uint16_t attack1, attack2, decay1, decay2 =0;
00033 TIMER_ID score1ID, score2ID, note1ID, note2ID;
00034 NotePtr score1note, score2note;
00035 static unsigned short durCount, durCount2=0;
00036 unsigned short I0,I1, Out, Out2 = 0;
00037
00038 static uint8_t VolDiv = 1;
00039
00040 // 12-bit 32-element sine wave
00041 const unsigned short wave[32] = {
00042   1024,1200,1368,1524,1660,1772,1855,1907,1924,1907,1855,
00043   1772,1660,1524,1368,1200,1024,848,680,524,388,276,
00044   193,141,124,141,193,276,388,524,680,848
00045 };
00046
00051 void Blox_Speaker_Init(void)
00052 {
00053   GPIO_InitTypeDef GPIO_InitStructure;
00054
00055   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
00056   RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
00057
00058   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
00059   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
00060   GPIO_Init(GPIOA, &GPIO_InitStructure);
00061
00062   DAC_InitStructure.DAC_Trigger = DAC_Trigger_None;
00063   DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None;
00064   DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
00065   DAC_Init(DAC_Channel_1, &DAC_InitStructure);
00066   DAC_Cmd(DAC_Channel_1, ENABLE);
00067 }
00068
00073 void PlayMusic(void)
00074 {
00075   CCR1_Val = score[0].noteName;
00076   CCR2_Val = score2[0].noteName;
00077
00078   Blox_Timer_Init(4, SpkClock);
00079   score1ID = Blox_Timer_Register_IRQ(4, CCR1_Val, &Sin_gen, ENABLE);
00080   score2ID = Blox_Timer_Register_IRQ(4, CCR2_Val, &Sin_gen2, ENABLE);
00081   note1ID = Blox_Timer_Register_IRQ(4, BEAT, &NoteHandler, ENABLE);
00082   note2ID = Blox_Timer_Register_IRQ(4, BEAT*4, &EnvelopeGen, ENABLE);
00083
00084   score1note = &score[0]; //set score to first note
00085   score2note = &score2[0]; //set score to first note
00086
00087   return;
00088 }
00089
00094 void StopMusic(void)
00095 {
00096   score1note = &score[0]; //reset song if it has reached the end
00097   score2note = &score2[0];
00098   CCR1_Val = 0;
00099   CCR2_Val = 0;
```

```
00100   TIM_Cmd(TIM4, DISABLE); //disable all interrupts
00101 }
00102
00107 void Sin_gen(void){
00108         if(score1note->noteName==0 && (Out==1024)) //check to see if at a rest
00109                 Out = 1024;
00110         else
00111          {
00112                  I0 = (I0+1)&0x1F;       // 0 to 31 - for 1024 point sine wave
00113                  Out = wave[I0];
00114          }
00115
00116          DAC_SetChannel1Data(DAC_Align_12b_R, (Out+Out2)/VolDiv);
00117 }
00118
00123 void Sin_gen2(void){
00124         if(score2note->noteName==0 && (Out2==1024)) //check to see if at a rest
00125                 Out2 = 1024;   //silence
00126         else
00127          {
00128                  I1 = (I1+1)&0x1F;       // 0 to 31 - for 32 point sine wave
00129                  Out2 = wave[I1];
00130          }
00131
00132          DAC_SetChannel1Data(DAC_Align_12b_R, (Out+Out2)/VolDiv);
00133 }
00134
00139 void EnvelopeGen(void)
00140 {
00141                 if(attack1 < CCR1_Val)  //attack, sine envelope
00142                   {
00143                         attack1 = attack1 + (CCR1_Val/4) ;
00144                         Blox_Timer_Modify_IRQ(score1ID, attack1);
00145                   }
00146
00147                  else if((attack1 >= CCR2_Val) && decay1 >= 100)  //decay, sine
00148                   {
00149                         decay1 = decay1/2;
00150                         Blox_Timer_Modify_IRQ(score1ID, decay1);
00151                   }
00152
00153                  if(attack2 < CCR2_Val)  //attack, sine envelope
00154                   {
00155                         attack2 = attack2 + (CCR2_Val/4) ;
00156                         Blox_Timer_Modify_IRQ(score2ID, attack2);
00157                   }
00158
00159                  else if((attack2 >= CCR2_Val) && decay2 >= 100)  //decay, sine
00160                   {
00161                         decay2 = decay2/2;
00162                         Blox_Timer_Modify_IRQ(score2ID, decay2);
00163                   }
00164 }
00165
00170 void NoteHandler(void){
00171
00172                 durCount++;  //increment duration counter
```

```
00173                 durCount2++;
00174
00175                 if(score1note->duration <= durCount)  //see if note has completed
      its duration
00176                   {
00177                     durCount = 0;
00178                     score1note++;       //go to next note
00179                         CCR1_Val = score1note->noteName;
00180                         Blox_Timer_Modify_IRQ(score1ID, CCR1_Val);
00181                         attack1 = 0;
00182                         decay1 = CCR1_Val;
00183                         Out = 1024;
00184                   }
00185
00186                 if(score2note->duration <= durCount2)  //see if note has complete
      d its duration
00187                   {
00188                     durCount2 = 0;
00189                     score2note++;       //go to next note
00190                         CCR2_Val = score2note->noteName;
00191                         Blox_Timer_Modify_IRQ(score2ID, CCR2_Val);
00192                         attack2 = 0;
00193                         decay2 = CCR1_Val;
00194                     Out2 = 1024;
00195                   }
00196
00197                 if((score1note->duration==0) ||( score2note->duration==0))
00198                   {
00199                     StopMusic();       //stop if score reaches end (should occur
      at the same time)
00200                   }
00201 }
```

6.55 drivers/src/blox_system.c File Reference

Defines system-wide concepts including the memory map, and deinitialization.

```
#include "blox_system.h"
#include "blox_filesystem.h"
```

- **SysVar** * **sys** = (**SysVar** *)MEM_SYS_VAR_START
- void **Blox_System_Init** (void)

  Initializes the pointer to the system variables array.

- void **Blox_System_DeInit** (void)

  De-initializes all the peripherals in the system.

- void **Blox_System_Register_DeInit** (ptrVoidFn fn)

  Registers a new function to be called when the system deInits. Only adds if it isn't already there.

- void **Blox_System_Create** (void)

  *Creates an initial **SysVar** (p. 110) struct.*

- uint32_t **Blox_System_GetId** (void)

  *Returns the current **SysVar** (p. 110) struct.*

- void **Blox_System_GetVars** (**SysVar** *retSys)

  *Returns the current **SysVar** (p. 110) struct.*

- void **Blox_System_WriteVars** (**SysVar** *newVars)

  *Writes over the **SysVar** (p. 110) in flash.*

6.55.1 Detailed Description

Defines system-wide concepts including the memory map, and deinitialization.

Author

Jesse Tannahill

Version

V0.1

Date

10/19/2010

Definition in file **blox_system.c**.

6.56 drivers/src/blox_system.c

```
00001
00010 #include "blox_system.h"
00011 #include "blox_filesystem.h"
00016 SysVar *sys = (SysVar *)MEM_SYS_VAR_START;;
00017 static ptrVoidFn deInit[MAX_DEINIT_FN];
00018 static uint32_t numDeInit;
00019
00020
```

```
00025 void Blox_System_Init(void) {
00026   Blox_Debug_Init();
00027   //sys = (SysVar *)MEM_SYS_VAR_START;
00028   if(sys->magic != SYS_MAGIC) {
00029     Blox_DebugPat("Blox_System_Init sys magic fails, got:%x\r\n", sys->magic);
00030     while(1) ;
00031   }
00032
00033   numDeInit = 0;
00034 }
00035
00040 void Blox_System_DeInit(void) {
00041   uint32_t i;
00042   for (i = 0; i < numDeInit; i++)
00043     deInit[i]();
00044 }
00045
00052 void Blox_System_Register_DeInit(ptrVoidFn fn) {
00053   uint32_t i;
00054   if (numDeInit == MAX_DEINIT_FN) {
00055     Blox_DebugPat("Blox_System_Register_DeInit numDeInit reached max of:%d\r\n",
00056         MAX_DEINIT_FN);
00057     while(1) ;
00058   }
00059
00060   for (i = 0; i < numDeInit; i++) {
00061     if(deInit[i] == fn)
00062       return;
00063   }
00064
00065   deInit[numDeInit++] = fn;
00066 }
00067
00072 void Blox_System_Create(void) {
00073   SysVar *sys_new = (SysVar *)malloc(PAGE_SIZE);
00074   sys_new->magic = SYS_MAGIC;
00075   sys_new->id = SYS_INV_ID;
00076   sys_new->ACCEL_X = 0;
00077   sys_new->ACCEL_Y = 0;
00078   sys_new->ACCEL_Z = 0;
00079   sys_new->TOUCH_1_X = 0;
00080   sys_new->TOUCH_1_Y = 0;
00081   sys_new->TOUCH_2_X = 0;
00082   sys_new->TOUCH_2_Y = 0;
00083   sys_new->TOUCH_3_X = 0;
00084   sys_new->TOUCH_3_Y = 0;
00085   sys_new->TOUCH_4_X = 0;
00086   sys_new->TOUCH_4_Y = 0;
00087
00088   sys = (SysVar *)MEM_SYS_VAR_START;
00089   FS_SwapPage((uint32_t *)sys_new, (uint32_t *)sys);
00090   free(sys_new);
00091 }
00092
00097 uint32_t Blox_System_GetId(void) {
00098   return sys->id;
00099 }
```

```
00100
00105 void Blox_System_GetVars(SysVar *retSys) {
00106   memcpy(retSys, sys, sizeof(SysVar));
00107 }
00108
00114 void Blox_System_WriteVars(SysVar *newVars) {
00115   FS_SwapPage((uint32_t *)newVars, (uint32_t *)sys);
00116 }
```

6.57 drivers/src/blox_tim.c File Reference

A basic wrapper around the timers on the STM32F103.

```
#include "blox_tim.h"
```

- uint16_t **TIM_PSC** [8]
- void(∗ **TIM_Handler** [28])(void)
- uint16_t **TIM_IRQ_period** [28]
- void **Blox_Timer_DeInit_Timer** (void)

  De-initializes the Timers for all the Timer interfaces.

- void **Blox_Timer_RCC_Configuration** (uint8_t TIMx)

  Initializes clocks for the given timer.

- void **Blox_Timer_NVIC_Configuration** (uint8_t TIMx)

  Initializes interrupts for the given timer.

- **TIMER_ID Timer_OC_IRQ_Configuration** (uint8_t TIMx, uint16_t period, void(∗Timer_-Handler)(void), FunctionalState NewState)

  Registers an output compare interrupt for a given timer.

- **TIMER_ID Timer_UP_IRQ_Configuration** (uint8_t TIMx, uint16_t period, void(∗Timer_-Handler)(void), FunctionalState NewState)

  Registers an update interrupt for a given timer.

- void **Blox_Timer_Init** (uint8_t TIMx, uint32_t TIM_CLK)

  Initializes Timer.

- **TIMER_ID Blox_Timer_Register_IRQ** (uint8_t TIMx, uint16_t period, void(∗Timer_-Handler)(void), FunctionalState NewState)

  Registers a timer interrupt for a given timer.

- void **Blox_Timer_Release_IRQ** (**TIMER_ID** id)

  Releases a given timer interrupt.

- void **Blox_Timer_Enable_IRQ** (**TIMER_ID** id)

  Enables interrupts for a given interrupt.

- void **Blox_Timer_Disable_IRQ** (**TIMER_ID** id)

  Disables interrupts for a given interrupt.

- void **Blox_Timer_Modify_IRQ** (**TIMER_ID** id, uint16_t period)

  Modifies the period for a given output compare interrupt.

- void **TIM1_CC_IRQHandler** (void)

  This function handles timer 1 capture/compare interrupt request.

- void **TIM2_IRQHandler** (void)

  This function handles timer 2 interrupt request.

- void **TIM3_IRQHandler** (void)

  This function handles timer 3 interrupt request.

- void **TIM4_IRQHandler** (void)

  This function handles timer 4 interrupt request.

- void **TIM5_IRQHandler** (void)

  This function handles timer 5 interrupt request.

- void **TIM6_IRQHandler** (void)

  This function handles timer 5 interrupt request.

- void **TIM7_IRQHandler** (void)

  This function handles timer 5 interrupt request.

- void **TIM8_CC_IRQHandler** (void)

  This function handles timer 8 capture/compare interrupt request.

6.57.1 Detailed Description

A basic wrapper around the timers on the STM32F103.

Author

Zach Wasson

Version

V0.1

Date

10/20/2010

Definition in file **blox_tim.c**.

6.57.2 Function Documentation

6.57.2.1 void Blox_Timer_DeInit_Timer (void)

De-initializes the Timers for all the Timer interfaces.

Return values

None	

Definition at line **86** of file **blox_tim.c**.

6.57.2.2 void Blox_Timer_Disable_IRQ (TIMER_ID *id*)

Disables interrupts for a given interrupt.

Parameters

id	specifies the id of the timer interrupt

Return values

None	

Definition at line **686** of file **blox_tim.c**.

6.57.2.3 void Blox_Timer_Enable_IRQ (TIMER_ID *id*)

Enables interrupts for a given interrupt.

Parameters

id	specifies the id of the timer interrupt

Return values

None	

Definition at line **537** of file **blox_tim.c**.

6.57.2.4 void Blox_Timer_Init (uint8_t *TIMx,* uint32_t *TIM_CLK*)

Initializes Timer.

Parameters

TIMx	where x can be (1...8) to select the timer.
TIM_CLK	specifies the TIM_CLK to set for the given timer where TIM_CLK can be in the range of 1.1kHz to 72MHz TIM_CLK = SYS_CLK / (PSC + 1)

Return values

None	

Definition at line **42** of file **blox_tim.c**.

6.57.2.5 void Blox_Timer_Modify_IRQ (TIMER_ID *id,* uint16_t *period*)

Modifies the period for a given output compare interrupt.

Parameters

id	specifies the id of the timer interrupt
period	specifies the new period between timer interrupts.

Return values

None	

Definition at line **781** of file **blox_tim.c**.

6.57.2.6 void Blox_Timer_NVIC_Configuration (uint8_t *TIMx*)

Initializes interrupts for the given timer.

Parameters

TIMx	where x can be (1...8) to select the timer.

Return values

None	

Definition at line **136** of file **blox_tim.c**.

6.57.2.7 void Blox_Timer_RCC_Configuration (uint8_t *TIMx*)

Initializes clocks for the given timer.

Parameters

TIMx	where x can be (1...8) to select the timer.

Return values

None	

Definition at line **102** of file **blox_tim.c**.

6.57.2.8 TIMER_ID Blox_Timer_Register_IRQ (uint8_t *TIMx,* uint16_t *period,* void(∗)(void) *Timer_Handler,* FunctionalState *NewState*)

Registers a timer interrupt for a given timer.

Parameters

TIMx	where x can be (1...8) to select the timer.
period	specifies the period between timer interrupts.
Timer_Handler	the handler function for the timer interrupt.
NewState	new state of the timer interrupt. ENABLE or DISABLE

Return values

the	id for the given interrupt or error

Definition at line **183** of file **blox_tim.c**.

6.57.2.9 void Blox_Timer_Release_IRQ (TIMER_ID *id*)

Releases a given timer interrupt.

Parameters

id	specifies the id of the timer interrupt

Return values

None	

Definition at line **526** of file **blox_tim.c**.

6.57.2.10 void TIM1_CC_IRQHandler (void)

This function handles timer 1 capture/compare interrupt request.

Return values

None	

Definition at line **789** of file **blox_tim.c**.

6.57.2.11 void TIM2_IRQHandler (void)

This function handles timer 2 interrupt request.

Return values

None	

Definition at line **824** of file **blox_tim.c**.

6.57.2.12 void TIM3_IRQHandler (void)

This function handles timer 3 interrupt request.

Return values

None	

Definition at line **859** of file **blox_tim.c**.

6.57.2.13 void TIM4_IRQHandler (void)

This function handles timer 4 interrupt request.

Return values

None	

Definition at line **894** of file **blox_tim.c**.

6.57.2.14 void TIM5_IRQHandler (void)

This function handles timer 5 interrupt request.

Return values

None	

Definition at line **929** of file **blox_tim.c**.

6.57.2.15 void TIM6_IRQHandler (void)

This function handles timer 5 interrupt request.

Return values

None	

Definition at line **964** of file **blox_tim.c**.

6.57.2.16 void TIM7_IRQHandler (void)

This function handles timer 5 interrupt request.

Return values

None	

Definition at line **977** of file **blox_tim.c**.

6.57.2.17 void TIM8_CC_IRQHandler (void)

This function handles timer 8 capture/compare interrupt request.

Return values

None	

Definition at line **990** of file **blox_tim.c**.

6.57.2.18 TIMER_ID Timer_OC_IRQ_Configuration (uint8_t *TIMx,* uint16_t *period,* void(*)(void) *Timer_Handler,* FunctionalState *NewState*)

Registers an output compare interrupt for a given timer.

Parameters

TIMx	where x can be (1,2,3,4,5,8) to select the timer.
period	specifies the period between timer interrupts.
Timer_Handler	the handler function for the timer interrupt.
NewState	new state of the timer interrupt. ENABLE or DISABLE

Return values

the	id for the given interrupt or error

Definition at line **210** of file **blox_tim.c**.

6.57.2.19 TIMER_ID Timer_UP_IRQ_Configuration (uint8_t *TIMx,* uint16_t *period,* void(*)(void) *Timer_Handler,* FunctionalState *NewState*)

Registers an update interrupt for a given timer.

Parameters

TIMx	where x can be (6,7) to select the timer.
period	specifies the period between timer interrupts.
Timer_Handler	the handler function for the timer interrupt.
NewState	new state of the timer interrupt. ENABLE or DISABLE

Return values

the	id for the given interrupt or error

Definition at line **485** of file **blox_tim.c**.

6.58 drivers/src/blox_tim.c

```
00001
00010 /* TO DO:
00011   add functionality to warn or block users from using Timer_Init on an initialize
      d timer
00012   check if range for TIM1 and TIM8 is less than counter value used for ARR
00013   TIM1 and TIM8 UP IRQ
```

```
00014   add code for ARR in advanced config function
00015   add priority stuff for NVIC
00016 */
00017
00018 #include "blox_tim.h"
00019
00024 uint16_t TIM_PSC[8];
00025 void (*TIM_Handler[28])(void);
00026 uint16_t TIM_IRQ_period[28];
00027
00028 void Blox_Timer_DeInit_Timer(void);
00029 void Blox_Timer_RCC_Configuration(uint8_t TIMx);
00030 void Blox_Timer_NVIC_Configuration(uint8_t TIMx);
00031 TIMER_ID Timer_OC_IRQ_Configuration(uint8_t TIMx, uint16_t period, void (*Timer_H
      andler)(void), FunctionalState NewState);
00032 TIMER_ID Timer_UP_IRQ_Configuration(uint8_t TIMx, uint16_t period, void (*Timer_H
      andler)(void), FunctionalState NewState);
00033
00042 void Blox_Timer_Init(uint8_t TIMx, uint32_t TIM_CLK) {
00043   TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
00044   TIM_PSC[TIMx-1] = (uint16_t) (SystemCoreClock / TIM_CLK) - 1;
00045   Blox_Timer_RCC_Configuration(TIMx);
00046   Blox_Timer_NVIC_Configuration(TIMx);
00047   TIM_TimeBaseStructure.TIM_Period = 65535;
00048   TIM_TimeBaseStructure.TIM_Prescaler = TIM_PSC[TIMx-1];
00049   TIM_TimeBaseStructure.TIM_ClockDivision = 0;
00050   TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
00051   switch(TIMx) {
00052     case 1:
00053       TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);
00054       break;
00055     case 2:
00056       TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
00057       break;
00058     case 3:
00059       TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
00060       break;
00061     case 4:
00062       TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);
00063       break;
00064     case 5:
00065       TIM_TimeBaseInit(TIM5, &TIM_TimeBaseStructure);
00066       break;
00067     case 6:
00068       TIM_TimeBaseInit(TIM6, &TIM_TimeBaseStructure);
00069       break;
00070     case 7:
00071       TIM_TimeBaseInit(TIM7, &TIM_TimeBaseStructure);
00072       break;
00073     case 8:
00074       TIM_TimeBaseInit(TIM8, &TIM_TimeBaseStructure);
00075       break;
00076    }
00077
00078    Blox_System_Register_DeInit(&RCC_DeInit);
00079    Blox_System_Register_DeInit(&Blox_Timer_DeInit_Timer);
00080 }
```

```
00081
00086 void Blox_Timer_DeInit_Timer(void) {
00087   TIM_DeInit(TIM1);
00088   TIM_DeInit(TIM2);
00089   TIM_DeInit(TIM3);
00090   TIM_DeInit(TIM4);
00091   TIM_DeInit(TIM5);
00092   TIM_DeInit(TIM6);
00093   TIM_DeInit(TIM7);
00094   TIM_DeInit(TIM8);
00095 }
00096
00102 void Blox_Timer_RCC_Configuration(uint8_t TIMx) {
00103   switch(TIMx) {
00104     case 1:
00105       RCC_APB2PeriphClockCmd(TIM1_CLK, ENABLE);
00106       break;
00107     case 2:
00108       RCC_APB1PeriphClockCmd(TIM2_CLK, ENABLE);
00109       break;
00110     case 3:
00111       RCC_APB1PeriphClockCmd(TIM3_CLK, ENABLE);
00112       break;
00113     case 4:
00114       RCC_APB1PeriphClockCmd(TIM4_CLK, ENABLE);
00115       break;
00116     case 5:
00117       RCC_APB1PeriphClockCmd(TIM5_CLK, ENABLE);
00118       break;
00119     case 6:
00120       RCC_APB1PeriphClockCmd(TIM6_CLK, ENABLE);
00121       break;
00122     case 7:
00123       RCC_APB1PeriphClockCmd(TIM7_CLK, ENABLE);
00124       break;
00125     case 8:
00126       RCC_APB2PeriphClockCmd(TIM8_CLK, ENABLE);
00127       break;
00128   }
00129 }
00130
00136 void Blox_Timer_NVIC_Configuration(uint8_t TIMx) {
00137   NVIC_InitTypeDef NVIC_InitStructure;
00138   NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);
00139   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 10;
00140   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
00141   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
00142   switch(TIMx) {
00143     case 1:
00144       NVIC_InitStructure.NVIC_IRQChannel = TIM1_UP_IRQn;
00145       NVIC_Init(&NVIC_InitStructure);
00146       NVIC_InitStructure.NVIC_IRQChannel = TIM1_CC_IRQn;
00147       break;
00148     case 2:
00149       NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
00150       break;
00151     case 3:
```

```
00152       NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
00153       break;
00154     case 4:
00155       NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;
00156       break;
00157     case 5:
00158       NVIC_InitStructure.NVIC_IRQChannel = TIM5_IRQn;
00159       break;
00160     case 6:
00161       NVIC_InitStructure.NVIC_IRQChannel = TIM6_IRQn;
00162       break;
00163     case 7:
00164       NVIC_InitStructure.NVIC_IRQChannel = TIM7_IRQn;
00165       break;
00166     case 8:
00167       NVIC_InitStructure.NVIC_IRQChannel = TIM8_UP_IRQn;
00168       NVIC_Init(&NVIC_InitStructure);
00169       NVIC_InitStructure.NVIC_IRQChannel = TIM8_CC_IRQn;
00170       break;
00171   }
00172   NVIC_Init(&NVIC_InitStructure);
00173 }
00174
00183 TIMER_ID Blox_Timer_Register_IRQ(uint8_t TIMx, uint16_t period, void (*Timer_Hand
      ler)(void), FunctionalState NewState) {
00184   TIMER_ID id;
00185   switch(TIMx) {
00186     case 1:
00187     case 2:
00188     case 3:
00189     case 4:
00190     case 5:
00191     case 8:
00192       id = Timer_OC_IRQ_Configuration(TIMx, period, Timer_Handler, NewState);
00193       break;
00194     case 6:
00195     case 7:
00196       id = Timer_UP_IRQ_Configuration(TIMx, period, Timer_Handler, NewState);
00197       break;
00198   }
00199   return id;
00200 }
00201
00210 TIMER_ID Timer_OC_IRQ_Configuration(uint8_t TIMx, uint16_t period, void (*Timer_H
      andler)(void), FunctionalState NewState) {
00211   TIMER_ID id;
00212   TIM_OCInitTypeDef TIM_OCInitStructure;
00213   TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;
00214   TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Disable;
00215   TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
00216   switch(TIMx) {
00217     case 1:
00218
00219       if(TIM_Handler[TIM1CH1] == NULL) {
00220         id = TIM1CH1;
00221         TIM_Handler[id] = Timer_Handler;
00222         TIM_IRQ_period[id] = period;
```

```
00223         TIM_OCInitStructure.TIM_Pulse = period;
00224         TIM_OC1Init(TIM1, &TIM_OCInitStructure);
00225         TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Disable);
00226         TIM_ITConfig(TIM1, TIM_IT_CC1, NewState);
00227       }
00228       else if(TIM_Handler[TIM1CH2] == NULL) {
00229         id = TIM1CH2;
00230         TIM_Handler[id] = Timer_Handler;
00231         TIM_IRQ_period[id] = period;
00232         TIM_OCInitStructure.TIM_Pulse = period;
00233         TIM_OC2Init(TIM1, &TIM_OCInitStructure);
00234         TIM_OC2PreloadConfig(TIM1, TIM_OCPreload_Disable);
00235         TIM_ITConfig(TIM1, TIM_IT_CC2, NewState);
00236       }
00237       else if(TIM_Handler[TIM1CH3] == NULL) {
00238         id = TIM1CH3;
00239         TIM_Handler[id] = Timer_Handler;
00240         TIM_IRQ_period[id] = period;
00241         TIM_OCInitStructure.TIM_Pulse = period;
00242         TIM_OC3Init(TIM1, &TIM_OCInitStructure);
00243         TIM_OC3PreloadConfig(TIM1, TIM_OCPreload_Disable);
00244         TIM_ITConfig(TIM1, TIM_IT_CC3, NewState);
00245       }
00246       else if(TIM_Handler[TIM1CH4] == NULL) {
00247         id = TIM1CH4;
00248         TIM_Handler[id] = Timer_Handler;
00249         TIM_IRQ_period[id] = period;
00250         TIM_OCInitStructure.TIM_Pulse = period;
00251         TIM_OC4Init(TIM1, &TIM_OCInitStructure);
00252         TIM_OC4PreloadConfig(TIM1, TIM_OCPreload_Disable);
00253         TIM_ITConfig(TIM1, TIM_IT_CC4, NewState);
00254       }
00255       else {
00256         id = IRQ_UNAVAILABLE;
00257       }
00258       TIM_Cmd(TIM1, ENABLE);
00259       break;
00260     case 2:
00261       if(TIM_Handler[TIM2CH1] == NULL) {
00262         id = TIM2CH1;
00263         TIM_Handler[id] = Timer_Handler;
00264         TIM_IRQ_period[id] = period;
00265         TIM_OCInitStructure.TIM_Pulse = period;
00266         TIM_OC1Init(TIM2, &TIM_OCInitStructure);
00267         TIM_OC1PreloadConfig(TIM2, TIM_OCPreload_Disable);
00268         TIM_ITConfig(TIM2, TIM_IT_CC1, NewState);
00269       }
00270       else if(TIM_Handler[TIM2CH2] == NULL) {
00271         id = TIM2CH2;
00272         TIM_Handler[id] = Timer_Handler;
00273         TIM_IRQ_period[id] = period;
00274         TIM_OCInitStructure.TIM_Pulse = period;
00275         TIM_OC2Init(TIM2, &TIM_OCInitStructure);
00276         TIM_OC2PreloadConfig(TIM2, TIM_OCPreload_Disable);
00277         TIM_ITConfig(TIM2, TIM_IT_CC2, NewState);
00278       }
00279       else if(TIM_Handler[TIM2CH3] == NULL) {
```

```
00280         id = TIM2CH3;
00281         TIM_Handler[id] = Timer_Handler;
00282         TIM_IRQ_period[id] = period;
00283         TIM_OCInitStructure.TIM_Pulse = period;
00284         TIM_OC3Init(TIM2, &TIM_OCInitStructure);
00285         TIM_OC3PreloadConfig(TIM2, TIM_OCPreload_Disable);
00286         TIM_ITConfig(TIM2, TIM_IT_CC3, NewState);
00287       }
00288       else if(TIM_Handler[TIM2CH4] == NULL) {
00289         id = TIM2CH4;
00290         TIM_Handler[id] = Timer_Handler;
00291         TIM_IRQ_period[id] = period;
00292         TIM_OCInitStructure.TIM_Pulse = period;
00293         TIM_OC4Init(TIM2, &TIM_OCInitStructure);
00294         TIM_OC4PreloadConfig(TIM2, TIM_OCPreload_Disable);
00295         TIM_ITConfig(TIM2, TIM_IT_CC4, NewState);
00296       }
00297       else {
00298         id = IRQ_UNAVAILABLE;
00299       }
00300       TIM_Cmd(TIM2, ENABLE);
00301       break;
00302     case 3:
00303       if(TIM_Handler[TIM3CH1] == NULL) {
00304         id = TIM3CH1;
00305         TIM_Handler[id] = Timer_Handler;
00306         TIM_IRQ_period[id] = period;
00307         TIM_OCInitStructure.TIM_Pulse = period;
00308         TIM_OC1Init(TIM3, &TIM_OCInitStructure);
00309         TIM_OC1PreloadConfig(TIM3, TIM_OCPreload_Disable);
00310         TIM_ITConfig(TIM3, TIM_IT_CC1, NewState);
00311       }
00312       else if(TIM_Handler[TIM3CH2] == NULL) {
00313         id = TIM3CH2;
00314         TIM_Handler[id] = Timer_Handler;
00315         TIM_IRQ_period[id] = period;
00316         TIM_OCInitStructure.TIM_Pulse = period;
00317         TIM_OC2Init(TIM3, &TIM_OCInitStructure);
00318         TIM_OC2PreloadConfig(TIM3, TIM_OCPreload_Disable);
00319         TIM_ITConfig(TIM3, TIM_IT_CC2, NewState);
00320       }
00321       else if(TIM_Handler[TIM3CH3] == NULL) {
00322         id = TIM3CH3;
00323         TIM_Handler[id] = Timer_Handler;
00324         TIM_IRQ_period[id] = period;
00325         TIM_OCInitStructure.TIM_Pulse = period;
00326         TIM_OC3Init(TIM3, &TIM_OCInitStructure);
00327         TIM_OC3PreloadConfig(TIM3, TIM_OCPreload_Disable);
00328         TIM_ITConfig(TIM3, TIM_IT_CC3, NewState);
00329       }
00330       else if(TIM_Handler[TIM3CH4] == NULL) {
00331         id = TIM3CH4;
00332         TIM_Handler[id] = Timer_Handler;
00333         TIM_IRQ_period[id] = period;
00334         TIM_OCInitStructure.TIM_Pulse = period;
00335         TIM_OC4Init(TIM3, &TIM_OCInitStructure);
00336         TIM_OC4PreloadConfig(TIM3, TIM_OCPreload_Disable);
```

```
00337         TIM_ITConfig(TIM3, TIM_IT_CC4, NewState);
00338       }
00339       else {
00340         id = IRQ_UNAVAILABLE;
00341       }
00342       TIM_Cmd(TIM3, ENABLE);
00343       break;
00344     case 4:
00345       if(TIM_Handler[TIM4CH1] == NULL) {
00346         id = TIM4CH1;
00347         TIM_Handler[id] = Timer_Handler;
00348         TIM_IRQ_period[id] = period;
00349         TIM_OCInitStructure.TIM_Pulse = period;
00350         TIM_OC1Init(TIM4, &TIM_OCInitStructure);
00351         TIM_OC1PreloadConfig(TIM4, TIM_OCPreload_Disable);
00352         TIM_ITConfig(TIM4, TIM_IT_CC1, NewState);
00353       }
00354       else if(TIM_Handler[TIM4CH2] == NULL) {
00355         id = TIM4CH2;
00356         TIM_Handler[id] = Timer_Handler;
00357         TIM_IRQ_period[id] = period;
00358         TIM_OCInitStructure.TIM_Pulse = period;
00359         TIM_OC2Init(TIM4, &TIM_OCInitStructure);
00360         TIM_OC2PreloadConfig(TIM4, TIM_OCPreload_Disable);
00361         TIM_ITConfig(TIM4, TIM_IT_CC2, NewState);
00362       }
00363       else if(TIM_Handler[TIM4CH3] == NULL) {
00364         id = TIM4CH3;
00365         TIM_Handler[id] = Timer_Handler;
00366         TIM_IRQ_period[id] = period;
00367         TIM_OCInitStructure.TIM_Pulse = period;
00368         TIM_OC3Init(TIM4, &TIM_OCInitStructure);
00369         TIM_OC3PreloadConfig(TIM4, TIM_OCPreload_Disable);
00370         TIM_ITConfig(TIM4, TIM_IT_CC3, NewState);
00371       }
00372       else if(TIM_Handler[TIM4CH4] == NULL) {
00373         id = TIM4CH4;
00374         TIM_Handler[id] = Timer_Handler;
00375         TIM_IRQ_period[id] = period;
00376         TIM_OCInitStructure.TIM_Pulse = period;
00377         TIM_OC4Init(TIM4, &TIM_OCInitStructure);
00378         TIM_OC4PreloadConfig(TIM4, TIM_OCPreload_Disable);
00379         TIM_ITConfig(TIM4, TIM_IT_CC4, NewState);
00380       }
00381       else {
00382         id = IRQ_UNAVAILABLE;
00383       }
00384       TIM_Cmd(TIM4, ENABLE);
00385       break;
00386     case 5:
00387       if(TIM_Handler[TIM5CH1] == NULL) {
00388         id = TIM5CH1;
00389         TIM_Handler[id] = Timer_Handler;
00390         TIM_IRQ_period[id] = period;
00391         TIM_OCInitStructure.TIM_Pulse = period;
00392         TIM_OC1Init(TIM5, &TIM_OCInitStructure);
00393         TIM_OC1PreloadConfig(TIM5, TIM_OCPreload_Disable);
```

```
00394         TIM_ITConfig(TIM5, TIM_IT_CC1, NewState);
00395       }
00396       else if(TIM_Handler[TIM5CH2] == NULL) {
00397         id = TIM5CH2;
00398         TIM_Handler[id] = Timer_Handler;
00399         TIM_IRQ_period[id] = period;
00400         TIM_OCInitStructure.TIM_Pulse = period;
00401         TIM_OC2Init(TIM5, &TIM_OCInitStructure);
00402         TIM_OC2PreloadConfig(TIM5, TIM_OCPreload_Disable);
00403         TIM_ITConfig(TIM5, TIM_IT_CC2, NewState);
00404       }
00405       else if(TIM_Handler[TIM5CH3] == NULL) {
00406         id = TIM5CH3;
00407         TIM_Handler[id] = Timer_Handler;
00408         TIM_IRQ_period[id] = period;
00409         TIM_OCInitStructure.TIM_Pulse = period;
00410         TIM_OC3Init(TIM5, &TIM_OCInitStructure);
00411         TIM_OC3PreloadConfig(TIM5, TIM_OCPreload_Disable);
00412         TIM_ITConfig(TIM5, TIM_IT_CC3, NewState);
00413       }
00414       else if(TIM_Handler[TIM5CH4] == NULL) {
00415         id = TIM5CH4;
00416         TIM_Handler[id] = Timer_Handler;
00417         TIM_IRQ_period[id] = period;
00418         TIM_OCInitStructure.TIM_Pulse = period;
00419         TIM_OC4Init(TIM5, &TIM_OCInitStructure);
00420         TIM_OC4PreloadConfig(TIM5, TIM_OCPreload_Disable);
00421         TIM_ITConfig(TIM5, TIM_IT_CC4, NewState);
00422       }
00423       else {
00424         id = IRQ_UNAVAILABLE;
00425       }
00426       TIM_Cmd(TIM5, ENABLE);
00427       break;
00428     case 8:
00429       if(TIM_Handler[TIM8CH1] == NULL) {
00430         id = TIM8CH1;
00431         TIM_Handler[id] = Timer_Handler;
00432         TIM_IRQ_period[id] = period;
00433         TIM_OCInitStructure.TIM_Pulse = period;
00434         TIM_OC1Init(TIM8, &TIM_OCInitStructure);
00435         TIM_OC1PreloadConfig(TIM8, TIM_OCPreload_Disable);
00436         TIM_ITConfig(TIM8, TIM_IT_CC1, NewState);
00437       }
00438       else if(TIM_Handler[TIM8CH2] == NULL) {
00439         id = TIM8CH2;
00440         TIM_Handler[id] = Timer_Handler;
00441         TIM_IRQ_period[id] = period;
00442         TIM_OCInitStructure.TIM_Pulse = period;
00443         TIM_OC2Init(TIM8, &TIM_OCInitStructure);
00444         TIM_OC2PreloadConfig(TIM8, TIM_OCPreload_Disable);
00445         TIM_ITConfig(TIM8, TIM_IT_CC2, NewState);
00446       }
00447       else if(TIM_Handler[TIM8CH3] == NULL) {
00448         id = TIM8CH3;
00449         TIM_Handler[id] = Timer_Handler;
00450         TIM_IRQ_period[id] = period;
```

```
00451         TIM_OCInitStructure.TIM_Pulse = period;
00452         TIM_OC3Init(TIM8, &TIM_OCInitStructure);
00453         TIM_OC3PreloadConfig(TIM8, TIM_OCPreload_Disable);
00454         TIM_ITConfig(TIM8, TIM_IT_CC3, NewState);
00455       }
00456       else if(TIM_Handler[TIM8CH4] == NULL) {
00457         id = TIM8CH4;
00458         TIM_Handler[id] = Timer_Handler;
00459         TIM_IRQ_period[id] = period;
00460         TIM_OCInitStructure.TIM_Pulse = period;
00461         TIM_OC4Init(TIM8, &TIM_OCInitStructure);
00462         TIM_OC4PreloadConfig(TIM8, TIM_OCPreload_Disable);
00463         TIM_ITConfig(TIM8, TIM_IT_CC4, NewState);
00464       }
00465       else {
00466         id = IRQ_UNAVAILABLE;
00467       }
00468       TIM_Cmd(TIM8, ENABLE);
00469       break;
00470     default:
00471       id = INVALID_TIMER;
00472       break;
00473   }
00474   return id;
00475 }
00476
00485 TIMER_ID Timer_UP_IRQ_Configuration(uint8_t TIMx, uint16_t period, void (*Timer_H
      andler)(void), FunctionalState NewState) {
00486   TIMER_ID id;
00487   switch(TIMx) {
00488     case 6:
00489       if(TIM_Handler[TIM6UP] == NULL) {
00490         id = TIM6UP;
00491         TIM_Handler[id] = Timer_Handler;
00492         TIM_IRQ_period[id] = period;
00493         TIM_SetAutoreload(TIM6, period);
00494         TIM_ITConfig(TIM6, TIM_IT_Update, NewState);
00495       }
00496       else {
00497         id = IRQ_UNAVAILABLE;
00498       }
00499       TIM_Cmd(TIM6, ENABLE);
00500       break;
00501     case 7:
00502       if(TIM_Handler[TIM7UP] == NULL) {
00503         id = TIM7UP;
00504         TIM_Handler[id] = Timer_Handler;
00505         TIM_IRQ_period[id] = period;
00506         TIM_SetAutoreload(TIM7, period);
00507         TIM_ITConfig(TIM7, TIM_IT_Update, NewState);
00508       }
00509       else {
00510         id = IRQ_UNAVAILABLE;
00511       }
00512       TIM_Cmd(TIM7, ENABLE);
00513       break;
00514     default:
```

```
00515       id = INVALID_TIMER;
00516       break;
00517   }
00518   return id;
00519 }
00520
00526 void Blox_Timer_Release_IRQ(TIMER_ID id) {
00527   Blox_Timer_Disable_IRQ(id);
00528   TIM_Handler[id] = NULL;
00529 }
00530
00531 // calculations to fix CH[1-4] for TIM1 and TIM8 since CNT is reset to 0
00537 void Blox_Timer_Enable_IRQ(TIMER_ID id) {
00538   switch(id) {
00539     case TIM1UP:
00540       TIM_ClearITPendingBit(TIM1, TIM_IT_Update);
00541       TIM_ITConfig(TIM1, TIM_IT_Update, ENABLE);
00542       break;
00543     case TIM1CH1:
00544       TIM_ClearITPendingBit(TIM1, TIM_IT_CC1);
00545       TIM_ITConfig(TIM1, TIM_IT_CC1, ENABLE);
00546       TIM_SetCompare1(TIM1, TIM_IRQ_period[id] + TIM_GetCounter(TIM1));
00547       break;
00548     case TIM1CH2:
00549       TIM_ClearITPendingBit(TIM1, TIM_IT_CC2);
00550       TIM_ITConfig(TIM1, TIM_IT_CC2, ENABLE);
00551       TIM_SetCompare2(TIM1, TIM_IRQ_period[id] + TIM_GetCounter(TIM1));
00552       break;
00553     case TIM1CH3:
00554       TIM_ClearITPendingBit(TIM1, TIM_IT_CC3);
00555       TIM_ITConfig(TIM1, TIM_IT_CC3, ENABLE);
00556       TIM_SetCompare3(TIM1, TIM_IRQ_period[id] + TIM_GetCounter(TIM1));
00557       break;
00558     case TIM1CH4:
00559       TIM_ClearITPendingBit(TIM1, TIM_IT_CC4);
00560       TIM_ITConfig(TIM1, TIM_IT_CC4, ENABLE);
00561       TIM_SetCompare4(TIM1, TIM_IRQ_period[id] + TIM_GetCounter(TIM1));
00562       break;
00563     case TIM2CH1:
00564       TIM_ClearITPendingBit(TIM2, TIM_IT_CC1);
00565       TIM_ITConfig(TIM2, TIM_IT_CC1, ENABLE);
00566       TIM_SetCompare1(TIM2, TIM_IRQ_period[id] + TIM_GetCounter(TIM2));
00567       break;
00568     case TIM2CH2:
00569       TIM_ClearITPendingBit(TIM2, TIM_IT_CC2);
00570       TIM_ITConfig(TIM2, TIM_IT_CC2, ENABLE);
00571       TIM_SetCompare2(TIM2, TIM_IRQ_period[id] + TIM_GetCounter(TIM2));
00572       break;
00573     case TIM2CH3:
00574       TIM_ClearITPendingBit(TIM2, TIM_IT_CC3);
00575       TIM_ITConfig(TIM2, TIM_IT_CC3, ENABLE);
00576       TIM_SetCompare3(TIM2, TIM_IRQ_period[id] + TIM_GetCounter(TIM2));
00577       break;
00578     case TIM2CH4:
00579       TIM_ClearITPendingBit(TIM2, TIM_IT_CC4);
00580       TIM_ITConfig(TIM2, TIM_IT_CC4, ENABLE);
00581       TIM_SetCompare4(TIM2, TIM_IRQ_period[id] + TIM_GetCounter(TIM2));
```

```
00582       break;
00583     case TIM3CH1:
00584       TIM_ClearITPendingBit(TIM3, TIM_IT_CC1);
00585       TIM_ITConfig(TIM3, TIM_IT_CC1, ENABLE);
00586       TIM_SetCompare1(TIM3, TIM_IRQ_period[id] + TIM_GetCounter(TIM3));
00587       break;
00588     case TIM3CH2:
00589       TIM_ClearITPendingBit(TIM3, TIM_IT_CC2);
00590       TIM_ITConfig(TIM3, TIM_IT_CC2, ENABLE);
00591       TIM_SetCompare2(TIM3, TIM_IRQ_period[id] + TIM_GetCounter(TIM3));
00592       break;
00593     case TIM3CH3:
00594       TIM_ClearITPendingBit(TIM3, TIM_IT_CC3);
00595       TIM_ITConfig(TIM3, TIM_IT_CC3, ENABLE);
00596       TIM_SetCompare3(TIM3, TIM_IRQ_period[id] + TIM_GetCounter(TIM3));
00597       break;
00598     case TIM3CH4:
00599       TIM_ClearITPendingBit(TIM3, TIM_IT_CC4);
00600       TIM_ITConfig(TIM3, TIM_IT_CC4, ENABLE);
00601       TIM_SetCompare4(TIM3, TIM_IRQ_period[id] + TIM_GetCounter(TIM3));
00602       break;
00603     case TIM4CH1:
00604       TIM_ClearITPendingBit(TIM4, TIM_IT_CC1);
00605       TIM_ITConfig(TIM4, TIM_IT_CC1, ENABLE);
00606       TIM_SetCompare1(TIM4, TIM_IRQ_period[id] + TIM_GetCounter(TIM4));
00607       break;
00608     case TIM4CH2:
00609       TIM_ClearITPendingBit(TIM4, TIM_IT_CC2);
00610       TIM_ITConfig(TIM4, TIM_IT_CC2, ENABLE);
00611       TIM_SetCompare2(TIM4, TIM_IRQ_period[id] + TIM_GetCounter(TIM4));
00612       break;
00613     case TIM4CH3:
00614       TIM_ClearITPendingBit(TIM4, TIM_IT_CC3);
00615       TIM_ITConfig(TIM4, TIM_IT_CC3, ENABLE);
00616       TIM_SetCompare3(TIM4, TIM_IRQ_period[id] + TIM_GetCounter(TIM4));
00617       break;
00618     case TIM4CH4:
00619       TIM_ClearITPendingBit(TIM4, TIM_IT_CC4);
00620       TIM_ITConfig(TIM4, TIM_IT_CC4, ENABLE);
00621       TIM_SetCompare4(TIM4, TIM_IRQ_period[id] + TIM_GetCounter(TIM4));
00622       break;
00623     case TIM5CH1:
00624       TIM_ClearITPendingBit(TIM5, TIM_IT_CC1);
00625       TIM_ITConfig(TIM5, TIM_IT_CC1, ENABLE);
00626       TIM_SetCompare1(TIM5, TIM_IRQ_period[id] + TIM_GetCounter(TIM5));
00627       break;
00628     case TIM5CH2:
00629       TIM_ClearITPendingBit(TIM5, TIM_IT_CC2);
00630       TIM_ITConfig(TIM5, TIM_IT_CC2, ENABLE);
00631       TIM_SetCompare2(TIM5, TIM_IRQ_period[id] + TIM_GetCounter(TIM5));
00632       break;
00633     case TIM5CH3:
00634       TIM_ClearITPendingBit(TIM5, TIM_IT_CC3);
00635       TIM_ITConfig(TIM5, TIM_IT_CC3, ENABLE);
00636       TIM_SetCompare3(TIM5, TIM_IRQ_period[id] + TIM_GetCounter(TIM5));
00637       break;
00638     case TIM5CH4:
```

```
00639       TIM_ClearITPendingBit(TIM5, TIM_IT_CC4);
00640       TIM_ITConfig(TIM5, TIM_IT_CC4, ENABLE);
00641       TIM_SetCompare4(TIM5, TIM_IRQ_period[id] + TIM_GetCounter(TIM5));
00642       break;
00643     case TIM6UP:
00644       TIM_ClearITPendingBit(TIM6, TIM_IT_Update);
00645       TIM_ITConfig(TIM6, TIM_IT_Update, ENABLE);
00646       TIM_SetCounter(TIM6, 0);
00647       break;
00648     case TIM7UP:
00649       TIM_ClearITPendingBit(TIM7, TIM_IT_Update);
00650       TIM_ITConfig(TIM7, TIM_IT_Update, ENABLE);
00651       TIM_SetCounter(TIM7, 0);
00652       break;
00653     case TIM8UP:
00654       TIM_ClearITPendingBit(TIM8, TIM_IT_Update);
00655       TIM_ITConfig(TIM8, TIM_IT_Update, ENABLE);
00656
00657       break;
00658     case TIM8CH1:
00659       TIM_ClearITPendingBit(TIM8, TIM_IT_CC1);
00660       TIM_ITConfig(TIM8, TIM_IT_CC1, ENABLE);
00661       TIM_SetCompare1(TIM8, TIM_IRQ_period[id] + TIM_GetCounter(TIM8));
00662       break;
00663     case TIM8CH2:
00664       TIM_ClearITPendingBit(TIM8, TIM_IT_CC2);
00665       TIM_ITConfig(TIM8, TIM_IT_CC2, ENABLE);
00666       TIM_SetCompare2(TIM8, TIM_IRQ_period[id] + TIM_GetCounter(TIM8));
00667       break;
00668     case TIM8CH3:
00669       TIM_ClearITPendingBit(TIM8, TIM_IT_CC3);
00670       TIM_ITConfig(TIM8, TIM_IT_CC3, ENABLE);
00671       TIM_SetCompare3(TIM8, TIM_IRQ_period[id] + TIM_GetCounter(TIM8));
00672       break;
00673     case TIM8CH4:
00674       TIM_ClearITPendingBit(TIM8, TIM_IT_CC4);
00675       TIM_ITConfig(TIM8, TIM_IT_CC4, ENABLE);
00676       TIM_SetCompare4(TIM8, TIM_IRQ_period[id] + TIM_GetCounter(TIM8));
00677       break;
00678   }
00679 }
00680
00686 void Blox_Timer_Disable_IRQ(TIMER_ID id) {
00687   switch(id) {
00688     case TIM1UP:
00689       TIM_ITConfig(TIM1, TIM_IT_Update, DISABLE);
00690       break;
00691     case TIM1CH1:
00692       TIM_ITConfig(TIM1, TIM_IT_CC1, DISABLE);
00693       break;
00694     case TIM1CH2:
00695       TIM_ITConfig(TIM1, TIM_IT_CC2, DISABLE);
00696       break;
00697     case TIM1CH3:
00698       TIM_ITConfig(TIM1, TIM_IT_CC3, DISABLE);
00699       break;
00700     case TIM1CH4:
```

```
00701       TIM_ITConfig(TIM1, TIM_IT_CC4, DISABLE);
00702       break;
00703     case TIM2CH1:
00704       TIM_ITConfig(TIM2, TIM_IT_CC1, DISABLE);
00705       break;
00706     case TIM2CH2:
00707       TIM_ITConfig(TIM2, TIM_IT_CC2, DISABLE);
00708       break;
00709     case TIM2CH3:
00710       TIM_ITConfig(TIM2, TIM_IT_CC3, DISABLE);
00711       break;
00712     case TIM2CH4:
00713       TIM_ITConfig(TIM2, TIM_IT_CC4, DISABLE);
00714       break;
00715     case TIM3CH1:
00716       TIM_ITConfig(TIM3, TIM_IT_CC1, DISABLE);
00717       break;
00718     case TIM3CH2:
00719       TIM_ITConfig(TIM3, TIM_IT_CC2, DISABLE);
00720       break;
00721     case TIM3CH3:
00722       TIM_ITConfig(TIM3, TIM_IT_CC3, DISABLE);
00723       break;
00724     case TIM3CH4:
00725       TIM_ITConfig(TIM3, TIM_IT_CC4, DISABLE);
00726       break;
00727     case TIM4CH1:
00728       TIM_ITConfig(TIM4, TIM_IT_CC1, DISABLE);
00729       break;
00730     case TIM4CH2:
00731       TIM_ITConfig(TIM4, TIM_IT_CC2, DISABLE);
00732       break;
00733     case TIM4CH3:
00734       TIM_ITConfig(TIM4, TIM_IT_CC3, DISABLE);
00735       break;
00736     case TIM4CH4:
00737       TIM_ITConfig(TIM4, TIM_IT_CC4, DISABLE);
00738       break;
00739     case TIM5CH1:
00740       TIM_ITConfig(TIM5, TIM_IT_CC1, DISABLE);
00741       break;
00742     case TIM5CH2:
00743       TIM_ITConfig(TIM5, TIM_IT_CC2, DISABLE);
00744       break;
00745     case TIM5CH3:
00746       TIM_ITConfig(TIM5, TIM_IT_CC3, DISABLE);
00747       break;
00748     case TIM5CH4:
00749       TIM_ITConfig(TIM5, TIM_IT_CC4, DISABLE);
00750       break;
00751     case TIM6UP:
00752       TIM_ITConfig(TIM6, TIM_IT_Update, DISABLE);
00753       break;
00754     case TIM7UP:
00755       TIM_ITConfig(TIM7, TIM_IT_Update, DISABLE);
00756       break;
00757     case TIM8UP:
```

```
00758       TIM_ITConfig(TIM8, TIM_IT_Update, DISABLE);
00759       break;
00760     case TIM8CH1:
00761       TIM_ITConfig(TIM8, TIM_IT_CC1, DISABLE);
00762       break;
00763     case TIM8CH2:
00764       TIM_ITConfig(TIM8, TIM_IT_CC2, DISABLE);
00765       break;
00766     case TIM8CH3:
00767       TIM_ITConfig(TIM8, TIM_IT_CC3, DISABLE);
00768       break;
00769     case TIM8CH4:
00770       TIM_ITConfig(TIM8, TIM_IT_CC4, DISABLE);
00771       break;
00772   }
00773 }
00774
00781 void Blox_Timer_Modify_IRQ(TIMER_ID id, uint16_t period) {
00782   TIM_IRQ_period[id] = period;
00783 }
00784
00789 void TIM1_CC_IRQHandler(void) {
00790   if (TIM_GetITStatus(TIM1, TIM_IT_CC1) != RESET) {
00791     TIM_ClearITPendingBit(TIM1, TIM_IT_CC1);
00792     if(TIM_Handler[TIM1CH1] != NULL) {
00793       TIM_Handler[TIM1CH1]();
00794     }
00795     TIM_SetCompare1(TIM1, TIM_GetCapture1(TIM1) + TIM_IRQ_period[TIM1CH1]);
00796   }
00797   else if (TIM_GetITStatus(TIM1, TIM_IT_CC2) != RESET) {
00798     TIM_ClearITPendingBit(TIM1, TIM_IT_CC2);
00799     if(TIM_Handler[TIM1CH2] != NULL) {
00800       TIM_Handler[TIM1CH2]();
00801     }
00802     TIM_SetCompare2(TIM1, TIM_GetCapture2(TIM1) + TIM_IRQ_period[TIM1CH2]);
00803   }
00804   else if (TIM_GetITStatus(TIM1, TIM_IT_CC3) != RESET) {
00805     TIM_ClearITPendingBit(TIM1, TIM_IT_CC3);
00806     if(TIM_Handler[TIM1CH3] != NULL) {
00807       TIM_Handler[TIM1CH3]();
00808     }
00809     TIM_SetCompare3(TIM1, TIM_GetCapture3(TIM1) + TIM_IRQ_period[TIM1CH3]);
00810   }
00811   else if (TIM_GetITStatus(TIM1, TIM_IT_CC4) != RESET) {
00812     TIM_ClearITPendingBit(TIM1, TIM_IT_CC4);
00813     if(TIM_Handler[TIM1CH4] != NULL) {
00814       TIM_Handler[TIM1CH4]();
00815     }
00816     TIM_SetCompare4(TIM1, TIM_GetCapture4(TIM1) + TIM_IRQ_period[TIM1CH4]);
00817   }
00818 }
00819
00824 void TIM2_IRQHandler(void) {
00825   if (TIM_GetITStatus(TIM2, TIM_IT_CC1) != RESET) {
00826     TIM_ClearITPendingBit(TIM2, TIM_IT_CC1);
00827     if(TIM_Handler[TIM2CH1] != NULL) {
00828       TIM_Handler[TIM2CH1]();
```

```
00829     }
00830     TIM_SetCompare1(TIM2, TIM_GetCapture1(TIM2) + TIM_IRQ_period[TIM2CH1]);
00831   }
00832   else if (TIM_GetITStatus(TIM2, TIM_IT_CC2) != RESET) {
00833     TIM_ClearITPendingBit(TIM2, TIM_IT_CC2);
00834     if(TIM_Handler[TIM2CH2] != NULL) {
00835       TIM_Handler[TIM2CH2]();
00836     }
00837     TIM_SetCompare2(TIM2, TIM_GetCapture2(TIM2) + TIM_IRQ_period[TIM2CH2]);
00838   }
00839   else if (TIM_GetITStatus(TIM2, TIM_IT_CC3) != RESET) {
00840     TIM_ClearITPendingBit(TIM2, TIM_IT_CC3);
00841     if(TIM_Handler[TIM2CH3] != NULL) {
00842       TIM_Handler[TIM2CH3]();
00843     }
00844     TIM_SetCompare3(TIM2, TIM_GetCapture3(TIM2) + TIM_IRQ_period[TIM2CH3]);
00845   }
00846   else if (TIM_GetITStatus(TIM2, TIM_IT_CC4) != RESET) {
00847     TIM_ClearITPendingBit(TIM2, TIM_IT_CC4);
00848     if(TIM_Handler[TIM2CH4] != NULL) {
00849       TIM_Handler[TIM2CH4]();
00850     }
00851     TIM_SetCompare4(TIM2, TIM_GetCapture4(TIM2) + TIM_IRQ_period[TIM2CH4]);
00852   }
00853 }
00854
00859 void TIM3_IRQHandler(void) {
00860   if (TIM_GetITStatus(TIM3, TIM_IT_CC1) != RESET) {
00861     TIM_ClearITPendingBit(TIM3, TIM_IT_CC1);
00862     if(TIM_Handler[TIM3CH1] != NULL) {
00863       TIM_Handler[TIM3CH1]();
00864     }
00865     TIM_SetCompare1(TIM3, TIM_GetCapture1(TIM3) + TIM_IRQ_period[TIM3CH1]);
00866   }
00867   else if (TIM_GetITStatus(TIM3, TIM_IT_CC2) != RESET) {
00868     TIM_ClearITPendingBit(TIM3, TIM_IT_CC2);
00869     if(TIM_Handler[TIM3CH2] != NULL) {
00870       TIM_Handler[TIM3CH2]();
00871     }
00872     TIM_SetCompare2(TIM3, TIM_GetCapture2(TIM3) + TIM_IRQ_period[TIM3CH2]);
00873   }
00874   else if (TIM_GetITStatus(TIM3, TIM_IT_CC3) != RESET) {
00875     TIM_ClearITPendingBit(TIM3, TIM_IT_CC3);
00876     if(TIM_Handler[TIM3CH3] != NULL) {
00877       TIM_Handler[TIM3CH3]();
00878     }
00879     TIM_SetCompare3(TIM3, TIM_GetCapture3(TIM3) + TIM_IRQ_period[TIM3CH3]);
00880   }
00881   else if (TIM_GetITStatus(TIM3, TIM_IT_CC4) != RESET) {
00882     TIM_ClearITPendingBit(TIM3, TIM_IT_CC4);
00883     if(TIM_Handler[TIM3CH4] != NULL) {
00884       TIM_Handler[TIM3CH4]();
00885     }
00886     TIM_SetCompare4(TIM3, TIM_GetCapture4(TIM3) + TIM_IRQ_period[TIM3CH4]);
00887   }
00888 }
00889
```

```
00894 void TIM4_IRQHandler(void) {
00895   if (TIM_GetITStatus(TIM4, TIM_IT_CC1) != RESET) {
00896     TIM_ClearITPendingBit(TIM4, TIM_IT_CC1);
00897     if(TIM_Handler[TIM4CH1] != NULL) {
00898       TIM_Handler[TIM4CH1]();
00899     }
00900     TIM_SetCompare1(TIM4, TIM_GetCapture1(TIM4) + TIM_IRQ_period[TIM4CH1]);
00901   }
00902   else if (TIM_GetITStatus(TIM4, TIM_IT_CC2) != RESET) {
00903     TIM_ClearITPendingBit(TIM4, TIM_IT_CC2);
00904     if(TIM_Handler[TIM4CH2] != NULL) {
00905       TIM_Handler[TIM4CH2]();
00906     }
00907     TIM_SetCompare2(TIM4, TIM_GetCapture2(TIM4) + TIM_IRQ_period[TIM4CH2]);
00908   }
00909   else if (TIM_GetITStatus(TIM4, TIM_IT_CC3) != RESET) {
00910     TIM_ClearITPendingBit(TIM4, TIM_IT_CC3);
00911     if(TIM_Handler[TIM4CH3] != NULL) {
00912       TIM_Handler[TIM4CH3]();
00913     }
00914     TIM_SetCompare3(TIM4, TIM_GetCapture3(TIM4) + TIM_IRQ_period[TIM4CH3]);
00915   }
00916   else if (TIM_GetITStatus(TIM4, TIM_IT_CC4) != RESET) {
00917     TIM_ClearITPendingBit(TIM4, TIM_IT_CC4);
00918     if(TIM_Handler[TIM4CH4] != NULL) {
00919       TIM_Handler[TIM4CH4]();
00920     }
00921     TIM_SetCompare4(TIM4, TIM_GetCapture4(TIM4) + TIM_IRQ_period[TIM4CH4]);
00922   }
00923 }
00924
00929 void TIM5_IRQHandler(void) {
00930   if (TIM_GetITStatus(TIM5, TIM_IT_CC1) != RESET) {
00931     TIM_ClearITPendingBit(TIM5, TIM_IT_CC1);
00932     if(TIM_Handler[TIM5CH1] != NULL) {
00933       TIM_Handler[TIM5CH1]();
00934     }
00935     TIM_SetCompare1(TIM5, TIM_GetCapture1(TIM5) + TIM_IRQ_period[TIM5CH1]);
00936   }
00937   else if (TIM_GetITStatus(TIM5, TIM_IT_CC2) != RESET) {
00938     TIM_ClearITPendingBit(TIM5, TIM_IT_CC2);
00939     if(TIM_Handler[TIM5CH2] != NULL) {
00940       TIM_Handler[TIM5CH2]();
00941     }
00942     TIM_SetCompare2(TIM5, TIM_GetCapture2(TIM5) + TIM_IRQ_period[TIM5CH2]);
00943   }
00944   else if (TIM_GetITStatus(TIM5, TIM_IT_CC3) != RESET) {
00945     TIM_ClearITPendingBit(TIM5, TIM_IT_CC3);
00946     if(TIM_Handler[TIM5CH3] != NULL) {
00947       TIM_Handler[TIM5CH3]();
00948     }
00949     TIM_SetCompare3(TIM5, TIM_GetCapture3(TIM5) + TIM_IRQ_period[TIM5CH3]);
00950   }
00951   else if (TIM_GetITStatus(TIM5, TIM_IT_CC4) != RESET) {
00952     TIM_ClearITPendingBit(TIM5, TIM_IT_CC4);
00953     if(TIM_Handler[TIM5CH4] != NULL) {
00954       TIM_Handler[TIM5CH4]();
```

```
00955     }
00956     TIM_SetCompare4(TIM5, TIM_GetCapture4(TIM5) + TIM_IRQ_period[TIM5CH4]);
00957   }
00958 }
00959
00964 void TIM6_IRQHandler(void) {
00965   if (TIM_GetITStatus(TIM6, TIM_IT_Update) != RESET) {
00966     TIM_ClearITPendingBit(TIM6, TIM_IT_Update);
00967     if(TIM_Handler[TIM6UP] != NULL) {
00968       TIM_Handler[TIM6UP]();
00969     }
00970   }
00971 }
00972
00977 void TIM7_IRQHandler(void) {
00978   if (TIM_GetITStatus(TIM7, TIM_IT_Update) != RESET) {
00979     TIM_ClearITPendingBit(TIM7, TIM_IT_Update);
00980     if(TIM_Handler[TIM7UP] != NULL) {
00981       TIM_Handler[TIM7UP]();
00982     }
00983   }
00984 }
00985
00990 void TIM8_CC_IRQHandler(void) {
00991   if (TIM_GetITStatus(TIM8, TIM_IT_CC1) != RESET) {
00992     TIM_ClearITPendingBit(TIM8, TIM_IT_CC1);
00993     if(TIM_Handler[TIM8CH1] != NULL) {
00994       TIM_Handler[TIM8CH1]();
00995     }
00996     TIM_SetCompare1(TIM8, TIM_GetCapture1(TIM8) + TIM_IRQ_period[TIM8CH1]);
00997   }
00998   else if (TIM_GetITStatus(TIM8, TIM_IT_CC2) != RESET) {
00999     TIM_ClearITPendingBit(TIM8, TIM_IT_CC2);
01000     if(TIM_Handler[TIM8CH2] != NULL) {
01001       TIM_Handler[TIM8CH2]();
01002     }
01003     TIM_SetCompare2(TIM8, TIM_GetCapture2(TIM8) + TIM_IRQ_period[TIM8CH2]);
01004   }
01005   else if (TIM_GetITStatus(TIM8, TIM_IT_CC3) != RESET) {
01006     TIM_ClearITPendingBit(TIM8, TIM_IT_CC3);
01007     if(TIM_Handler[TIM8CH3] != NULL) {
01008       TIM_Handler[TIM8CH3]();
01009     }
01010     TIM_SetCompare3(TIM8, TIM_GetCapture3(TIM8) + TIM_IRQ_period[TIM8CH3]);
01011   }
01012   else if (TIM_GetITStatus(TIM8, TIM_IT_CC4) != RESET) {
01013     TIM_ClearITPendingBit(TIM8, TIM_IT_CC4);
01014     if(TIM_Handler[TIM8CH4] != NULL) {
01015       TIM_Handler[TIM8CH4]();
01016     }
01017     TIM_SetCompare4(TIM8, TIM_GetCapture4(TIM8) + TIM_IRQ_period[TIM8CH4]);
01018   }
01019 }
```

6.59 drivers/src/blox_touch.c File Reference

Driver that interacts with touchpanels over SPI.

```
#include "blox_touch.h"
```

Functions

- void **Touch_RCC_Init** ()
  Initializes clocks for SPI and the GPIO the SPI & BUSY Pins are on.
- void **Touch_GPIO_Init** ()
  Initializes the gpio for the SPI pins, and BUSY.
- void **Touch_GPIO_DeInit** ()
  De-initializes the gpio for the SPI pins, and BUSY.
- void **Touch_SPI_Init** ()
  Initializes the SPI for the Touchpanel.
- void **Touch_SPI_DeInit** ()
  De-initializes the SPI for the Touchpanel.
- void **Blox_Touch_Init** (void)
  Initializes the IR module. Basically a wrapper on USART.
- uint16_t **Blox_Touch_GetX** (int numTouch)
  Get the X-value of a press on the touchpanel.
- uint16_t **Blox_Touch_GetY** (int numTouch)
  Get the Y-value of a press on the touchpanel./.
- uint16_t **Blox_Touch_GetZ1** (int numTouch)
  Get the Z1-value of a press on the touchpanel.
- uint16_t **Blox_Touch_GetZ2** (int numTouch)
  Get the Z2-value of a press on the touchpanel.
- void **Touch_SPI_Send** (uint16_t data)
  Sends a byte out on SPI to the touchpanel.
- uint16_t **Touch_SPI_Receive** (void)
  Receive a byte from the touchpanel.

6.59.1 Detailed Description

Driver that interacts with touchpanels over SPI.

Author

Ankita Kaul & Jesse Tannahill

Version

V0.1

Date

11/01/2010

Definition in file **blox_touch.c**.

6.60 drivers/src/blox_touch.c

```
00001
00009 #include "blox_touch.h"
00010
00016 /* Private function prototypes */
00017 void Touch_RCC_Init(void);
00018 void Touch_GPIO_Init(void);
00019 void Touch_GPIO_DeInit(void);
00020 void Touch_SPI_Init(void);
00021 void Touch_SPI_DeInit(void);
00022
00027 void Blox_Touch_Init(void) {
00028   Touch_RCC_Init();
00029   Touch_GPIO_Init();
00030   Touch_SPI_Init();
00031
00032   Blox_System_Register_DeInit(&RCC_DeInit);
00033   Blox_System_Register_DeInit(&Touch_GPIO_DeInit);
00034   Blox_System_Register_DeInit(&Touch_SPI_DeInit);
00035 }
00036
00041 void Touch_RCC_Init() {
00042   RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
00043         RCC_APB2PeriphClockCmd(TOUCH_SPI_GPIO_CLK, ENABLE);
00044   RCC_APB2PeriphClockCmd(TOUCH_CS_GPIO_CLK, ENABLE);
00045         RCC_APB2PeriphClockCmd(TOUCH_SPI_CLK, ENABLE);
00046
00047   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
00048 }
00049
00054 void Touch_GPIO_Init() {
00055 GPIO_InitTypeDef GPIO_InitStructure;
00056         GPIO_InitStructure.GPIO_Pin = TOUCH_SPI_SCK_PIN | TOUCH_SPI_MOSI_PIN;
```

```
00057         GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
00058         GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
00059         GPIO_Init(TOUCH_SPI_GPIO, &GPIO_InitStructure);
00060
00061         GPIO_InitStructure.GPIO_Pin = TOUCH_SPI_MISO_PIN;// | TOUCH_BUSY_PIN;
00062         GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
00063         GPIO_Init(TOUCH_SPI_GPIO, &GPIO_InitStructure);
00064
00065   GPIO_InitStructure.GPIO_Pin = TOUCH1_CS_PIN | TOUCH2_CS_PIN | TOUCH3_CS_PIN | T
      OUCH4_CS_PIN;
00066         GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
00067         GPIO_Init(TOUCH_CS_GPIO, &GPIO_InitStructure);
00068
00069   GPIO_InitStructure.GPIO_Pin = TOUCH1_PENIRQ_PIN | TOUCH2_PENIRQ_PIN | TOUCH3_PE
      NIRQ_PIN | TOUCH4_PENIRQ_PIN;
00070   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
00071         GPIO_Init(TOUCH_PENIRQ_GPIO, &GPIO_InitStructure);
00072
00073   /*GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;
00074   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
00075         GPIO_Init(GPIOA, &GPIO_InitStructure);*/
00076
00077   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
00078   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
00079         GPIO_Init(GPIOA, &GPIO_InitStructure);
00080 }
00081
00086 void Touch_GPIO_DeInit() {
00087   GPIO_DeInit(TOUCH_SPI_GPIO);
00088 }
00089
00094 void Touch_SPI_Init() {
00095         SPI_InitTypeDef SPIInitStruct;
00096         SPIInitStruct.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
00097         SPIInitStruct.SPI_Mode = SPI_Mode_Master;
00098         SPIInitStruct.SPI_DataSize = SPI_DataSize_8b;
00099         SPIInitStruct.SPI_CPOL = SPI_CPOL_Low;
00100         SPIInitStruct.SPI_CPHA = SPI_CPHA_1Edge;
00101         SPIInitStruct.SPI_NSS = SPI_NSS_Soft;
00102         SPIInitStruct.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_256;
00103         SPIInitStruct.SPI_FirstBit = SPI_FirstBit_MSB;
00104         SPIInitStruct.SPI_CRCPolynomial = 7;
00105         SPI_Init(TOUCH_SPI, &SPIInitStruct);
00106
00107         SPI_Cmd(TOUCH_SPI, ENABLE);
00108   //SPI_SSOutputCmd(TOUCH_SPI, ENABLE);
00109 }
00110
00115 void Touch_SPI_DeInit() {
00116   SPI_I2S_DeInit(TOUCH_SPI);
00117 }
00118
00124 uint16_t Blox_Touch_GetX(int numTouch) {
00125   uint16_t ret = 0;
00126   //Chip-select is active low.
00127   TOUCH_CS_GPIO->ODR |= (TOUCH1_CS_PIN | TOUCH2_CS_PIN | TOUCH3_CS_PIN
00128                          | TOUCH4_CS_PIN);
```

```
00129
00130   switch(numTouch){
00131         case 1: TOUCH_CS_GPIO->ODR &= ~TOUCH1_CS_PIN;
00132                         break;
00133         case 2: TOUCH_CS_GPIO->ODR &= ~TOUCH2_CS_PIN;
00134                         break;
00135         case 3: TOUCH_CS_GPIO->ODR &= ~TOUCH3_CS_PIN;
00136                         break;
00137         case 4: TOUCH_CS_GPIO->ODR &= ~TOUCH4_CS_PIN;
00138                         break;
00139   }
00140
00141   if (SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_RXNE) == SET)
00142     SPI_I2S_ReceiveData(TOUCH_SPI);
00143
00144   while (SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_TXE)== RESET) ;
00145   SPI_I2S_SendData(TOUCH_SPI, TOUCH_CTL_X);
00146   while(SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_RXNE) == RESET) ;
00147   SPI_I2S_ReceiveData(TOUCH_SPI); //Toss junk
00148
00149   while (SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_TXE)== RESET) ;
00150   SPI_I2S_SendData(TOUCH_SPI, 0x0);
00151   while(SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_RXNE) == RESET) ;
00152   ret = SPI_I2S_ReceiveData(TOUCH_SPI); //Data
00153
00154   while (SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_TXE)== RESET) ;
00155   SPI_I2S_SendData(TOUCH_SPI, 0x0);
00156   while(SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_RXNE) == RESET) ;
00157   SPI_I2S_ReceiveData(TOUCH_SPI); //tosss
00158
00159   return ret;
00160 }
00161
00167 uint16_t Blox_Touch_GetY(int numTouch) {
00168   uint16_t ret = 0;
00169   //Chip-select is active low.
00170   TOUCH_CS_GPIO->ODR |= (TOUCH1_CS_PIN | TOUCH2_CS_PIN | TOUCH3_CS_PIN
00171                         | TOUCH4_CS_PIN);
00172   switch(numTouch){
00173         case 1: TOUCH_CS_GPIO->ODR &= ~TOUCH1_CS_PIN;
00174                         break;
00175         case 2: TOUCH_CS_GPIO->ODR &= ~TOUCH2_CS_PIN;
00176                         break;
00177         case 3: TOUCH_CS_GPIO->ODR &= ~TOUCH3_CS_PIN;
00178                         break;
00179         case 4: TOUCH_CS_GPIO->ODR &= ~TOUCH4_CS_PIN;
00180                         break;
00181         }
00182
00183   if (SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_RXNE) == SET)
00184     SPI_I2S_ReceiveData(TOUCH_SPI);
00185
00186   while (SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_TXE)== RESET) ;
00187   SPI_I2S_SendData(TOUCH_SPI, TOUCH_CTL_Y);
00188   while(SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_RXNE) == RESET) ;
00189   SPI_I2S_ReceiveData(TOUCH_SPI); //Toss junk
00190
```

```
00191   while (SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_TXE)== RESET) ;
00192   SPI_I2S_SendData(TOUCH_SPI, 0x0);
00193   while(SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_RXNE) == RESET) ;
00194   ret = SPI_I2S_ReceiveData(TOUCH_SPI); //Data
00195
00196   while (SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_TXE)== RESET) ;
00197   SPI_I2S_SendData(TOUCH_SPI, 0x0);
00198   while(SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_RXNE) == RESET) ;
00199   SPI_I2S_ReceiveData(TOUCH_SPI); //Test toss
00200
00201   return ret;
00202 }
00203
00209 uint16_t Blox_Touch_GetZ1(int numTouch) {
00210   uint16_t ret = 0;
00211
00212   //Chip-select is active low.
00213   TOUCH_CS_GPIO->ODR |= (TOUCH1_CS_PIN | TOUCH2_CS_PIN | TOUCH3_CS_PIN
00214                          | TOUCH4_CS_PIN);
00215
00216   switch(numTouch){
00217         case 1: TOUCH_CS_GPIO->ODR &= ~TOUCH1_CS_PIN;
00218                         break;
00219         case 2: TOUCH_CS_GPIO->ODR &= ~TOUCH2_CS_PIN;
00220                         break;
00221         case 3: TOUCH_CS_GPIO->ODR &= ~TOUCH3_CS_PIN;
00222                         break;
00223         case 4: TOUCH_CS_GPIO->ODR &= ~TOUCH4_CS_PIN;
00224                         break;
00225         }
00226
00227   if (SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_RXNE) == SET)
00228     SPI_I2S_ReceiveData(TOUCH_SPI);
00229
00230   while (SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_TXE)== RESET) ;
00231   SPI_I2S_SendData(TOUCH_SPI, TOUCH_CTL_Z1);
00232   while(SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_RXNE) == RESET) ;
00233   SPI_I2S_ReceiveData(TOUCH_SPI); //Toss junk
00234
00235   while (SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_TXE)== RESET) ;
00236   SPI_I2S_SendData(TOUCH_SPI, 0x0);
00237   while(SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_RXNE) == RESET) ;
00238   ret = SPI_I2S_ReceiveData(TOUCH_SPI); //Data
00239
00240   while (SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_TXE)== RESET) ;
00241   SPI_I2S_SendData(TOUCH_SPI, 0x0);
00242   while(SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_RXNE) == RESET) ;
00243   SPI_I2S_ReceiveData(TOUCH_SPI); //Test toss
00244
00245   return ret;
00246 }
00247
00253 uint16_t Blox_Touch_GetZ2(int numTouch) {
00254   uint16_t ret = 0;
00255
00256   //Chip-select is active low.
00257   TOUCH_CS_GPIO->ODR |= (TOUCH1_CS_PIN | TOUCH2_CS_PIN | TOUCH3_CS_PIN
```

```
00258                         | TOUCH4_CS_PIN);
00259
00260   switch(numTouch){
00261         case 1: TOUCH_CS_GPIO->ODR &= ~TOUCH1_CS_PIN;
00262                         break;
00263         case 2: TOUCH_CS_GPIO->ODR &= ~TOUCH2_CS_PIN;
00264                         break;
00265         case 3: TOUCH_CS_GPIO->ODR &= ~TOUCH3_CS_PIN;
00266                         break;
00267         case 4: TOUCH_CS_GPIO->ODR &= ~TOUCH4_CS_PIN;
00268                         break;
00269         }
00270
00271   if (SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_RXNE) == SET)
00272     SPI_I2S_ReceiveData(TOUCH_SPI);
00273
00274   while (SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_TXE)== RESET) ;
00275   SPI_I2S_SendData(TOUCH_SPI, TOUCH_CTL_Z2);
00276   while(SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_RXNE) == RESET) ;
00277   SPI_I2S_ReceiveData(TOUCH_SPI); //Toss junk
00278
00279   while (SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_TXE)== RESET) ;
00280   SPI_I2S_SendData(TOUCH_SPI, 0x0);
00281   while(SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_RXNE) == RESET) ;
00282   ret = SPI_I2S_ReceiveData(TOUCH_SPI); //Data
00283
00284   while (SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_TXE)== RESET) ;
00285   SPI_I2S_SendData(TOUCH_SPI, 0x0);
00286   while(SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_RXNE) == RESET) ;
00287   SPI_I2S_ReceiveData(TOUCH_SPI); //Test toss
00288
00289   return ret;
00290 }
00291
00297 void Touch_SPI_Send(uint16_t data) {
00298   while (SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_TXE)== RESET) ;
00299   SPI_I2S_SendData(TOUCH_SPI, data);
00300   //USB_SendPat("Sent data %d\r\n", data);
00301   return;
00302 }
00303
00308 uint16_t Touch_SPI_Receive(void) {
00309   uint16_t debugger =0;
00310   while(SPI_I2S_GetFlagStatus(TOUCH_SPI, SPI_I2S_FLAG_RXNE) == RESET) ;
00311   debugger = SPI_I2S_ReceiveData(TOUCH_SPI);
00312     //USB_SendPat("Recieved data %d\r\n", debugger);
00313   return debugger;
00314 }
00315
```

6.61 drivers/src/blox_usart.c File Reference

A very basic wrapper around the USARTs on the STM32F103.

```
#include "blox_usart.h"
```

- void(∗ **USARTn_Handler** [5])(void) = {NULL}
  Array of handlers to call on interrupt.
- void **Blox_USART_DeInit_USART** (void)
  De-initializes all the USART interfaces.
- void **Blox_USART_DeInit_GPIO** (void)
  De-initializes the GPIOs for all the USART interfaces.
- void **Blox_USART_RCC_Configuration** (uint8_t id)
  Initializes clocks for the given the USART interface.
- void **Blox_USART_GPIO_Configuration** (uint8_t id)
  Initializes the gpios for the given the USART interface.
- void **Blox_USART_NVIC_Configuration** (uint8_t id)
  Initializes the NVIC for the given the USART interface.
- void **Blox_USART_Init** (uint8_t id)
  Initializes the USART module.
- uint8_t **Blox_USART_Receive** (uint8_t id)
  Receive a byte on the given USART.
- int16_t **Blox_USART_TryReceive** (uint8_t id)
  Receive a byte on the given USART.
- void **Blox_USART_Send** (uint8_t id, uint8_t data)
  Sends a byte out on the given USART.
- void **Blox_USART_Register_RXNE_IRQ** (uint8_t id, void(∗RXNE_Handler)(void))
  Registers a USART Interrupt on RXNE.
- void **Blox_USART_Release_RXNE_IRQ** (uint8_t id)
  Releases a USART Interrupt on RXNE.
- void **Blox_USART_Enable_RXNE_IRQ** (uint8_t id)
  Disables the USART Interrupt on RXNE.

- void **Blox_USART_Disable_RXNE_IRQ** (uint8_t id)
  Disables the USART Interrupt on RXNE.
- void **USART1_IRQHandler** (void)
  This function handles USART1 interrupt request.
- void **USART2_IRQHandler** (void)
  This function handles USART2 interrupt request.
- void **USART3_IRQHandler** (void)
  This function handles USART3 interrupt request.
- void **UART4_IRQHandler** (void)
  This function handles UART4 interrupt request.
- void **UART5_IRQHandler** (void)
  This function handles UART5 interrupt request.

6.61.1 Detailed Description

A very basic wrapper around the USARTs on the STM32F103.

Author

Jesse Tannahill

Version

V0.1

Date

10/18/2010

Definition in file **blox_usart.c**.

6.61.2 Function Documentation

6.61.2.1 void Blox_USART_DeInit_GPIO (void)

De-initializes the GPIOs for all the USART interfaces.

Return values

None	

Definition at line **99** of file **blox_usart.c**.

6.61.2.2 void Blox_USART_Disable_RXNE_IRQ (uint8_t *id*)

Disables the USART Interrupt on RXNE.

Parameters

id	the USART id to use.

Return values

The	current status of the USART.

Definition at line **377** of file **blox_usart.c**.

6.61.2.3 void Blox_USART_Enable_RXNE_IRQ (uint8_t *id*)

Disables the USART Interrupt on RXNE.

Parameters

id	the USART id to use.

Return values

The	current status of the USART.

Definition at line **352** of file **blox_usart.c**.

6.61.2.4 void Blox_USART_GPIO_Configuration (uint8_t *id*)

Initializes the gpios for the given the USART interface.

Parameters

id	the id of the USART interface.

Return values

None	

Definition at line **143** of file **blox_usart.c**.

6.61.2.5 void Blox_USART_Init (uint8_t *id*)

Initializes the USART module.

Parameters

id,:	the id of the USART interface.

Return values

None	

Definition at line **32** of file **blox_usart.c**.

6.61.2.6 void Blox_USART_NVIC_Configuration (uint8_t *id*)

Initializes the NVIC for the given the USART interface.

Parameters

id	the id of the USART interface.

Return values

None	

Definition at line **202** of file **blox_usart.c**.

6.61.2.7 void Blox_USART_RCC_Configuration (uint8_t *id*)

Initializes clocks for the given the USART interface.

Parameters

id	the id of the USART interface.

Return values

None	

Definition at line **112** of file **blox_usart.c**.

6.61.2.8 uint8_t Blox_USART_Receive (uint8_t *id*)

Receive a byte on the given USART.

Parameters

id	the USART id to use.

Return values

The	received byte.

Definition at line **233** of file **blox_usart.c**.

6.61.2.9 void Blox_USART_Register_RXNE_IRQ (uint8_t *id,* void(∗)(void) *RXNE_Handler*)

Registers a USART Interrupt on RXNE.

Parameters

id	the USART id to use.
RXNE_Handler	the handler function for the USART RXNE IRQ.

Return values

The	current status of the USART.

Definition at line **327** of file **blox_usart.c**.

6.61.2.10 void Blox_USART_Release_RXNE_IRQ (uint8_t *id*)

Releases a USART Interrupt on RXNE.

Parameters

id	the USART id to use.

Return values

The	current status of the USART.

Definition at line **342** of file **blox_usart.c**.

6.61.2.11 void Blox_USART_Send (uint8_t *id*, uint8_t *data*)

Sends a byte out on the given USART.

Parameters

id	the USART id to use
data	the byte to send

Return values

None.	

Definition at line **296** of file **blox_usart.c**.

6.61.2.12 int16_t Blox_USART_TryReceive (uint8_t *id*)

Receive a byte on the given USART.

Parameters

id	the USART id to use.

Return values

The	received byte.

Definition at line **259** of file **blox_usart.c**.

6.61.2.13 void UART4_IRQHandler (void)

This function handles UART4 interrupt request.

Return values

None	

Definition at line **445** of file **blox_usart.c**.

6.61.2.14 void UART5_IRQHandler (void)

This function handles UART5 interrupt request.

Return values

None	

Definition at line **459** of file **blox_usart.c**.

6.61.2.15 void USART1_IRQHandler (void)

This function handles USART1 interrupt request.

Return values

None	

Definition at line **403** of file **blox_usart.c**.

6.61.2.16 void USART2_IRQHandler (void)

This function handles USART2 interrupt request.

Return values

None	

Definition at line **417** of file **blox_usart.c**.

6.61.2.17 void USART3_IRQHandler (void)

This function handles USART3 interrupt request.

Return values

None	

Definition at line **431** of file **blox_usart.c**.

6.62 drivers/src/blox_usart.c

```
00001
00009 #include "blox_usart.h"
00010
00015 /* Private function prototypes */
00016 void Blox_USART_DeInit_USART(void);
00017 void Blox_USART_DeInit_GPIO(void);
00018 void Blox_USART_RCC_Configuration(uint8_t id);
00019 void Blox_USART_GPIO_Configuration(uint8_t id);
00020 void Blox_USART_NVIC_Configuration(uint8_t id);
00021
```

```
00025 void (*USARTn_Handler[5])(void) = {NULL};
00026
00032 void Blox_USART_Init(uint8_t id) {
00033   USART_InitTypeDef USART_InitStructure;
00034   Blox_USART_RCC_Configuration(id);
00035   Blox_USART_GPIO_Configuration(id);
00036
00037   USART_InitStructure.USART_BaudRate = 115200;
00038   USART_InitStructure.USART_WordLength = USART_WordLength_9b;
00039   USART_InitStructure.USART_StopBits = USART_StopBits_1;
00040   USART_InitStructure.USART_Parity = USART_Parity_Even;
00041   USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;

00042   USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
00043
00044   switch(id) {
00045   case 1:
00046     USART_Init(USART1, &USART_InitStructure);
00047     USART_Cmd(USART1, ENABLE);
00048     break;
00049   case 2: /* IR 3 */
00050     USART_Init(USART2, &USART_InitStructure);
00051     USART_Cmd(USART2, ENABLE);
00052     USART_SetPrescaler(USART2, 0x1);
00053     USART_IrDAConfig(USART2, USART_IrDAMode_LowPower);
00054           USART_IrDACmd(USART2, ENABLE);
00055     break;
00056   case 3: /* IR 4 */
00057     USART_Init(USART3, &USART_InitStructure);
00058     USART_Cmd(USART3, ENABLE);
00059     USART_SetPrescaler(USART3, 0x1);
00060     USART_IrDAConfig(USART3, USART_IrDAMode_LowPower);
00061           USART_IrDACmd(USART3, ENABLE);
00062     break;
00063   case 4: /* IR 1 */
00064     USART_Init(UART4, &USART_InitStructure);
00065     USART_Cmd(UART4, ENABLE);
00066     USART_SetPrescaler(UART4, 0x1);
00067     USART_IrDAConfig(UART4, USART_IrDAMode_LowPower);
00068           USART_IrDACmd(UART4, ENABLE);
00069     break;
00070   case 5: /* IR 2 */
00071     USART_Init(UART5, &USART_InitStructure);
00072     USART_Cmd(UART5, ENABLE);
00073     USART_SetPrescaler(UART5, 0x1);
00074     USART_IrDAConfig(UART5, USART_IrDAMode_LowPower);
00075           USART_IrDACmd(UART5, ENABLE);
00076     break;
00077   }
00078
00079   Blox_System_Register_DeInit(&RCC_DeInit);
00080   Blox_System_Register_DeInit(&Blox_USART_DeInit_USART);
00081   Blox_System_Register_DeInit(&Blox_USART_DeInit_GPIO);
00082 }
00083
00088 void Blox_USART_DeInit_USART(void) {
00089   USART_DeInit(USART1);
```

```
00090   USART_DeInit(USART2);
00091   USART_DeInit(USART3);
00092   USART_DeInit(UART4);
00093   USART_DeInit(UART5);
00094 }
00099 void Blox_USART_DeInit_GPIO(void) {
00100   GPIO_DeInit(USART2_GPIO);
00101   GPIO_DeInit(USART3_GPIO);
00102   GPIO_DeInit(UART4_GPIO);
00103   GPIO_DeInit(UART5_GPIO_RX);
00104   GPIO_DeInit(UART5_GPIO_TX);
00105 }
00106
00112 void Blox_USART_RCC_Configuration(uint8_t id) {
00113   switch(id) {
00114   case 1:
00115     RCC_APB2PeriphClockCmd(USART1_GPIO_CLK, ENABLE);
00116     RCC_APB2PeriphClockCmd(USART1_CLK, ENABLE);
00117     break;
00118   case 2:
00119     RCC_APB2PeriphClockCmd(USART2_GPIO_CLK, ENABLE);
00120     RCC_APB1PeriphClockCmd(USART2_CLK, ENABLE);
00121     break;
00122   case 3:
00123     RCC_APB2PeriphClockCmd(USART3_GPIO_CLK, ENABLE);
00124     RCC_APB1PeriphClockCmd(USART3_CLK, ENABLE);
00125     break;
00126   case 4:
00127     RCC_APB2PeriphClockCmd(UART4_GPIO_CLK, ENABLE);;
00128     RCC_APB1PeriphClockCmd(UART4_CLK, ENABLE);
00129     break;
00130   case 5:
00131     RCC_APB2PeriphClockCmd(UART5_GPIO_TX_CLK, ENABLE);
00132     RCC_APB2PeriphClockCmd(UART5_GPIO_RX_CLK, ENABLE);
00133     RCC_APB1PeriphClockCmd(UART5_CLK, ENABLE);
00134     break;
00135   }
00136 }
00137
00143 void Blox_USART_GPIO_Configuration(uint8_t id) {
00144 GPIO_InitTypeDef GPIO_InitStructure;
00145   //Set up Rx as Floating
00146   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
00147   switch(id) {
00148     case 1:
00149       GPIO_InitStructure.GPIO_Pin = USART1_RxPin;
00150       GPIO_Init(USART1_GPIO, &GPIO_InitStructure);
00151       break;
00152    case 2:
00153       GPIO_InitStructure.GPIO_Pin = USART2_RxPin;
00154       GPIO_Init(USART2_GPIO, &GPIO_InitStructure);
00155       break;
00156    case 3:
00157       GPIO_InitStructure.GPIO_Pin = USART3_RxPin;
00158       GPIO_Init(USART3_GPIO, &GPIO_InitStructure);
00159       break;
00160    case 4:
```

```
00161       GPIO_InitStructure.GPIO_Pin = UART4_RxPin;
00162       GPIO_Init(UART4_GPIO, &GPIO_InitStructure);
00163       break;
00164    case 5:
00165       GPIO_InitStructure.GPIO_Pin = UART5_RxPin;
00166       GPIO_Init(UART5_GPIO_RX, &GPIO_InitStructure);
00167       break;
00168   }
00169
00170   //Set Tx as 50Mhz and Floating
00171   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
00172   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
00173   switch(id) {
00174   case 1:
00175     GPIO_InitStructure.GPIO_Pin = USART1_TxPin;
00176     GPIO_Init(USART1_GPIO, &GPIO_InitStructure);
00177     break;
00178   case 2:
00179     GPIO_InitStructure.GPIO_Pin = USART2_TxPin;
00180     GPIO_Init(USART2_GPIO, &GPIO_InitStructure);
00181     break;
00182   case 3:
00183     GPIO_InitStructure.GPIO_Pin = USART3_TxPin;
00184     GPIO_Init(USART3_GPIO, &GPIO_InitStructure);
00185     break;
00186   case 4:
00187     GPIO_InitStructure.GPIO_Pin = UART4_TxPin;
00188     GPIO_Init(UART4_GPIO, &GPIO_InitStructure);
00189     break;
00190   case 5:
00191     GPIO_InitStructure.GPIO_Pin = UART5_TxPin;
00192     GPIO_Init(UART5_GPIO_TX, &GPIO_InitStructure);
00193     break;
00194   }
00195 }
00196
00202 void Blox_USART_NVIC_Configuration(uint8_t id) {
00203   NVIC_InitTypeDef NVIC_InitStructure;
00204   NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);
00205   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 10;
00206   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
00207   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
00208   switch(id) {
00209     case 1:
00210       NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
00211       break;
00212     case 2:
00213       NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
00214       break;
00215     case 3:
00216       NVIC_InitStructure.NVIC_IRQChannel = USART3_IRQn;
00217       break;
00218     case 4:
00219       NVIC_InitStructure.NVIC_IRQChannel = UART4_IRQn;
00220       break;
00221     case 5:
00222       NVIC_InitStructure.NVIC_IRQChannel = UART5_IRQn;
```

```
00223       break;
00224   }
00225   NVIC_Init(&NVIC_InitStructure);
00226 }
00227
00233 uint8_t Blox_USART_Receive(uint8_t id) {
00234   switch(id) {
00235   case 1:
00236     while(USART_GetFlagStatus(USART1, USART_FLAG_RXNE) == RESET) ;
00237     return USART_ReceiveData(USART1) & 0xFF;
00238   case 2:
00239     while(USART_GetFlagStatus(USART2, USART_FLAG_RXNE) == RESET) ;
00240     return USART_ReceiveData(USART2) & 0xFF;
00241   case 3:
00242     while(USART_GetFlagStatus(USART3, USART_FLAG_RXNE) == RESET) ;
00243     return USART_ReceiveData(USART3) & 0xFF;
00244   case 4:
00245     while(USART_GetFlagStatus( UART4, USART_FLAG_RXNE) == RESET) ;
00246     return USART_ReceiveData(UART4) & 0xFF;
00247   case 5:
00248     while(USART_GetFlagStatus( UART5, USART_FLAG_RXNE) == RESET) ;
00249     return USART_ReceiveData(UART5) & 0xFF;
00250   }
00251   return 0;
00252 }
00253
00259 int16_t Blox_USART_TryReceive(uint8_t id) {
00260   switch(id) {
00261   case 1:
00262     if (USART_GetFlagStatus(USART1, USART_FLAG_RXNE) == RESET)
00263       return -1;
00264     else
00265       return USART_ReceiveData(USART1) & 0xFF;
00266   case 2:
00267     if (USART_GetFlagStatus(USART2, USART_FLAG_RXNE) == RESET)
00268       return -1;
00269     else
00270       return USART_ReceiveData(USART2) & 0xFF;
00271   case 3:
00272     if (USART_GetFlagStatus(USART3, USART_FLAG_RXNE) == RESET)
00273       return -1;
00274     else
00275       return USART_ReceiveData(USART3) & 0xFF;
00276   case 4:
00277     if (USART_GetFlagStatus( UART4, USART_FLAG_RXNE) == RESET)
00278       return -1;
00279     else
00280       return USART_ReceiveData(UART4) & 0xFF;
00281   case 5:
00282     if (USART_GetFlagStatus( UART5, USART_FLAG_RXNE) == RESET)
00283       return -1;
00284     else
00285       return USART_ReceiveData(UART5) & 0xFF;
00286   }
00287   return -1;
00288 }
00289
```

```
00296 void Blox_USART_Send(uint8_t id, uint8_t data) {
00297   switch(id) {
00298   case 1:
00299         while (USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET) ;
00300     USART_SendData(USART1, data);
00301     break;
00302   case 2:
00303         while (USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RESET) ;
00304     USART_SendData(USART2, data);
00305     break;
00306   case 3:
00307           while (USART_GetFlagStatus(USART3, USART_FLAG_TXE) == RESET) ;
00308     USART_SendData(USART3, data);
00309     break;
00310   case 4:
00311         while (USART_GetFlagStatus( UART4, USART_FLAG_TXE) == RESET) ;
00312     USART_SendData(UART4, data);
00313     break;
00314   case 5:
00315         while (USART_GetFlagStatus( UART5, USART_FLAG_TXE) == RESET) ;
00316     USART_SendData(UART5, data);
00317     break;
00318   }
00319 }
00320
00327 void Blox_USART_Register_RXNE_IRQ(uint8_t id, void (*RXNE_Handler)(void)) {
00328   if(USARTn_Handler[id-1] == NULL) {
00329     USARTn_Handler[id-1] = RXNE_Handler;
00330   } else {
00331     //IRQ already set
00332   }
00333   Blox_USART_Disable_RXNE_IRQ(id);
00334   Blox_USART_NVIC_Configuration(id);
00335 }
00336
00342 void Blox_USART_Release_RXNE_IRQ(uint8_t id) {
00343   Blox_USART_Disable_RXNE_IRQ(id);
00344   USARTn_Handler[id-1] = NULL;
00345 }
00346
00352 void Blox_USART_Enable_RXNE_IRQ(uint8_t id) {
00353   switch(id) {
00354     case 1:
00355       USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
00356       break;
00357     case 2:
00358       USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);
00359       break;
00360     case 3:
00361       USART_ITConfig(USART3, USART_IT_RXNE, ENABLE);
00362       break;
00363     case 4:
00364       USART_ITConfig(UART4, USART_IT_RXNE, ENABLE);
00365       break;
00366     case 5:
00367       USART_ITConfig(UART5, USART_IT_RXNE, ENABLE);
00368       break;
```

```
00369   }
00370 }
00371
00377 void Blox_USART_Disable_RXNE_IRQ(uint8_t id) {
00378   switch(id) {
00379     case 1:
00380       USART_ITConfig(USART1, USART_IT_RXNE, DISABLE);
00381       break;
00382     case 2:
00383       USART_ITConfig(USART2, USART_IT_RXNE, DISABLE);
00384       break;
00385     case 3:
00386       USART_ITConfig(USART3, USART_IT_RXNE, DISABLE);
00387       break;
00388     case 4:
00389       USART_ITConfig(UART4, USART_IT_RXNE, DISABLE);
00390       break;
00391     case 5:
00392       USART_ITConfig(UART5, USART_IT_RXNE, DISABLE);
00393       break;
00394   }
00395 }
00396
00397
00398
00403 void USART1_IRQHandler(void)
00404 {
00405   if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
00406   {
00407     if(USARTn_Handler[0] != NULL) {
00408       (*USARTn_Handler[0])();
00409     }
00410   }
00411 }
00412
00417 void USART2_IRQHandler(void)
00418 {
00419   if(USART_GetITStatus(USART2, USART_IT_RXNE) != RESET)
00420   {
00421     if(USARTn_Handler[1] != NULL) {
00422       (*USARTn_Handler[1])();;
00423     }
00424   }
00425 }
00426
00431 void USART3_IRQHandler(void)
00432 {
00433   if(USART_GetITStatus(USART3, USART_IT_RXNE) != RESET)
00434   {
00435     if(USARTn_Handler[2] != NULL) {
00436       (*USARTn_Handler[2])();;
00437     }
00438   }
00439 }
00440
00445 void UART4_IRQHandler(void)
00446 {
```

```
00447   if(USART_GetITStatus(UART4, USART_IT_RXNE) != RESET)
00448   {
00449     if(USARTn_Handler[3] != NULL) {
00450       (*USARTn_Handler[3])();;
00451     }
00452   }
00453 }
00454
00459 void UART5_IRQHandler(void)
00460 {
00461   if(USART_GetITStatus(UART5, USART_IT_RXNE) != RESET)
00462   {
00463     if(USARTn_Handler[4] != NULL) {
00464       (*USARTn_Handler[4])();;
00465     }
00466   }
00467 }
```

6.63 drivers/src/blox_usb.c File Reference

A wrapper class for the USB interface that uses USART.

```
#include "blox_usb.h"
```

- void **USB_Init** (void)

  Initializes the USB module. Basically a wrapper on USART.

- uint8_t **USB_Receive** (void)

  Blocking receive of a byte over USB. A wrapper around USART.

- int16_t **USB_TryReceive** (void)

  Non-blocking receive of a byte over USB. A wrapper around USART.

- void **USB_Send** (uint8_t data)

  Sends a byte over USB. A wrapper around USART.

- void **USB_SendData** (uint8_t *data, uint32_t len)

  Sends len bytes over USB. A wrapper around USART.

- void **USB_SendPat** (char *format,...)

  Sends a string based on pattern passed over USB. A wrapper around USART.

6.63.1 Detailed Description

A wrapper class for the USB interface that uses USART.

Author

Jesse Tannahill

Version

V0.1

Date

10/19/2010

Definition in file **blox_usb.c**.

6.64 drivers/src/blox_usb.c

```
00001
00009 #include "blox_usb.h"
00018 static uint8_t usb_init = 0;
00019
00024 void USB_Init(void) {
00025   if(!usb_init) {
00026     usb_init = 1;
00027     Blox_USART_Init(USB_USART_ID);
00028   }
00029 }
00030
00036 uint8_t USB_Receive(void) {
00037     return Blox_USART_Receive(USB_USART_ID);
00038 }
00039
00045 int16_t USB_TryReceive(void) {
00046     return Blox_USART_TryReceive(USB_USART_ID);
00047 }
00048
00054 void USB_Send(uint8_t data) {
00055   Blox_USART_Send(USB_USART_ID, data);
00056 }
00057
00064 void USB_SendData(uint8_t *data, uint32_t len) {
00065   while(len--)
00066     USB_Send(*data++);
00067 }
00068
00076 void USB_SendPat(char *format, ...) {
00077   va_list args;
00078   char buffer[100];
00079   va_start(args, format);
```

```
00080
00081   vsprintf(buffer, format, args);
00082   USB_SendData((uint8_t *)buffer, strlen(buffer));
00083 }
00084
```

6.65 drivers/src/blox_vusart.c File Reference

A virtual USART driver for the STM32F103.

```
#include "blox_vusart.h"
```

- **TIMER_ID VUSART1_RxDataID** = INVALID_TIMER
- **TIMER_ID VUSART1_TxDataID** = INVALID_TIMER
- **TIMER_ID VUSART2_RxDataID** = INVALID_TIMER
- **TIMER_ID VUSART2_TxDataID** = INVALID_TIMER
- **EXTI_ID VUSART1_RxStartID** = EXTI_INVALID_LINE
- **EXTI_ID VUSART2_RxStartID** = EXTI_INVALID_LINE
- **EXTI_ID VUSART1_RXNE_IRQ_ID** = EXTI_INVALID_LINE
- uint8_t **VUSART1_RXNE_IRQ_Enable** = FALSE
- **EXTI_ID VUSART2_RXNE_IRQ_ID** = EXTI_INVALID_LINE
- uint8_t **VUSART2_RXNE_IRQ_Enable** = FALSE
- uint16_t **VUSART1_BaudRate**
- uint16_t **VUSART1_DoubleBaudRate**
- uint8_t **VUSART1_TxDataRegister**
- uint8_t **VUSART1_RxDataRegister**
- uint16_t **VUSART2_BaudRate**
- uint16_t **VUSART2_DoubleBaudRate**
- uint8_t **VUSART2_TxDataRegister**
- uint8_t **VUSART2_RxDataRegister**
- uint8_t **VUSART1_TxComplete**
- uint8_t **VUSART1_TxEmpty**
- uint8_t **VUSART1_RxNotEmpty**
- uint8_t **VUSART2_TxComplete**
- uint8_t **VUSART2_TxEmpty**
- uint8_t **VUSART2_RxNotEmpty**
- uint16_t **VUSART1_RxError**
- uint16_t **VUSART2_RxError**
- void **Blox_VUSART_RCC_Configuration** (uint8_t id)

  Initializes clocks for the given the virtual USART interface.

- void **Blox_VUSART_GPIO_Configuration** (uint8_t id)

  Initializes the gpios for the given the virtual USART interface.

- void **VUSART1_RxStart** (void)

  Turns on a timer for VUSART1 which samples at the specified baud rate when a falling edge is received.

- void **VUSART1_RxData** (void)

  Samples incoming data at the specified baud rate for VUSART1.

- void **VUSART1_TxData** (void)

  Outputs data at the specified baud rate for VUSART1.

- void **VUSART2_RxStart** (void)

  Turns on a timer for VUSART2 which samples at the specified baud rate when a falling edge is received.

- void **VUSART2_RxData** (void)

  Samples incoming data at the specified baud rate for VUSART2.

- void **VUSART2_TxData** (void)

  Outputs data at the specified baud rate for VUSART1.

- void **Blox_VUSART_Init** (uint8_t id)

  Initializes the virtual USART module.

- void **Blox_VUSART_SetBaudrate** (uint8_t id, uint16_t baudrate)

  Sets the baudrate of the given ID.

- **VUSART_STATUS Blox_VUSART_TryReceive** (uint8_t id, uint8_t ∗data)

  Tries to receive a byte on the given virtual USART.

- **VUSART_STATUS Blox_VUSART_TrySend** (uint8_t id, uint8_t data)

  Tries to send a byte out on the given virtual USART.

- **VUSART_STATUS Blox_VUSART_Receive** (uint8_t id, uint8_t ∗data)

  Receives a blocking byte on the given virtual USART.

- **VUSART_STATUS Blox_VUSART_Send** (uint8_t id, uint8_t data)

  Sends a blocking byte out on the given virtual USART.

- **VUSART_STATUS Blox_VUSART_SendData** (uint8_t id, uint8_t ∗data, uint32_t len)

Sends a blocking byte out on the given virtual USART.

- **VUSART_STATUS Blox_VUSART_Register_RXNE_IRQ** (uint8_t id, void(∗RXNE_Handler)(void))

  Registers a function to be called in the SWInterrupt that occurs when a receive happens.

- **VUSART_STATUS Blox_VUSART_Enable_RXNE_IRQ** (uint8_t id)

  Enables the SW Interrupt on RXNE.

- **VUSART_STATUS Blox_VUSART_Disable_RXNE_IRQ** (uint8_t id)

  Disables the SW Interrupt on RXNE.

6.65.1 Detailed Description

A virtual USART driver for the STM32F103.

Author

Zach Wasson

Version

V0.1

Date

11/01/2010

Definition in file **blox_vusart.c**.

6.65.2 Function Documentation

6.65.2.1 VUSART_STATUS Blox_VUSART_Disable_RXNE_IRQ (uint8_t *id*)

Disables the SW Interrupt on RXNE.

Parameters

id	the virtual USART id to use.

Return values

The	current status of the VUSART.

Definition at line **479** of file **blox_vusart.c**.

6.65.2.2 VUSART_STATUS Blox_VUSART_Enable_RXNE_IRQ (uint8_t *id*)

Enables the SW Interrupt on RXNE.

Parameters

id	the virtual USART id to use.

Return values

The	current status of the VUSART.

Definition at line **461** of file **blox_vusart.c**.

6.65.2.3 void Blox_VUSART_GPIO_Configuration (uint8_t *id*)

Initializes the gpios for the given the virtual USART interface.

Parameters

id	the id of the virtual USART interface.

Return values

None	

Definition at line **294** of file **blox_vusart.c**.

6.65.2.4 void Blox_VUSART_Init (uint8_t *id*)

Initializes the virtual USART module.

Parameters

id	the id of the virtual USART interface.

Return values

None	

Definition at line **61** of file **blox_vusart.c**.

6.65.2.5 VUSART_STATUS Blox_VUSART_Receive (uint8_t *id,* uint8_t * *data*)

Receives a blocking byte on the given virtual USART.

Parameters

id	the virtual USART id to use.
data	a pointer to the location the data will be returned

Return values

The	current status of the VUSART.

Definition at line **400** of file **blox_vusart.c**.

6.65.2.6 VUSART_STATUS Blox_VUSART_Register_RXNE_IRQ (uint8_t *id,* void(*)(void) *RXNE_Handler*)

Registers a function to be called in the SWInterrupt that occurs when a receive happens.

Parameters

id	the virtual USART id to use.
RXNE_Handler	the function to be called.

Return values

The	current status of the VUSART.

Definition at line **439** of file **blox_vusart.c**.

6.65.2.7 VUSART_STATUS Blox_VUSART_Send (uint8_t *id,* uint8_t *data*)

Sends a blocking byte out on the given virtual USART.

Parameters

id	the virtual USART id to use
data	the byte to send

Return values

The	current status of the VUSART.

Definition at line **411** of file **blox_vusart.c**.

6.65.2.8 VUSART_STATUS Blox_VUSART_SendData (uint8_t *id,* uint8_t * *data,* uint32_t *len*)

Sends a blocking byte out on the given virtual USART.

Parameters

id	the virtual USART id to use
data	the byte to send
len	the length of the data

Return values

The	current status of the VUSART.

Definition at line **423** of file **blox_vusart.c**.

6.65.2.9 void Blox_VUSART_SetBaudrate (uint8_t *id,* uint16_t *baudrate*)

Sets the baudrate of the given ID.

Parameters

id	the id of the virtual USART interface./
baudrate	the baudrate to set to (_9600, etc)

Return values

None	

Definition at line **102** of file **blox_vusart.c**.

6.65.2.10 VUSART_STATUS Blox_VUSART_TryReceive (uint8_t *id,* uint8_t * *data*)

Tries to receive a byte on the given virtual USART.

Parameters

id	the virtual USART id to use.
data	a pointer to the location the data will be returned

Return values

The	current status of the VUSART.

Definition at line **329** of file **blox_vusart.c**.

6.65.2.11 VUSART_STATUS Blox_VUSART_TrySend (uint8_t *id,* uint8_t *data*)

Tries to send a byte out on the given virtual USART.

Parameters

id	the virtual USART id to use
data	the byte to send

Return values

The	current status of the VUSART.

Definition at line **363** of file **blox_vusart.c**.

6.65.2.12 void VUSART1_RxData (void)

Samples incoming data at the specified baud rate for VUSART1.

Return values

None	

Definition at line **132** of file **blox_vusart.c**.

6.65.2.13 void VUSART1_RxStart (void)

Turns on a timer for VUSART1 which samples at the specified baud rate when a falling edge is received.

Return values

None	

Definition at line **119** of file **blox_vusart.c**.

6.65.2.14 void VUSART1_TxData (void)

Outputs data at the specified baud rate for VUSART1.

Return values

None	

Definition at line **167** of file **blox_vusart.c**.

6.65.2.15 void VUSART2_RxData (void)

Samples incoming data at the specified baud rate for VUSART2.

Return values

None	

Definition at line **209** of file **blox_vusart.c**.

6.65.2.16 void VUSART2_RxStart (void)

Turns on a timer for VUSART2 which samples at the specified baud rate when a falling edge is received.

Return values

None	

Definition at line **198** of file **blox_vusart.c**.

6.65.2.17 void VUSART2_TxData (void)

Outputs data at the specified baud rate for VUSART1.

Return values

None	

Definition at line **244** of file **blox_vusart.c**.

6.66 drivers/src/blox_vusart.c

```
00001
00009 #include "blox_vusart.h"
00010
00015 /* Private function prototypes */
00016 void Blox_VUSART_RCC_Configuration(uint8_t id);
00017 void Blox_VUSART_GPIO_Configuration(uint8_t id);
00018
00019 TIMER_ID VUSART1_RxDataID = INVALID_TIMER;
00020 TIMER_ID VUSART1_TxDataID = INVALID_TIMER;
```

```
00021 TIMER_ID VUSART2_RxDataID = INVALID_TIMER;
00022 TIMER_ID VUSART2_TxDataID = INVALID_TIMER;
00023 EXTI_ID VUSART1_RxStartID = EXTI_INVALID_LINE;
00024 EXTI_ID VUSART2_RxStartID = EXTI_INVALID_LINE;
00025 EXTI_ID VUSART1_RXNE_IRQ_ID = EXTI_INVALID_LINE;
00026 uint8_t VUSART1_RXNE_IRQ_Enable = FALSE;
00027 EXTI_ID VUSART2_RXNE_IRQ_ID = EXTI_INVALID_LINE;
00028 uint8_t VUSART2_RXNE_IRQ_Enable = FALSE;
00029
00030 void VUSART1_RxStart(void);
00031 void VUSART1_RxData(void);
00032 void VUSART1_TxData(void);
00033 void VUSART2_RxStart(void);
00034 void VUSART2_RxData(void);
00035 void VUSART2_TxData(void);
00036
00037 uint16_t VUSART1_BaudRate, VUSART1_DoubleBaudRate;
00038 uint8_t VUSART1_TxDataRegister;
00039 uint8_t VUSART1_RxDataRegister;
00040
00041 uint16_t VUSART2_BaudRate, VUSART2_DoubleBaudRate;
00042 uint8_t VUSART2_TxDataRegister;
00043 uint8_t VUSART2_RxDataRegister;
00044
00045 uint8_t VUSART1_TxComplete;      //TC
00046 uint8_t VUSART1_TxEmpty;         //TXE
00047 uint8_t VUSART1_RxNotEmpty;      //RXNE
00048
00049 uint8_t VUSART2_TxComplete;      //TC
00050 uint8_t VUSART2_TxEmpty;         //TXE
00051 uint8_t VUSART2_RxNotEmpty;      //RXNE
00052
00053 uint16_t VUSART1_RxError;
00054 uint16_t VUSART2_RxError;
00055
00061 void Blox_VUSART_Init(uint8_t id) {
00062   Blox_VUSART_RCC_Configuration(id);
00063   Blox_VUSART_GPIO_Configuration(id);
00064   Blox_Timer_Init(VUSART_TIMx, VUSART_TIM_CLK);
00065   NVIC_SetPriority(VUSART_TIM_IRQn, 1);
00066   Blox_EXTI_Init();
00067
00068   switch(id) {
00069   case 1: /* XBee */
00070     //initialize data and flags
00071     VUSART1_BaudRate = _9600bps;
00072     VUSART1_DoubleBaudRate = VUSART1_BaudRate / 2;
00073     VUSART1_TxEmpty = 1;
00074     VUSART1_TxComplete = 1;
00075     /* set Tx high while idle */
00076     VUSART1_GPIO->ODR |= (VUSART1_TxPin);
00077     VUSART1_RxDataID = Blox_Timer_Register_IRQ(VUSART_TIMx, VUSART1_DoubleBaudRat
      e, &VUSART1_RxData, DISABLE);
00078     VUSART1_TxDataID = Blox_Timer_Register_IRQ(VUSART_TIMx, VUSART1_BaudRate, &
      VUSART1_TxData, DISABLE);
00079     VUSART1_RxStartID = Blox_EXTI_Register_HW_IRQ(VUSART1_RxPortSource, VUSART1_R
      xPinSource, &VUSART1_RxStart);
```

```
00080     break;
00081   case 2: /* OLED Display */
00082     //initialize data and flags
00083     VUSART2_BaudRate = _38400bps;
00084     VUSART2_DoubleBaudRate = VUSART2_BaudRate / 2;
00085     VUSART2_TxEmpty = 1;
00086     VUSART2_TxComplete = 1;
00087     /* set Tx high while idle */
00088     VUSART2_GPIO->ODR |= (VUSART2_TxPin);
00089     VUSART2_RxDataID = Blox_Timer_Register_IRQ(VUSART_TIMx, VUSART2_DoubleBaudRat
      e, &VUSART2_RxData, DISABLE);
00090     VUSART2_TxDataID = Blox_Timer_Register_IRQ(VUSART_TIMx, VUSART2_BaudRate, &
      VUSART2_TxData, DISABLE);
00091     VUSART2_RxStartID = Blox_EXTI_Register_HW_IRQ(VUSART2_RxPortSource, VUSART2_R
      xPinSource, &VUSART2_RxStart);
00092     break;
00093   }
00094 }
00095
00102 void Blox_VUSART_SetBaudrate(uint8_t id, uint16_t baudrate) {
00103   switch(id) {
00104   case 1: /* XBee */
00105     VUSART1_BaudRate = baudrate;
00106     VUSART1_DoubleBaudRate = VUSART1_BaudRate / 2;
00107     break;
00108   case 2: /* OLED Display */
00109     VUSART2_BaudRate = baudrate;
00110     VUSART2_DoubleBaudRate = VUSART2_BaudRate / 2;
00111     break;
00112   }
00113 }
00114
00119 void VUSART1_RxStart(void) {
00120   if(VUSART1_RxNotEmpty == 0) {
00121     Blox_Timer_Enable_IRQ(VUSART1_RxDataID);
00122     Blox_EXTI_Disable_IRQ(VUSART1_RxStartID);
00123   } else if (VUSART1_RXNE_IRQ_ID != EXTI_INVALID_LINE && VUSART1_RXNE_IRQ_ID != E
      XTI_IRQ_UNAVAILABLE && VUSART1_RXNE_IRQ_Enable == TRUE) {
00124     Blox_EXTI_Trigger_SW_IRQ(VUSART1_RXNE_IRQ_ID);
00125   }
00126 }
00127
00132 void VUSART1_RxData(void) {
00133   static uint8_t bit_num;
00134   static uint8_t data;
00135   if(bit_num == 0) {
00136     Blox_Timer_Modify_IRQ(VUSART1_RxDataID, VUSART1_BaudRate);
00137     /* error if start bit != 0 */
00138     if((VUSART1_GPIO->IDR & VUSART1_RxPin) >> VUSART1_RxPinSource) {
00139       VUSART1_RxError++;
00140       Blox_Timer_Disable_IRQ(VUSART1_RxDataID);
00141       Blox_Timer_Modify_IRQ(VUSART1_RxDataID, VUSART1_DoubleBaudRate);
00142       Blox_EXTI_Enable_IRQ(VUSART1_RxStartID);
00143     }
00144   }
00145   else if(bit_num <= 8) {
00146     uint8_t tmp = ((VUSART1_GPIO->IDR & VUSART1_RxPin)>>VUSART1_RxPinSource);
```

```
00147     data |= (tmp << (bit_num-1));
00148     if(bit_num == 8) {
00149       VUSART1_RxDataRegister = data;
00150       VUSART1_RxNotEmpty = 1;
00151       data = 0;
00152       Blox_Timer_Disable_IRQ(VUSART1_RxDataID);
00153       Blox_Timer_Modify_IRQ(VUSART1_RxDataID, VUSART1_DoubleBaudRate);
00154       Blox_EXTI_Enable_IRQ(VUSART1_RxStartID);
00155       if(VUSART1_RXNE_IRQ_ID != EXTI_INVALID_LINE && VUSART1_RXNE_IRQ_ID != EXTI_
      IRQ_UNAVAILABLE &&
00156          VUSART1_RXNE_IRQ_Enable == TRUE)
00157         Blox_EXTI_Trigger_SW_IRQ(VUSART1_RXNE_IRQ_ID);
00158     }
00159   }
00160   bit_num = (bit_num + 1) % 9;
00161 }
00162
00167 void VUSART1_TxData(void) {
00168   static uint8_t bit_num;
00169   static uint8_t data;
00170   //static uint8_t parity;
00171   if(bit_num == 0) {
00172     data = VUSART1_TxDataRegister;
00173     VUSART1_TxEmpty = 1;
00174     /* start bit = 0 */
00175     VUSART1_GPIO->ODR &= ~(VUSART1_TxPin);
00176   }
00177   else if(bit_num <= 8) {
00178     if((data >> (bit_num - 1)) & 0x01) {
00179       VUSART1_GPIO->ODR |= (VUSART1_TxPin);
00180     }
00181     else {
00182       VUSART1_GPIO->ODR &= ~(VUSART1_TxPin);
00183     }
00184   }
00185   else if(bit_num == 9) {
00186     /* stop bit = 1 */
00187     VUSART1_GPIO->ODR |= (VUSART1_TxPin);
00188     VUSART1_TxComplete = 1;
00189     Blox_Timer_Disable_IRQ(VUSART1_TxDataID);
00190   }
00191   bit_num = (bit_num + 1) % 10;
00192 }
00193
00198 void VUSART2_RxStart(void) {
00199   if(VUSART2_RxNotEmpty == 0) {
00200     Blox_Timer_Enable_IRQ(VUSART2_RxDataID);
00201     Blox_EXTI_Disable_IRQ(VUSART2_RxStartID);
00202   }
00203 }
00204
00209 void VUSART2_RxData(void) {
00210   static uint8_t bit_num;
00211   static uint8_t data;
00212   if(bit_num == 0) {
00213     Blox_Timer_Modify_IRQ(VUSART2_RxDataID, VUSART2_BaudRate);
00214     /* error if start bit != 0 */
```

```
00215     if((VUSART2_GPIO->IDR & VUSART2_RxPin) >> VUSART2_RxPinSource) {
00216       VUSART2_RxError++;
00217       Blox_EXTI_Enable_IRQ(VUSART2_RxStartID);
00218       Blox_Timer_Disable_IRQ(VUSART2_RxDataID);
00219       Blox_Timer_Modify_IRQ(VUSART2_RxDataID, VUSART2_DoubleBaudRate);
00220     }
00221   }
00222   else if(bit_num <= 8) {
00223     uint8_t tmp = ((VUSART2_GPIO->IDR & VUSART2_RxPin)>>VUSART2_RxPinSource);
00224     data |= (tmp << (bit_num-1));
00225     if(bit_num == 8) {
00226       VUSART2_RxDataRegister = data;
00227       VUSART2_RxNotEmpty = 1;
00228       data = 0;
00229       Blox_Timer_Disable_IRQ(VUSART2_RxDataID);
00230       Blox_Timer_Modify_IRQ(VUSART2_RxDataID, VUSART2_DoubleBaudRate);
00231       Blox_EXTI_Enable_IRQ(VUSART2_RxStartID);
00232       if(VUSART2_RXNE_IRQ_ID != EXTI_INVALID_LINE && VUSART2_RXNE_IRQ_ID != EXTI_
      IRQ_UNAVAILABLE &&
00233          VUSART2_RXNE_IRQ_Enable == TRUE)
00234         Blox_EXTI_Trigger_SW_IRQ(VUSART2_RXNE_IRQ_ID);
00235     }
00236   }
00237   bit_num = (bit_num + 1) % 9;
00238 }
00239
00244 void VUSART2_TxData(void) {
00245   static uint8_t bit_num;
00246   static uint8_t data;
00247   //static uint8_t parity;
00248   if(bit_num == 0) {
00249     data = VUSART2_TxDataRegister;
00250     VUSART2_TxEmpty = 1;
00251     /* start bit = 0 */
00252     VUSART2_GPIO->ODR &= ~(VUSART2_TxPin);
00253   }
00254   else if(bit_num <= 8) {
00255     if((data >> (bit_num - 1)) & 0x01) {
00256       VUSART2_GPIO->ODR |= (VUSART2_TxPin);
00257     }
00258     else {
00259       VUSART2_GPIO->ODR &= ~(VUSART2_TxPin);
00260     }
00261   }
00262   else if(bit_num == 9) {
00263     /* stop bit = 1 */
00264     VUSART2_GPIO->ODR |= (VUSART2_TxPin);
00265     VUSART2_TxComplete = 1;
00266     Blox_Timer_Disable_IRQ(VUSART2_TxDataID);
00267   }
00268   bit_num = (bit_num + 1) % 10;
00269 }
00270
00276 void Blox_VUSART_RCC_Configuration(uint8_t id) {
00277   switch(id) {
00278   case 1:
00279     RCC_APB2PeriphClockCmd(VUSART1_GPIO_CLK, ENABLE);
```

```
00280     //RCC_APB2PeriphClockCmd(VUSART1_CLK, ENABLE);
00281     break;
00282   case 2:
00283     RCC_APB2PeriphClockCmd(VUSART2_GPIO_CLK, ENABLE);
00284     //RCC_APB1PeriphClockCmd(VUSART2_CLK, ENABLE);
00285     break;
00286   }
00287 }
00288
00294 void Blox_VUSART_GPIO_Configuration(uint8_t id) {
00295   GPIO_InitTypeDef GPIO_InitStructure;
00296   //Set Rx as Floating input
00297   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
00298   switch(id) {
00299     case 1:
00300       GPIO_InitStructure.GPIO_Pin = VUSART1_RxPin;
00301       GPIO_Init(VUSART1_GPIO, &GPIO_InitStructure);
00302       break;
00303     case 2:
00304       GPIO_InitStructure.GPIO_Pin = VUSART2_RxPin;
00305       GPIO_Init(VUSART2_GPIO, &GPIO_InitStructure);
00306       break;
00307   }
00308   //Set Tx as 50Mhz output
00309   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
00310   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
00311   switch(id) {
00312     case 1:
00313       GPIO_InitStructure.GPIO_Pin = VUSART1_TxPin;
00314       GPIO_Init(VUSART1_GPIO, &GPIO_InitStructure);
00315       break;
00316     case 2:
00317       GPIO_InitStructure.GPIO_Pin = VUSART2_TxPin;
00318       GPIO_Init(VUSART2_GPIO, &GPIO_InitStructure);
00319       break;
00320   }
00321 }
00322
00329 VUSART_STATUS Blox_VUSART_TryReceive(uint8_t id, uint8_t *data) {
00330   //uint8_t data;
00331   switch(id) {
00332     case 1:
00333     //return Virtual USART2 Rx FIFO
00334       if(VUSART1_RxNotEmpty == 1) {
00335         *data = VUSART1_RxDataRegister;
00336         VUSART1_RxNotEmpty = 0;
00337         return VUSART_SUCCESS;
00338       }
00339       else {
00340         return RX_EMPTY;
00341       }
00342     case 2:
00343     //return Virtual USART2 Rx FIFO
00344       if(VUSART2_RxNotEmpty == 1) {
00345         *data = VUSART2_RxDataRegister;
00346         VUSART2_RxNotEmpty = 0;
00347         return VUSART_SUCCESS;
```

```
00348       }
00349       else {
00350         return RX_EMPTY;
00351       }
00352     default:
00353       return INVALID_ID;
00354   }
00355 }
00356
00363 VUSART_STATUS Blox_VUSART_TrySend(uint8_t id, uint8_t data) {
00364   switch(id) {
00365     case 1:
00366       /* check for transmit data register empty and transmission complete */
00367       if(VUSART1_TxEmpty == 1 && VUSART1_TxComplete == 1) {
00368         VUSART1_TxDataRegister = data;
00369         VUSART1_TxEmpty = 0;
00370         VUSART1_TxComplete = 0;
00371         Blox_Timer_Enable_IRQ(VUSART1_TxDataID);
00372         return VUSART_SUCCESS;
00373       }
00374           else {
00375         return TX_BUSY;
00376           }
00377     case 2:
00378       /* check for transmit data register empty and transmission complete */
00379       if(VUSART2_TxEmpty == 1 && VUSART2_TxComplete == 1) {
00380         VUSART2_TxDataRegister = data;
00381         VUSART2_TxEmpty = 0;
00382         VUSART2_TxComplete = 0;
00383         Blox_Timer_Enable_IRQ(VUSART2_TxDataID);
00384         return VUSART_SUCCESS;
00385       }
00386           else {
00387         return TX_BUSY;
00388           }
00389     default:
00390       return INVALID_ID;
00391   }
00392 }
00393
00400 VUSART_STATUS Blox_VUSART_Receive(uint8_t id, uint8_t *data) {
00401   while (Blox_VUSART_TryReceive(id, data) == RX_EMPTY);
00402   return VUSART_SUCCESS;
00403 }
00404
00411 VUSART_STATUS Blox_VUSART_Send(uint8_t id, uint8_t data) {
00412   while (Blox_VUSART_TrySend(id, data) == TX_BUSY);
00413   return VUSART_SUCCESS;
00414 }
00415
00423 VUSART_STATUS Blox_VUSART_SendData(uint8_t id, uint8_t *data, uint32_t len) {
00424   VUSART_STATUS ret;
00425   int i;
00426   for (i = 0; i < len; i++) {
00427     if ((ret = Blox_VUSART_Send(id, data[i])) != VUSART_SUCCESS)
00428       return ret;
00429   }
```

```
00430   return VUSART_SUCCESS;
00431 }
00432
00439 VUSART_STATUS Blox_VUSART_Register_RXNE_IRQ(uint8_t id, void (*RXNE_Handler)(void
      )) {
00440   switch(id) {
00441     case 1:
00442       VUSART1_RXNE_IRQ_ID = Blox_EXTI_Register_SW_IRQ(RXNE_Handler);
00443       if(VUSART1_RXNE_IRQ_ID == EXTI_IRQ_UNAVAILABLE)
00444         return RXNE_IRQ_UNAVAILABLE;
00445       return VUSART_SUCCESS;
00446     case 2:
00447       VUSART2_RXNE_IRQ_ID = Blox_EXTI_Register_SW_IRQ(RXNE_Handler);
00448       if(VUSART2_RXNE_IRQ_ID == EXTI_IRQ_UNAVAILABLE)
00449         return RXNE_IRQ_UNAVAILABLE;
00450       return VUSART_SUCCESS;
00451     default:
00452       return INVALID_ID;
00453   }
00454 }
00455
00461 VUSART_STATUS Blox_VUSART_Enable_RXNE_IRQ(uint8_t id) {
00462   switch(id) {
00463     case 1:
00464       VUSART1_RXNE_IRQ_Enable = TRUE;
00465       return VUSART_SUCCESS;
00466     case 2:
00467        VUSART2_RXNE_IRQ_Enable = TRUE;
00468       return VUSART_SUCCESS;
00469     default:
00470       return INVALID_ID;
00471   }
00472 }
00473
00479 VUSART_STATUS Blox_VUSART_Disable_RXNE_IRQ(uint8_t id) {
00480   switch(id) {
00481     case 1:
00482       VUSART1_RXNE_IRQ_Enable = FALSE;
00483       return VUSART_SUCCESS;
00484     case 2:
00485        VUSART2_RXNE_IRQ_Enable = FALSE;
00486       return VUSART_SUCCESS;
00487     default:
00488       return INVALID_ID;
00489   }
00490 }
```

6.67 drivers/src/blox_xbee.c File Reference

Driver for Blox XBee wireless module.

```
#include "blox_xbee.h"
```

- void(∗ **XBee_RX_Handler**)(**BloxFrame** ∗)

  The function pointer called when an interrupt occurs.

- uint8_t **XBee_RX_Enable** = FALSE

  Flag if the RX interrupt will call the user registered function.

- uint8_t **XBee_TxStatus_Flag** = XBEE_TXSTATUS_NORMAL

  Flag used to communicate information between interrupt and TxStatus.

- void **XBee_RCC_Configuration** (void)

  Initializes clocks for XBee sleep and reset pins.

- void **XBee_GPIO_Configuration** ()

  Initializes the gpio for the XBee reset and sleep pins.

- uint8_t **XBee_CheckOkResponse** (void)

  Checks if "OK<CR>" is waiting in the buffer.

- **XBEE_STATUS XBee_SendTxFrame** (**XBeeTxFrame** ∗frame)

  *Sends a **XBeeTxFrame*** (p. *115*).

- **XBEE_STATUS XBee_TxStatus** (void)
- void **Blox_XBee_VUSART_RXNE_IRQ** (void)

  The function that XBee registers with VUSART to execute on byte received.

- **XBEE_STATUS Blox_XBee_Config** (void)

  Configures the XBee and writes the configuration to non-volatile mem.

- **XBEE_STATUS Blox_XBee_Print** (void)

  Prints out the configuration options of the XBee.

- **XBEE_STATUS Blox_XBee_Init** (void)

  Initializes the XBees sleep and reset pins, and then resets the XBee.

- void **Blox_XBee_Register_RX_IRQ** (void(∗RX_Handler)(**BloxFrame** ∗frame))

  Registers a function to execute when a complete XBee frame is received.

- void **Blox_XBee_Enable_RX_IRQ** (void)

  Enables the sw interrupt that occurs when a XBee reads a byte.

- void **Blox_XBee_Disable_RX_IRQ** (void)

Disables the sw interrupt that occurs when a XBee reads a byte.

- **BloxFrame** * **Blox_XBee_Receive** (void)

  *Receives a **BloxFrame** (p. 97) from the XBee.*

- **XBEE_STATUS Blox_XBee_Send** (uint8_t *data, uint32_t len, **BloxFrameType** type, uint32_t dst_id)

  Sends data out of a specific type on the XBee.

- void **Blox_XBee_Send_Period** (uint8_t *data, uint32_t len, **BloxFrameType** type, uint32_t dst_id, uint32_t millis)

  Sends data out of a specific type on the XBee for a period of time.

6.67.1 Detailed Description

Driver for Blox XBee wireless module.

Author

Dan Cleary

Version

V0.1

Date

10/27/2010

Definition in file **blox_xbee.c**.

6.68 drivers/src/blox_xbee.c

```
00001
00009 #include "blox_xbee.h"
00010
00015 /* Globals */
00019 void (*XBee_RX_Handler)(BloxFrame *);
00023 uint8_t XBee_RX_Enable = FALSE;
00027 uint8_t XBee_TxStatus_Flag = XBEE_TXSTATUS_NORMAL;
00028
00029 /* Private function prototypes */
00030 void XBee_RCC_Configuration(void);
00031 void XBee_GPIO_Configuration(void);
00032 uint8_t XBee_CheckOkResponse(void);
00033 XBEE_STATUS XBee_SendTxFrame (XBeeTxFrame *frame);
```

```
00034 XBEE_STATUS XBee_TxStatus (void);
00035 void Blox_XBee_VUSART_RXNE_IRQ(void);
00036
00041 XBEE_STATUS Blox_XBee_Config(void) {
00042   char buffer[10];
00043   uint8_t garbage;
00044   SysVar sys;
00045   XBee_RCC_Configuration();
00046   XBee_GPIO_Configuration();
00047
00048   Blox_VUSART_Init(XBEE_VUSART_ID);
00049   SysTick_Init();
00050   Blox_System_Init();
00051   Blox_System_GetVars(&sys);
00052
00053   XBEE_SLEEP_GPIO->ODR &= ~(XBEE_SLEEP_PIN);
00054   XBEE_RESET_GPIO->ODR |= XBEE_RESET_PIN;
00055   SysTick_Wait(1);
00056   XBEE_RESET_GPIO->ODR &= ~(XBEE_RESET_PIN);
00057   SysTick_Wait(1);
00058   XBEE_RESET_GPIO->ODR |= XBEE_RESET_PIN;
00059   SysTick_Wait(1100);
00060
00061   Blox_VUSART_TryReceive(XBEE_VUSART_ID, &garbage);   //clear VUSART buffer befor
      e send/receive
00062   Blox_VUSART_Send(XBEE_VUSART_ID, 'X');           // Junk character to before in
      it
00063   SysTick_Wait(1100);
00064   Blox_VUSART_SendData(XBEE_VUSART_ID, "+++", 3) ; //Enter Command Mode
00065   if (XBee_CheckOkResponse() == FALSE)
00066     return XBEE_INIT_FAIL;
00067   SysTick_Wait(1100);
00068
00069   sprintf(buffer, "ATMY%d\r", sys.id);
00070   Blox_VUSART_SendData(XBEE_VUSART_ID, (uint8_t *)buffer, strlen(buffer)) ; //Ent
      er Command Mode
00071   if (XBee_CheckOkResponse() == FALSE)
00072     return XBEE_INIT_FAIL;
00073   SysTick_Wait(20);
00074
00075   Blox_VUSART_SendData(XBEE_VUSART_ID, "ATDLFFFF\r", 9) ; //Set broadcast dest ID

00076   if (XBee_CheckOkResponse() == FALSE)
00077     return XBEE_INIT_FAIL;
00078   SysTick_Wait(20);
00079
00080   Blox_VUSART_SendData(XBEE_VUSART_ID, "ATDH0\r", 6) ; //Set high of dest ID
00081   if (XBee_CheckOkResponse() == FALSE)
00082     return XBEE_INIT_FAIL;
00083   SysTick_Wait(20);
00084
00085   Blox_VUSART_SendData(XBEE_VUSART_ID, "ATAP1\r", 6) ; //API Mode 1 (no escapes)
00086   if (XBee_CheckOkResponse() == FALSE)
00087     return XBEE_INIT_FAIL;
00088   SysTick_Wait(20);
00089
00090   Blox_VUSART_SendData(XBEE_VUSART_ID, "ATRN1\r", 6) ; //API Mode 1 (no escapes)
```

```
00091   if (XBee_CheckOkResponse() == FALSE)
00092     return XBEE_INIT_FAIL;
00093   SysTick_Wait(20);
00094
00095   Blox_VUSART_SendData(XBEE_VUSART_ID, "ATWR\r", 9) ; //Write to non-volatile mem
      ory
00096   if (XBee_CheckOkResponse() == FALSE)
00097     return XBEE_INIT_FAIL;
00098   SysTick_Wait(20);
00099
00100   Blox_VUSART_SendData(XBEE_VUSART_ID, "ATCN\r", 5) ; //Exit command mode
00101   if (XBee_CheckOkResponse() == FALSE)
00102     return XBEE_INIT_FAIL;
00103   SysTick_Wait(20);
00104
00105   return XBEE_OK;
00106 }
00107
00112 XBEE_STATUS Blox_XBee_Print(void) {
00113   uint8_t garbage;
00114
00115   Blox_VUSART_TryReceive(XBEE_VUSART_ID, &garbage);   //clear VUSART buffer befor
      e send/receive
00116   Blox_VUSART_Send(XBEE_VUSART_ID, 'X');           // Junk character to before in
      it
00117   SysTick_Wait(1100);
00118   Blox_VUSART_SendData(XBEE_VUSART_ID, "+++", 3) ; //Enter Command Mode
00119   if (XBee_CheckOkResponse() == FALSE)
00120     return XBEE_INIT_FAIL;
00121   SysTick_Wait(1100);
00122
00123   Blox_VUSART_SendData(XBEE_VUSART_ID, "ATMY\r", 5) ; //Read source address
00124   Blox_DebugStr("ATMY:");
00125   Blox_VUSART_Receive(XBEE_VUSART_ID, &garbage);
00126   Blox_DebugPat("%c\r\n", garbage);
00127   Blox_VUSART_Receive(XBEE_VUSART_ID, &garbage); //\r
00128
00129   Blox_VUSART_SendData(XBEE_VUSART_ID, "ATDL\r", 5) ; //Read source address
00130   Blox_DebugStr("ATDL:");
00131   Blox_VUSART_Receive(XBEE_VUSART_ID, &garbage);
00132   Blox_DebugPat("%c", garbage);
00133   Blox_VUSART_Receive(XBEE_VUSART_ID, &garbage);
00134   Blox_DebugPat("%c", garbage);
00135   Blox_VUSART_Receive(XBEE_VUSART_ID, &garbage);
00136   Blox_DebugPat("%c", garbage);
00137   Blox_VUSART_Receive(XBEE_VUSART_ID, &garbage);
00138   Blox_DebugPat("%c\r\n", garbage);
00139   Blox_VUSART_Receive(XBEE_VUSART_ID, &garbage); //\r
00140
00141   Blox_VUSART_SendData(XBEE_VUSART_ID, "ATDH\r", 5) ; //Read source address
00142   Blox_DebugStr("ATDH:");
00143   Blox_VUSART_Receive(XBEE_VUSART_ID, &garbage);
00144   Blox_DebugPat("%c\r\n", garbage);
00145   Blox_VUSART_Receive(XBEE_VUSART_ID, &garbage); //\r
00146
00147   Blox_VUSART_SendData(XBEE_VUSART_ID, "ATAP\r", 5) ; //Read source address
00148   Blox_DebugStr("ATAP:");
```

```
00149   Blox_VUSART_Receive(XBEE_VUSART_ID, &garbage);
00150   Blox_DebugPat("%c\r\n", garbage);
00151   Blox_VUSART_Receive(XBEE_VUSART_ID, &garbage); //\r
00152
00153   Blox_VUSART_SendData(XBEE_VUSART_ID, "ATCN\r", 5) ; //Exit command mode
00154   if (XBee_CheckOkResponse() == FALSE)
00155     return XBEE_INIT_FAIL;
00156   SysTick_Wait(20);
00157
00158   return XBEE_OK;
00159 }
00160
00165 XBEE_STATUS Blox_XBee_Init(void) {
00166   uint8_t garbage;
00167   XBee_RCC_Configuration();
00168   XBee_GPIO_Configuration();
00169
00170   Blox_System_Init();
00171   SysTick_Init();
00172
00173   XBEE_SLEEP_GPIO->ODR &= ~(XBEE_SLEEP_PIN);
00174   XBEE_RESET_GPIO->ODR |= XBEE_RESET_PIN;
00175   SysTick_Wait(1);
00176   XBEE_RESET_GPIO->ODR &= ~(XBEE_RESET_PIN);
00177   SysTick_Wait(1);
00178   XBEE_RESET_GPIO->ODR |= XBEE_RESET_PIN;
00179   SysTick_Wait(300);
00180
00181   Blox_VUSART_Init(XBEE_VUSART_ID);
00182   Blox_VUSART_TryReceive(XBEE_VUSART_ID, &garbage);   //clear VUSART buffer befor
      e send/receive
00183   Blox_VUSART_Disable_RXNE_IRQ(XBEE_VUSART_ID);
00184   Blox_VUSART_Register_RXNE_IRQ(XBEE_VUSART_ID, &Blox_XBee_VUSART_RXNE_IRQ);
00185   Blox_VUSART_Enable_RXNE_IRQ(XBEE_VUSART_ID);
00186
00187   return XBEE_OK;
00188 }
00189
00194 void XBee_RCC_Configuration(void) {
00195     RCC_APB2PeriphClockCmd(XBEE_RESET_GPIO_CLK, ENABLE);
00196 }
00197
00202 void XBee_GPIO_Configuration() {
00203   GPIO_InitTypeDef GPIO_InitStructure;
00204
00205   //XBee Reset is push-pull, 50Mz
00206   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
00207   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
00208   GPIO_InitStructure.GPIO_Pin = XBEE_RESET_PIN;
00209   GPIO_Init(XBEE_RESET_GPIO, &GPIO_InitStructure);
00210
00211   //XBee Sleep is push-pull, 50Mz
00212   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
00213   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
00214   GPIO_InitStructure.GPIO_Pin = XBEE_SLEEP_PIN;
00215   GPIO_Init(XBEE_SLEEP_GPIO, &GPIO_InitStructure);
00216 }
```

```
00217
00222 uint8_t XBee_CheckOkResponse(void) {
00223   uint8_t ret;
00224   Blox_VUSART_Receive(XBEE_VUSART_ID, &ret);
00225   if (ret != 'O') {
00226     Blox_DebugPat("XBee_CheckOKResponse received %x instead of O\r\n", (char)ret)
      ;
00227     return FALSE;
00228   }
00229   Blox_VUSART_Receive(XBEE_VUSART_ID, &ret);
00230   if (ret != 'K') {
00231     Blox_DebugPat("XBee_CheckOKResponse received %x instead of K\r\n", (char)ret)
      ;
00232     return FALSE;
00233   }
00234   Blox_VUSART_Receive(XBEE_VUSART_ID, &ret);
00235   if (ret != CR) {
00236     Blox_DebugPat("XBee_CheckOKResponse received %x instead of <CR>\r\n", (char)r
      et);
00237     return FALSE;
00238   }
00239   return TRUE;
00240 }
00241
00242
00247 void Blox_XBee_VUSART_RXNE_IRQ(void) {
00248   static uint8_t num = 0;
00249   static uint8_t checksum = 0;
00250   static XBeeFrame frame;
00251   uint8_t data;
00252
00253   if (Blox_VUSART_TryReceive(XBEE_VUSART_ID, &data) == RX_EMPTY)
00254     return;
00255
00256   if (num == 0) {
00257     if (data == 0x7E) {
00258       num = 1; //Start of a frame
00259       checksum = 0;
00260     }
00261   } else if (num == 1) {
00262     frame.length = data << 8;
00263     num++;
00264   } else if (num == 2) {
00265     frame.length |= data;
00266     if (frame.length > 100)
00267       num = 0;
00268     else
00269       num++;
00270   } else if (num > 2) {
00271     if (num == frame.length+3) {
00272       if(0xFF-checksum != data)
00273         num = 0;
00274       else {
00275         XBeeTxStatusFrame *status;
00276         switch(frame.data[0]) {
00277         case API_TX_STATUS:
00278           status = (XBeeTxStatusFrame *)&frame;
```

```
00279           if (status->status == 0 && data == 0xFF-checksum)
00280             XBee_TxStatus_Flag = XBEE_TXSTATUS_SUCCESS;
00281           else
00282             XBee_TxStatus_Flag = XBEE_TXSTATUS_ERROR;
00283           break;
00284         case API_RX_FRAME:
00285           if (XBee_RX_Handler != NULL && XBee_RX_Enable == TRUE) {
00286             XBeeRxFrame rx_frame;
00287             BloxFrame *retFrame;
00288             rx_frame.length = frame.length;
00289             rx_frame.api = frame.data[0];
00290             rx_frame.source = frame.data[1] << 8;
00291             rx_frame.source = frame.data[2];
00292             rx_frame.rssi = frame.data[3];
00293             rx_frame.options = frame.data[4];
00294             memcpy(&(rx_frame.blox_frame), &(frame.data[5]), rx_frame.length-5);
00295             rx_frame.checksum = data;
00296             retFrame = (BloxFrame *)malloc(sizeof(BloxFrame));
00297             memcpy(retFrame, &(rx_frame.blox_frame), sizeof(BloxFrame));
00298             XBee_RX_Handler(retFrame);
00299             free(retFrame);
00300           }
00301           break;
00302         }
00303       }
00304       num = 0;
00305     } else {
00306       frame.data[num-3] = data;
00307       checksum += data;
00308       num++;
00309     }
00310   }
00311 }
00312
00317 void Blox_XBee_Register_RX_IRQ(void (*RX_Handler)(BloxFrame *frame)) {
00318   XBee_RX_Handler = RX_Handler;
00319 }
00320
00325 void Blox_XBee_Enable_RX_IRQ(void) {
00326   XBee_RX_Enable = TRUE;
00327 }
00328
00333 void Blox_XBee_Disable_RX_IRQ(void) {
00334   XBee_RX_Enable = FALSE;
00335 }
00336
00341 BloxFrame * Blox_XBee_Receive(void) {
00342   XBeeRxFrame frame;
00343   BloxFrame *retFrame;
00344   uint8_t ret = 0;
00345   uint8_t checksum = 0;
00346   uint32_t i;
00347
00348   if (Blox_VUSART_TryReceive(XBEE_VUSART_ID, &ret) != VUSART_SUCCESS)
00349     return NULL;
00350
00351   if (ret != 0x7E)
```

```
00352     return NULL;
00353
00354   Blox_VUSART_Receive(XBEE_VUSART_ID, &ret);
00355   frame.length = ret << 8;
00356   Blox_VUSART_Receive(XBEE_VUSART_ID, &ret);
00357   frame.length |= ret;
00358   if(frame.length > 100) {
00359     return NULL;
00360   }
00361   Blox_VUSART_Receive(XBEE_VUSART_ID, &(frame.api));
00362   checksum += frame.api;
00363
00364   Blox_VUSART_Receive(XBEE_VUSART_ID, &ret); //Source Address MSB
00365   frame.source = ret << 8;
00366   checksum += ret;
00367   Blox_VUSART_Receive(XBEE_VUSART_ID, &ret); //Source Address LSB
00368   frame.source |= ret;
00369   checksum += ret;
00370   Blox_VUSART_Receive(XBEE_VUSART_ID, &ret); //RSSI
00371   frame.rssi = ret;
00372   checksum += ret;
00373   Blox_VUSART_Receive(XBEE_VUSART_ID, &ret); //Options
00374   frame.options = ret;
00375   checksum += ret;
00376
00377   for(i = 0; i < frame.length-5; i++) {
00378     Blox_VUSART_Receive(XBEE_VUSART_ID, ((uint8_t *)&(frame.blox_frame))+i);
00379     checksum += ((char *)&(frame.blox_frame))[i];
00380   }
00381   Blox_VUSART_Receive(XBEE_VUSART_ID, &(frame.checksum));
00382
00383   if (frame.checksum != 0xFF-checksum)
00384     return NULL;
00385
00386   retFrame = (BloxFrame *)malloc(sizeof(BloxFrame));
00387   memcpy(retFrame, &(frame.blox_frame), sizeof(BloxFrame));
00388
00389   return retFrame;
00390 }
00391
00400 XBEE_STATUS Blox_XBee_Send (uint8_t *data, uint32_t len, BloxFrameType type, uint
      32_t dst_id) {
00401   XBeeTxFrame frame;
00402   uint16_t i;
00403   if(len > BLOX_FRAME_DATA_LEN)
00404     return XBEE_TX_FAIL;
00405
00406   frame.start = 0x7E;
00407   frame.api = 0x01;
00408   frame.length = sizeof(BloxFrame) + 5; // length includes checksum
00409   frame.dest_addr = 0xFFFF;
00410   frame.options = 0x04;
00411   frame.id = 1;
00412   frame.blox_frame.src_id = Blox_System_GetId();
00413   frame.blox_frame.dst_id = dst_id;
00414   frame.blox_frame.type = type;
00415   frame.blox_frame.len = len;
```

```
00416   for (i = 0; i < len; i++)
00417     frame.blox_frame.data[i] = data[i];
00418 
00419   //checksum split up for hand debugging
00420   frame.checksum = 0xFF;
00421   frame.checksum -= frame.api;
00422   frame.checksum -= frame.id;
00423   frame.checksum -= (frame.dest_addr >> 8) & 0xFF;
00424   frame.checksum -= frame.dest_addr & 0xFF;
00425   frame.checksum -= frame.options;
00426   for (i = 0; i < sizeof(BloxFrame); i++)
00427     frame.checksum -= ((uint8_t *)&(frame.blox_frame))[i];
00428 
00429   return XBee_SendTxFrame(&frame);
00430 }
00431 
00441 void Blox_XBee_Send_Period (uint8_t *data, uint32_t len, BloxFrameType type, uint
      32_t dst_id, uint32_t millis) {
00442   uint32_t cur_time = SysTick_Get_Milliseconds();
00443   while (SysTick_Get_Milliseconds() < cur_time+millis) {
00444     Blox_XBee_Send(data, len, type, dst_id);
00445   }
00446 }
00447 
00453 XBEE_STATUS XBee_SendTxFrame (XBeeTxFrame *frame) {
00454   uint8_t i;
00455   uint8_t len = frame->length-5;
00456 
00457   XBee_TxStatus_Flag = XBEE_TXSTATUS_NORMAL;
00458 
00459   Blox_VUSART_Send(XBEE_VUSART_ID, frame->start);
00460         Blox_VUSART_Send(XBEE_VUSART_ID, (uint8_t)(frame->length >> 8));
00461         Blox_VUSART_Send(XBEE_VUSART_ID, (uint8_t)frame->length);
00462         Blox_VUSART_Send(XBEE_VUSART_ID, frame->api);
00463         Blox_VUSART_Send(XBEE_VUSART_ID, frame->id);
00464         Blox_VUSART_Send(XBEE_VUSART_ID, (uint8_t)(frame->dest_addr >> 8));
00465         Blox_VUSART_Send(XBEE_VUSART_ID, (uint8_t)frame->dest_addr);
00466         Blox_VUSART_Send(XBEE_VUSART_ID, frame->options);
00467         for (i = 0; i < len; i++)
00468           Blox_VUSART_Send(XBEE_VUSART_ID, ((uint8_t *)&(frame->blox_frame))[i]);

00469         Blox_VUSART_Send(XBEE_VUSART_ID, frame->checksum);
00470 
00471   SysTick_Wait(1); //Give it a chance to send.
00472   if(XBee_TxStatus_Flag == XBEE_TXSTATUS_SUCCESS)
00473     return XBEE_OK;
00474 
00475   return XBEE_TX_FAIL;
00476 }
```

6.69 drivers/src/example.c File Reference

Describe the purpose of the module here.

Functions

- void **Blox_MyModule_MyFunc** (type1 Arg1, type2 Arg2, type3 Arg3)

  A brief description of the function's purpose.

6.69.1 Detailed Description

Describe the purpose of the module here.

Author

[Name]

Version

V0.1

Date

10/19/2010

Definition in file **example.c**.

6.69.2 Function Documentation

6.69.2.1 void Blox_MyModule_MyFunc (type1 *Arg1,* type2 *Arg2,* type3 *Arg3*)

A brief description of the function's purpose.

Parameters

Arg1,:	Arg1 is used for ... .
Arg2,:	Arg2 is used for ... . This parameter can be any combination of the following values: • val1: Indicates ... • val2: Indicates ... • val3: Indicates ...
Arg3,:	Arg3 is used for ... . This parameter can be: ENABLE or DISABLE.

Return values

None	

Definition at line **21** of file **example.c**.

6.70 drivers/src/example.c

```
00001
00021 void Blox_MyModule_MyFunc(type1 Arg1, type2 Arg2, type3 Arg3) {
00022         int x;
00023         x += 1;
00024 }
00025
```

6.71 feature_modules/blox_gesture.c File Reference

Driver that performs touchpanel gesture detection.

```
#include "blox_gesture.h"
#include "blox_oled.h"
#include "blox_led.h"
```

- **TIMER_ID touch1ID**
- **TIMER_ID touch2ID**
- **TIMER_ID touch3ID**
- **TIMER_ID touch4ID**
- uint16_t **val** [4] = {0,0,0,0}
- uint16_t **XVals** [4][50]
- uint16_t **YVals** [4][50]
- **GestureRecord LastGesture** [4]
- void **Blox_touch1_isTouched** (void)

  Determines if user is touching Blox, interrupts every 0.02seconds.

- void **Blox_touch2_isTouched** (void)

  Determines if user is touching Blox, interrupts every 0.02seconds.

- void **Blox_touch3_isTouched** (void)

  Determines if user is touching Blox, interrupts every 0.02seconds.

- void **Blox_touch4_isTouched** (void)

  Determines if user is touching Blox, interrupts every 0.02seconds.

- void **Blox_gestureHandler** (int touchNumber)

  Determines gesture movement from pre-populated tracking gesture array.

- void **Blox_Gesture_Init** (void)

  Initializes timer interrupt and array that hold touch values.

- void **Blox_Gesture_DeInit** (void)

  Clears timer interrupt and array that holds touch value.

- void **Blox_touch1_tracker** (void)

  Populated array tracking gesture movement.

- void **Blox_touch2_tracker** (void)

  Populated array tracking gesture movement.

- void **Blox_touch3_tracker** (void)

  Populated array tracking gesture movement.

- void **Blox_touch4_tracker** (void)

  Populated array tracking gesture movement.

- int **Blox_Gesture_GetGesture** (int touchNumber)

  Returns most recent gesture id for a specified touch panel.

- int **Blox_Gesture_GetGestureTime** (int touchNumber)

  Returns most recent gesture timestamp for a specified touch panel.

6.71.1 Detailed Description

Driver that performs touchpanel gesture detection.

Author

Ankita Kaul

Version

V0.1

Date

11/08/2010

Definition in file **blox_gesture.c**.

6.72 feature_modules/blox_gesture.c

```
00001
00009 #include "blox_gesture.h"
00010 #include "blox_oled.h"
00011 #include "blox_led.h"
00012
00017 TIMER_ID touch1ID, touch2ID, touch3ID, touch4ID;
00018 uint16_t val[4]={0,0,0,0}; //[Touch1 #datapoints, Touch2 #datapoints, Touch3 #dat
      apoints, Touch4 #datapoints]
00019 uint16_t XVals[4][50];
00020 uint16_t YVals[4][50];
00021 GestureRecord LastGesture[4]; //Timestamp and last gesture recorded for each touc
      hpanel
00022
00023 //private functions/*
00024 void Blox_touch1_isTouched(void);
00025 void Blox_touch2_isTouched(void);
00026 void Blox_touch3_isTouched(void);
00027 void Blox_touch4_isTouched(void);
00028 void Blox_gestureHandler(int touchNumber);
00029
00034 void Blox_Gesture_Init(void){
00035
00036   Blox_Touch_Init(); //Start out by initializing the touchpanel, before gesture h
      andling
00037
00038   Blox_Timer_Init(TOUCH_TIMx, TOUCH_CLK);
00039   touch1ID = Blox_Timer_Register_IRQ(TOUCH_TIMx, TOUCH_DETECT_FREQ, &
      Blox_touch1_tracker, DISABLE);
00040   touch2ID = Blox_Timer_Register_IRQ(TOUCH_TIMx, TOUCH_DETECT_FREQ, &
      Blox_touch2_tracker, DISABLE);
00041   touch3ID = Blox_Timer_Register_IRQ(TOUCH_TIMx, TOUCH_DETECT_FREQ, &
      Blox_touch3_tracker, DISABLE);
00042   touch4ID = Blox_Timer_Register_IRQ(TOUCH_TIMx, TOUCH_DETECT_FREQ, &
      Blox_touch4_tracker, DISABLE);
00043
00044   Blox_EXTI_Init();
00045   Blox_EXTI_Register_HW_IRQ(GPIO_PortSourceGPIOE, GPIO_PinSource10, &
      Blox_touch1_isTouched);
00046   Blox_EXTI_Register_HW_IRQ(GPIO_PortSourceGPIOE, GPIO_PinSource11, &
      Blox_touch2_isTouched);
00047   Blox_EXTI_Register_HW_IRQ(GPIO_PortSourceGPIOE, GPIO_PinSource14, &
      Blox_touch3_isTouched);
00048   Blox_EXTI_Register_HW_IRQ(GPIO_PortSourceGPIOE, GPIO_PinSource7, &
      Blox_touch4_isTouched);
00049
00050   Blox_Touch_GetY(1);//trash, to enable PENIRQ
00051
00052   //DEBUG - REMOVE LATER
00053   //Blox_LED_Init();
00054 }
00055
00060 void Blox_Gesture_DeInit(void){
00061
00062         Blox_Timer_Disable_IRQ(touch1ID);
00063         Blox_Timer_Disable_IRQ(touch2ID);
```

```
00064         Blox_Timer_Disable_IRQ(touch3ID);
00065         Blox_Timer_Disable_IRQ(touch4ID);
00066
00067         Blox_EXTI_Disable_IRQ(GPIO_PinSource10);
00068         Blox_EXTI_Disable_IRQ(GPIO_PinSource11);
00069         Blox_EXTI_Disable_IRQ(GPIO_PinSource14);
00070         Blox_EXTI_Disable_IRQ(GPIO_PinSource7);
00071
00072 }
00073
00078 void Blox_touch1_isTouched(void){
00079
00080         if(Blox_Touch_GetZ1(1)> PRESSURE_THRESHOLD) //check if pressure detected
      and over specified threshold
00081         {
00082           //disable gesture detection on touchpanel
00083           Blox_EXTI_Disable_IRQ(GPIO_PinSource10);
00084
00085           //enable timer interrupt to collect gesture data
00086           Blox_Timer_Enable_IRQ(touch1ID);
00087         }
00088 }
00089
00094 void Blox_touch2_isTouched(void){
00095
00096         if(Blox_Touch_GetZ1(2)> PRESSURE_THRESHOLD) //is pressure detected and ov
      er specified threshold?
00097         {
00098           //disable gesture detection on touchpanel
00099           Blox_EXTI_Disable_IRQ(GPIO_PinSource11);
00100
00101           //enable timer interrupt to collect gesture data
00102           Blox_Timer_Enable_IRQ(touch2ID);
00103
00104           //DEBUG
00105           //Blox_LED_Toggle(1);
00106
00107         }
00108 }
00109
00114 void Blox_touch3_isTouched(void){
00115
00116         if(Blox_Touch_GetZ1(3)> PRESSURE_THRESHOLD) //is pressure detected and ov
      er specified threshold?
00117         {
00118           //disable gesture detection on touchpanel
00119           Blox_EXTI_Disable_IRQ(GPIO_PinSource14);
00120
00121           //enable timer interrupt to collect gesture data
00122           Blox_Timer_Enable_IRQ(touch3ID);
00123         }
00124 }
00125
00130 void Blox_touch4_isTouched(void){
00131
00132         if(Blox_Touch_GetZ1(4)> PRESSURE_THRESHOLD) //is pressure detected and ov
      er specified threshold?
```

```
00133         {
00134           //disable gesture detection on touchpanel
00135           Blox_EXTI_Disable_IRQ(GPIO_PinSource7);
00136 
00137           //enable timer interrupt to collect gesture data
00138           Blox_Timer_Enable_IRQ(touch4ID);
00139         }
00140 }
00141 
00146 void Blox_touch1_tracker(void){
00147 
00148           if(val[0]<50 && (Blox_Touch_GetZ1(1)>PRESSURE_THRESHOLD)){ //include ti
      meout after 50 samples collected or gesture done
00149                 XVals[0][val[0]]=Blox_Touch_GetX(1);
00150                 YVals[0][val[0]]=Blox_Touch_GetY(1);
00151                 val[0]++;
00152             }
00153 
00154           else{ //gesture done or timeout
00155                 Blox_Timer_Disable_IRQ(touch1ID);
00156                   Blox_gestureHandler(1);
00157                   Blox_EXTI_Enable_IRQ(GPIO_PinSource10);
00158           }
00159 }
00160 
00165 void Blox_touch2_tracker(void){
00166 
00167 //Blox_LED_Toggle(2);
00168 
00169   if(val[1]<50 && (Blox_Touch_GetZ1(2)>PRESSURE_THRESHOLD)){ //include timeout af
      ter 50 samples collected or gesture done
00170         XVals[1][val[1]]=Blox_Touch_GetX(2);
00171         YVals[1][val[1]]=Blox_Touch_GetY(2);
00172         val[1]++;
00173   }
00174 
00175          else{ //gesture done or timeout
00176                 Blox_Timer_Disable_IRQ(touch2ID);
00177                 Blox_gestureHandler(2);
00178                 Blox_EXTI_Enable_IRQ(GPIO_PinSource11);
00179         }
00180 }
00181 
00186 void Blox_touch3_tracker(void){
00187 
00188           if(val[2]<50 && (Blox_Touch_GetZ1(3)> PRESSURE_THRESHOLD)){ //include t
      imeout after 50 samples collected or gesture done
00189                 XVals[2][val[2]]=Blox_Touch_GetX(3);
00190                 YVals[2][val[2]]=Blox_Touch_GetY(3);
00191                 val[2]++;
00192             }
00193 
00194           else{ //gesture done or timeout
00195                 Blox_Timer_Disable_IRQ(touch3ID);
00196                 Blox_gestureHandler(3);
00197                 Blox_EXTI_Enable_IRQ(GPIO_PinSource14);
00198           }
```

```
00199 }
00200
00205 void Blox_touch4_tracker(void){
00206
00207   if(val[3]<50 && (Blox_Touch_GetZ1(4)>PRESSURE_THRESHOLD)){ //include timeout af
      ter 50 samples collected or gesture done
00208           XVals[3][val[3]]=Blox_Touch_GetX(4);
00209           YVals[3][val[3]]=Blox_Touch_GetY(4);
00210           val[3]++;
00211   }
00212
00213   else{ //gesture done or timeout
00214         Blox_Timer_Disable_IRQ(touch4ID);
00215         Blox_gestureHandler(4);
00216                 Blox_EXTI_Enable_IRQ(GPIO_PinSource7);
00217         }
00218 }
00219
00224 void Blox_gestureHandler(int touchNumber){
00225
00226 //   uint16_t  xavg_1, xavg_2, yavg_1, yavg_2 = 0;
00227   int gestureNow=0;
00228   int counter =0;
00229   int XMov =0;
00230   int YMov = 0;
00231   int XStart = XVals[touchNumber-1][0];
00232   int YStart = YVals[touchNumber-1][0];
00233   int XEnd = XVals[touchNumber-1][val[touchNumber-1]-1];
00234   int YEnd = YVals[touchNumber-1][val[touchNumber-1]-1];
00235
00236   if((XStart-XEnd)>XTHRESH)
00237                 XMov = TOUCH_GESTURE_RL;
00238   else if((XEnd-XStart)>XTHRESH)
00239                 XMov = TOUCH_GESTURE_LR;
00240   else if((XStart-XEnd)<XNOMOV)
00241                 XMov = TOUCH_X_STABLE;
00242
00243    if((YStart-YEnd)>YTHRESH)
00244      YMov = TOUCH_GESTURE_DU;
00245    else if((YEnd-YStart)>YTHRESH)
00246      YMov = TOUCH_GESTURE_UD;
00247    else if((YStart-YEnd)<YNOMOV)
00248      YMov = TOUCH_Y_STABLE;
00249
00250    gestureNow = XMov*YMov;
00251
00252    switch(gestureNow){
00253         case 3: // LR and DU = Diagonal bottom left to top right
00254       gestureNow = TOUCH_DIAG_DLUR;
00255       break;
00256     case 4: // LR and UD = Diagonal top left to bottom right
00257       gestureNow = TOUCH_DIAG_ULDR;
00258       break;
00259     case -1: // LR (Y Stable)
00260       gestureNow = TOUCH_GESTURE_LR;
00261       break;
00262     case 6: // RL and DU = Diagonal bottom right to top left
```

```
00263       gestureNow = TOUCH_DIAG_DRUL;
00264       break;
00265     case 8: // RL and UD = Diagonal top right to bottom left
00266       gestureNow = TOUCH_DIAG_URDL;
00267       break;
00268       case -2: // RL (Y stable)
00269       gestureNow = TOUCH_GESTURE_RL;
00270       break;
00271     case -4: //UD (X stable)
00272       gestureNow = TOUCH_GESTURE_UD;
00273       break;
00274     case -3 : //DU (X stable)
00275       gestureNow = TOUCH_GESTURE_DU;
00276       break;
00277     case 1: // Tap (X stable and Y stable)
00278       gestureNow = TOUCH_GESTURE_TAP;
00279       break;
00280   }
00281
00282         //Blox_LED_Toggle(3); //DEBUG
00283
00284         LastGesture[touchNumber-1].timestamp=SysTick_Get_Milliseconds();
00285         LastGesture[touchNumber-1].gesture=gestureNow;
00286         val[0]=val[1]=val[2]=val[3]=0; //clear
00287
00288   for(counter=0; counter<50; counter++) //clear memory
00289   {
00290     XVals[touchNumber-1][counter]=0;
00291           YVals[touchNumber-1][counter]=0;
00292   }
00293
00294 }
00295
00300 int Blox_Gesture_GetGesture(int touchNumber){
00301   return LastGesture[touchNumber-1].gesture;
00302 }
00303
00308 int Blox_Gesture_GetGestureTime(int touchNumber){
00309   return LastGesture[touchNumber-1].timestamp;
00310 }
```

6.73 feature_modules/blox_gesture.h File Reference

Driver that performs touchpanel gesture detection.

```
#include "stm32f10x_rcc.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_spi.h"
#include "blox_system.h"
#include "blox_counter.h"
```

```
#include "blox_exti.h"
#include "blox_touch.h"
#include "blox_tim.h"
```

Data Structures

- struct **GestureRecord**

  Contains gesture data.

- #define **TOUCH_TIMx** 3
- #define **TOUCH_CLK** 180000
- #define **TOUCH_DETECT_FREQ** 3600
- #define **PRESSURE_THRESHOLD** 5
- #define **XTHRESH** 25
- #define **YTHRESH** 25
- #define **XNOMOV** 7
- #define **YNOMOV** 7
- #define **TOUCH_X_STABLE** -1
- #define **TOUCH_Y_STABLE** -1
- #define **TOUCH_GESTURE_LR** 1
- #define **TOUCH_GESTURE_RL** 2
- #define **TOUCH_GESTURE_DU** 3
- #define **TOUCH_GESTURE_UD** 4
- #define **TOUCH_DIAG_DLUR** 5
- #define **TOUCH_DIAG_DRUL** 6
- #define **TOUCH_DIAG_ULDR** 7
- #define **TOUCH_DIAG_URDL** 8
- #define **TOUCH_GESTURE_TAP** 9
- void **Blox_Gesture_Init** (void)

  Initializes timer interrupt and array that hold touch values.

- void **Blox_Gesture_DeInit** (void)

  Clears timer interrupt and array that holds touch value.

- void **Blox_touch1_tracker** (void)

  Populated array tracking gesture movement.

- void **Blox_touch2_tracker** (void)

Populated array tracking gesture movement.

- void **Blox_touch3_tracker** (void)

  Populated array tracking gesture movement.

- void **Blox_touch4_tracker** (void)

  Populated array tracking gesture movement.

- int **Blox_Gesture_GetGesture** (int touchNumber)

  Returns most recent gesture id for a specified touch panel.

- int **Blox_Gesture_GetGestureTime** (int touchNumber)

  Returns most recent gesture timestamp for a specified touch panel.

6.73.1 Detailed Description

Driver that performs touchpanel gesture detection.

Author

Ankita Kaul

Version

V0.1

Date

11/08/2010

Definition in file **blox_gesture.h**.

6.74 feature_modules/blox_gesture.h

```
00001
00009 #ifndef __BLOX_GESTURE_H
00010 #define __BLOX_GESTURE_H
00011
00012 #include "stm32f10x_rcc.h"
00013 #include "stm32f10x_gpio.h"
00014 #include "stm32f10x_spi.h"
00015
00016 #include "blox_system.h"
00017 #include "blox_counter.h"
00018 #include "blox_exti.h"
```

```
00019 #include "blox_touch.h"
00020 #include "blox_tim.h"
00021
00026 #define TOUCH_TIMx                      3
00027 #define TOUCH_CLK                       180000
00028 #define TOUCH_DETECT_FREQ       3600          //interrupt every .02s
00029
00030 #define PRESSURE_THRESHOLD  5
00031 #define XTHRESH                         25
00032 #define YTHRESH                         25
00033 #define XNOMOV                          7
00034 #define YNOMOV                          7
00035
00036 #define TOUCH_X_STABLE          -1
00037 #define TOUCH_Y_STABLE          -1
00038 #define TOUCH_GESTURE_LR        1
00039 #define TOUCH_GESTURE_RL        2
00040 #define TOUCH_GESTURE_DU        3
00041 #define TOUCH_GESTURE_UD        4
00042 #define TOUCH_DIAG_DLUR         5
00043 #define TOUCH_DIAG_DRUL         6
00044 #define TOUCH_DIAG_ULDR         7
00045 #define TOUCH_DIAG_URDL         8
00046 #define TOUCH_GESTURE_TAP       9
00047
00051 typedef struct {
00052   uint32_t timestamp;
00053   int gesture;
00054 } GestureRecord;
00055
00056 void Blox_Gesture_Init(void);
00057 void Blox_Gesture_DeInit(void);
00058 void Blox_touch1_tracker(void);
00059 void Blox_touch2_tracker(void);
00060 void Blox_touch3_tracker(void);
00061 void Blox_touch4_tracker(void);
00062 int Blox_Gesture_GetGesture(int touchNumber);
00063 int Blox_Gesture_GetGestureTime(int touchNumber);
00065 #endif
```

6.75 feature_modules/blox_role.c File Reference

Module that allows a distributed program to be stored in a single program and then started on multiple Blox automatically.

```
#include "blox_role.h"

#include "blox_led.h"
```

- #define **ROLE_FLAG_LOC** 0x20006000

- #define **ROLE_FN_LOC** 0x20006004
- enum **State** { **STATE_EMPTY**, **STATE_PARENT**, **STATE_CHILD** }
- void **Blox_Role_RX** (**BloxFrame** ∗frame)

  *Executes when XBee receives a **BloxFrame** (p. 97).*

- uint8_t **Role_NextID** (void)

  Returns the next role_id to be allocated All the role have their minimum number required filled first in order. Then the maxiumum number are filled in a fair manner by adding one to each role in order.

- **ROLE_STATUS Blox_Role_Init** (char ∗name, uint8_t len)

  Initializes the role driver's data structures.

- **ROLE_STATUS Blox_Role_Add** (ptrVoidFn fn, uint8_t min, uint8_t max)

  Adds a new role to the role driver.

- **ROLE_STATUS Blox_Role_Run** (void)

  Runs the role, which is a multiple-step procedure. First the Blox should determine if it is the original starter of this application. if so, then it will query Blox running base programs to see if they can run the requested application. It then requests those base programs begin running the application and designates a role to the application once it requests a role.

6.75.1 Detailed Description

Module that allows a distributed program to be stored in a single program and then started on multiple Blox automatically.

Author

Jesse Tannahill

Version

V0.1

Date

11/03/2010

Definition in file **blox_role.c**.

6.76 feature_modules/blox_role.c

```
00001
00010 #include "blox_role.h"
00011 #include "blox_led.h"
00012
00017 #define ROLE_FLAG_LOC 0x20006000
00018 #define ROLE_FN_LOC   0x20006004
00019
00020 /* Private function prototypes */
00021 void Blox_Role_RX(BloxFrame *frame);
00022 uint8_t Role_NextID(void);
00023
00027 static RoleInfo info;
00028
00032 typedef enum {
00033   STATE_EMPTY,
00034   STATE_PARENT,
00035   STATE_CHILD
00036 } State;
00037
00041 static State parent = STATE_EMPTY;
00045 static uint8_t role_id = 0;
00049 static uint8_t allocating = FALSE;
00050
00057 ROLE_STATUS Blox_Role_Init(char *name, uint8_t len) {
00058   if (*(uint32_t *)ROLE_FLAG_LOC == 1) {
00059     *(uint32_t *)ROLE_FLAG_LOC = 0;
00060     (*(ptrVoidFn *)ROLE_FN_LOC)();
00061   }
00062
00063   SysTick_Init();
00064   Blox_XBee_Init();
00065   Blox_XBee_Register_RX_IRQ(&Blox_Role_RX);
00066   Blox_XBee_Enable_RX_IRQ();
00067   Blox_LED_Init();
00068
00069   memset(&info, 0, sizeof(RoleInfo));
00070   memcpy(info.name, name, len);
00071   info.name_len = len;
00072   info.num_blox_found = 1;
00073   info.num_blox_started = 1;
00074
00075   return ROLE_OK;
00076 }
00077
00083 void Blox_Role_RX(BloxFrame *frame) {
00084   static uint16_t id_in_progress = 0xFFFF;
00085   static uint16_t count = 0;
00086   count++;
00087   //Addressed to me?
00088   if(frame->dst_id == XBEE_BLOX_BROADCAST_ID || frame->dst_id ==
      Blox_System_GetId()
00089           && (frame->src_id == id_in_progress || allocating == FALSE)) {
00090     if (frame->type == FRAME_TYPE_ROLE) {
00091       RoleFrame respFrame;
00092       RoleFrame *rFrame = (RoleFrame *)(&(frame->data));
```

```
00093       switch(rFrame->opcode) {
00094       case PARENT_QUERY:
00095         if(parent == STATE_PARENT) {
00096           if (memcmp(info.name, ((QueryFrame *)&(rFrame->data))->name, info.
      name_len) == 0) {
00097             SysTick_Wait(XBEE_HOLD_PERIOD); //Wait for other to stop transmitting
       PARENT_QUERY
00098             respFrame.opcode = PARENT_ACK;
00099             ((ParentAckFrame *)&(respFrame.data))->role_id = Role_NextID(); //TOD
      O: Get a Role ID here
00100             Blox_LED_On(LED2);
00101             Blox_XBee_Send_Period((uint8_t *)&respFrame, sizeof(RoleFrame), FRAME
      _TYPE_ROLE, frame->src_id, XBEE_HOLD_PERIOD);
00102             Blox_LED_Off(LED2);
00103             info.num_blox_started++;
00104             allocating = FALSE;
00105             id_in_progress = 0xFFFF;
00106           }
00107         }
00108         break;
00109       case PARENT_ACK:
00110         Blox_LED_On(LED4);
00111         role_id = ((ParentAckFrame *)(&(rFrame->data)))->role_id;
00112         parent = STATE_CHILD;
00113         break;
00114       case PROG_ACK:
00115         id_in_progress = frame->src_id;
00116         allocating = TRUE;
00117         Blox_LED_On(LED4);
00118         SysTick_Wait(XBEE_HOLD_PERIOD); //Hold period
00119         Blox_LED_Off(LED4);
00120         respFrame.opcode = PROG_START;
00121         memcpy(&(((QueryFrame *)&respFrame.data)->name), info.name, FS_FILE_MAX_N
      AME_LEN);
00122         Blox_XBee_Send_Period((uint8_t *)&respFrame, sizeof(RoleFrame), FRAME_TYP
      E_ROLE, frame->src_id, XBEE_HOLD_PERIOD);
00123         info.num_blox_found++;
00124       }
00125     }
00126   }
00127 }
00128
00138 ROLE_STATUS Blox_Role_Add(ptrVoidFn fn, uint8_t min, uint8_t max) {
00139   if (info.num_roles == ROLE_MAX)
00140     return ROLE_OOM;
00141   if (min == 0 || min > max)
00142     return ROLE_ADD_FAIL;
00143
00144   info.roles[info.num_roles].fn = fn;
00145   info.roles[info.num_roles].min = min;
00146   info.num_needed += min;
00147   info.roles[info.num_roles].max = max;
00148   info.num_wanted += max;
00149   info.roles[info.num_roles++].num_allocated = 0;
00150
00151   if(info.num_roles == 1) {
00152     role_id = 0;
```

```
00153     info.roles[0].num_allocated = 1;
00154   }
00155   return ROLE_OK;
00156 }
00157
00168 ROLE_STATUS Blox_Role_Run(void) {
00169   RoleFrame frame;
00170   uint8_t num_found;
00171
00172   //Am I the parent?
00173   frame.opcode = PARENT_QUERY;
00174   memcpy(&(((QueryFrame *)frame.data)->name), info.name, FS_FILE_MAX_NAME_LEN);
00175   Blox_XBee_Send_Period((uint8_t *)&frame, sizeof(RoleFrame), FRAME_TYPE_ROLE, XB
      EE_BLOX_BROADCAST_ID, XBEE_HOLD_PERIOD);
00176   Blox_LED_On(LED1);
00177   SysTick_Wait(XBEE_HOLD_PERIOD*2); //Let parent respond.
00178   Blox_LED_Off(LED1);
00179
00180   if (parent == STATE_CHILD) {
00181     *(uint32_t *)ROLE_FLAG_LOC = 1;
00182     FS_SetAppFlag(1);
00183     *(uint32_t *)ROLE_FN_LOC = (uint32_t)info.roles[role_id].fn;
00184     NVIC_SystemReset();
00185   }
00186
00187   //I'm the parent. Find base programs with the program.
00188   parent = STATE_PARENT;
00189   frame.opcode = PROG_QUERY;
00190   memcpy(&(((QueryFrame *)&frame.data)->name), info.name, FS_FILE_MAX_NAME_LEN);
00191   num_found = info.num_blox_found;
00192   while (info.num_blox_found < info.num_wanted) {
00193     allocating = FALSE;
00194     Blox_LED_On(LED3);
00195     Blox_XBee_Send_Period((uint8_t *)&frame, sizeof(RoleFrame), FRAME_TYPE_ROLE,
      XBEE_BLOX_BROADCAST_ID, XBEE_HOLD_PERIOD);
00196     Blox_LED_Off(LED3);
00197     SysTick_Wait(XBEE_HOLD_PERIOD*4);
00198     if(info.num_blox_found == num_found)
00199       break;
00200     num_found = info.num_blox_found;
00201     while (allocating == TRUE) ; //Wait for the other program to PROG_QUERY
00202   }
00203
00204   //Done finding nodes, start yourself.
00205   *(uint32_t *)ROLE_FLAG_LOC = 1;
00206   FS_SetAppFlag(1);
00207   *(uint32_t *)ROLE_FN_LOC = (uint32_t)info.roles[role_id].fn;
00208   NVIC_SystemReset();
00209
00210   return ROLE_OK;
00211 }
00212
00220 uint8_t Role_NextID(void) {
00221   static uint8_t cur_role = 0;
00222   static uint8_t needs_full = FALSE, wants_full = FALSE;
00223
00224   if(needs_full == FALSE) {
```

```
00225     uint8_t tmp_cur = cur_role;
00226     do {
00227       if (info.roles[cur_role].num_allocated < info.roles[cur_role].min) {
00228         info.roles[cur_role].num_allocated++;
00229         return cur_role;
00230       }
00231       cur_role = (cur_role + 1) % info.num_roles;
00232     } while (tmp_cur != cur_role) ;
00233     needs_full = TRUE;
00234     return Role_NextID();
00235   } else {
00236     if (wants_full == TRUE)
00237       return 0; //ERROR CONDITION
00238     else {
00239       uint8_t tmp_cur = cur_role;
00240       uint8_t ret_role;
00241       do {
00242         if (info.roles[cur_role].num_allocated < info.roles[cur_role].max) {
00243           info.roles[cur_role].num_allocated++;
00244           ret_role = cur_role;
00245           cur_role = (cur_role + 1) % info.num_roles;
00246           return ret_role;
00247         }
00248         cur_role = (cur_role + 1) % info.num_roles;
00249       } while (tmp_cur != cur_role) ;
00250       wants_full = TRUE;
00251       return Role_NextID();
00252     }
00253   }
00254 }
```

6.77 feature_modules/neighbor_detect.c File Reference

A driver for the neighbor detection module.

```
#include "neighbor_detect.h"
```

- void(∗ **IR_North_User_Handler**)(**IRFrame** ∗frame) = NULL
- void(∗ **IR_East_User_Handler**)(**IRFrame** ∗frame) = NULL
- void(∗ **IR_South_User_Handler**)(**IRFrame** ∗frame) = NULL
- void(∗ **IR_West_User_Handler**)(**IRFrame** ∗frame) = NULL
- uint8_t **neighbors** [4] = {FALSE}
- uint8_t **neighbors_id** [4] = {0,0,0,0}
- uint8_t **neighbor_update_flag** [4] = {FALSE}
- void **IR_Ping** (void)

  The function that Neighbor Detection registers with timer to ping IRs.

- void **IR_North_Neighbor_Handler** (**IRFrame** ∗frame)

  The function that Neighbor Detection registers with IR1 to execute on frame received.

- void **IR_East_Neighbor_Handler** (**IRFrame** ∗frame)

  The function that Neighbor Detection registers with IR2 to execute on frame received.

- void **IR_South_Neighbor_Handler** (**IRFrame** ∗frame)

  The function that Neighbor Detection registers with IR3 to execute on frame received.

- void **IR_West_Neighbor_Handler** (**IRFrame** ∗frame)

  The function that Neighbor Detection registers with IR4 to execute on frame received.

- void **Neighbor_Detect_Init** (void)

  Initializes the Neighbor Detection module.

- void **Neighbor_Register_IR_RX_IRQ** (uint8_t id, void(∗IR_User_Handler)(**IRFrame** ∗frame))

  Register a receive interrupt for user frames.

- uint8_t **IR_Get_Neighbor** (**IR_DIR** face_dir)

  Gets the neighbor for a given face.

6.77.1 Detailed Description

A driver for the neighbor detection module.

Author

Zach Wasson

Version

V0.1

Date

11/17/2010

Definition in file **neighbor_detect.c**.

6.77.2 Function Documentation

6.77.2.1 void IR_East_Neighbor_Handler (IRFrame * *frame*)

The function that Neighbor Detection registers with IR2 to execute on frame received.

Return values

None.	

Definition at line **124** of file **neighbor_detect.c**.

6.77.2.2 uint8_t IR_Get_Neighbor (IR_DIR *face_dir*)

Gets the neighbor for a given face.

Return values

The	id of a neighbor if one is present

Definition at line **172** of file **neighbor_detect.c**.

6.77.2.3 void IR_North_Neighbor_Handler (IRFrame * *frame*)

The function that Neighbor Detection registers with IR1 to execute on frame received.

Return values

None.	

Definition at line **108** of file **neighbor_detect.c**.

6.77.2.4 void IR_Ping (void)

The function that Neighbor Detection registers with timer to ping IRs.

Return values

None.	

Definition at line **82** of file **neighbor_detect.c**.

6.77.2.5 void IR_South_Neighbor_Handler (IRFrame * *frame*)

The function that Neighbor Detection registers with IR3 to execute on frame received.

Return values

None.	

Definition at line **140** of file **neighbor_detect.c**.

6.77.2.6 void IR_West_Neighbor_Handler (IRFrame * *frame*)

The function that Neighbor Detection registers with IR4 to execute on frame received.

Return values

None.	

Definition at line **156** of file **neighbor_detect.c**.

6.77.2.7 void Neighbor_Detect_Init (void)

Initializes the Neighbor Detection module.

Return values

None	

Definition at line **33** of file **neighbor_detect.c**.

6.77.2.8 void Neighbor_Register_IR_RX_IRQ (uint8_t *id,* void(*)(IRFrame *frame) *IR_User_Handler*)

Register a receive interrupt for user frames.

Parameters

id	the id of the IR
IR_User_Handler	the function that will be called when a user frame is received

Return values

None	

Definition at line **61** of file **neighbor_detect.c**.

6.78 feature_modules/neighbor_detect.c

```
00001
00008 #include "neighbor_detect.h"
00009
00014 void IR_Ping(void);
00015 void IR_North_Neighbor_Handler(IRFrame *frame);
00016 void IR_East_Neighbor_Handler(IRFrame *frame);
00017 void IR_South_Neighbor_Handler(IRFrame *frame);
00018 void IR_West_Neighbor_Handler(IRFrame *frame);
00019
00020 void (*IR_North_User_Handler)(IRFrame *frame) = NULL;
00021 void (*IR_East_User_Handler)(IRFrame *frame) = NULL;
00022 void (*IR_South_User_Handler)(IRFrame *frame) = NULL;
00023 void (*IR_West_User_Handler)(IRFrame *frame) = NULL;
00024
00025 uint8_t neighbors[4] = {FALSE};
00026 uint8_t neighbors_id[4] = {0,0,0,0};
00027 uint8_t neighbor_update_flag[4] = {FALSE};
00028
00033 void Neighbor_Detect_Init(void) {
00034   USB_Init();
00035
00036   IR_Init(IR_NORTH_ID);
00037   IR_Init(IR_EAST_ID);
00038   IR_Init(IR_SOUTH_ID);
00039   IR_Init(IR_WEST_ID);
00040   Blox_Timer_Init(NEIGHBOR_TIMx, NEIGHBOR_TIM_CLK);
00041
00042   Blox_IR_Register_RX_IRQ(IR_NORTH_ID, &IR_North_Neighbor_Handler);
00043   Blox_IR_Register_RX_IRQ(IR_EAST_ID, &IR_East_Neighbor_Handler);
00044   Blox_IR_Register_RX_IRQ(IR_SOUTH_ID, &IR_South_Neighbor_Handler);
00045   Blox_IR_Register_RX_IRQ(IR_WEST_ID, &IR_West_Neighbor_Handler);
00046
00047   Blox_Timer_Register_IRQ(NEIGHBOR_TIMx, NEIGHBOR_SAMPLE_PERIOD, &IR_Ping, ENABLE
      );
00048
00049   Blox_IR_Enable_RX_IRQ(IR_NORTH_ID);
00050   Blox_IR_Enable_RX_IRQ(IR_EAST_ID);
00051   Blox_IR_Enable_RX_IRQ(IR_SOUTH_ID);
00052   Blox_IR_Enable_RX_IRQ(IR_WEST_ID);
00053 }
00054
00061 void Neighbor_Register_IR_RX_IRQ(uint8_t id, void (*IR_User_Handler)(IRFrame *fra
      me)) {
00062   switch(id) {
00063     case IR_NORTH_ID:
00064       IR_North_User_Handler = IR_User_Handler;
00065       break;
00066     case IR_EAST_ID:
00067       IR_East_User_Handler = IR_User_Handler;
00068       break;
```

```
00069     case IR_SOUTH_ID:
00070       IR_South_User_Handler = IR_User_Handler;
00071       break;
00072     case IR_WEST_ID:
00073       IR_West_User_Handler = IR_User_Handler;
00074       break;
00075   }
00076 }
00077
00082 void IR_Ping(void) {
00083   uint8_t i;
00084   IR_SendFrame(IR_NORTH_ID, IR_FRAME_TYPE_NEIGHBOR, "Yo", 2);
00085   IR_SendFrame(IR_EAST_ID, IR_FRAME_TYPE_NEIGHBOR, "Sup", 3);
00086   IR_SendFrame(IR_SOUTH_ID, IR_FRAME_TYPE_NEIGHBOR, "Hey", 3);
00087   IR_SendFrame(IR_WEST_ID, IR_FRAME_TYPE_NEIGHBOR, "Hi", 2);
00088   for(i = 0; i < 4; i++) {
00089     if(neighbor_update_flag[i] == TRUE) {
00090       neighbor_update_flag[i] = FALSE;
00091     } else {
00092       neighbors[i] = FALSE;
00093       neighbors_id[i] = 0;
00094     }
00095   }
00096
00097   for(i = 0; i < 4; i++) {
00098     if(neighbors[i] == TRUE) {
00099       USB_SendPat("N: %d\t", neighbors_id[i]);
00100     }
00101   }
00102 }
00103
00108 void IR_North_Neighbor_Handler(IRFrame *frame) {
00109   if(frame->type == IR_FRAME_TYPE_NEIGHBOR) {
00110     neighbors[IR_NORTH_ID-1] = TRUE;
00111     neighbors_id[IR_NORTH_ID-1] = frame->src_id;
00112     neighbor_update_flag[IR_NORTH_ID-1] = TRUE;
00113   } else if (frame->type == IR_FRAME_TYPE_USER && IR_North_User_Handler != NULL)
     {
00114     (*IR_North_User_Handler)(frame);
00115   }
00116   free(frame->data);
00117   free(frame);
00118 }
00119
00124 void IR_East_Neighbor_Handler(IRFrame *frame) {
00125   if(frame->type == IR_FRAME_TYPE_NEIGHBOR) {
00126     neighbors[IR_EAST_ID-1] = TRUE;
00127     neighbors_id[IR_EAST_ID-1] = frame->src_id;
00128     neighbor_update_flag[IR_EAST_ID-1] = TRUE;
00129   } else if (frame->type == IR_FRAME_TYPE_USER && IR_East_User_Handler != NULL) {

00130     (*IR_East_User_Handler)(frame);
00131   }
00132   free(frame->data);
00133   free(frame);
00134 }
00135
```

```
00140 void IR_South_Neighbor_Handler(IRFrame *frame) {
00141   if(frame->type == IR_FRAME_TYPE_NEIGHBOR) {
00142     neighbors[IR_SOUTH_ID-1] = TRUE;
00143     neighbors_id[IR_SOUTH_ID-1] = frame->src_id;
00144     neighbor_update_flag[IR_SOUTH_ID-1] = TRUE;
00145   } else if (frame->type == IR_FRAME_TYPE_USER && IR_South_User_Handler != NULL)
      {
00146     (*IR_South_User_Handler)(frame);
00147   }
00148   free(frame->data);
00149   free(frame);
00150 }
00151
00156 void IR_West_Neighbor_Handler(IRFrame *frame) {
00157   if(frame->type == IR_FRAME_TYPE_NEIGHBOR) {
00158     neighbors[IR_WEST_ID-1] = TRUE;
00159     neighbors_id[IR_WEST_ID-1] = frame->src_id;
00160     neighbor_update_flag[IR_WEST_ID-1] = TRUE;
00161   } else if (frame->type == IR_FRAME_TYPE_USER && IR_West_User_Handler != NULL) {

00162     (*IR_West_User_Handler)(frame);
00163   }
00164   free(frame->data);
00165   free(frame);
00166 }
00167
00172 uint8_t IR_Get_Neighbor(IR_DIR face_dir) {
00173   return neighbors_id[(uint8_t)face_dir-1];
00174 }
```

6.79 feature_modules/neighbor_detect.h File Reference

Contains function prototypes for the neighbor detection module.

```
#include "stm32f10x.h"
#include "blox_ir.h"
#include "blox_tim.h"
#include "blox_usb.h"
```

- #define **NEIGHBOR_TIMx** 6
- #define **NEIGHBOR_TIM_CLK** 10000
- #define **NEIGHBOR_SAMPLE_PERIOD** 1000
- void **Neighbor_Detect_Init** (void)

  Initializes the Neighbor Detection module.

- void **Neighbor_Register_IR_RX_IRQ** (uint8_t id, void(∗IR_User_Handler)(**IRFrame** ∗frame))

  Register a receive interrupt for user frames.

- uint8_t **IR_Get_Neighbor** (**IR_DIR** face_id)

  Gets the neighbor for a given face.

6.79.1 Detailed Description

Contains function prototypes for the neighbor detection module.

Author

Zach Wasson

Version

V0.1

Date

11/17/2010

Definition in file **neighbor_detect.h**.

6.79.2 Function Documentation

6.79.2.1 uint8_t IR_Get_Neighbor (IR_DIR *face_dir*)

Gets the neighbor for a given face.

Return values

The	id of a neighbor if one is present

Definition at line **172** of file **neighbor_detect.c**.

6.79.2.2 void Neighbor_Detect_Init (void)

Initializes the Neighbor Detection module.

Return values

None	

Definition at line **33** of file **neighbor_detect.c**.

6.79.2.3 void Neighbor_Register_IR_RX_IRQ (uint8_t *id*, void(*)(IRFrame *frame) *IR_User_Handler*)

Register a receive interrupt for user frames.

Parameters

id	the id of the IR
IR_User_Handler	the function that will be called when a user frame is received

Return values

None	

Definition at line **61** of file **neighbor_detect.c**.

6.80 feature_modules/neighbor_detect.h

```
00001
00008 #ifndef __NEIGHBOR_DETECT
00009 #define __NEIGHBOR_DETECT
00010
00011 #include "stm32f10x.h"
00012 #include "blox_ir.h"
00013 #include "blox_tim.h"
00014
00015 #include "blox_usb.h"
00016
00021 #define NEIGHBOR_TIMx             6
00022 #define NEIGHBOR_TIM_CLK          10000
00023 #define NEIGHBOR_SAMPLE_PERIOD    1000
00024
00025 void Neighbor_Detect_Init(void);
00026 void Neighbor_Register_IR_RX_IRQ(uint8_t id, void (*IR_User_Handler)(IRFrame *fra
      me));
00027 uint8_t IR_Get_Neighbor(IR_DIR face_id);
00029 #endif
```

6.81 feature_modules/power.c File Reference

Defines power management functions.

```
#include "power.h"
```

- static uint32_t **numPowerSleep**
- void **Blox_Power_Register_Power** (void(∗Power_Wake)(void), void(∗Power_Sleep)(void))

  Registers a new function to be called when the system wakes or sleeps. Only adds if it isn't already there.

- void **Blox_Power_Sleep** (void)

  Puts the system to sleep.

- void **Blox_Power_Wake** (void)

  Wakes the system up.

6.81.1 Detailed Description

Defines power management functions.

Author

Zach Wasson

Version

V0.1

Date

11/20/2010

Definition in file **power.c**.

6.82 feature_modules/power.c

```
00001
00008 #include "power.h"
00009
00014 static void (*powerWake[MAX_POWER_WAKE_FN])(void);
00015 static void (*powerSleep[MAX_POWER_SLEEP_FN])(void);
00016 static uint32_t numPowerWake, numPowerSleep;
00017
00025 void Blox_Power_Register_Power(void (*Power_Wake)(void), void (*Power_Sleep)(void
      )) {
00026   uint8_t i;
00027   if (numPowerWake == MAX_POWER_WAKE_FN) {
```

```
00028     Blox_DebugPat("Blox_Power_Register_Power numPowerWake reached max of:%d\r\n",

00029         MAX_POWER_WAKE_FN);
00030     while(1) ;
00031   }
00032
00033   for (i = 0; i < numPowerWake; i++) {
00034     if(powerWake[i] == Power_Wake)
00035       return;
00036   }
00037
00038   powerWake[numPowerWake++] = Power_Wake;
00039
00040   if (numPowerSleep == MAX_POWER_SLEEP_FN) {
00041     Blox_DebugPat("Blox_Power_Register_Power numPowerSleep reached max of:%d\r\n"
    ,
00042         MAX_POWER_SLEEP_FN);
00043     while(1) ;
00044   }
00045
00046   for (i = 0; i < numPowerSleep; i++) {
00047     if(powerSleep[i] == Power_Sleep)
00048       return;
00049   }
00050
00051   powerSleep[numPowerSleep++] = Power_Sleep;
00052 }
00053
00058 void Blox_Power_Sleep(void) {
00059   uint8_t i;
00060   for (i = 0; i < numPowerSleep; i++) {
00061     if(powerSleep[i] != NULL)
00062       (*powerSleep[i])();
00063   }
00064 }
00065
00070 void Blox_Power_Wake(void) {
00071   uint8_t i;
00072   for (i = 0; i < numPowerWake; i++) {
00073     if(powerWake[i] != NULL)
00074       (*powerWake[i])();
00075   }
00076 }
```

6.83 feature_modules/power.h File Reference

Prototypes for power management functions and definitions.

```
#include "blox_system.h"
```

- #define **MAX_POWER_WAKE_FN** 32
- #define **MAX_POWER_SLEEP_FN** 32
- void **Blox_Power_Register_Power** (void(∗Power_Wake)(void), void(∗Power_Sleep)(void))

  Registers a new function to be called when the system wakes or sleeps. Only adds if it isn't already there.

- void **Blox_Power_Sleep** (void)

  Puts the system to sleep.

- void **Blox_Power_Wake** (void)

  Wakes the system up.

6.83.1 Detailed Description

Prototypes for power management functions and definitions.

Author

Zach Wasson

Version

V0.1

Date

11/20/2010

Definition in file **power.h**.

6.84 feature_modules/power.h

```
00001
00008 #ifndef __BLOX_POWER_H
00009 #define __BLOX_POWER_H
00010
00011 #include "blox_system.h"
00012
00017 #define MAX_POWER_WAKE_FN     32
00018 #define MAX_POWER_SLEEP_FN    32
00019
00020 void Blox_Power_Register_Power(void (*Power_Wake)(void), void (*Power_Sleep)(void
```

```
    ));
00021 void Blox_Power_Sleep(void);
00022 void Blox_Power_Wake(void);
00024 #endif
```

6.85 system_programs/base_program/blox_base_ui.c File Reference

Functions used to create the base program UI.

```
#include "blox_base_ui.h"
```

- char **entry_names** [MAX_ENTRIES][FS_FILE_MAX_NAME_LEN]
- ptrVoidFn **entry_handlers** [MAX_ENTRIES]
- ptrVoidFn **back_handler**
- uint8_t **selected_entry**
- uint8_t **total_entries**
- void **Blox_UI_DrawEntry** (char ∗entry, uint8_t index, uint16_t color)

  Draws a single entry.

- void **Blox_UI_DrawHeader** (void)

  Draws the header for the UI.

- void **Blox_UI_DrawFooter** (void)

  Draws the footer for the UI.

- void **Blox_UI_DrawTitle** (char ∗title)

  Draws the title for the UI.

- void **Blox_UI_SetEntries** (char ∗∗entries, ptrVoidFn ∗entries_handler, ptrVoidFn Back-Handler, uint8_t numEntries)

  Sets the entries for the UI page.

- void **Blox_UI_DrawEntries** (void)

  Draws the entries for the UI page.

- void **Blox_UI_SelectEntry** (char ∗entry)

  Selects an entry for the UI.

- void **Blox_UI_SelectEntryAbove** (void)

  Selects the entry above the currently selected one.

- void **Blox_UI_SelectEntryBelow** (void)

  Selects the entry below the currently selected one.

- void **Blox_UI_RunEntry** (void)

  Runs the currently selected entry.

- uint8_t **Blox_UI_GetEntryID** (void)

  Gets the id of the selected entry.

- void **Blox_UI_Back** (void)

  Calls the Back function for the UI page.

- void **Blox_UI_ClearScreen** (void)

  Clears the current UI page.

6.85.1 Detailed Description

Functions used to create the base program UI.

Author

Zach Wasson

Version

V0.1

Date

11/21/2010

Definition in file **blox_base_ui.c**.

6.85.2 Function Documentation

6.85.2.1 void Blox_UI_Back (void)

Calls the Back function for the UI page.

Return values

None	

Definition at line **179** of file **blox_base_ui.c**.

6.85.2.2 void Blox_UI_ClearScreen (void)

Clears the current UI page.

Return values

None	

Definition at line **189** of file **blox_base_ui.c**.

6.85.2.3 void Blox_UI_DrawEntries (void)

Draws the entries for the UI page.

Return values

None	

Definition at line **80** of file **blox_base_ui.c**.

6.85.2.4 void Blox_UI_DrawFooter (void)

Draws the footer for the UI.

Return values

None	

Definition at line **40** of file **blox_base_ui.c**.

6.85.2.5 void Blox_UI_DrawHeader (void)

Draws the header for the UI.

Return values

None	

Definition at line **26** of file **blox_base_ui.c**.

6.85.2.6 void Blox_UI_DrawTitle (char * *title*)

Draws the title for the UI.

Parameters

title	The title text to be displayed

Return values

None	

Definition at line **51** of file **blox_base_ui.c**.

6.85.2.7 uint8_t Blox_UI_GetEntryID (void)

Gets the id of the selected entry.

Return values

The	id of the currently selected entry

Definition at line **171** of file **blox_base_ui.c**.

6.85.2.8 void Blox_UI_RunEntry (void)

Runs the currently selected entry.

Return values

None	

Definition at line **161** of file **blox_base_ui.c**.

6.85.2.9 void Blox_UI_SelectEntry (char * *entry*)

Selects an entry for the UI.

Parameters

entry	The entry to be selected

Return values

None	

Definition at line **116** of file **blox_base_ui.c**.

6.85.2.10 void Blox_UI_SelectEntryAbove (void)

Selects the entry above the currently selected one.

Return values

None	

Definition at line **131** of file **blox_base_ui.c**.

6.85.2.11 void Blox_UI_SelectEntryBelow (void)

Selects the entry below the currently selected one.

Return values

None	

Definition at line **146** of file **blox_base_ui.c**.

6.85.2.12 void Blox_UI_SetEntries (char ** *entries,* ptrVoidFn * *entries_handler,* ptrVoidFn *BackHandler,* uint8_t *numEntries*)

Sets the entries for the UI page.

Parameters

entries	The names of entries for the UI page
entries_handler	The handler functions for the entries
BackHandler	The handler function for going back a page
numEntries	The number of entries for the UI page

Return values

None	

Definition at line **65** of file **blox_base_ui.c**.

6.86 system_programs/base_program/blox_base_ui.c

```
00001
00008 #include "blox_base_ui.h"
00009
00014 void Blox_UI_DrawEntry(char *entry, uint8_t index, uint16_t color);
00015
00016 char entry_names[MAX_ENTRIES][FS_FILE_MAX_NAME_LEN];
00017 ptrVoidFn entry_handlers[MAX_ENTRIES];
00018 ptrVoidFn back_handler;
00019 uint8_t selected_entry;
00020 uint8_t total_entries;
00021
00026 void Blox_UI_DrawHeader(void) {
00027   char buffer[13];
00028   Blox_OLED_DrawRectangle(0, 0, 127, HEADER_HEIGHT, HEADER_BACKGROUND_COLOR);
00029   Blox_OLED_DrawStringGraphics(HEADER_TITLE_LOCATION_X, HEADER_TITLE_LOCATION_Y,
      OLED_FONT_5X7,
00030                                HEADER_TEXT_COLOR, 1, 1, "Base Program");
00031   sprintf(buffer, "ID: %.1d", Blox_System_GetId());
00032   Blox_OLED_DrawStringGraphics(HEADER_VERSION_LOCATION_X, HEADER_VERSION_LOCATION
      _Y, OLED_FONT_5X7,
00033                                HEADER_TEXT_COLOR, 1, 1, buffer);
00034 }
00035
00040 void Blox_UI_DrawFooter(void) {
00041   Blox_OLED_DrawRectangle(0, 127-FOOTER_HEIGHT, 127, 127, FOOTER_BACKGROUND_COLOR
      );
00042   Blox_OLED_DrawStringGraphics(FOOTER_COPYRIGHT_LOCATION_X, FOOTER_COPYRIGHT_LOCA
      TION_Y, OLED_FONT_5X7,
00043                                FOOTER_TEXT_COLOR, 1, 1, FOOTER_TEXT);
00044 }
00045
00051 void Blox_UI_DrawTitle(char *title) {
00052   Blox_OLED_DrawStringGraphics(CENTER_TEXT(title, 8), TITLE_LOCATION_Y, OLED_FONT
      _8X12,
00053                                TITLE_COLOR, TITLE_SIZE, TITLE_SIZE, title);
00054   Blox_OLED_DrawLine(TITLE_LINE_X_START, TITLE_LINE_Y, TITLE_LINE_X_END, TITLE_LI
      NE_Y, TITLE_LINE_COLOR);
00055 }
00056
00065 void Blox_UI_SetEntries(char **entries, ptrVoidFn *entries_handler, ptrVoidFn Bac
      kHandler, uint8_t numEntries) {
00066   uint8_t i;
00067   for(i = 0; i < numEntries; i++) {
00068     strcpy(entry_names[i], entries[i]);
00069     entry_handlers[i] = entries_handler[i];
00070   }
00071   total_entries = numEntries;
00072   selected_entry = 0;
00073   back_handler = BackHandler;
00074 }
00075
00080 void Blox_UI_DrawEntries(void) {
00081   uint8_t i;
00082   for(i = 0; i < total_entries; i++) {
00083     Blox_UI_DrawEntry(entry_names[i], i, ENTRY_COLOR);
```

```
00084   }
00085   Blox_UI_SelectEntry(entry_names[selected_entry]);
00086 }
00087
00095 void Blox_UI_DrawEntry(char *entry, uint8_t index, uint16_t color) {
00096   if(strlen(entry) > MAX_ENTRY_LENGTH) {
00097     char buffer[MAX_ENTRY_LENGTH+1];
00098     strncpy(buffer, entry, MAX_ENTRY_LENGTH-3);
00099     buffer[MAX_ENTRY_LENGTH-3] = '.';
00100     buffer[MAX_ENTRY_LENGTH-2] = '.';
00101     buffer[MAX_ENTRY_LENGTH-1] = '.';
00102     buffer[MAX_ENTRY_LENGTH] = 0;
00103     Blox_OLED_DrawStringGraphics(CENTER_TEXT(buffer, 6), ENTRY_START_LOCATION+ind
      ex*ENTRY_LOCATION_OFFSET,
00104                                  OLED_FONT_5X7, color, 1, 1, buffer);
00105   } else {
00106     Blox_OLED_DrawStringGraphics(CENTER_TEXT(entry, 6), ENTRY_START_LOCATION+inde
      x*ENTRY_LOCATION_OFFSET,
00107                                  OLED_FONT_5X7, color, 1, 1, entry);
00108   }
00109 }
00110
00116 void Blox_UI_SelectEntry(char *entry) {
00117   uint8_t i;
00118   for(i = 0; i < total_entries; i++) {
00119     if(strcmp(entry_names[i], entry) == 0) {
00120       Blox_UI_DrawEntry(entry_names[selected_entry], selected_entry, ENTRY_COLOR)
      ;
00121       Blox_UI_DrawEntry(entry, i, ENTRY_SELECTED_COLOR);
00122       selected_entry = i;
00123     }
00124   }
00125 }
00126
00131 void Blox_UI_SelectEntryAbove(void) {
00132   if(total_entries == 0 || total_entries == 1)
00133     return;
00134   Blox_UI_DrawEntry(entry_names[selected_entry], selected_entry, ENTRY_COLOR);
00135   if(selected_entry == 0)
00136     selected_entry = total_entries - 1;
00137   else
00138     selected_entry = selected_entry - 1;
00139   Blox_UI_DrawEntry(entry_names[selected_entry], selected_entry, ENTRY_SELECTED_C
      OLOR);
00140 }
00141
00146 void Blox_UI_SelectEntryBelow(void) {
00147   if(total_entries == 0 || total_entries == 1)
00148     return;
00149   Blox_UI_DrawEntry(entry_names[selected_entry], selected_entry, ENTRY_COLOR);
00150   if(selected_entry == total_entries - 1)
00151     selected_entry = 0;
00152   else
00153     selected_entry = selected_entry + 1;
00154   Blox_UI_DrawEntry(entry_names[selected_entry], selected_entry, ENTRY_SELECTED_C
      OLOR);
00155 }
```

```
00156
00161 void Blox_UI_RunEntry(void) {
00162   if(total_entries != 0 && entry_handlers[selected_entry] != NULL) {
00163     (*entry_handlers[selected_entry])();
00164   }
00165 }
00166
00171 uint8_t Blox_UI_GetEntryID(void) {
00172   return selected_entry;
00173 }
00174
00179 void Blox_UI_Back(void) {
00180   if(back_handler != NULL) {
00181     (*back_handler)();
00182   }
00183 }
00184
00189 void Blox_UI_ClearScreen(void) {
00190   total_entries = 0;
00191   Blox_OLED_Clear();
00192 }
```

6.87 system_programs/base_program/blox_base_ui.h File Reference

Function prototypes for the base program user interface.

```
#include "string.h"

#include "stdio.h"

#include "blox_oled.h"

#include "blox_filesystem.h"
```

- #define **CENTER_TEXT**(text, width) (64-width∗TITLE_SIZE∗strlen(text)/2)
- #define **HEADER_BACKGROUND_COLOR** COLOR_BLUE
- #define **HEADER_TEXT_COLOR** COLOR_WHITE
- #define **HEADER_HEIGHT** 12
- #define **HEADER_TITLE_LOCATION_X** 3
- #define **HEADER_TITLE_LOCATION_Y** 2
- #define **HEADER_VERSION_LOCATION_X** 96
- #define **HEADER_VERSION_LOCATION_Y** 2
- #define **TITLE_SIZE** 1
- #define **TITLE_LOCATION_Y** 19
- #define **TITLE_COLOR** COLOR_WHITE
- #define **TITLE_LINE_X_START** 15
- #define **TITLE_LINE_X_END** 127-(TITLE_LINE_X_START+1)

- #define **TITLE_LINE_Y** 35
- #define **TITLE_LINE_COLOR** COLOR_BLUE
- #define **ENTRY_START_LOCATION** 43
- #define **ENTRY_LOCATION_OFFSET** 14
- #define **MAX_ENTRIES** 5
- #define **ENTRY_COLOR** COLOR_WHITE
- #define **ENTRY_SELECTED_COLOR** COLOR_YELLOW
- #define **MAX_ENTRY_LENGTH** 20
- #define **FOOTER_TEXT** "Project Blox"
- #define **FOOTER_BACKGROUND_COLOR** COLOR_BLUE
- #define **FOOTER_TEXT_COLOR** COLOR_WHITE
- #define **FOOTER_HEIGHT** 12
- #define **FOOTER_COPYRIGHT_LOCATION_X** CENTER_TEXT(FOOTER_TEXT, 6)
- #define **FOOTER_COPYRIGHT_LOCATION_Y** 117
- void **Blox_UI_DrawHeader** (void)

  Draws the header for the UI.

- void **Blox_UI_DrawFooter** (void)

  Draws the footer for the UI.

- void **Blox_UI_DrawTitle** (char ∗title)

  Draws the title for the UI.

- void **Blox_UI_SetEntries** (char ∗∗entries, ptrVoidFn ∗entries_handler, ptrVoidFn Back-Handler, uint8_t numEntries)

  Sets the entries for the UI page.

- void **Blox_UI_DrawEntries** (void)

  Draws the entries for the UI page.

- void **Blox_UI_SelectEntry** (char ∗entry)

  Selects an entry for the UI.

- void **Blox_UI_SelectEntryAbove** (void)

  Selects the entry above the currently selected one.

- void **Blox_UI_SelectEntryBelow** (void)

  Selects the entry below the currently selected one.

- void **Blox_UI_RunEntry** (void)

  Runs the currently selected entry.

- uint8_t **Blox_UI_GetEntryID** (void)

Gets the id of the selected entry.

- void **Blox_UI_Back** (void)

  Calls the Back function for the UI page.

- void **Blox_UI_ClearScreen** (void)

  Clears the current UI page.

6.87.1 Detailed Description

Function prototypes for the base program user interface.

Author

Zach Wasson

Version

V0.1

Date

11/21/2010

Definition in file **blox_base_ui.h**.

6.88 system_programs/base_program/blox_base_ui.h

```
00001
00008 #ifndef __BLOX_BASE_UI_H
00009 #define __BLOX_BASE_UI_H
00010
00011 #include "string.h"
00012 #include "stdio.h"
00013 #include "blox_oled.h"
00014 #include "blox_filesystem.h"
00015
00020 #define CENTER_TEXT(text, width) (64-width*TITLE_SIZE*strlen(text)/2)
00021
00022 //header
00023 #define HEADER_BACKGROUND_COLOR       COLOR_BLUE
00024 #define HEADER_TEXT_COLOR             COLOR_WHITE
00025 #define HEADER_HEIGHT                 12
00026 #define HEADER_TITLE_LOCATION_X       3
00027 #define HEADER_TITLE_LOCATION_Y       2
00028 #define HEADER_VERSION_LOCATION_X     96
00029 #define HEADER_VERSION_LOCATION_Y     2
```

```
00030
00031 //title
00032 #define TITLE_SIZE                  1
00033 #define TITLE_LOCATION_Y            19
00034 #define TITLE_COLOR                 COLOR_WHITE
00035 #define TITLE_LINE_X_START          15
00036 #define TITLE_LINE_X_END            127-(TITLE_LINE_X_START+1)
00037 #define TITLE_LINE_Y                35
00038 #define TITLE_LINE_COLOR            COLOR_BLUE
00039
00040 //entry
00041 #define ENTRY_START_LOCATION        43
00042 #define ENTRY_LOCATION_OFFSET       14
00043 #define MAX_ENTRIES                 5
00044 #define ENTRY_COLOR                 COLOR_WHITE
00045 #define ENTRY_SELECTED_COLOR        COLOR_YELLOW
00046 #define MAX_ENTRY_LENGTH            20
00047
00048 //footer
00049 #define FOOTER_TEXT                 "Project Blox"
00050 #define FOOTER_BACKGROUND_COLOR     COLOR_BLUE
00051 #define FOOTER_TEXT_COLOR           COLOR_WHITE
00052 #define FOOTER_HEIGHT               12
00053 #define FOOTER_COPYRIGHT_LOCATION_X CENTER_TEXT(FOOTER_TEXT, 6)
00054 #define FOOTER_COPYRIGHT_LOCATION_Y 117
00055
00056 void Blox_UI_DrawHeader(void);
00057 void Blox_UI_DrawFooter(void);
00058 void Blox_UI_DrawTitle(char *title);
00059 void Blox_UI_SetEntries(char **entries, ptrVoidFn *entries_handler,
00060                         ptrVoidFn BackHandler, uint8_t numEntries);
00061 void Blox_UI_DrawEntries(void);
00062 void Blox_UI_SelectEntry(char *entry);
00063 void Blox_UI_SelectEntryAbove(void);
00064 void Blox_UI_SelectEntryBelow(void);
00065 void Blox_UI_RunEntry(void);
00066 uint8_t Blox_UI_GetEntryID(void);
00067 void Blox_UI_Back(void);
00068 void Blox_UI_ClearScreen(void);
00070 #endif
```

6.89 system_programs/base_program/blox_transfer.c File Reference

This module facilitates two systems to manage applications on each other.

```
#include "blox_transfer.h"
```

Functions

- **TRANSFER_STATUS Cmd_RCV_APP** (void)

  Receives an application from a sender and stores it in the fs.

- **TRANSFER_STATUS Cmd_DEL_APP** (void)

  Receives an app. id from a sender and removes it from the fs.

- **TRANSFER_STATUS Cmd_LST_APPS** (void)

  Sends a command with headers for all the applications in the fs.

- **TRANSFER_STATUS Cmd_RUN_APP** (void)

  Receives an app. id, retrieves it from the fs, and runs the app.

- void **Transfer_Init** (void)

  Initializes the transfer module.

- void **Transfer_Slave** (void)

  Run in slave mode accepting and processing commands.

6.89.1 Detailed Description

This module facilitates two systems to manage applications on each other.

Author

Jesse Tannahill

Version

V0.1

Date

10/18/2010

Definition in file **blox_transfer.c**.

6.90 system_programs/base_program/blox_transfer.c

```
00001
00010 #include "blox_transfer.h"
00011
00016 /* Private function prototypes */
00017 TRANSFER_STATUS Cmd_RCV_APP(void);
00018 TRANSFER_STATUS Cmd_DEL_APP(void);
00019 TRANSFER_STATUS Cmd_LST_APPS(void);
00020 TRANSFER_STATUS Cmd_RUN_APP(void);
00021
```

```
00026 void Transfer_Init(void) {
00027   USB_Init();
00028   FS_Init(1);
00029 }
00030
00034 void Transfer_Slave(void) {
00035   uint8_t checksum;
00036   TRANSFER_OPCODE opcode;
00037
00038   while (1) {
00039     opcode = (TRANSFER_OPCODE)USB_Receive();
00040     checksum = USB_Receive();
00041     if ((checksum+opcode) != 0xFF) {
00042       USB_Send(TRANSFER_NAK);
00043       continue;
00044     }
00045     USB_Send(TRANSFER_ACK);
00046
00047     switch(opcode) {
00048     case RCV_APP:
00049       Cmd_RCV_APP();
00050       break;
00051           case DEL_APP:
00052       Cmd_DEL_APP();
00053       break;
00054           case LST_APPS:
00055       Cmd_LST_APPS();
00056       break;
00057           case RUN_APP:
00058       Cmd_RUN_APP();
00059       break;
00060     }
00061   }
00062 }
00063
00069 TRANSFER_STATUS Cmd_RCV_APP(void) {
00070   uint8_t checksum, name_len, id, numPages, *page;
00071   uint16_t i, j;
00072   char name[FS_FILE_MAX_NAME_LEN];
00073   uint32_t size, remaining, read_amt;
00074   /*** Get # of characters in filename ***/
00075   checksum = 0;
00076   name_len = USB_Receive();
00077   checksum = USB_Receive();
00078   if((checksum+name_len) != 0xFF || name_len > FS_FILE_MAX_NAME_LEN-1) {
00079     USB_Send(TRANSFER_NAK);
00080     return TRANSFER_CMD_FAIL;
00081   }
00082   USB_Send(TRANSFER_ACK);
00083   /*** Get filename ***/
00084   checksum = 0;
00085   for(i = 0; i < name_len; i++) {
00086     name[i] = USB_Receive();
00087     checksum += name[i];
00088   }
00089   name[i] = '\0';
00090   checksum += USB_Receive();
```

```
00091   if(checksum != 0xFF) {
00092     USB_Send(TRANSFER_NAK);
00093     return TRANSFER_CMD_FAIL;
00094   }
00095   USB_Send(TRANSFER_ACK);
00096
00097   /*** Get file size in pages ***/
00098   checksum = 0;
00099   size = USB_Receive()
00100        | (USB_Receive() << 8)
00101        | (USB_Receive() << 16)
00102        | (USB_Receive() << 24);
00103   checksum += size & 0xFF;
00104   checksum += (size >> 8) & 0xFF;
00105   checksum += (size >> 16) & 0xFF;
00106   checksum += (size >> 24) & 0xFF;
00107   checksum += USB_Receive();
00108
00109   numPages = FS_RoundPageUp(size);
00110   id = FS_CreateFile(name, numPages);
00111   if (checksum != 0xFF || id == FS_MAX_FILES) {
00112     USB_Send(TRANSFER_NAK);
00113     return TRANSFER_CMD_FAIL;
00114   }
00115   USB_Send(TRANSFER_ACK);
00116
00117   /*** Receive pages 1 at a time ***/
00118   remaining = size;
00119   page = (uint8_t *)malloc(PAGE_SIZE);
00120   for(i = 0; i < numPages; i++) {
00121     memset(page, 0, PAGE_SIZE); //Zero for funs
00122     checksum = 0;
00123     if(remaining < PAGE_SIZE)
00124       read_amt = remaining; //Don't read too much on the last page
00125     else
00126       read_amt = PAGE_SIZE;
00127     for(j = 0; j < read_amt; j++) {
00128       page[j] = USB_Receive();
00129       checksum += page[j];
00130     }
00131     checksum += USB_Receive();
00132     FS_WriteFilePage(id, (uint32_t *)page, i);
00133     if (checksum != 0xFF || id == FS_MAX_FILES) {
00134       USB_Send(TRANSFER_NAK);
00135       FS_DeleteFile(id);
00136       free(page);
00137       return TRANSFER_CMD_FAIL;
00138     }
00139     USB_Send(TRANSFER_ACK);
00140     remaining -= PAGE_SIZE;
00141   }
00142   free(page);
00143   return TRANSFER_OK;
00144 }
00145
00151 TRANSFER_STATUS Cmd_DEL_APP(void) {
00152 uint8_t checksum, id;
```

```
00153   /*** Receive file id ***/
00154   id = USB_Receive();
00155   checksum = id;
00156   checksum += USB_Receive();
00157   if(checksum != 0xFF
00158      || id >= FS_GetNumFiles()
00159      || FS_DeleteFile(id) != FS_OK) {
00160     USB_Send(TRANSFER_NAK);
00161     return TRANSFER_CMD_FAIL;
00162   }
00163   USB_Send(TRANSFER_ACK);
00164
00165   return TRANSFER_OK;
00166 }
00167
00173 TRANSFER_STATUS Cmd_LST_APPS(void) {
00174   uint8_t checksum, numFiles, i, j;
00175   FS_File *file;
00176   /*** Send number of files in FS ***/
00177   numFiles = FS_GetNumFiles();
00178   checksum = 0xFF - numFiles;
00179   USB_Send(numFiles);
00180   USB_Send(checksum);
00181   if (USB_Receive() == TRANSFER_NAK)
00182     return TRANSFER_CMD_FAIL;
00183   /*** Send files 1 at a time ***/
00184   for (i = 0; i < numFiles; i++) {
00185     file = FS_GetFile(i);
00186     checksum = 0;
00187     USB_Send(file->id);
00188     checksum += file->id;
00189     for (j = 0; j < FS_FILE_MAX_NAME_LEN; j++) {
00190       USB_Send(file->name[j]);
00191       checksum += file->name[j];
00192     }
00193     checksum += file->numPages;
00194     USB_Send(file->numPages);
00195     USB_Send(0xFF-checksum);
00196
00197     if (USB_Receive() != TRANSFER_ACK)
00198       return TRANSFER_CMD_FAIL;
00199   }
00200
00201   return TRANSFER_OK;
00202 }
00203
00208 TRANSFER_STATUS Cmd_RUN_APP(void) {
00209   uint8_t checksum, id;
00210
00211   /*** Receive file id ***/
00212   id = USB_Receive();
00213   checksum = id;
00214   checksum += USB_Receive();
00215   if(checksum != 0xFF
00216      || id >= FS_GetNumFiles()) {
00217     USB_Send(TRANSFER_NAK);
00218     return TRANSFER_CMD_FAIL;
```

```
00219   }
00220   USB_Send(TRANSFER_ACK);
00221
00222   //Run the application in the file
00223   FS_RunFile(id);
00224
00225   return TRANSFER_CMD_FAIL; //Shouldn't ever get here.
00226 }
```

6.91 system_programs/base_program/blox_transfer.h File Reference

Contains function prototypes for the transfer interface.

```
#include "blox_system.h"
#include "blox_usb.h"
#include "blox_filesystem.h"
```

- #define **TRANSFER_ACK** 0x79
- #define **TRANSFER_NAK** 0x1F
- enum **TRANSFER_STATUS** { **TRANSFER_OK** = 0, **TRANSFER_CMD_FAIL**, **TRANSFER_INV_OPCODE** }

  Enum of the possible transfer statuses.

- enum **TRANSFER_OPCODE** {

  OP_BOT = 0, **RCV_APP**, **DEL_APP**, **LST_APPS**,

  RUN_APP, **OP_TOP** }

  Enum of the available opcodes.

- void **Transfer_Init** (void)

  Initializes the transfer module.

- void **Transfer_Slave** (void)

  Run in slave mode accepting and processing commands.

6.91.1 Detailed Description

Contains function prototypes for the transfer interface.

Author

Jesse Tannahill

Version

V0.1

Date

10/18/2010

Definition in file **blox_transfer.h**.

6.92 system_programs/base_program/blox_transfer.h

```
00001
00008 #ifndef __BLOX_TRANSFER_H
00009 #define __BLOX_TRANSFER_H
00010
00011 #include "blox_system.h"
00012 #include "blox_usb.h"
00013 #include "blox_filesystem.h"
00014
00019 #define TRANSFER_ACK 0x79
00020 #define TRANSFER_NAK 0x1F
00021
00025 typedef enum {
00026         TRANSFER_OK = 0,
00027         TRANSFER_CMD_FAIL,
00028         TRANSFER_INV_OPCODE
00029 } TRANSFER_STATUS;
00030
00034 typedef enum {
00035   OP_BOT = 0,
00036         RCV_APP,
00037         DEL_APP,
00038         LST_APPS,
00039         RUN_APP,
00040   OP_TOP
00041 } TRANSFER_OPCODE;
00042
00043 void Transfer_Init(void);
00044 void Transfer_Slave(void);
00046 #endif
```

6.93 system_programs/base_program/user/base_program.c File Reference

Provides a filesystem interface to a section of flash memory. The base program sits on a Blox and is the only program that runs at startup. The user interface is largely

interrupt-driven, responding to stimulus from the touchpanels until a user wants to a perform an action. Main actions are to load new programs via the Transfer driver and running a program that is already loaded in flash via the Flash driver.

```
#include "blox_system.h"
#include "blox_transfer.h"
#include "blox_usb.h"
#include "blox_xbee.h"
#include "blox_filesystem.h"
#include "blox_role.h"
#include "blox_led.h"
#include "blox_base_ui.h"
#include "blox_gesture.h"
#include "string.h"
#include "stdio.h"
```

- #define **MAIN_MENU_NUM_ENTRIES** 4
- #define **TEXT_START_LOCATION** 43
- #define **TEXT_LOCATION_LEFT_ALIGNED** 16
- #define **TEXT_LOCATION_OFFSET** 14
- void **Base_RX** (**BloxFrame** *frame)

  Listens for other Blox asking for new participants.

- void **Base_UI_MainMenu** (void)

  Draws the main menu.

- void **Base_UI_ApplicationsMenu** (void)

  Draws the applications menu.

- void **Base_UI_CalibrationMenu** (void)

  Draws the calibration menu.

- void **Base_UI_SysInfoMenu** (void)

  Draws the system info menu. The system info menu displays the ID of the Blox, the current number of programs, and the amount of free and used space.

- void **Base_UI_USBMenu** (void)

Draws the USB menu and starts listening for USB packets.

- void **Base_UI_Loading** (void)

  Draws a loading screen.

- int **main** (void)

  Provides a GUI for the base program. The base program allows users to select applications to run, calibrate system variables, receive USB commands, and view system info.

- void **Application_Handler** (void)

  Handler for calling applications using blox_base_ui.

6.93.1 Detailed Description

Provides a filesystem interface to a section of flash memory. The base program sits on a Blox and is the only program that runs at startup. The user interface is largely interrupt-driven, responding to stimulus from the touchpanels until a user wants to a perform an action. Main actions are to load new programs via the Transfer driver and running a program that is already loaded in flash via the Flash driver.

Author

Jesse Tannahill/Zach Wasson

Version

V0.2

Date

10/18/2010

Definition in file **base_program.c**.

6.94 system_programs/base_program/user/base_program.c

```
00001
00015 #include "blox_system.h"
00016 #include "blox_transfer.h"
00017 #include "blox_usb.h"
00018 #include "blox_xbee.h"
00019 #include "blox_filesystem.h"
00020 #include "blox_role.h"
00021 #include "blox_led.h"
```

```
00022 #include "blox_base_ui.h"
00023 #include "blox_gesture.h"
00024 #include "string.h"
00025 #include "stdio.h"
00026
00031 #define MAIN_MENU_NUM_ENTRIES     4
00032
00033 #define TEXT_START_LOCATION          43
00034 #define TEXT_LOCATION_LEFT_ALIGNED   16
00035 #define TEXT_LOCATION_OFFSET         14
00036
00037 void Base_RX(BloxFrame *frame);
00038
00039 void Base_UI_MainMenu(void);
00040 void Base_UI_ApplicationsMenu(void);
00041 void Base_UI_CalibrationMenu(void);
00042 void Base_UI_SysInfoMenu(void);
00043 void Base_UI_USBMenu(void);
00044 void Base_UI_Loading(void);
00045
00053 int main(void)
00054 {
00055   if (FS_GetAppFlag() == 1) {
00056     FS_RunStage();
00057   }
00058
00059   Blox_System_Init();
00060   FS_Init(1);
00061   Blox_LED_Init();
00062   Blox_XBee_Init();
00063   Blox_XBee_Register_RX_IRQ(&Base_RX);
00064   Blox_XBee_Enable_RX_IRQ();
00065   SysTick_Init();
00066   USB_Init();
00067   Blox_OLED_Init();
00068   Blox_Gesture_Init();
00069
00070   Base_UI_MainMenu();
00071
00072   while(1) {
00073     static uint32_t old_north_time, old_south_time, old_east_time, old_west_time;

00074
00075     if(Blox_Gesture_GetGesture(TOUCH_NORTH_ID) == TOUCH_GESTURE_TAP &&
00076        Blox_Gesture_GetGestureTime(TOUCH_NORTH_ID) != old_north_time) {
00077       Blox_UI_SelectEntryAbove();
00078       old_north_time = Blox_Gesture_GetGestureTime(TOUCH_NORTH_ID);
00079       Blox_LED_Toggle(LED_NORTH);
00080     }
00081     if(Blox_Gesture_GetGesture(TOUCH_SOUTH_ID) == TOUCH_GESTURE_TAP &&
00082        Blox_Gesture_GetGestureTime(TOUCH_SOUTH_ID) != old_south_time) {
00083       Blox_UI_SelectEntryBelow();
00084       old_south_time = Blox_Gesture_GetGestureTime(TOUCH_SOUTH_ID);
00085       Blox_LED_Toggle(LED_SOUTH);
00086     }
00087     if(Blox_Gesture_GetGesture(TOUCH_EAST_ID) == TOUCH_GESTURE_TAP &&
00088        Blox_Gesture_GetGestureTime(TOUCH_EAST_ID) != old_east_time) {
```

```
00089       Blox_UI_RunEntry();
00090       old_east_time = Blox_Gesture_GetGestureTime(TOUCH_EAST_ID);
00091       Blox_LED_Toggle(LED_EAST);
00092     }
00093     if(Blox_Gesture_GetGesture(TOUCH_WEST_ID) == TOUCH_GESTURE_TAP &&
00094        Blox_Gesture_GetGestureTime(TOUCH_WEST_ID) != old_west_time) {
00095       Blox_UI_Back();
00096       old_west_time = Blox_Gesture_GetGestureTime(TOUCH_WEST_ID);
00097       Blox_LED_Toggle(LED_WEST);
00098     }
00099   }
00100 }
00101
00106 void Base_UI_MainMenu(void) {
00107   char *entries[MAIN_MENU_NUM_ENTRIES];
00108   ptrVoidFn entry_handlers[MAIN_MENU_NUM_ENTRIES];
00109   entries[0] = "Applications";
00110   entries[1] = "Calibration";
00111   entries[2] = "System Info";
00112   entries[3] = "USB Mode";
00113
00114   entry_handlers[0] = Base_UI_ApplicationsMenu;
00115   entry_handlers[1] = Base_UI_CalibrationMenu;
00116   entry_handlers[2] = Base_UI_SysInfoMenu;
00117   entry_handlers[3] = Base_UI_USBMenu;
00118
00119   Blox_UI_ClearScreen();
00120   Blox_UI_DrawHeader();
00121   Blox_UI_DrawFooter();
00122   Blox_UI_DrawTitle("MAIN MENU");
00123
00124   Blox_UI_SetEntries(entries, entry_handlers, NULL, MAIN_MENU_NUM_ENTRIES);
00125   Blox_UI_DrawEntries();
00126 }
00127
00132 void Application_Handler(void) {
00133   Base_UI_Loading();
00134   FS_RunFile(Blox_UI_GetEntryID());
00135 }
00136
00141 void Base_UI_ApplicationsMenu(void) {
00142   uint8_t applications_num_entries = 0;
00143   char *entries[MAX_ENTRIES];
00144   ptrVoidFn entry_handlers[MAX_ENTRIES];
00145   uint8_t i;
00146
00147   Blox_UI_ClearScreen();
00148   Blox_UI_DrawHeader();
00149   Blox_UI_DrawFooter();
00150   Blox_UI_DrawTitle("APPLICATIONS");
00151
00152   if((applications_num_entries = FS_GetNumFiles()) > 5)
00153     applications_num_entries = 5;
00154
00155   for(i = 0; i < applications_num_entries; i++) {
00156     FS_File *file = FS_GetFile(i);
00157     entries[i] = file->name;
```

```
00158     entry_handlers[i] = Application_Handler;
00159   }
00160
00161   Blox_UI_SetEntries(entries, entry_handlers, &Base_UI_MainMenu, applications_num
      _entries);
00162   Blox_UI_DrawEntries();
00163 }
00164
00169 void Base_UI_CalibrationMenu(void) {
00170   Blox_UI_ClearScreen();
00171   Blox_UI_DrawHeader();
00172   Blox_UI_DrawFooter();
00173   Blox_UI_DrawTitle("CALIBRATION");
00174
00175   Blox_UI_SetEntries(NULL, NULL, &Base_UI_MainMenu, 0);
00176 }
00177
00184 void Base_UI_SysInfoMenu(void) {
00185   uint8_t numFiles;
00186   uint32_t freeSpace = 0;
00187   uint32_t usedSpace = 0;
00188   char buffer[20];
00189   uint8_t i;
00190
00191   Blox_UI_ClearScreen();
00192   Blox_UI_DrawHeader();
00193   Blox_UI_DrawFooter();
00194   Blox_UI_DrawTitle("SYSTEM INFO");
00195
00196   Blox_UI_SetEntries(NULL, NULL, &Base_UI_MainMenu, 0);
00197
00198   sprintf(buffer, "Blox ID: %.1d", Blox_System_GetId());
00199   Blox_OLED_DrawStringGraphics(TEXT_LOCATION_LEFT_ALIGNED, TEXT_START_LOCATION, O
      LED_FONT_5X7,
00200                                COLOR_WHITE, 1, 1, buffer);
00201
00202   numFiles = FS_GetNumFiles();
00203   sprintf(buffer, "Num Apps: %.1d", numFiles);
00204   Blox_OLED_DrawStringGraphics(TEXT_LOCATION_LEFT_ALIGNED, TEXT_START_LOCATION +
      TEXT_LOCATION_OFFSET,
00205                                OLED_FONT_5X7, COLOR_WHITE, 1, 1, buffer);
00206
00207   for(i = 0; i < numFiles; i++) {
00208     FS_File *file = FS_GetFile(i);
00209     usedSpace += file->numPages;
00210   }
00211   sprintf(buffer, "Used Space: %d", usedSpace);
00212   Blox_OLED_DrawStringGraphics(TEXT_LOCATION_LEFT_ALIGNED, TEXT_START_LOCATION +
      2*TEXT_LOCATION_OFFSET,
00213                                OLED_FONT_5X7, COLOR_WHITE, 1, 1, buffer);
00214   freeSpace = (MEM_STORE_SIZE/PAGE_SIZE) - usedSpace;
00215   sprintf(buffer, "Free Space: %d", freeSpace);
00216   Blox_OLED_DrawStringGraphics(TEXT_LOCATION_LEFT_ALIGNED, TEXT_START_LOCATION +
      3*TEXT_LOCATION_OFFSET,
00217                                OLED_FONT_5X7, COLOR_WHITE, 1, 1, buffer);
00218 }
00219
```

```
00225 void Base_UI_USBMenu(void) {
00226   Blox_UI_ClearScreen();
00227   Blox_UI_DrawHeader();
00228   Blox_UI_DrawFooter();
00229   Blox_UI_DrawTitle("USB MODE");
00230 
00231   Blox_UI_SetEntries(NULL, NULL, &Base_UI_MainMenu, 0);
00232 
00233   //connected text
00234 
00235   Transfer_Init();
00236   Transfer_Slave();
00237 }
00238 
00243 void Base_UI_Loading(void) {
00244   Blox_UI_ClearScreen();
00245   Blox_UI_DrawHeader();
00246   Blox_UI_DrawFooter();
00247   Blox_OLED_DrawStringGraphics(CENTER_TEXT("LOADING", 8), 64,
00248                                OLED_FONT_8X12, COLOR_WHITE, 1, 1, "LOADING");
00249 }
00250 
00255 void Base_RX(BloxFrame *frame) {
00256   //Addressed to me?
00257   if(frame->dst_id == XBEE_BLOX_BROADCAST_ID || frame->dst_id ==
      Blox_System_GetId()) {
00258     if (frame->type == FRAME_TYPE_ROLE) {
00259       RoleFrame *rFrame = (RoleFrame *)(&(frame->data));
00260       switch(rFrame->opcode) {
00261       case PROG_QUERY:
00262         Blox_LED_Toggle(LED1);
00263         //See if I have this application
00264         if (FS_GetFileFromName(((QueryFrame *)&(rFrame->data))->name) != NULL) {
00265           RoleFrame respFrame;
00266           SysTick_Wait(XBEE_HOLD_PERIOD); //Wait for hold period.
00267           respFrame.opcode = PROG_ACK;
00268           Blox_LED_Toggle(LED2);
00269           Blox_XBee_Send_Period((uint8_t *)&respFrame, sizeof(RoleFrame), FRAME_T
      YPE_ROLE, frame->src_id, XBEE_HOLD_PERIOD);
00270         }
00271         break;
00272       case PROG_START:
00273       {
00274         //Get the file
00275         FS_File *file;
00276         SysTick_Wait(XBEE_HOLD_PERIOD);
00277         file = FS_GetFileFromName(((QueryFrame *)&(rFrame->data))->name);
00278         FS_RunFile(file->id);
00279       }
00280         break;
00281       case PARENT_QUERY:
00282         Blox_LED_Toggle(LED2);
00283         break;
00284       }
00285     }
00286   }
00287 }
```

6.95 system_programs/blox_setup/user/blox_setup.c File Reference

Sets up the system variables and filesystem to a known good state.

```
#include "blox_system.h"
#include "blox_filesystem.h"
#include "blox_xbee.h"
#include "blox_usb.h"
```

- #define **BLOX_ID** 3
- #define **DEF_ACCEL_X** 128
- #define **DEF_ACCEL_Y** 128
- #define **DEF_ACCEL_Z** 128
- #define **DEF_TOUCH_X** 64
- #define **DEF_TOUCH_Y** 64
- void **RCC_Configuration** (void)

  Configures the clocks.

- void **GPIO_Configuration** (void)

  Configures the GPIOs.

- int **main** (void)

  Sets up the the blox system.

6.95.1 Detailed Description

Sets up the system variables and filesystem to a known good state.

Author

Jesse Tannahill

Version

V0.1

Date

11/7/2010

Definition in file **blox_setup.c**.

6.96 system_programs/blox_setup/user/blox_setup.c

```
00001
00009 #include "blox_system.h"
00010 #include "blox_filesystem.h"
00011 #include "blox_xbee.h"
00012 #include "blox_usb.h"
00013
00018 #define BLOX_ID 3
00019
00020 #define DEF_ACCEL_X 128
00021 #define DEF_ACCEL_Y 128
00022 #define DEF_ACCEL_Z 128
00023
00024 #define DEF_TOUCH_X 64
00025 #define DEF_TOUCH_Y 64
00026
00027 void RCC_Configuration (void);
00028 void GPIO_Configuration (void);
00029
00034 int main(void)
00035 {
00036   SysVar vars;
00037   RCC_Configuration();
00038   GPIO_Configuration();
00039
00040   Blox_System_Create();
00041   Blox_System_Init();
00042   USB_Init();
00043   Blox_System_GetVars(&vars);
00044
00045   vars.id = BLOX_ID;
00046   vars.ACCEL_X = DEF_ACCEL_X;
00047   vars.ACCEL_Y = DEF_ACCEL_Y;
00048   vars.ACCEL_Z = DEF_ACCEL_Z;
00049   vars.TOUCH_1_X = DEF_TOUCH_X;
00050   vars.TOUCH_1_Y = DEF_TOUCH_Y;
00051   vars.TOUCH_2_X = DEF_TOUCH_X;
00052   vars.TOUCH_2_Y = DEF_TOUCH_Y;
00053   vars.TOUCH_3_X = DEF_TOUCH_X;
00054   vars.TOUCH_3_Y = DEF_TOUCH_Y;
00055   vars.TOUCH_4_X = DEF_TOUCH_X;
00056   vars.TOUCH_4_Y = DEF_TOUCH_Y;
00057   Blox_System_WriteVars(&vars);
00058
00059   if (FS_CreateFS() == FS_CREATE_FAIL) {
00060     //Trap
00061     while (1);
00062   }
00063   USB_SendPat("Configuring Xbee...\r\n");
00064   if (Blox_XBee_Config() == XBEE_INIT_FAIL)
00065     USB_SendPat("Failed\r\n");
00066   else {
00067     USB_SendPat("Success\r\n");
00068     Blox_XBee_Print();
00069   }
00070
```

```
00071   /* Infinite loop */
00072   while (1) {
00073     long i;
00074     GPIOA->ODR ^= (1<<8);
00075     for(i = 0; i < 1000000; i++);
00076   }
00077 }
00078
00083 void RCC_Configuration (void)
00084 {
00085   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
00086 }
00087
00092 void GPIO_Configuration (void)
00093 {
00094   GPIO_InitTypeDef GPIO_InitStructure;
00095   GPIO_StructInit(&GPIO_InitStructure);
00096   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
00097   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
00098   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
00099   GPIO_Init(GPIOA, &GPIO_InitStructure);
00100
00101   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
00102   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
00103   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
00104   GPIO_Init(GPIOC, &GPIO_InitStructure);
00105 }
```

Index