**Motorboard: Code documentation**

Table of Contents

[Adc.h 1](#_Toc478493411)

[void adc\_init(void) 1](#_Toc478493412)

[uint16\_t adc\_read(uint8\_t channel) 1](#_Toc478493413)

[uint16\_t getPreviousAdc4Value(void); 1](#_Toc478493414)

[uint16\_t getPreviousAdc8Value(void); 1](#_Toc478493415)

[Bluetooth.h 2](#_Toc478493416)

[void blue\_init(void) 2](#_Toc478493417)

[void blue\_putc(unsigned char data) 2](#_Toc478493418)

[void blue\_puts(char \* s) 2](#_Toc478493419)

[uint16\_t blue\_available() 2](#_Toc478493420)

[void blue\_flush\_buffer() 2](#_Toc478493421)

[unsigned char blue\_read\_data() 3](#_Toc478493422)

[void blue\_read\_buffer(void \* buff, uint16\_t len) 3](#_Toc478493423)

[void blue\_set\_new\_device() 3](#_Toc478493424)

[Global.h 3](#_Toc478493425)

[void global\_init(void) 4](#_Toc478493426)

[Inout.h 4](#_Toc478493427)

[void io\_init(void) 4](#_Toc478493428)

[void wheel\_sensor\_init(uint8\_t numberOfMagnetsPerRevolution, uint32\_t mMagicWheelConstant) 4](#_Toc478493429)

[void setHornHigh(void) 4](#_Toc478493430)

[void setHornLow(void) 5](#_Toc478493431)

[void setStarterHigh(void) 5](#_Toc478493432)

[void setStarterLow(void) 5](#_Toc478493433)

[uint8\_t digitalReadGSensor(void) 5](#_Toc478493434)

[uint16\_t getWheelSensorPeriod(void) 5](#_Toc478493435)

[uint32\_t getDistanceCompleted(void) 5](#_Toc478493436)

[void SetPWMDutyGear(uint16\_t duty); 6](#_Toc478493437)

[void SetPWMDutySpeed(uint16\_t duty); 6](#_Toc478493438)

[void pwm\_init(void); 6](#_Toc478493439)

[LEDs.h 6](#_Toc478493440)

[void LED\_init(void) 6](#_Toc478493441)

[void LEDnOn(void) 6](#_Toc478493442)

[void LEDnOff(void) 7](#_Toc478493443)

[void LED1Toggle(void) 7](#_Toc478493444)

[void LEDBlink(int LED, int times) 7](#_Toc478493445)

[Rs232.h 7](#_Toc478493446)

[void rs232\_init(void) 7](#_Toc478493447)

[void rs232\_putc(unsigned char data) 7](#_Toc478493448)

[void rs232\_wait\_transmit(void) 8](#_Toc478493449)

[uint16\_t rs232\_available(void) 8](#_Toc478493450)

[Rs232sync.h 8](#_Toc478493451)

[void rs232\_set\_car(uint8\_t carId) 8](#_Toc478493452)

[void rs232\_tx(void) 8](#_Toc478493453)

[Rs485.h 9](#_Toc478493454)

[void rs485\_init(void) 9](#_Toc478493455)

[void rs485\_set\_tx\_mode(void) 9](#_Toc478493456)

[void rs485\_set\_rx\_mode(void) 9](#_Toc478493457)

[void rs485\_putc(unsigned char data) 9](#_Toc478493458)

[void rs485\_wait\_transmit(void) 9](#_Toc478493459)

[Rs485sync.h 10](#_Toc478493460)

[void rs485\_sync(void) 10](#_Toc478493461)

[Tunes.h 10](#_Toc478493462)

[void tunes\_init(void) 10](#_Toc478493463)

[void sing(int s) 10](#_Toc478493464)

[void buzz(int targetPin, long frequency, long length) 11](#_Toc478493465)

**Disclaimer**

This document might be faulty in terms of both its analysis and its included references in regards to the code. To avoid an erroneous understanding of the aforementioned it is recommended to RTFC.

# Adc.h

Contains functions for initializing and reading from the Analog to Digital Converter (ADC).

## void adc\_init(void)

|  |  |
| --- | --- |
| Description: | Initializes the Analog to Digital Converter, with the following attributes:   * Prescaler set to 128 * Voltage reference 5V |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | N/A |

## uint16\_t adc\_read(uint8\_t channel)

|  |  |
| --- | --- |
| Description: | Reads a value via the ADC from a specified channel, which corresponds to a certain pin. If the channel read is the one designated to reading battery voltage the value is stored |
| Inputs: | 8 bits, specifying the channel/pin |
| Outputs: | 16 bits, value returned from the ADC |
| External libraries: | N/A |

## uint16\_t getPreviousAdc4Value(void);

|  |  |
| --- | --- |
| Description: | Returns the previously read value from the designated battery voltage which is channel 4 for the Dynamo |
| Inputs: | N/A |
| Outputs: | 16 bits, previously read battery voltage |
| External libraries: | N/A |

## uint16\_t getPreviousAdc8Value(void);

|  |  |
| --- | --- |
| Description: | Returns the previously read value from the designated battery voltage which is channel 8 for the Innovator |
| Inputs: | N/A |
| Outputs: | 16 bits, previously read battery voltage |
| External libraries: | N/A |

# Bluetooth.h

Contains functions for initializing, reading, and checking the status of the Bluetooth module (Adafruit Bluefruit LE UART Friend).

## void blue\_init(void)

|  |  |
| --- | --- |
| Description: | Initializes the Bluetooth module @ UART0 |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | Uart.h: uart0\_init(BAUD, F\_CPU) |

## void blue\_putc(unsigned char data)

|  |  |
| --- | --- |
| Description: | Transmits a character via UART0 |
| Inputs: | 8 bit, desired character to be send |
| Outputs: | N/A |
| External libraries: | Uart.h: uart0\_putc(data) |

## void blue\_puts(char \* s)

|  |  |
| --- | --- |
| Description: | Transmits a string/an array of characters via UART0 |
| Inputs: | 8 bit array, desired string/characters to be transmitted |
| Outputs: | N/A |
| External libraries: | Uart.h: uart0\_puts(s) |

## uint16\_t blue\_available()

|  |  |
| --- | --- |
| Description: | Determines the number of bytes waiting in the receive buffer related to the Bluetooth module |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | Uart.h: uart0\_available() |

## void blue\_flush\_buffer()

|  |  |
| --- | --- |
| Description: | Flushes/ignores the bytes waiting in the receive buffer related to the Bluetooth module |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | Uart.h: uart0\_flush() |

## unsigned char blue\_read\_data()

|  |  |
| --- | --- |
| Description: | Returns byte from the Bluetooth buffer if no errors are detected, i.e. stop bit error, over run error, and overflow error. If the buffer is empty or data is not available, it returns 0 |
| Inputs: | N/A |
| Outputs: | 8 bit, character from the buffer |
| External libraries: | Uart.h: uart0\_getc() |

## void blue\_read\_buffer(void \* buff, uint16\_t len)

|  |  |
| --- | --- |
| Description: | Loads data into an array used as a buffer via blue\_read\_data() |
| Inputs: | 8 bit array, buffer; 16 bit, length of buffer |
| Outputs: | N/A |
| External libraries: | Uart.h: uart0\_flush() |

## void blue\_set\_new\_device()

|  |  |
| --- | --- |
| Description: | Gives a Bluetooth module a new name, e.g. “DTU Dynamo”, the string must be set in the method itself |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | N/A |

# Global.h

Contains global variables used throughout the code extensively, i.e. declarations for buffer sizes used for communication and the global up-time (how long the board has been running since last reset). A quick recap of the buffers contained within:

1. Motor <-> RIO (<-> counts for two buffers, one each way)
2. Motor <-> Steering
3. Motor <-> Front lights
4. Motor <-> Back lights

The header file also contains simple “Helping functions”, such as:

* TESTBIT(var, bit) (var & (1<<bit))
* SETBIT(var, bit) (var |= (1<<bit))
* CLRBIT(var, bit) (var &= ~(1<<bit))
* FLIPBIT(var, bit) (var ^= (1<<bit))

## void global\_init(void)

|  |  |
| --- | --- |
| Description: | Performs any initializations needed in regards to the global variables. *This method is currently empty!* |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | N/A |

# Inout.h

Contains methods dealing with the inputs and outputs start motor, horn, gear sensor, and wheel sensor. *N.B. The magicWheelConstant is some Henning magic.*

## void io\_init(void)

|  |  |
| --- | --- |
| Description: | Initiates/declares the designated pins for starter motor and horn as output and gear sensor as an input |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | Global.h: SETBIT(var, bit), CLRBIT(var, bit) |

## void wheel\_sensor\_init(uint8\_t numberOfMagnetsPerRevolution, uint32\_t mMagicWheelConstant)

|  |  |
| --- | --- |
| Description: | Initiates the wheel sensor, toggles an LED, and sets the magicWheelConstant |
| Inputs: | 8 bit, number of magnets pr. revolution regarding the wheel sensor; 32 bit for the magic wheel constant |
| Outputs: | N/A |
| External libraries: | Global.h: CLRBIT(var, bit) |

## void setHornHigh(void)

|  |  |
| --- | --- |
| Description: | Turns the horn on |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | Global.h: SETBIT(var, bit) |

## void setHornLow(void)

|  |  |
| --- | --- |
| Description: | Turns the horn off |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | Global.h: CLRBIT(var, bit) |

## void setStarterHigh(void)

|  |  |
| --- | --- |
| Description: | Turns the starter motor on |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | Global.h: SETBIT(var, bit) |

## void setStarterLow(void)

|  |  |
| --- | --- |
| Description: | Turns the starter motor off |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | Global.h: CLRBIT(var, bit) |

## uint8\_t digitalReadGSensor(void)

|  |  |
| --- | --- |
| Description: | Reads the gear sensor value |
| Inputs: | N/A |
| Outputs: | 8 bit, digital value of the gear sensor |
| External libraries: | Global.h: TESTBIT(var, bit) |

## uint16\_t getWheelSensorPeriod(void)

|  |  |
| --- | --- |
| Description: | Returns the period of time passed since the last wheel sensor read |
| Inputs: | N/A |
| Outputs: | 16 bits, time in ms |
| External libraries: | Global.h: variable time |

## uint32\_t getDistanceCompleted(void)

|  |  |
| --- | --- |
| Description: | Returns the total distance completed |
| Inputs: | N/A |
| Outputs: | 32 bits, distance in meters |
| External libraries: | N/A |

## void SetPWMDutyGear(uint16\_t duty);

|  |  |
| --- | --- |
| Description: | Sets the duty cycle of the PWM designated to the gear sensor |
| Inputs: | 16 bit, desired duty cycle of the PWM |
| Outputs: | N/A |
| External libraries: | N/A |

## void SetPWMDutySpeed(uint16\_t duty);

|  |  |
| --- | --- |
| Description: | Sets the duty cycle of the PWM designated to the speed servo. *Currently not in use!* |
| Inputs: | 16 bit, desired duty cycle of the PWM |
| Outputs: | N/A |
| External libraries: | N/A |

## void pwm\_init(void);

|  |  |
| --- | --- |
| Description: | Initializes the PWM Output for the gear sensor and speed servo. *N.B. The speed servo’s initialization is currently out commented!* |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | N/A |

# LEDs.h

Contains methods for initializing, turning on/off, and toggling the various LEDs.

## void LED\_init(void)

|  |  |
| --- | --- |
| Description: | Initiates/declares the designated pins for the various LEDs used |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | Global.h: SETBIT(var, bit) |

## void LEDnOn(void)

|  |  |
| --- | --- |
| Description: | Turns on the n’th LED |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | N/A |

## void LEDnOff(void)

|  |  |
| --- | --- |
| Description: | Turns off the n’th LED |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | N/A |

## void LED1Toggle(void)

|  |  |
| --- | --- |
| Description: | Toggles the n’th LED |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | N/A |

## void LEDBlink(int LED, int times)

|  |  |
| --- | --- |
| Description: | Blinks an LED as many times as the input “times” dependent on the “LED” input. Provides an easy way to test all the LED’s |
| Inputs: | 32 bits, LED number; 32 bits, desired times that the LED/LED’s should blink |
| Outputs: | N/A |
| External libraries: | N/A |

# Rs232.h

Contains functions for initializing, reading, and checking the status of the Reconfigurable Input/Output (RIO) module. This library contains inline methods which are used extensively throughout the library. *Be aware there is inconsistency between the c file and h file (rs232\_wait… in c file, rs485\_wait… in h file)!*

## void rs232\_init(void)

|  |  |
| --- | --- |
| Description: | Initiates the communication with the RIO module |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | N/A |

## void rs232\_putc(unsigned char data)

|  |  |
| --- | --- |
| Description: | Transmits a character via UART2 to the RIO |
| Inputs: | 8 bits, desired data to be transmitted |
| Outputs: | N/A |
| External libraries: | N/A |

## void rs232\_wait\_transmit(void)

|  |  |
| --- | --- |
| Description: | Used to stall the main loop until transmission from the RIO is complete or has timed out |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | N/A |

## uint16\_t rs232\_available(void)

|  |  |
| --- | --- |
| Description: | Determines the number of bytes waiting in the receive buffer related to the Bluetooth module |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | Uart.h: uart2\_available() |

# Rs232sync.h

The “big brother” of rs232.h, contains interrupt handler for receiving data from the RIO, as well as a method to transmit larger chunks of data to the RIO. The interrupt handler is not included in this document.

## void rs232\_set\_car(uint8\_t carId)

|  |  |
| --- | --- |
| Description: | Sets the carID which determines the way the interrupt handler handles receiving data from the RIO. The carID only needs to be set if we are dealing with the innovator, which is out of use |
| Inputs: | 8 bits, specified carID |
| Outputs: | N/A |
| External libraries: | N/A |

## void rs232\_tx(void)

|  |  |
| --- | --- |
| Description: | Transmits data to the RIO after syncing the two nodes. The data and length thereof is determined outside of the method |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | Rs232.h: rs232\_putc(unsigned char data) |

# Rs485.h

Contains methods for initializing, transmitting, and stall the UART communication via the rs485 protocol.

## void rs485\_init(void)

|  |  |
| --- | --- |
| Description: | Initializes the module to be ready for UART communication. The module starts off in receive mode |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | Uart.h: uart1\_init(uint16\_t baudrate) |

## void rs485\_set\_tx\_mode(void)

|  |  |
| --- | --- |
| Description: | Sets the module in transmission mode |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | N/A |

## void rs485\_set\_rx\_mode(void)

|  |  |
| --- | --- |
| Description: | Sets the module in receive mode |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | N/A |

## void rs485\_putc(unsigned char data)

|  |  |
| --- | --- |
| Description: | Transmits a character via UART1 |
| Inputs: | 8 bits, desired data to be transmitted |
| Outputs: | N/A |
| External libraries: | Uart.h: uart1\_putc(uint8\_t data) |

## void rs485\_wait\_transmit(void)

|  |  |
| --- | --- |
| Description: | Used to stall the main loop until the transmission has been completed |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | N/A |

# Rs485sync.h

The big brother of rs485.h, contains the structure that synchronises all the communication from and to the motorboard. It also contains the interrupt handler for receiving data automatically. The interrupt handler is not included in this document.

## void rs485\_sync(void)

|  |  |
| --- | --- |
| Description: | Handles all the synchronization needed for transmitting and receiving data from and to the motorboard. In broad strokes, it does the following sequentially:   1. Reads data from the steering and light (front) board 2. Reacts to said data read, e.g. sets bits needed to blink left, starts wipers, etc. 3. Synchronises data with the RIO 4. Sends data to the steering, front light, and back light board |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | Too extensive to include |

# Tunes.h

Contains the methods and the definitions used to make the buzzer play songs.

## void tunes\_init(void)

|  |  |
| --- | --- |
| Description: | Initiates output pin used to control the buzzer |
| Inputs: | N/A |
| Outputs: | N/A |
| External libraries: | N/A |

## void sing(int s)

|  |  |
| --- | --- |
| Description: | Makes the buzzer play a tune |
| Inputs: | 16 bits, value of the desired tune to be played |
| Outputs: | N/A |
| External libraries: | Delay.h: delay\_ms(uint16\_t count) |

## void buzz(int targetPin, long frequency, long length)

|  |  |
| --- | --- |
| Description: | Makes the buzzer play a tone for a certain duration. *Cation: The targetPin argument is not being used as of right now. Instead the output pin is hardcoded in the method!* |
| Inputs: | 16 bits, specifies the output pin; 32 bits, specifies the frequency to be buzzed; 32 bits, specifies the duration of the note |
| Outputs: | N/A |
| External libraries: | Delay.h: delay\_us(uint16\_t count) |