AVA LLC Documentation

1.0

Generated by Doxygen 1.13.2

Chapter 1

AVA LLC Documentation

Documentation for the AVA LLC embedded platform.

Documentation for the AVA LLC embedded platform.

1.1 Introduction

The **AVA LLC** firmware runs on an STM32 MCU and manages several subsystems:

- MAINBOARD Subsystem
- ULTRASONIC Subsystem
- STEERING Subsystem
- BRAKE Subsystem

Each subsystem is implemented as a Doxygen module (`@defgroup`) and linked from these pages.

1.2 Author

Developed by **Shiddieqy**
Version 1.0.0
Date: 2025-11-10

Chapter 2

MAINBOARD Subsystem

The central control unit.

The MAINBOARD subsystem is responsible for:

- System startup and RTOS initialization
- CAN and UART communication
- Safety and telemetry tasks

2.1 Related Module

For implementation details, see: MAINBOARD

Chapter 3

ULTRASONIC Subsystem

Handles distance measurement and obstacle detection.

The ULTRASONIC subsystem monitors front and rear sensors and provides safety feedback to other subsystems.

3.1 Related Module

For implementation details, see: ULTRASONIC

Chapter 4

STEERING Subsystem

Controls the steering actuator.

The STEERING subsystem interprets angle commands from the MAINBOARD and drives the steering servo.

4.1 Related Module

For implementation details, see: STEERING

Chapter 5

BRAKE Subsystem

Manages braking actuation and safety overrides.

The BRAKE subsystem applies brake force commands from the MAINBOARD and responds to safety triggers from ULTRASONIC sensors.

5.1 Related Module

For implementation details, see: BRAKE

Chapter 6

Topic Index

6.1 Topics

Here is a list of all topics with brief descriptions:

Chapter 7

Topic Documentation

7.1 MAINBOARD

Initializes and manages the core system.

Macros

- #define THROTTLE_FILTER 0.1

  Smoothing factor for throttle input (0..1).
- #define ULTRASONIC_THRESHOLD 0

  Threshold (in raw units) under which ultrasonic sensors trigger proximity actions.
- #define ULTRASONIC_BOT_THRESHOLD 70

  Lower bound threshold for ultrasonic sensors mapping to brake action.
- #define UART_BUF_SIZE 32
- #define UART1_BUF_SIZE 5

Functions

- int main (void)

  The application entry point.
- void SystemClock_Config (void)

  System Clock Configuration.
- void PeriphCommonClock_Config (void)

  Peripherals Common Clock Configuration.
- int _write (int file, char ∗ptr, int len)

  Reimplements write syscall used by printf to route through UART8.
- uint8_t ceksum (uint8_t ∗msg, uint16_t array_size)

  Compute checksum of a message buffer.
- void query_request_data (uint8_t ∗msg)

  Handle query request messages received over UART.
- void parse_command (uint8_t ∗msg)

  Parse incoming UART commands and dispatch appropriate actions.
- void HAL_UART_RxCpltCallback (UART_HandleTypeDef ∗huart)

  UART receive complete callback used by HAL interrupt mode.
- void send_uart (void ∗argument)

Function implementing the defaultTask thread (send_uart).

- void ezkontrol (void *argument)

  Function implementing the myTask02 thread (ezkontrol).
- void canbus_sampling (void *argument)

  Function implementing the myTask03 thread (canbus_sampling).
- void Steering (void *argument)

  Function implementing the myTask04 thread (Steering).
- void Brake (void *argument)

  Function implementing the myTask05 thread (Brake).
- void safety_system (void *argument)

  Function implementing the myTask06 thread (safety_system).
- void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef *htim)

  Period elapsed callback in non blocking mode.
- void Error_Handler (void)

  This function is executed in case of error occurrence.

Variables

- FDCAN_HandleTypeDef hfdcan2

  FDCAN handle for FDCAN2 peripheral.
- UART_HandleTypeDef huart8

  UART handle for UART8.
- UART_HandleTypeDef huart1

  UART handle for USART1.
- UART_HandleTypeDef huart2

  UART handle for USART2.
- osThreadId_t defaultTaskHandle
- const osThreadAttr_t defaultTask_attributes
- osThreadId_t myTask02Handle
- const osThreadAttr_t myTask02_attributes
- osThreadId_t myTask03Handle
- const osThreadAttr_t myTask03_attributes
- osThreadId_t myTask04Handle
- const osThreadAttr_t myTask04_attributes
- osThreadId_t myTask05Handle
- const osThreadAttr_t myTask05_attributes
- osThreadId_t myTask06Handle
- const osThreadAttr_t myTask06_attributes
- uint8_t uart_data [34]

  Telemetry packet to be streamed over UART.
- uint8_t uart1_data [100]

  Local buffer for incoming UART1 data copy.
- uint8_t uart_byte8

  Single-byte temporary storage for UART interrupts (UART8).
- uint8_t uart_byte1

  Single-byte temporary storage for UART interrupts (USART1/2).
- uint8_t uart_rx_buffer [UART_BUF_SIZE]

  Circular RX buffer for UART8 asynchronous reception.
- uint16_t uart_index = 0

  Index into uart_rx_buffer for next write position.
- char uart1_rx_buffer [UART1_BUF_SIZE]

Temporary buffer for incoming ASCII remote packets (USART1/2).

- uint16_t uart1_index =0

  Index into uart1_rx_buffer for next write position.
- uint8_t is_broadcast = 0

  Flag indicating whether telemetry broadcast streaming is enabled.
- FDCAN_RxHeaderTypeDef rxHeader

  FDCAN Rx header used for received messages (global scratch).
- uint16_t remote_cmd [3]

  Parsed remote control channels (3 channels).
- uint8_t remote_bin

  Parsed remote binary flags nibble.
- uint16_t uart_cmd [3]

  Parsed UART command channels stored as integers.
- struct ControlCMD control_cmd

  Global instance of ControlCMD used by tasks to read/update control signals.
- uint32_t rx_mailbox = 0

  Mailbox index for CAN Tx mailbox (unused/placeholder).
- volatile uint32_t uart_last_remote_time = 0

  Last tick time when a remote UART byte was received.
- volatile uint32_t ez_last_can_time = 0

  Last tick time when ezkontrol CAN message was seen.
- uint8_t is_remote_ready =0

  Flag indicating remote control readiness.
- uint8_t rxData_ezkontrol [8]
- uint8_t rxData_brake [8]
- uint8_t rxData_steering [8]
- uint8_t rxData_ultrasonic_front [8]
- uint8_t rxData_ultrasonic_rear [8]
- uint8_t rxData_analog_throttle [8]
- uint8_t flag_ezkontrol = 0
- uint8_t flag_brake = 0
- uint8_t flag_steering = 0
- uint8_t flag_ultrasonic_front = 0
- uint8_t flag_ultrasonic_rear = 0
- uint8_t flag_emergency = 0
- uint8_t flag_key = 0
- uint8_t flag_remote = 0
- uint8_t flag_analog_throttle = 0
- uint32_t total_rotation = 0

  Cumulative wheel rotation counter (units from integration of rpm).
- float rotation = 0

  Floating point rotation accumulator.
- float prev_rpm = 0

  Previous RPM value for trapezoidal integration.
- uint32_t rpm_sampling = 0

  Timestamp used when sampling RPM from ezkontrol module.
- uint32_t last_ezkontrol_rpm = 0

  Last instant when an ezkontrol RPM was observed.
- uint16_t closest_radar

  Closest radar distance (cached).
- uint8_t ready_to_recieve_radar = 0

  Flag indicating radar ready-to-receive sequence (unused in current logic).

7.1.1 Detailed Description

Initializes and manages the core system.

See also

mainboard_page

The MAINBOARD subsystem is responsible for:

- RTOS initialization
- CAN and UART communication
- Safety monitoring
- Task orchestration

Attention

7.1.2 Macro Definition Documentation

7.1.2.1 THROTTLE_FILTER

```
#define THROTTLE_FILTER 0.1
```

Smoothing factor for throttle input (0..1).

Used to low-pass filter sudden changes in throttle commands.

7.1.2.2 UART1_BUF_SIZE

```
#define UART1_BUF_SIZE 5
```

7.1.2.3 UART_BUF_SIZE

```
#define UART_BUF_SIZE 32
```

7.1.2.4 ULTRASONIC_BOT_THRESHOLD

```
#define ULTRASONIC_BOT_THRESHOLD 70
```

Lower bound threshold for ultrasonic sensors mapping to brake action.

Values below this will be clamped to maximum braking.

7.1.2.5 ULTRASONIC_THRESHOLD

```
#define ULTRASONIC_THRESHOLD 0
```

Threshold (in raw units) under which ultrasonic sensors trigger proximity actions.

Compared against UART telemetry fields for front and rear ultrasonic measurements.

7.1.3 Function Documentation

7.1.3.1 _write()

```
int _write (
            int file,
            char * ptr,
            int len)
```

Reimplements write syscall used by printf to route through UART8.

Parameters

	file	File descriptor (ignored).
	ptr	Pointer to buffer to write.
	len	Number of bytes to write.

Returns

Number of bytes written.

This function forwards the buffer to HAL_UART_Transmit using huart8. It intentionally uses blocking HAL with HAL_MAX_DELAY for simplicity.

Parameters

	file	File descriptor (ignored).
	ptr	Pointer to buffer to write.
	len	Number of bytes to write.

Returns

Number of bytes written.

7.1.3.2 Brake()

```
void Brake (
            void * argument)
```

Function implementing the myTask05 thread (Brake).

Function implementing the myTask05 thread.

Parameters

	argument	Not used

Return values

	None

7.1.3.3 canbus_sampling()

```
void canbus_sampling (
            void * argument)
```

Function implementing the myTask03 thread (canbus_sampling).

Function implementing the myTask03 thread.

Parameters

	argument	Not used

Return values

	None

7.1.3.4 ceksum()

```
uint8_t ceksum (
            uint8_t * msg,
            uint16_t array_size)
```

Compute checksum of a message buffer.

Parameters

	msg	Pointer to message buffer.
	array_size	Size of the buffer (including checksum byte).

Returns

8-bit sum of bytes msg[0]..msg[array_size-2].

Sums all bytes in the buffer except last and returns 8-bit sum.

Parameters

	msg	Pointer to message buffer.
	array_size	Size of the buffer (including checksum byte).

Returns

8-bit sum of bytes msg[0]..msg[array_size-2].

7.1.3.5 Error_Handler()

```
void Error_Handler (
             void )
```

This function is executed in case of error occurrence.

Return values

	None

7.1.3.6 ezkontrol()

```
void ezkontrol (
             void * argument)
```

Function implementing the myTask02 thread (ezkontrol).

Function implementing the myTask02 thread.

Parameters

	argument	Not used

Return values

	None

7.1.3.7 HAL_TIM_PeriodElapsedCallback()

```
void HAL_TIM_PeriodElapsedCallback (
             TIM_HandleTypeDef * htim)
```

Period elapsed callback in non blocking mode.

Parameters

	htim	: TIM handle

Return values

	None

Note

This function is called when TIM4 interrupt took place, inside HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to increment a global variable "uwTick" used as application time base.

Parameters

htim	: TIM handle

Return values

None

7.1.3.8 HAL_UART_RxCpltCallback()

```
void HAL_UART_RxCpltCallback (
            UART_HandleTypeDef * huart)
```

UART receive complete callback used by HAL interrupt mode.

Parameters

huart	Pointer to UART handle that triggered the callback.

Return values

None

Handles per-byte reception from UART8 (telemetry control) and USART1/2 (remote control). Re-arms the receive interrupt for continuous asynchronous operation.

Parameters

huart	Pointer to UART handle that triggered the callback.

7.1.3.9 main()

```
int main (
            void )
```

The application entry point.
Function Definitions (documented prototypes) - appear here in the same order

7.1.4 as the implementations later in the file.

Return values

int
int

7.1.4.1 parse_command()

```
void parse_command (
            uint8_t * msg)
```

Parse incoming UART commands and dispatch appropriate actions.

Parameters

msg	Pointer to incoming command buffer.

Supports functions codes:

- 0x01: Start streaming
- 0x00: Stop streaming (sends ack)
- 0x02: Query request
- 0x03..0x06: Various command/response flows (control and query)

Parameters

msg	Pointer to incoming command buffer.

7.1.4.2 PeriphCommonClock_Config()

```
void PeriphCommonClock_Config (
            void )
```

Peripherals Common Clock Configuration.

Return values

None

Initializes the peripherals clock

7.1.4.3 query_request_data()

```
void query_request_data (
            uint8_t * msg)
```

Handle query request messages received over UART.

Parameters

msg	Pointer to incoming 5-byte or variable-length query message.

Verifies checksum and responds by sending a subset of the telemetry packet back over UART8 according to the requested indices.

Parameters

msg	Pointer to incoming 5-byte or variable-length query message.

7.1.4.4 safety_system()

```
void safety_system (
            void * argument)
```

Function implementing the myTask06 thread (safety_system).
Function implementing the myTask06 thread.

Parameters

argument	Not used

Return values

	None

7.1.4.5 send_uart()

```
void send_uart (
            void * argument)
```

Function implementing the defaultTask thread (send_uart).
Function implementing the defaultTask thread.

Parameters

	argument	Not used

Return values

	None

7.1.4.6 Steering()

```
void Steering (
            void * argument)
```

Function implementing the myTask04 thread (Steering).
Function implementing the myTask04 thread.

Parameters

	argument	Not used

Return values

	None

7.1.4.7 SystemClock_Config()

```
void SystemClock_Config (
            void )
```

System Clock Configuration.

Return values

	None

Supply configuration update enable
Configure the main internal regulator output voltage
Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef structure.
Initializes the CPU, AHB and APB buses clocks
Enables the Clock Security System

7.1.5 Variable Documentation

7.1.5.1 closest_radar

```
uint16_t closest_radar
```

Closest radar distance (cached).

7.1.5.2 control_cmd

```
struct ControlCMD control_cmd
```

Global instance of ControlCMD used by tasks to read/update control signals.

7.1.5.3 defaultTask_attributes

```
const osThreadAttr_t defaultTask_attributes
```

Initial value:

```
= {
  .name = "defaultTask",
  .stack_size = 256 * 4,
  .priority = (osPriority_t) osPriorityNormal,
}
```

7.1.5.4 defaultTaskHandle

```
osThreadId_t defaultTaskHandle
```

7.1.5.5 ez_last_can_time

```
volatile uint32_t ez_last_can_time = 0
```

Last tick time when ezkontrol CAN message was seen.

7.1.5.6 flag_analog_throttle

```
uint8_t flag_analog_throttle = 0
```

7.1.5.7 flag_brake

```
uint8_t flag_brake = 0
```

7.1.5.8 flag_emergency

```
uint8_t flag_emergency = 0
```

7.1.5.9 flag_ezkontrol

```
uint8_t flag_ezkontrol = 0
```

7.1.5.10 flag_key

```
uint8_t flag_key = 0
```

7.1.5.11 flag_remote

```
uint8_t flag_remote = 0
```

7.1.5.12 flag_steering

```
uint8_t flag_steering = 0
```

7.1.5.13 flag_ultrasonic_front

```
uint8_t flag_ultrasonic_front = 0
```

7.1.5.14 flag_ultrasonic_rear

```
uint8_t flag_ultrasonic_rear = 0
```

7.1.5.15 hfdcan2

```
FDCAN_HandleTypeDef hfdcan2
```

FDCAN handle for FDCAN2 peripheral.
Used for CAN communication with motor controllers, sensors, etc.

7.1.5.16 huart1

```
UART_HandleTypeDef huart1
```

UART handle for USART1.
Used to receive remote control commands / telemetry.

7.1.5.17 huart2

```
UART_HandleTypeDef huart2
```

UART handle for USART2.
Used to receive remote control commands / telemetry.

7.1.5.18 huart8

```
UART_HandleTypeDef huart8
```

UART handle for UART8.
Used for telemetry and host communication (printf output via _write).

7.1.5.19 is_broadcast

```
uint8_t is_broadcast = 0
```

Flag indicating whether telemetry broadcast streaming is enabled.

7.1.5.20 is_remote_ready

```
uint8_t is_remote_ready =0
```

Flag indicating remote control readiness.

7.1.5.21 last_ezkontrol_rpm

```
uint32_t last_ezkontrol_rpm = 0
```

Last instant when an ezkontrol RPM was observed.

7.1.5.22 myTask02_attributes

```
const osThreadAttr_t myTask02_attributes
```

Initial value:

```
= {
  .name = "myTask02",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityNormal,
}
```

7.1.5.23 myTask02Handle

```
osThreadId_t myTask02Handle
```

7.1.5.24 myTask03_attributes

```
const osThreadAttr_t myTask03_attributes
```

Initial value:

```
= {
  .name = "myTask03",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityNormal,
}
```

7.1.5.25 myTask03Handle

```
osThreadId_t myTask03Handle
```

7.1.5.26 myTask04_attributes

```
const osThreadAttr_t myTask04_attributes
```

Initial value:

```
= {
  .name = "myTask04",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityNormal,
}
```

7.1.5.27 myTask04Handle

```
osThreadId_t myTask04Handle
```

7.1.5.28 myTask05_attributes

```
const osThreadAttr_t myTask05_attributes
```

Initial value:

```
= {
  .name = "myTask05",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityNormal,
}
```

7.1.5.29 myTask05Handle

```
osThreadId_t myTask05Handle
```

7.1.5.30 myTask06_attributes

```
const osThreadAttr_t myTask06_attributes
```

Initial value:

```
= {
  .name = "myTask06",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityNormal,
}
```

7.1.5.31 myTask06Handle

```
osThreadId_t myTask06Handle
```

7.1.5.32 prev_rpm

```
float prev_rpm = 0
```

Previous RPM value for trapezoidal integration.

7.1.5.33 ready_to_recieve_radar

```
uint8_t ready_to_recieve_radar = 0
```

Flag indicating radar ready-to-receive sequence (unused in current logic).

7.1.5.34 remote_bin

```
uint8_t remote_bin
```

Parsed remote binary flags nibble.

7.1.5.35 remote_cmd

```
uint16_t remote_cmd[3]
```

Parsed remote control channels (3 channels).

7.1.5.36 rotation

```
float rotation = 0
```
Floating point rotation accumulator.

7.1.5.37 rpm_sampling

```
uint32_t rpm_sampling = 0
```
Timestamp used when sampling RPM from ezkontrol module.

7.1.5.38 rx_mailbox

```
uint32_t rx_mailbox = 0
```
Mailbox index for CAN Tx mailbox (unused/placeholder).

7.1.5.39 rxData_analog_throttle

```
uint8_t rxData_analog_throttle[8]
```

7.1.5.40 rxData_brake

```
uint8_t rxData_brake[8]
```

7.1.5.41 rxData_ezkontrol

```
uint8_t rxData_ezkontrol[8]
```

7.1.5.42 rxData_steering

```
uint8_t rxData_steering[8]
```

7.1.5.43 rxData_ultrasonic_front

```
uint8_t rxData_ultrasonic_front[8]
```

7.1.5.44 rxData_ultrasonic_rear

```
uint8_t rxData_ultrasonic_rear[8]
```

7.1.5.45 rxHeader

```
FDCAN_RxHeaderTypeDef rxHeader
```
FDCAN Rx header used for received messages (global scratch).

7.1.5.46 total_rotation

```
uint32_t total_rotation = 0
```
Cumulative wheel rotation counter (units from integration of rpm).

7.1.5.47 uart1_data

```
uint8_t uart1_data[100]
```
Local buffer for incoming UART1 data copy.

7.1.5.48 uart1_index

```
uint16_t uart1_index =0
```
Index into uart1_rx_buffer for next write position.

7.1.5.49 uart1_rx_buffer

```
char uart1_rx_buffer[UART1_BUF_SIZE]
```

Temporary buffer for incoming ASCII remote packets (USART1/2).

7.1.5.50 uart_byte1

```
uint8_t uart_byte1
```

Single-byte temporary storage for UART interrupts (USART1/2).

7.1.5.51 uart_byte8

```
uint8_t uart_byte8
```

Single-byte temporary storage for UART interrupts (UART8).

7.1.5.52 uart_cmd

```
uint16_t uart_cmd[3]
```

Parsed UART command channels stored as integers.

7.1.5.53 uart_data

```
uint8_t uart_data[34]
```

Initial value:

```
= {
    0xA5,
    0x11,
    0x00,
    0x00,
    0x00, 0x00,
    0x00,
    0x00,
    0x00, 0x00,
    0x00, 0x00,
    0x00, 0x00,
    0x00, 0x00,
    0x00, 0x00,
    0x00, 0x00,
    0x00,
    0xFF,
    0xFF,
    0x00,
    0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00
}
```

Telemetry packet to be streamed over UART.
The array is a fixed 34-byte packet with the following mapping:

- [0] Header (0xA5)
- [1] Function code (streaming data)
- [2] Battery percentage
- [3] Charging status
- [4-5] RPM (int16, divided by 10 in original format)
- [6] Brake percentage
- [7] Steer angle
- [8-19] IMU fields (x,y,z,yaw,pitch,roll)
- [20] Radar closest object
- [21] Ultrasonic front
- [22] Ultrasonic rear

- [23] Front bumper switch
- [24] Rear bumper switch
- [25-28] GPS latitude (float)
- [29-32] GPS longitude (float)
- [33] Checksum

7.1.5.54 uart_index

```
uint16_t uart_index = 0
```

Index into uart_rx_buffer for next write position.

7.1.5.55 uart_last_remote_time

```
volatile uint32_t uart_last_remote_time = 0
```

Last tick time when a remote UART byte was received.

7.1.5.56 uart_rx_buffer

```
uint8_t uart_rx_buffer[UART_BUF_SIZE]
```

Circular RX buffer for UART8 asynchronous reception.