Reversing Radar Project

1.0

Generated by Doxygen 1.15.0

1 Reversing Radar Distance Monitoring System

1.1 Introduction

This project implements a **Reversing Radar system** using two STM32F103-based nodes: a **Transmitter Node** and a **Receiver Node**. The system measures distances to obstacles behind a vehicle using ultrasonic sensors and provides feedback via LEDs, a buzzer, and a 2-digit 7-segment display. CAN bus is used for communication between the two nodes.

The software is developed with **STM32Cube HAL** and **FreeRTOS** for task scheduling.

1.2 Hardware Overview

The following components are used in this project:

- **MCU:** STM32F103C8T6 (ARM Cortex-M3)
- **CAN Transceivers:** MCP2551
- **Ultrasonic Sensors:** HC-SR04 or equivalent
- **Buzzer:** Generic active buzzer
- **LEDs:** Red, Green, Blue standard 5mm LEDs
- **7-segment Display:** 2281AS 2-digit common anode 7-segment display

1.3 Software Structure

- **Transmitter Node**
  - Measures distances from ultrasonic sensors.
  - Sends measured distances via CAN bus.
  - Generates FreeRTOS tasks for sensor reading and CAN transmission.
- **Receiver Node**
  - Receives CAN messages containing distance data.
  - Controls LEDs, buzzer, and 7-segment display based on distance thresholds.
  - Implements FreeRTOS tasks for display updates, buzzer, and UART debug output.

1.4 Usage

1. Compile the project in Keil uVision for each node.
2. Flash the firmware to the respective STM32F103 boards.
3. Connect CAN bus between the two nodes.
4. Power the system. The receiver node will display distance and activate LEDs/buzzer according to obstacle proximity.

1.5 Notes

- The project uses symbolic names for GPIO pins and peripherals (e.g., `Buzzer_Pin`, `DIG1_Pin`).
- All hardware references in this documentation use actual component names in the Hardware Overview section.

2 Topic Documentation

2.1 Transmitter Node

CAN Transmitter node for the Reversing Radar system.

Files

- file app_tasks.h

  FreeRTOS task declarations for the transmitter node.
- file main.h

  Header file for the main application of the STM32 transmitter node.
- file usensor.h

  Ultrasonic sensor driver interface for STM32.
- file app_tasks.c

  FreeRTOS task implementations for the transmitter node.
- file usensor.c

  Ultrasonic sensor measurement driver using STM32 timers.

2.1.1 Detailed Description

CAN Transmitter node for the Reversing Radar system.

This node acquires distance information and transmits it over CAN to the receiver node.

This file contains the main function for the transmitter node, which:

- Initializes STM32 peripherals (GPIO, TIM, CAN, UART)
- Handles ultrasonic sensor measurements
- Starts FreeRTOS scheduler with tasks for sensor reading and CAN transmission

Note

HAL_TIM_IC_CaptureCallback is forwarded to the ultrasonic sensor module.

2.2 Receiver Node

CAN Receiver node for the Reversing Radar system.

Files

- file app_tasks.h
  FreeRTOS task declarations for the CAN receiver application.
- file leds.h
  LED control functions for the distance indicator.
- file main.h
  Main application header for the CAN receiver node.
- file sevenseg.h
  Update the 7-segment display with a digit.
- file app_tasks.c
  FreeRTOS task implementations for the CAN receiver node.
- file leds.c
  LED control implementation for distance indication.
- file sevenseg.c
  7-segment display driver

2.2.1 Detailed Description

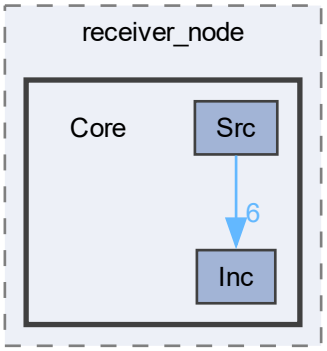
CAN Receiver node for the Reversing Radar system.

This node receives distance data via CAN and displays it using LEDs and a 7-segment display.

3 Directory Documentation

3.1 Core Directory Reference

Directory dependency graph for Core:

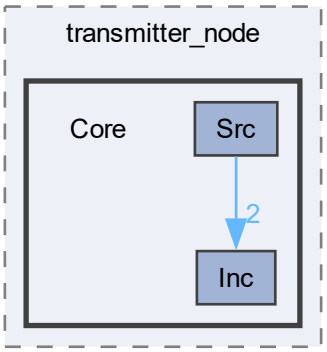


Directories

- directory Inc
- directory Src

3.2 Core Directory Reference

Directory dependency graph for Core:

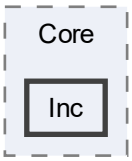


Directories

- directory Inc
- directory Src

3.3 Inc Directory Reference

Directory dependency graph for Inc:



Files

- file app_tasks.h

  FreeRTOS task declarations for the CAN receiver application.
- file leds.h

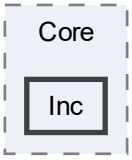
  LED control functions for the distance indicator.
- file main.h

  Main application header for the CAN receiver node.
- file sevenseg.h

  Update the 7-segment display with a digit.

3.4 Inc Directory Reference

Directory dependency graph for Inc:



Files

- file app_tasks.h

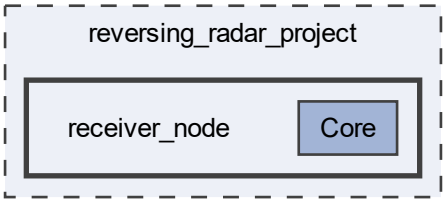
  FreeRTOS task declarations for the transmitter node.
- file main.h

  Header file for the main application of the STM32 transmitter node.
- file mainpage.h
- file usensor.h

  Ultrasonic sensor driver interface for STM32.

3.5 receiver_node Directory Reference

Directory dependency graph for receiver_node:



Directories

- directory Core

3.6 reversing_radar_project Directory Reference

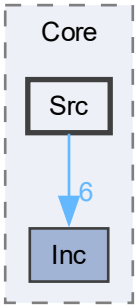
Directory dependency graph for reversing_radar_project:

Directories

- directory receiver_node
- directory transmitter_node

3.7 Src Directory Reference

Directory dependency graph for Src:



Files

- file app_tasks.c

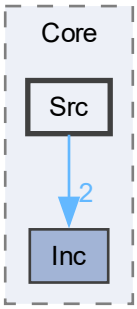
  FreeRTOS task implementations for the CAN receiver node.
- file freertos.c
- file leds.c

  LED control implementation for distance indication.
- file main.c
- file sevenseg.c

  7-segment display driver

3.8 Src Directory Reference

Directory dependency graph for Src:



Files

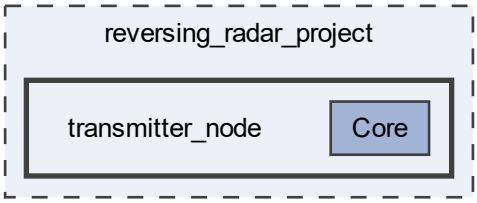
- file app_tasks.c

  FreeRTOS task implementations for the transmitter node.
- file freertos.c
- file main.c
- file usensor.c

  Ultrasonic sensor measurement driver using STM32 timers.

3.9 transmitter_node Directory Reference

Directory dependency graph for transmitter_node:



Directories

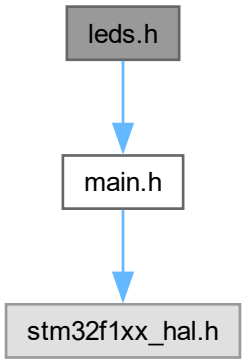
- directory Core

4 File Documentation

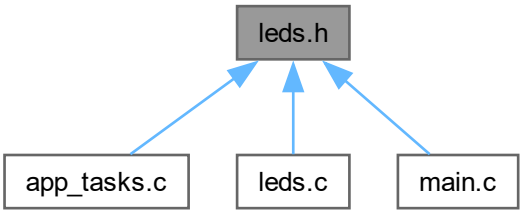
4.1 leds.h File Reference

LED control functions for the distance indicator.

Include dependency graph for leds.h:



This graph shows which files directly or indirectly include this file:



Functions

- void leds_1 (void)

  Turn on LED pattern 1 (closest distance indication).
- void leds_2 (void)

  Turn on LED pattern 2.
- void leds_3 (void)

  Turn on LED pattern 3.
- void leds_4 (void)

  Turn on LED pattern 4.
- void leds_5 (void)

  Turn on LED pattern 5.
- void leds_6 (void)

  Turn on LED pattern 6.
- void leds_7 (void)

  Turn on LED pattern 7 (all LEDs on for very close distance).

4.1.1 Detailed Description

LED control functions for the distance indicator.

This module provides functions to turn on a specific number of LEDs according to the measured distance from the CAN receiver node.

4.1.2 Function Documentation

leds_1()

```
void leds_1 (
            void )
```

Turn on LED pattern 1 (closest distance indication).

Turn on LED pattern 1 (closest distance indication).

References Blue1_Pin, Blue2_Pin, Green1_Pin, Green2_Pin, Green3_Pin, Red1_Pin, and Red2_Pin.

Referenced by StartDefaultTask().

Here is the caller graph for this function:

leds_2()

```
void leds_2 (
            void )
```

Turn on LED pattern 2.

Turn on LED pattern 2.

References Blue1_Pin, Blue2_Pin, Green1_Pin, Green2_Pin, Green3_Pin, Red1_Pin, and Red2_Pin.

Referenced by StartDefaultTask().

Here is the caller graph for this function:

leds_3()

```
void leds_3 (
             void )
```

Turn on LED pattern 3.

Turn on LED pattern 3.

References Blue1_Pin, Blue2_Pin, Green1_Pin, Green2_Pin, Green3_Pin, Red1_Pin, and Red2_Pin.

Referenced by StartDefaultTask().

Here is the caller graph for this function:

leds_4()

```
void leds_4 (
             void )
```

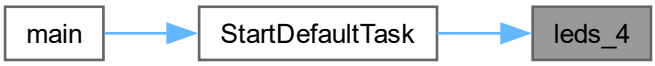
Turn on LED pattern 4.

Turn on LED pattern 4.

References Blue1_Pin, Blue2_Pin, Green1_Pin, Green2_Pin, Green3_Pin, Red1_Pin, and Red2_Pin.

Referenced by StartDefaultTask().

Here is the caller graph for this function:



leds_5()

```
void leds_5 (
             void )
```

Turn on LED pattern 5.

Turn on LED pattern 5.

References Blue1_Pin, Blue2_Pin, Green1_Pin, Green2_Pin, Green3_Pin, Red1_Pin, and Red2_Pin.

Referenced by StartDefaultTask().

Here is the caller graph for this function:

leds_6()

```
void leds_6 (
             void )
```

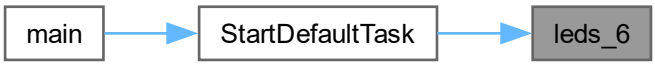
Turn on LED pattern 6.

Turn on LED pattern 6.

References Blue1_Pin, Blue2_Pin, Green1_Pin, Green2_Pin, Green3_Pin, Red1_Pin, and Red2_Pin.

Referenced by StartDefaultTask().

Here is the caller graph for this function:



leds_7()

```
void leds_7 (
             void )
```

Turn on LED pattern 7 (all LEDs on for very close distance).

Turn on LED pattern 7 (all LEDs on for very close distance).

References Blue1_Pin, Blue2_Pin, Green1_Pin, Green2_Pin, Green3_Pin, Red1_Pin, and Red2_Pin.

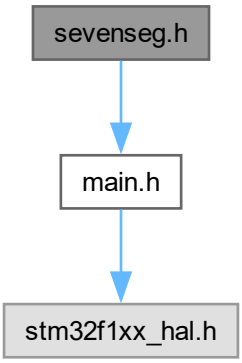
Referenced by StartDefaultTask().

Here is the caller graph for this function:

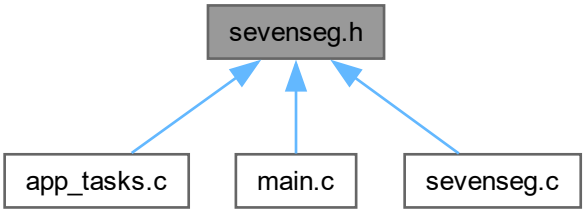
4.2 sevenseg.h File Reference

Update the 7-segment display with a digit.

Include dependency graph for sevenseg.h:



This graph shows which files directly or indirectly include this file:



Functions

- void SevenSegment_Update (uint8_t number)

  Update the 7-segment display segments for a given number.

4.2.1 Detailed Description

Update the 7-segment display with a digit.

Sets the GPIO outputs corresponding to the given digit value. The function does not handle digit multiplexing; it only updates the segment lines (AG, DP).

Parameters

number	Digit to display (09 or predefined segment pattern).

4.2.2 Function Documentation

SevenSegment_Update()

```
void SevenSegment_Update (
            uint8_t number)
```

Update the 7-segment display segments for a given number.

Parameters

number	The number to display encoded as bits for segments A-G. Each bit corresponds to a segment (bit 0 = A, bit 6 = G).

Return values

None	

References A_Pin, B_Pin, C_Pin, D_Pin, E_Pin, F_Pin, and G_Pin.

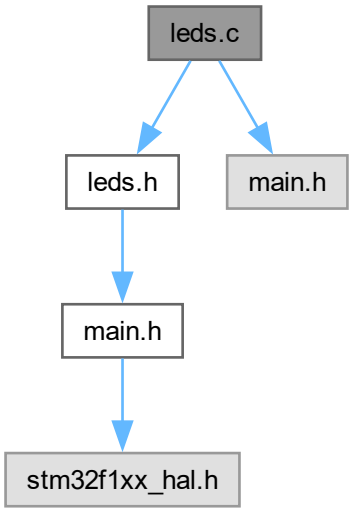
Referenced by lcdTask_init().

Here is the caller graph for this function:

4.3 leds.c File Reference

LED control implementation for distance indication.

Include dependency graph for leds.c:



Functions

- void leds_7 ()

  Turn on all LEDs (pattern 7) closest distance.
- void leds_6 ()

Turn on LEDs pattern 6.

- void leds_5 ()

  Turn on LEDs pattern 5.

- void leds_4 ()

  Turn on LEDs pattern 4.

- void leds_3 ()

  Turn on LEDs pattern 3.

- void leds_2 ()

  Turn on LEDs pattern 2.

- void leds_1 ()

  Turn on LEDs pattern 1 only Green1 for farthest distance.

4.3.1 Detailed Description

LED control implementation for distance indication.

Each function corresponds to a specific LED pattern, representing different distance thresholds from the CAN receiver.

4.3.2 Function Documentation

leds_1()

```
void leds_1 (
             void )
```

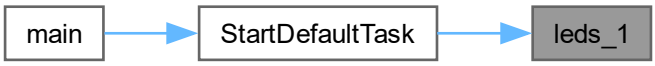
Turn on LEDs pattern 1 only Green1 for farthest distance.

Turn on LED pattern 1 (closest distance indication).

References Blue1_Pin, Blue2_Pin, Green1_Pin, Green2_Pin, Green3_Pin, Red1_Pin, and Red2_Pin.

Referenced by StartDefaultTask().

Here is the caller graph for this function:



leds_2()

```
void leds_2 (
             void )
```

Turn on LEDs pattern 2.

Turn on LED pattern 2.

References Blue1_Pin, Blue2_Pin, Green1_Pin, Green2_Pin, Green3_Pin, Red1_Pin, and Red2_Pin.

Referenced by StartDefaultTask().

Here is the caller graph for this function:

leds_3()

```
void leds_3 (
             void )
```

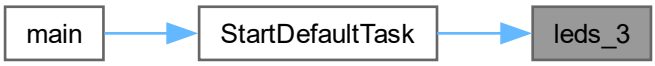
Turn on LEDs pattern 3.

Turn on LED pattern 3.

References Blue1_Pin, Blue2_Pin, Green1_Pin, Green2_Pin, Green3_Pin, Red1_Pin, and Red2_Pin.

Referenced by StartDefaultTask().

Here is the caller graph for this function:



leds_4()

```
void leds_4 (
             void )
```

Turn on LEDs pattern 4.

Turn on LED pattern 4.

References Blue1_Pin, Blue2_Pin, Green1_Pin, Green2_Pin, Green3_Pin, Red1_Pin, and Red2_Pin.

Referenced by StartDefaultTask().

Here is the caller graph for this function:

leds_5()

```
void leds_5 (
             void )
```

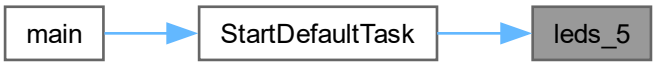
Turn on LEDs pattern 5.

Turn on LED pattern 5.

References Blue1_Pin, Blue2_Pin, Green1_Pin, Green2_Pin, Green3_Pin, Red1_Pin, and Red2_Pin.

Referenced by StartDefaultTask().

Here is the caller graph for this function:



leds_6()

```
void leds_6 (
             void )
```

Turn on LEDs pattern 6.

Turn on LED pattern 6.

References Blue1_Pin, Blue2_Pin, Green1_Pin, Green2_Pin, Green3_Pin, Red1_Pin, and Red2_Pin.

Referenced by StartDefaultTask().

Here is the caller graph for this function:

leds_7()

```
void leds_7 (
             void )
```

Turn on all LEDs (pattern 7) closest distance.

Turn on LED pattern 7 (all LEDs on for very close distance).

References Blue1_Pin, Blue2_Pin, Green1_Pin, Green2_Pin, Green3_Pin, Red1_Pin, and Red2_Pin.

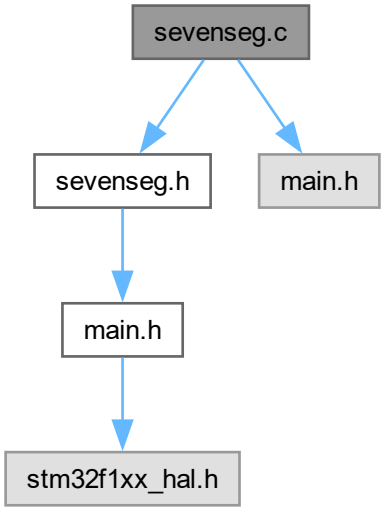
Referenced by StartDefaultTask().

Here is the caller graph for this function:

4.4 sevenseg.c File Reference

7-segment display driver

Include dependency graph for sevenseg.c:



Functions

- void SevenSegment_Update (uint8_t number)

  Update the 7-segment display segments for a given number.

Variables

- uint8_t segmentNumber [10]

4.4.1 Detailed Description

7-segment display driver

This module implements the control of a multiplexed 7-segment display using GPIO pins on STM32. It provides a function to update the segments according to a numeric value (0-9).

4.4.2 Function Documentation

SevenSegment_Update()

```
void SevenSegment_Update (
            uint8_t number)
```

Update the 7-segment display segments for a given number.

Parameters

number	The number to display encoded as bits for segments A-G. Each bit corresponds to a segment (bit 0 = A, bit 6 = G).

Return values

None	

References A_Pin, B_Pin, C_Pin, D_Pin, E_Pin, F_Pin, and G_Pin.

Referenced by lcdTask_init().

Here is the caller graph for this function:

4.4.3 Variable Documentation

segmentNumber

```
uint8_t segmentNumber[10]
```

Initial value:

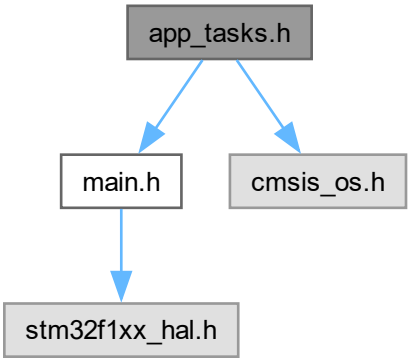
```
= {
    0x3f, 0x06, 0x5b, 0x4f, 0x66,
    0x6d, 0x7d, 0x07, 0x7f, 0x67
}
```

Segment encodings for digits 0-9 (GFEDCBA)

4.5 receiver_node/Core/Inc/app_tasks.h File Reference

FreeRTOS task declarations for the CAN receiver application.

Include dependency graph for receiver_node/Core/Inc/app_tasks.h:



Functions

- void StartDefaultTask (void ∗argument)

  Default FreeRTOS task.
- void serialTask_init (void ∗argument)

  UART serial communication task.
- void buzzerTask_init (void ∗argument)

  Buzzer control task.
- void lcdTask_init (void ∗argument)

  LCD display task.

Variables

- osThreadId_t defaultTaskHandle
- osThreadId_t serialTaskHandle
- osThreadId_t buzzerTaskHandle
- osThreadId_t lcdTaskHandle

4.5.1 Detailed Description

FreeRTOS task declarations for the CAN receiver application.

This file contains the task handles and task function prototypes used by the receiver node. Each task is created in main.c and executed under the FreeRTOS scheduler.

4.5.2 Function Documentation

buzzerTask_init()

```
void buzzerTask_init (
             void * argument)
```

Buzzer control task.

Controls the buzzer behavior based on application events.

Parameters

argument	Pointer passed to the task (not used).

Controls the buzzer behavior based on the measured distance. The buzzer toggles faster as the distance decreases.

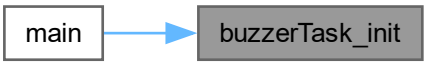
Parameters

argument	Pointer passed to the task (not used).

References Buzzer_Pin, Distance, and time.

Referenced by main().

Here is the caller graph for this function:



lcdTask_init()

```
void lcdTask_init (
            void * argument)
```

LCD display task.

Updates the LCD or display elements based on received CAN data.

Parameters

argument	Pointer passed to the task (not used).

LCD display task.

Displays the shortest received distance on a multiplexed 2-digit 7-segment display. If the distance exceeds the maximum threshold, a warning pattern is shown.

Parameters

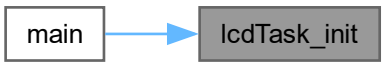
argument	Pointer passed to the task (not used).

References DIG1_HIGH, DIG1_LOW, DIG2_HIGH, DIG2_LOW, digit1, digit2, Distance, DP_OFF, DP_ON, RxData, and SevenSegment_Update().

Referenced by main().

Here is the call graph for this function:

Here is the caller graph for this function:



serialTask_init()

```
void serialTask_init (
            void * argument)
```

UART serial communication task.

Handles serial data transmission and/or reception over UART.

Parameters

argument	Pointer passed to the task (not used).

UART serial communication task.

Periodically transmits the measured distance value via UART for debugging or monitoring purposes.

Parameters

argument	Pointer passed to the task (not used).

References Buffer, Distance, and huart2.

Referenced by main().

Here is the caller graph for this function:

StartDefaultTask()

```
void StartDefaultTask (
            void * argument)
```

Default FreeRTOS task.

This task typically runs background or system-level operations.

Parameters

argument	Pointer passed to the task (not used).

Default FreeRTOS task.

This task selects the shortest distance received via CAN and updates LED patterns accordingly. It also updates the buzzer timing variable.

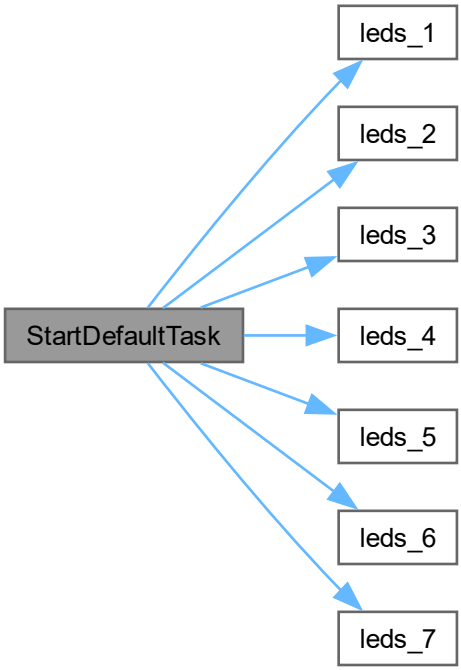
Parameters

argument	Pointer passed to the task (not used).

References Distance, leds_1(), leds_2(), leds_3(), leds_4(), leds_5(), leds_6(), leds_7(), RxData, and time.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

4.5.3 Variable Documentation

buzzerTaskHandle

```
osThreadId_t buzzerTaskHandle  [extern]
```

Handle for the buzzer control task

Referenced by main().

defaultTaskHandle

```
osThreadId_t defaultTaskHandle  [extern]
```

Handle for the default system task

Referenced by main().

lcdTaskHandle

```
osThreadId_t lcdTaskHandle  [extern]
```

Handle for the LCD display task

Referenced by main().

serialTaskHandle

```
osThreadId_t serialTaskHandle  [extern]
```

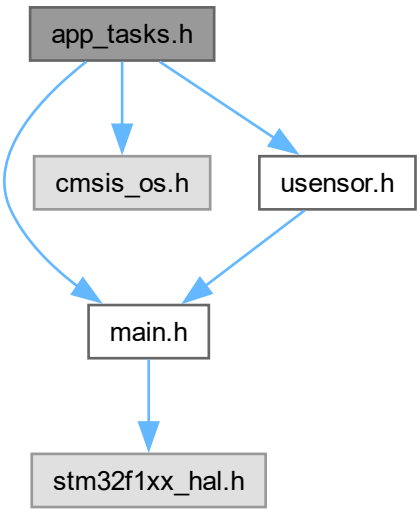
Handle for the UART serial communication task

Referenced by main().

4.6 transmitter_node/Core/Inc/app_tasks.h File Reference

FreeRTOS task declarations for the transmitter node.

Include dependency graph for transmitter_node/Core/Inc/app_tasks.h:



Functions

- void us1Task_init (void *argument)

  Task for first ultrasonic sensor reading.
- void us2Task_init (void *argument)

  Task for second ultrasonic sensor reading.
- void TxTask_init (void *argument)

  Task for transmitting measured distances via CAN.

Variables

- CAN_TxHeaderTypeDef TxHeader
- CAN_FilterTypeDef canfilterconfig
- uint32_t TxMailbox
- uint8_t TxData [8]

4.6.1 Detailed Description

FreeRTOS task declarations for the transmitter node.

This header exposes:

- Prototypes for ultrasonic sensor tasks
- The CAN transmission task
- External CAN-related variables shared with the main program

All tasks declared here are created and managed in main.c.

4.6.2 Function Documentation

TxTask_init()

```
void TxTask_init (
            void * argument)
```

Task for transmitting measured distances via CAN.

Parameters

argument	Pointer passed to the task (not used)

Return values

None	

Task for transmitting measured distances via CAN.
Task: TxTask_init

Parameters

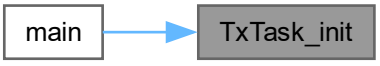
argument	Not used

4.6.2.1 autotoc_md2

Return values

None	

< Unused parameter
< Fill CAN transmit buffer
< Transmit CAN message
< CAN transmission failed, signal with LED (optional)
< Wait 60 ms before next transmission
References Distance_1, Distance_2, hcan, TxData, TxHeader, and TxMailbox.
Referenced by main().
Here is the caller graph for this function:



us1Task_init()

```
void us1Task_init (
            void * argument)
```

Task for first ultrasonic sensor reading.
< FreeRTOS CMSIS-RTOS API < Ultrasonic sensor module

Parameters

argument	Pointer passed to the task (not used)

Return values

None	

Task for first ultrasonic sensor reading.
Task: us1Task_init

Parameters

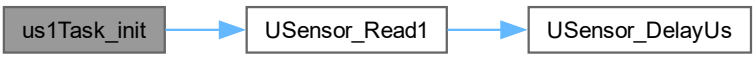
argument	Not used

4.6.2.2 autotoc_md0

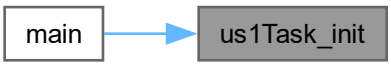
Return values

None	

< Unused parameter
< Trigger first sensor measurement
< Wait minimum 60 ms between measurements
References USensor_Read1().
Referenced by main().
Here is the call graph for this function:



Here is the caller graph for this function:



us2Task_init()

```
void us2Task_init (
            void * argument)
```

Task for second ultrasonic sensor reading.

Parameters

argument	Pointer passed to the task (not used)

Return values

None	

Task for second ultrasonic sensor reading.
Task: us2Task_init

Parameters

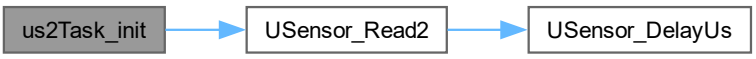
argument	Not used

4.6.2.3 autotoc_md1

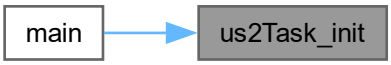
Return values

None	

< Unused parameter
< Trigger second sensor measurement
< Wait minimum 60 ms between measurements
References USensor_Read2().
Referenced by main().
Here is the call graph for this function:



Here is the caller graph for this function:



4.6.3 Variable Documentation

canfilterconfig

```
CAN_FilterTypeDef canfilterconfig  [extern]
```

CAN filter configuration
CAN filter config
Referenced by MX_CAN_Init(), and MX_CAN_Init().

TxData

```
uint8_t TxData[8]  [extern]
```

CAN transmit data buffer (8 bytes)
CAN transmit buffer
Referenced by TxTask_init().

TxHeader

```
CAN_TxHeaderTypeDef TxHeader  [extern]
```

CAN transmit header

Referenced by MX_CAN_Init(), MX_CAN_Init(), and TxTask_init().

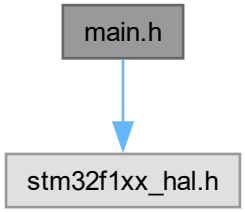
TxMailbox

```
uint32_t TxMailbox  [extern]
```

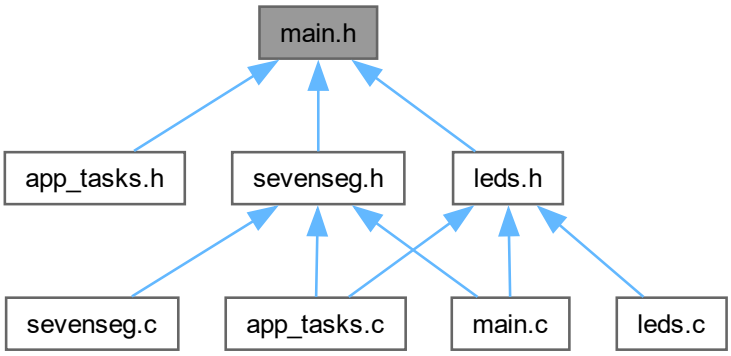
CAN transmit mailbox index
CAN mailbox index
Referenced by TxTask_init().

4.7 receiver_node/Core/Inc/main.h File Reference

Main application header for the CAN receiver node.
Include dependency graph for receiver_node/Core/Inc/main.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define Green1_Pin GPIO_PIN_6
- #define Green1_GPIO_Port GPIOA
- #define Green2_Pin GPIO_PIN_7
- #define Green2_GPIO_Port GPIOA
- #define Green3_Pin GPIO_PIN_14
- #define Green3_GPIO_Port GPIOB

- #define Blue1_Pin GPIO_PIN_15
- #define Blue1_GPIO_Port GPIOB
- #define Blue2_Pin GPIO_PIN_8
- #define Blue2_GPIO_Port GPIOA
- #define Red1_Pin GPIO_PIN_9
- #define Red1_GPIO_Port GPIOA
- #define Red2_Pin GPIO_PIN_10
- #define Red2_GPIO_Port GPIOA
- #define Buzzer_Pin GPIO_PIN_15
- #define Buzzer_GPIO_Port GPIOA
- #define A_Pin GPIO_PIN_13
- #define A_GPIO_Port GPIOB
- #define B_Pin GPIO_PIN_3
- #define B_GPIO_Port GPIOB
- #define C_Pin GPIO_PIN_1
- #define C_GPIO_Port GPIOB
- #define D_Pin GPIO_PIN_0
- #define D_GPIO_Port GPIOB
- #define E_Pin GPIO_PIN_9
- #define E_GPIO_Port GPIOB
- #define F_Pin GPIO_PIN_8
- #define F_GPIO_Port GPIOB
- #define G_Pin GPIO_PIN_10
- #define G_GPIO_Port GPIOB
- #define DP_Pin GPIO_PIN_11
- #define DP_GPIO_Port GPIOB
- #define DIG1_Pin GPIO_PIN_5
- #define DIG1_GPIO_Port GPIOB
- #define DIG2_Pin GPIO_PIN_12
- #define DIG2_GPIO_Port GPIOB

Functions

- void Error_Handler (void)

  Handles unrecoverable application errors.
- void StartDefaultTask (void ∗argument)

  Default FreeRTOS task.
- void serialTask_init (void ∗argument)

  Serial communication task.
- void buzzerTask_init (void ∗argument)

  Buzzer control task.
- void lcdTask_init (void ∗argument)

  LCD / 7-segment display handling task.

Variables

- CAN_HandleTypeDef hcan
- UART_HandleTypeDef huart2

4.7.1 Detailed Description

Main application header for the CAN receiver node.
This header file contains GPIO pin definitions, external peripheral handles, and FreeRTOS task prototypes used by the CAN receiver application.

4.7.2 Macro Definition Documentation

A_GPIO_Port

```
#define A_GPIO_Port GPIOB
```

A_Pin

```
#define A_Pin GPIO_PIN_13
```
Segment A
Referenced by MX_GPIO_Init(), and SevenSegment_Update().

B_GPIO_Port

```
#define B_GPIO_Port GPIOB
```

B_Pin

```
#define B_Pin GPIO_PIN_3
```
Segment B
Referenced by MX_GPIO_Init(), and SevenSegment_Update().

Blue1_GPIO_Port

```
#define Blue1_GPIO_Port GPIOB
```

Blue1_Pin

```
#define Blue1_Pin GPIO_PIN_15
```
Blue LED 1
Referenced by leds_1(), leds_2(), leds_3(), leds_4(), leds_5(), leds_6(), leds_7(), and MX_GPIO_Init().

Blue2_GPIO_Port

```
#define Blue2_GPIO_Port GPIOA
```

Blue2_Pin

```
#define Blue2_Pin GPIO_PIN_8
```
Blue LED 2
Referenced by leds_1(), leds_2(), leds_3(), leds_4(), leds_5(), leds_6(), leds_7(), and MX_GPIO_Init().

Buzzer_GPIO_Port

```
#define Buzzer_GPIO_Port GPIOA
```

Buzzer_Pin

```
#define Buzzer_Pin GPIO_PIN_15
```
Buzzer output pin
Referenced by buzzerTask_init(), and MX_GPIO_Init().

C_GPIO_Port

```
#define C_GPIO_Port GPIOB
```

C_Pin

```
#define C_Pin GPIO_PIN_1
```
Segment C
Referenced by MX_GPIO_Init(), and SevenSegment_Update().

D_GPIO_Port

```
#define D_GPIO_Port GPIOB
```

D_Pin

```
#define D_Pin GPIO_PIN_0
```
Segment D
Referenced by MX_GPIO_Init(), and SevenSegment_Update().

DIG1_GPIO_Port

```
#define DIG1_GPIO_Port GPIOB
```

DIG1_Pin

```
#define DIG1_Pin GPIO_PIN_5
```
Digit 1 enable
Referenced by MX_GPIO_Init().

DIG2_GPIO_Port

```
#define DIG2_GPIO_Port GPIOB
```

DIG2_Pin

```
#define DIG2_Pin GPIO_PIN_12
```
Digit 2 enable
Referenced by MX_GPIO_Init().

DP_GPIO_Port

```
#define DP_GPIO_Port GPIOB
```

DP_Pin

```
#define DP_Pin GPIO_PIN_11
```
Decimal point
Referenced by MX_GPIO_Init().

E_GPIO_Port

```
#define E_GPIO_Port GPIOB
```

E_Pin

```
#define E_Pin GPIO_PIN_9
```
Segment E
Referenced by MX_GPIO_Init(), and SevenSegment_Update().

F_GPIO_Port

```
#define F_GPIO_Port GPIOB
```

F_Pin

```
#define F_Pin GPIO_PIN_8
```
Segment F
Referenced by MX_GPIO_Init(), and SevenSegment_Update().

G_GPIO_Port

```
#define G_GPIO_Port GPIOB
```

G_Pin

```
#define G_Pin GPIO_PIN_10
```
Segment G
Referenced by MX_GPIO_Init(), and SevenSegment_Update().

Green1_GPIO_Port

```
#define Green1_GPIO_Port GPIOA
```

Green1_Pin

```
#define Green1_Pin GPIO_PIN_6
```
Green LED 1
Referenced by leds_1(), leds_2(), leds_3(), leds_4(), leds_5(), leds_6(), leds_7(), and MX_GPIO_Init().

Green2_GPIO_Port

```
#define Green2_GPIO_Port GPIOA
```

Green2_Pin

```
#define Green2_Pin GPIO_PIN_7
```
Green LED 2
Referenced by leds_1(), leds_2(), leds_3(), leds_4(), leds_5(), leds_6(), leds_7(), and MX_GPIO_Init().

Green3_GPIO_Port

```
#define Green3_GPIO_Port GPIOB
```

Green3_Pin

```
#define Green3_Pin GPIO_PIN_14
```
Green LED 3
Referenced by leds_1(), leds_2(), leds_3(), leds_4(), leds_5(), leds_6(), leds_7(), and MX_GPIO_Init().

Red1_GPIO_Port

```
#define Red1_GPIO_Port GPIOA
```

Red1_Pin

```
#define Red1_Pin GPIO_PIN_9
```
Red LED 1
Referenced by leds_1(), leds_2(), leds_3(), leds_4(), leds_5(), leds_6(), leds_7(), and MX_GPIO_Init().

Red2_GPIO_Port

```
#define Red2_GPIO_Port GPIOA
```

Red2_Pin

```
#define Red2_Pin GPIO_PIN_10
```
Red LED 2
Referenced by leds_1(), leds_2(), leds_3(), leds_4(), leds_5(), leds_6(), leds_7(), and MX_GPIO_Init().

4.7.3 Function Documentation

buzzerTask_init()

```
void buzzerTask_init (
            void * argument)
```

Buzzer control task.

Parameters

argument	Pointer to task parameters (unused)

Controls the buzzer behavior based on the measured distance. The buzzer toggles faster as the distance decreases.

Parameters

argument	Pointer passed to the task (not used).

Error_Handler()

```
void Error_Handler (
            void )
```

Handles unrecoverable application errors.

Return values

None	

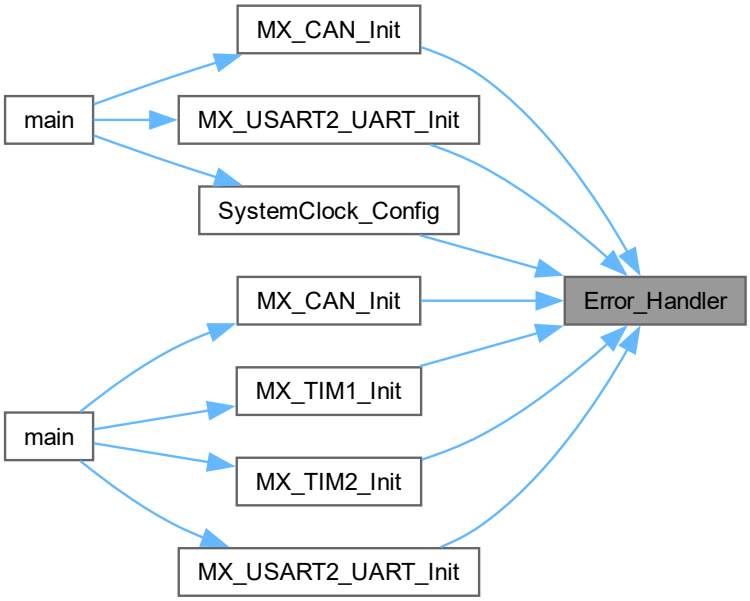
Handles unrecoverable application errors.
Handles HAL errors.
Handles unrecoverable application errors.
Handles HAL errors.

Return values

None	

Referenced by MX_CAN_Init(), MX_CAN_Init(), MX_TIM1_Init(), MX_TIM2_Init(), MX_USART2_UART_Init(), MX_USART2_UART_Init(), and SystemClock_Config().

Here is the caller graph for this function:



lcdTask_init()

```
void lcdTask_init (
            void * argument)
```

LCD / 7-segment display handling task.

Parameters

argument	Pointer to task parameters (unused)

LCD / 7-segment display handling task.
LCD display task.
Displays the shortest received distance on a multiplexed 2-digit 7-segment display. If the distance exceeds the maximum threshold, a warning pattern is shown.

Parameters

argument	Pointer passed to the task (not used).

serialTask_init()

```
void serialTask_init (
            void * argument)
```

Serial communication task.

Parameters

argument	Pointer to task parameters (unused)

Serial communication task.
UART serial communication task.
Periodically transmits the measured distance value via UART for debugging or monitoring purposes.

Parameters

argument	Pointer passed to the task (not used).

StartDefaultTask()

```
void StartDefaultTask (
            void * argument)
```

Default FreeRTOS task.

Parameters

argument	Pointer to task parameters (unused)

Default FreeRTOS task.
This task selects the shortest distance received via CAN and updates LED patterns accordingly. It also updates the buzzer timing variable.

Parameters

argument	Pointer passed to the task (not used).

4.7.4 Variable Documentation

hcan

```
CAN_HandleTypeDef hcan  [extern]
```

CAN peripheral handle
CAN handle

huart2

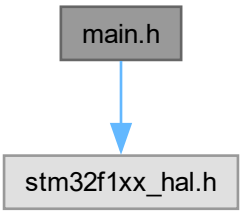
```
UART_HandleTypeDef huart2  [extern]
```

UART2 peripheral handle (debug/serial output)
UART2 handle

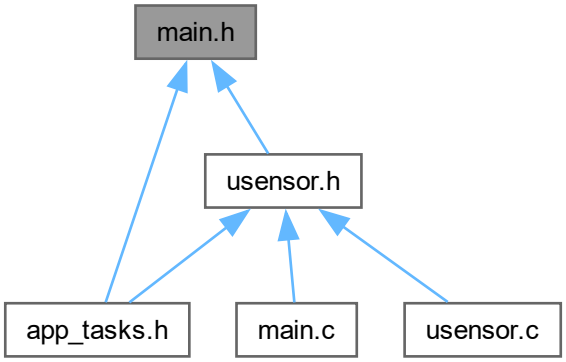
4.8 transmitter_node/Core/Inc/main.h File Reference

Header file for the main application of the STM32 transmitter node.

Include dependency graph for transmitter_node/Core/Inc/main.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define TRIG1_Pin GPIO_PIN_9

  Trigger pins for Ultrasonic sensors.
- #define TRIG1_GPIO_Port GPIOA
- #define TRIG2_Pin GPIO_PIN_12
- #define TRIG2_GPIO_Port GPIOA

Functions

- void SystemClock_Config (void)

  Configures the system clock.
- void Error_Handler (void)

  Handles HAL errors.
- static void MX_GPIO_Init (void)
- static void MX_TIM1_Init (void)
- static void MX_TIM2_Init (void)
- static void MX_CAN_Init (void)
- static void MX_USART2_UART_Init (void)

Variables

- TIM_HandleTypeDef htim1

  External peripheral handles used in main.c.
- TIM_HandleTypeDef htim2
- CAN_HandleTypeDef hcan
- UART_HandleTypeDef huart2

4.8.1 Detailed Description

Header file for the main application of the STM32 transmitter node.
This file provides:

- Core system includes
- Global peripheral handle declarations
- GPIO pin definitions
- Prototypes for system configuration and initialization functions

It is included by all modules requiring access to global peripherals such as timers, CAN, and UART, as well as shared GPIO definitions.

Note

Peripheral initialization functions declared here are implemented in main.c and should not be modified unless hardware changes.

4.8.2 Macro Definition Documentation

TRIG1_GPIO_Port

```
#define TRIG1_GPIO_Port GPIOA
```

TRIG1_Pin

```
#define TRIG1_Pin GPIO_PIN_9
```

Trigger pins for Ultrasonic sensors.
Referenced by MX_GPIO_Init().

TRIG2_GPIO_Port

```
#define TRIG2_GPIO_Port GPIOA
```

TRIG2_Pin

```
#define TRIG2_Pin GPIO_PIN_12
```

Referenced by MX_GPIO_Init().

4.8.3 Function Documentation

Error_Handler()

```
void Error_Handler (
            void )
```

Handles HAL errors.

Note

Typically halts the system when an unrecoverable error occurs

Handles HAL errors.
Handles HAL errors.

Return values

None	

MX_CAN_Init()

```
void MX_CAN_Init (
            void )  [static]
```

MX_GPIO_Init()

```
void MX_GPIO_Init (
            void )  [static]
```

MX_TIM1_Init()

```
void MX_TIM1_Init (
            void )  [static]
```

MX_TIM2_Init()

```
void MX_TIM2_Init (
            void )  [static]
```

MX_USART2_UART_Init()

```
void MX_USART2_UART_Init (
            void )  [static]
```

SystemClock_Config()

```
void SystemClock_Config (
            void )
```

Configures the system clock.
System Clock Configuration.

Note

Generated by CubeMX or manually implemented in main.c

Return values

None	

Configures the system clock.
Configures the system clock.

Return values

None	

Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef structure.
Initializes the CPU, AHB and APB buses clocks
Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef structure.
Initializes the CPU, AHB and APB buses clocks
References Error_Handler().
Referenced by main().

Here is the call graph for this function:

Here is the caller graph for this function:

4.8.4 Variable Documentation

hcan

`CAN_HandleTypeDef hcan [extern]`
CAN handle
Referenced by HAL_CAN_RxFifo0MsgPendingCallback(), main(), MX_CAN_Init(), MX_CAN_Init(), and TxTask_init().

htim1

`TIM_HandleTypeDef htim1 [extern]`
External peripheral handles used in main.c.
Timer 1 handle
Referenced by main(), MX_TIM1_Init(), USensor_DelayUs(), USensor_Read1(), and USensor_TIM_IC_Callback().

htim2

`TIM_HandleTypeDef htim2 [extern]`
Timer 2 handle
Referenced by main(), MX_TIM2_Init(), USensor_Read2(), and USensor_TIM_IC_Callback().

huart2

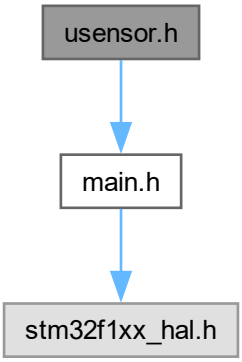
`UART_HandleTypeDef huart2 [extern]`
UART2 handle
Referenced by MX_USART2_UART_Init(), MX_USART2_UART_Init(), and serialTask_init().

4.9 mainpage.h File Reference

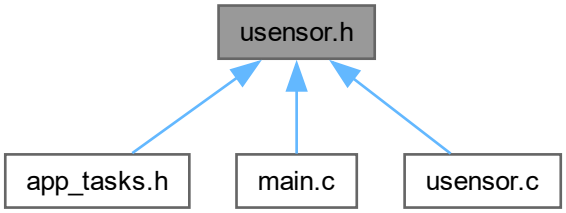
4.10 usensor.h File Reference

Ultrasonic sensor driver interface for STM32.

Include dependency graph for usensor.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define USENSOR1_TRIG_PIN TRIG1_Pin

  Ultrasonic sensor 1 trigger pin.
- #define USENSOR2_TRIG_PIN TRIG2_Pin

  Ultrasonic sensor 2 trigger pin.
- #define USENSOR_GPIO_PORT GPIOA

  GPIO port for ultrasonic sensors.

Functions

- void USensor_Init (void)

  Initialize timers and GPIO for ultrasonic sensors.
- void USensor_Read1 (void)

  Trigger the first ultrasonic sensor.
- void USensor_Read2 (void)

  Trigger the second ultrasonic sensor.
- void USensor_DelayUs (uint16_t us)

Delay for a specified number of microseconds.

- void USensor_TIM_IC_Callback (TIM_HandleTypeDef *htim)

  Callback function for Timer Input Capture events.

Variables

- uint8_t Distance_1

  Distance measured by sensor 1 in cm.

- uint8_t Distance_2

  Distance measured by sensor 2 in cm.

4.10.1 Detailed Description

Ultrasonic sensor driver interface for STM32.
This header provides:

- Definitions of ultrasonic trigger pins
- Access to measured distance variables
- Function prototypes for sensor triggering and IC handling

The implementation relies on hardware timers configured in main.c.

4.10.2 Macro Definition Documentation

USENSOR1_TRIG_PIN

```
#define USENSOR1_TRIG_PIN TRIG1_Pin
```

Ultrasonic sensor 1 trigger pin.
Referenced by USensor_Read1().

USENSOR2_TRIG_PIN

```
#define USENSOR2_TRIG_PIN TRIG2_Pin
```

Ultrasonic sensor 2 trigger pin.
Referenced by USensor_Read2().

USENSOR_GPIO_PORT

```
#define USENSOR_GPIO_PORT GPIOA
```

GPIO port for ultrasonic sensors.
Referenced by USensor_Read1(), and USensor_Read2().

4.10.3 Function Documentation

USensor_DelayUs()

```
void USensor_DelayUs (
            uint16_t us)
```

Delay for a specified number of microseconds.

Parameters

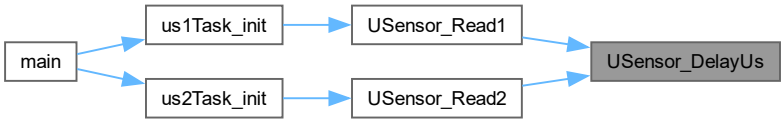
us	Number of microseconds to delay

Delay for a specified number of microseconds.

Parameters

us	Number of microseconds to delay

References htim1.
Referenced by USensor_Read1(), and USensor_Read2().
Here is the caller graph for this function:



USensor_Init()

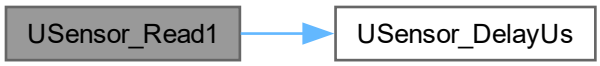
```
void USensor_Init (
            void )
```

Initialize timers and GPIO for ultrasonic sensors.

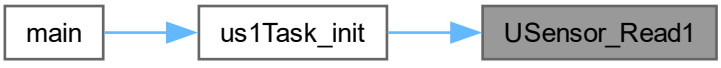
USensor_Read1()

```
void USensor_Read1 (
            void )
```

Trigger the first ultrasonic sensor.
< 10us trigger pulse
References htim1, USENSOR1_TRIG_PIN, USensor_DelayUs(), and USENSOR_GPIO_PORT.
Referenced by us1Task_init().
Here is the call graph for this function:



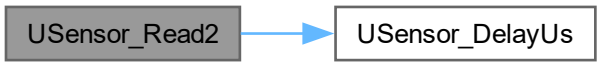
Here is the caller graph for this function:



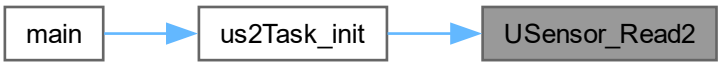
USensor_Read2()

```
void USensor_Read2 (
            void )
```

Trigger the second ultrasonic sensor.
References htim2, USENSOR2_TRIG_PIN, USensor_DelayUs(), and USENSOR_GPIO_PORT.
Referenced by us2Task_init().
Here is the call graph for this function:



Here is the caller graph for this function:



USensor_TIM_IC_Callback()

```
void USensor_TIM_IC_Callback (
            TIM_HandleTypeDef * htim)
```

Callback function for Timer Input Capture events.

Parameters

htim	Pointer to the TIM handle

Callback function for Timer Input Capture events.

Parameters

htim	Pointer to the TIM handle

Measures the pulse width for both ultrasonic sensors and calculates distance.
References Distance_1, Distance_2, htim1, htim2, IC1_Diff, IC1_FirstCaptured, IC1_Val1, IC1_Val2, IC2_Diff, IC2_FirstCaptured, IC2_Val1, and IC2_Val2.
Referenced by HAL_TIM_IC_CaptureCallback().

Here is the caller graph for this function:

4.10.4 Variable Documentation

Distance_1

```
uint8_t Distance_1  [extern]
```

Distance measured by sensor 1 in cm.
Distance measured by sensor 1
Referenced by TxTask_init(), and USensor_TIM_IC_Callback().

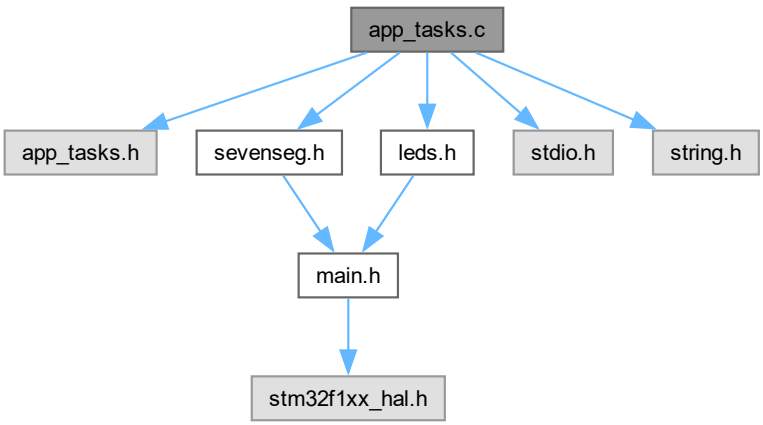
Distance_2

```
uint8_t Distance_2  [extern]
```

Distance measured by sensor 2 in cm.
Distance measured by sensor 2
Referenced by TxTask_init(), and USensor_TIM_IC_Callback().

4.11 receiver_node/Core/Src/app_tasks.c File Reference

FreeRTOS task implementations for the CAN receiver node.
Include dependency graph for receiver_node/Core/Src/app_tasks.c:



Macros

- #define DIG1_HIGH()
- #define DIG1_LOW()
- #define DIG2_HIGH()

- #define DIG2_LOW()
- #define DP_ON()
- #define DP_OFF()

Functions

- void StartDefaultTask (void *argument)

  Default task handling LED indication logic.
- void serialTask_init (void *argument)

  UART serial output task.
- void buzzerTask_init (void *argument)

  Buzzer control task.
- void lcdTask_init (void *argument)

  7-segment display task.

Variables

- float Distance
- int time
- uint8_t RxData [8]
- uint8_t digit1
- uint8_t digit2
- char Buffer [8]
- char lcdBuffer [8]

4.11.1 Detailed Description

FreeRTOS task implementations for the CAN receiver node.
This file contains all FreeRTOS task functions used by the receiver node. Tasks handle LED indication, UART output, buzzer control, and 7-segment display updates based on received CAN distance data.

4.11.2 Macro Definition Documentation

DIG1_HIGH

```
#define DIG1_HIGH()
```

Value:

```
HAL_GPIO_WritePin(GPIOB, DIG1_Pin, GPIO_PIN_SET)
```

Referenced by lcdTask_init().

DIG1_LOW

```
#define DIG1_LOW()
```

Value:

```
HAL_GPIO_WritePin(GPIOB, DIG1_Pin, GPIO_PIN_RESET)
```

Referenced by lcdTask_init().

DIG2_HIGH

```
#define DIG2_HIGH()
```

Value:

```
HAL_GPIO_WritePin(GPIOB, DIG2_Pin, GPIO_PIN_SET)
```

Referenced by lcdTask_init().

DIG2_LOW

```
#define DIG2_LOW()
```

Value:

```
HAL_GPIO_WritePin(GPIOB, DIG2_Pin, GPIO_PIN_RESET)
```

Referenced by lcdTask_init().

DP_OFF

```
#define DP_OFF()
```

Value:

```
HAL_GPIO_WritePin(GPIOB, DP_Pin, GPIO_PIN_RESET)
```

Referenced by lcdTask_init().

DP_ON

```
#define DP_ON()
```

Value:

```
HAL_GPIO_WritePin(GPIOB, DP_Pin, GPIO_PIN_SET)
```

Referenced by lcdTask_init().

4.11.3 Function Documentation

buzzerTask_init()

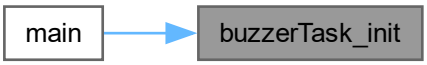
```
void buzzerTask_init (
            void * argument)
```

Buzzer control task.
Controls the buzzer behavior based on the measured distance. The buzzer toggles faster as the distance decreases.

Parameters

argument	Pointer passed to the task (not used).

References Buzzer_Pin, Distance, and time.
Referenced by main().
Here is the caller graph for this function:



lcdTask_init()

```
void lcdTask_init (
            void * argument)
```

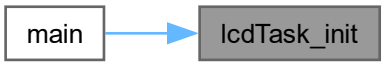
7-segment display task.
LCD / 7-segment display handling task.
LCD display task.
Displays the shortest received distance on a multiplexed 2-digit 7-segment display. If the distance exceeds the maximum threshold, a warning pattern is shown.

Parameters

argument	Pointer passed to the task (not used).

References DIG1_HIGH, DIG1_LOW, DIG2_HIGH, DIG2_LOW, digit1, digit2, Distance, DP_OFF, DP_ON, RxData, and SevenSegment_Update().
Referenced by main().
Here is the call graph for this function:

Here is the caller graph for this function:



serialTask_init()

```
void serialTask_init (
            void * argument)
```

UART serial output task.
Serial communication task.
UART serial communication task.
Periodically transmits the measured distance value via UART for debugging or monitoring purposes.

Parameters

argument	Pointer passed to the task (not used).

References Buffer, Distance, and huart2.
Referenced by main().

Here is the caller graph for this function:

StartDefaultTask()

```
void StartDefaultTask (
            void * argument)
```

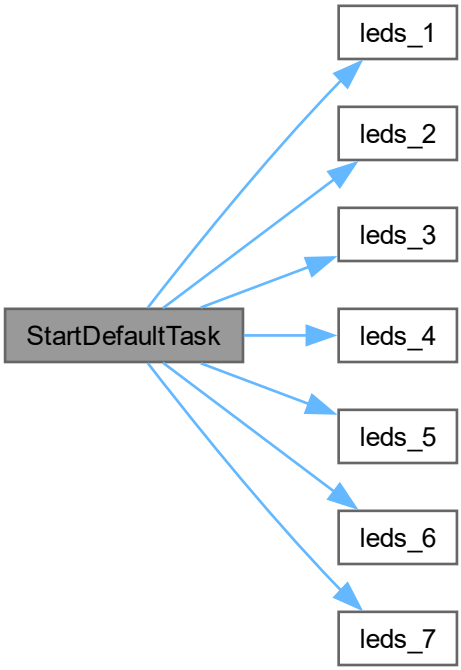
Default task handling LED indication logic.
Default FreeRTOS task.
This task selects the shortest distance received via CAN and updates LED patterns accordingly. It also updates the buzzer timing variable.

Parameters

argument	Pointer passed to the task (not used).

References Distance, leds_1(), leds_2(), leds_3(), leds_4(), leds_5(), leds_6(), leds_7(), RxData, and time.
Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

4.11.4 Variable Documentation

Buffer

`char Buffer[8] [extern]`
UART transmission buffer
Referenced by serialTask_init().

digit1

`uint8_t digit1 [extern]`
First displayed digit
Referenced by lcdTask_init().

digit2

```
uint8_t digit2  [extern]
```
Second displayed digit
Referenced by lcdTask_init().

Distance

```
float Distance  [extern]
```
Distance value computed from received CAN data (meters)
Referenced by buzzerTask_init(), lcdTask_init(), serialTask_init(), and StartDefaultTask().

lcdBuffer

```
char lcdBuffer[8]  [extern]
```
LCD/7-segment buffer

RxData

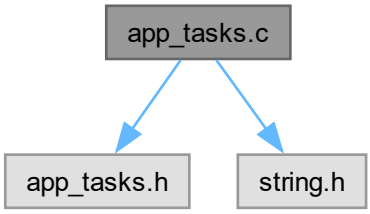
```
uint8_t RxData[8]  [extern]
```
CAN received data buffer
Referenced by HAL_CAN_RxFifo0MsgPendingCallback(), lcdTask_init(), and StartDefaultTask().

time

```
int time  [extern]
```
Buzzer timing variable (ms)
Referenced by buzzerTask_init(), and StartDefaultTask().

4.12 transmitter_node/Core/Src/app_tasks.c File Reference

FreeRTOS task implementations for the transmitter node.
Include dependency graph for transmitter_node/Core/Src/app_tasks.c:



Functions

- void us1Task_init (void *argument)

  RTOS thread to periodically read the first ultrasonic sensor.
- void us2Task_init (void *argument)

  RTOS thread to periodically read the second ultrasonic sensor.
- void TxTask_init (void *argument)

  RTOS thread to transmit sensor distances via CAN bus.

4.12.1 Detailed Description

FreeRTOS task implementations for the transmitter node.
This file contains the RTOS task loops responsible for:

- Periodic ultrasonic sensor triggering (Sensor 1 and Sensor 2)
- Packaging and transmitting distance measurements through CAN

Each task runs independently under FreeRTOS and uses modules provided in usensor.c and the CAN HAL driver.

4.12.2 Function Documentation

TxTask_init()

```
void TxTask_init (
            void * argument)
```

RTOS thread to transmit sensor distances via CAN bus.
Task for transmitting measured distances via CAN.
Task: TxTask_init

Parameters

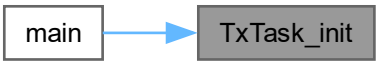
argument	Not used

4.12.2.1 autotoc_md2

Return values

None	

< Unused parameter
< Fill CAN transmit buffer
< Transmit CAN message
< CAN transmission failed, signal with LED (optional)
< Wait 60 ms before next transmission
References Distance_1, Distance_2, hcan, TxData, TxHeader, and TxMailbox.
Referenced by main().
Here is the caller graph for this function:



us1Task_init()

```
void us1Task_init (
            void * argument)
```

RTOS thread to periodically read the first ultrasonic sensor.
Task for first ultrasonic sensor reading.
Task: us1Task_init

Parameters

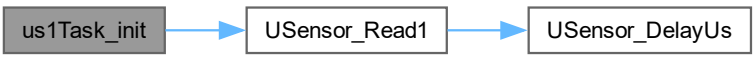
argument	Not used

4.12.2.2 autotoc_md0

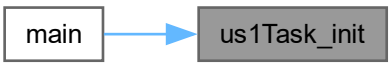
Return values

None	

< Unused parameter
< Trigger first sensor measurement
< Wait minimum 60 ms between measurements
References USensor_Read1().
Referenced by main().
Here is the call graph for this function:



Here is the caller graph for this function:



us2Task_init()

```
void us2Task_init (
            void * argument)
```

RTOS thread to periodically read the second ultrasonic sensor.
Task for second ultrasonic sensor reading.
Task: us2Task_init

Parameters

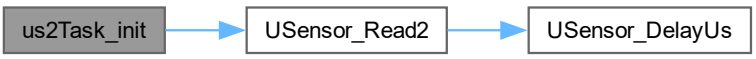
argument	Not used

4.12.2.3 autotoc_md1

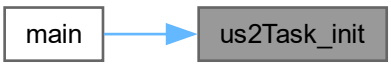
Return values

None	

< Unused parameter
< Trigger second sensor measurement
< Wait minimum 60 ms between measurements
References USensor_Read2().
Referenced by main().
Here is the call graph for this function:

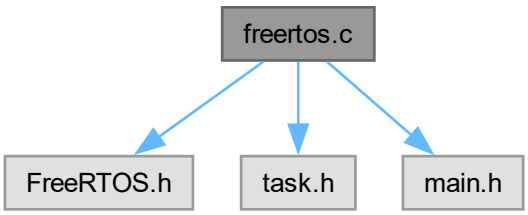


Here is the caller graph for this function:



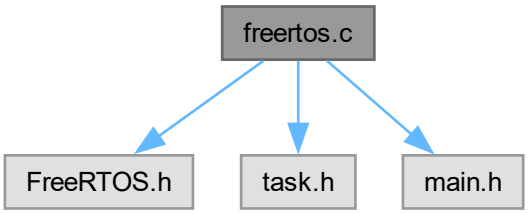
4.13 receiver_node/Core/Src/freertos.c File Reference

Include dependency graph for receiver_node/Core/Src/freertos.c:



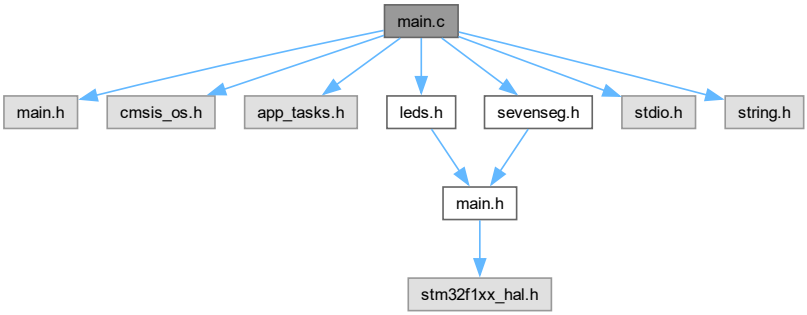
4.14 transmitter_node/Core/Src/freertos.c File Reference

Include dependency graph for transmitter_node/Core/Src/freertos.c:



4.15 receiver_node/Core/Src/main.c File Reference

Include dependency graph for receiver_node/Core/Src/main.c:



Functions

- void SystemClock_Config (void)

  System Clock Configuration.
- static void MX_GPIO_Init (void)

  GPIO Initialization Function.
- static void MX_CAN_Init (void)

  CAN Initialization Function.
- static void MX_USART2_UART_Init (void)

  USART2 Initialization Function.
- void HAL_CAN_RxFifo0MsgPendingCallback (CAN_HandleTypeDef *hcan)

  CAN RX FIFO 0 message pending callback.
- int main (void)
- 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

- CAN_HandleTypeDef hcan
- UART_HandleTypeDef huart2
- osThreadId_t defaultTaskHandle
- osThreadId_t serialTaskHandle
- osThreadId_t buzzerTaskHandle
- osThreadId_t lcdTaskHandle
- float Distance
- int time = 500
- char Buffer [8]
- char lcdBuffer [8]
- uint8_t RxData [8]
- uint32_t TxMailbox
- uint8_t digit1
- uint8_t digit2
- CAN_TxHeaderTypeDef TxHeader
- CAN_RxHeaderTypeDef RxHeader

4.15.1 Function Documentation

Error_Handler()

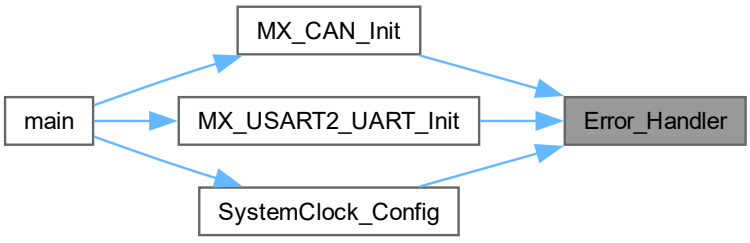
```
void Error_Handler (
            void )
```

This function is executed in case of error occurrence.
Handles unrecoverable application errors.
Handles HAL errors.

Return values

None	

Referenced by MX_CAN_Init(), MX_USART2_UART_Init(), and SystemClock_Config().
Here is the caller graph for this function:



HAL_CAN_RxFifo0MsgPendingCallback()

```
void HAL_CAN_RxFifo0MsgPendingCallback (
            CAN_HandleTypeDef * hcan)
```

CAN RX FIFO 0 message pending callback.

Note

This callback is invoked by the HAL when a CAN message is received in FIFO0. The received frame is read and stored in the RX buffer. In case of reception error, an error indicator LED is activated.

Parameters

hcan	Pointer to the CAN handle.

Return values

None	

References hcan, RxData, and RxHeader.

HAL_TIM_PeriodElapsedCallback()

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

Period elapsed callback in non blocking mode.

Note

This function is called when TIM2 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

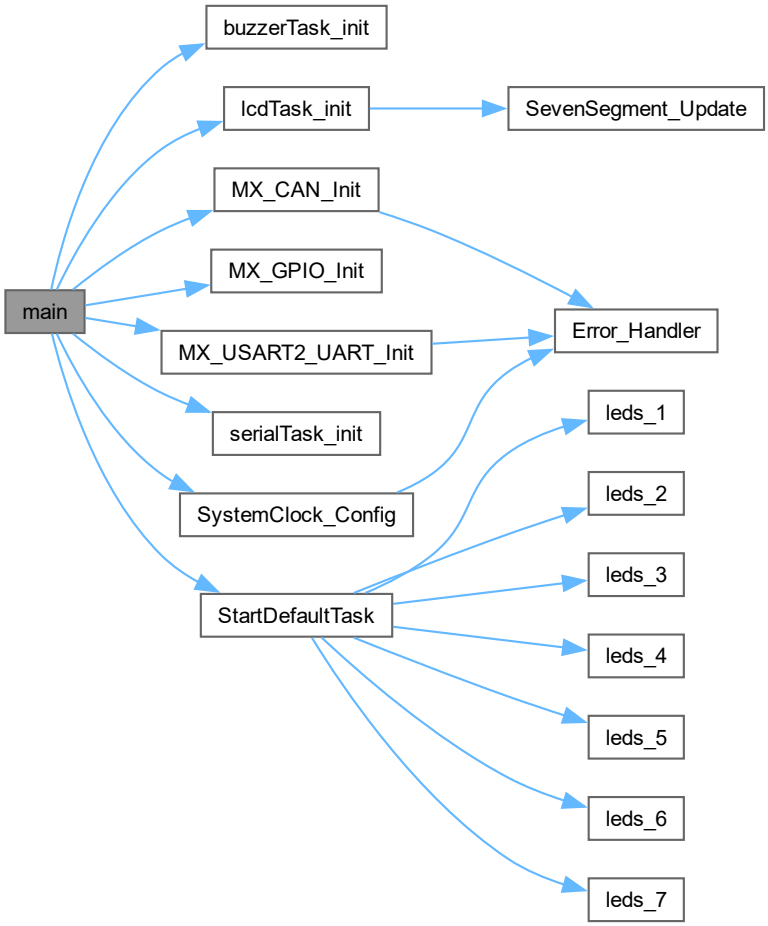
None	

main()

```
int main (
            void )
```

References buzzerTask_init(), buzzerTaskHandle, defaultTaskHandle, hcan, lcdTask_init(), lcdTaskHandle, MX_CAN_Init(), MX_GPIO_Init(), MX_USART2_UART_Init(), serialTask_init(), serialTaskHandle, StartDefaultTask(), and SystemClock_Config().

Here is the call graph for this function:



MX_CAN_Init()

```
void MX_CAN_Init (
            void )  [static]
```

CAN Initialization Function.

Parameters

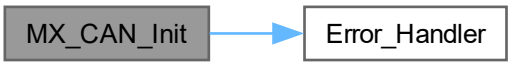
None	

Return values

None	

Configure CAN filter for receiving messages
This block sets up the CAN filter to accept messages from the transmitter node with ID 0x103. Only messages matching this ID will trigger the RX FIFO0 message pending callback.

< Standard CAN frame
< Data frame
< STM32F103 transmitter ID
< Filter bank index
< Filter for ID 0x103
< Mask for ID 0x103
< Number of filters for master CAN
References canfilterconfig, Error_Handler(), hcan, and TxHeader.
Referenced by main().
Here is the call graph for this function:



Here is the caller graph for this function:

MX_GPIO_Init()

```
void MX_GPIO_Init (
            void )  [static]
```

GPIO Initialization Function.

Parameters

None	

Return values

None	

References A_Pin, B_Pin, Blue1_Pin, Blue2_Pin, Buzzer_Pin, C_Pin, D_Pin, DIG1_Pin, DIG2_Pin, DP_Pin, E_Pin, F_Pin, G_Pin, Green1_Pin, Green2_Pin, Green3_Pin, Red1_Pin, and Red2_Pin.
Referenced by main().

Here is the caller graph for this function:

MX_USART2_UART_Init()

```
void MX_USART2_UART_Init (
            void )  [static]
```

USART2 Initialization Function.

Parameters

None	

Return values

None	

References Error_Handler(), and huart2.
Referenced by main().
Here is the call graph for this function:

Here is the caller graph for this function:

SystemClock_Config()

```
void SystemClock_Config (
            void )
```

System Clock Configuration.
Configures the system clock.

Return values

None	

Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef structure.
Initializes the CPU, AHB and APB buses clocks
References Error_Handler().
Referenced by main().
Here is the call graph for this function:

Here is the caller graph for this function:

4.15.2 Variable Documentation

Buffer

```
char Buffer[8]
```

UART transmission buffer
Referenced by serialTask_init().

buzzerTaskHandle

```
osThreadId_t buzzerTaskHandle
```

Handle for the buzzer control task
Referenced by main().

defaultTaskHandle

```
osThreadId_t defaultTaskHandle
```

Handle for the default system task
Referenced by main().

digit1

```
uint8_t digit1
```
First displayed digit
Referenced by lcdTask_init().

digit2

```
uint8_t digit2
```
Second displayed digit
Referenced by lcdTask_init().

Distance

```
float Distance
```
Distance value computed from received CAN data (meters)
Referenced by buzzerTask_init(), lcdTask_init(), serialTask_init(), and StartDefaultTask().

hcan

```
CAN_HandleTypeDef hcan
```
CAN handle
Referenced by HAL_CAN_RxFifo0MsgPendingCallback(), main(), MX_CAN_Init(), MX_CAN_Init(), and TxTask_init().

huart2

```
UART_HandleTypeDef huart2
```
UART2 handle
Referenced by MX_USART2_UART_Init(), MX_USART2_UART_Init(), and serialTask_init().

lcdBuffer

```
char lcdBuffer[8]
```
LCD/7-segment buffer

lcdTaskHandle

```
osThreadId_t lcdTaskHandle
```
Handle for the LCD display task
Referenced by main().

RxData

```
uint8_t RxData[8]
```
CAN received data buffer
Referenced by HAL_CAN_RxFifo0MsgPendingCallback(), lcdTask_init(), and StartDefaultTask().

RxHeader

```
CAN_RxHeaderTypeDef RxHeader
```
Referenced by HAL_CAN_RxFifo0MsgPendingCallback().

serialTaskHandle

```
osThreadId_t serialTaskHandle
```
Handle for the UART serial communication task
Referenced by main().

time

```
int time = 500
```
Buzzer timing variable (ms)
Referenced by buzzerTask_init(), and StartDefaultTask().

TxHeader

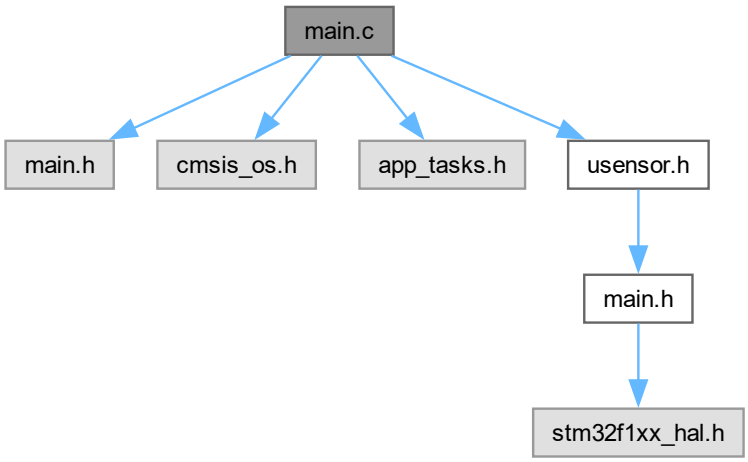
```
CAN_TxHeaderTypeDef TxHeader
```
CAN transmit header
Referenced by MX_CAN_Init(), MX_CAN_Init(), and TxTask_init().

TxMailbox

```
uint32_t TxMailbox
```
CAN transmit mailbox index
CAN mailbox index
Referenced by TxTask_init().

4.16 transmitter_node/Core/Src/main.c File Reference

Include dependency graph for transmitter_node/Core/Src/main.c:



Functions

- void SystemClock_Config (void)

  System Clock Configuration.
- static void MX_GPIO_Init (void)

  GPIO Initialization Function.
- static void MX_TIM1_Init (void)

  TIM1 Initialization Function for input capture.
- static void MX_TIM2_Init (void)

  TIM2 Initialization Function for input capture.
- static void MX_CAN_Init (void)

  CAN Initialization Function.
- static void MX_USART2_UART_Init (void)

USART2 Initialization Function.

- int main (void)

  Main program entry point.
- void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef *htim)

  HAL Timer Input Capture callback.
- void Error_Handler (void)

  Error Handler.

Variables

- TIM_HandleTypeDef htim1

  External peripheral handles used in main.c.
- TIM_HandleTypeDef htim2
- CAN_HandleTypeDef hcan
- UART_HandleTypeDef huart2
- osThreadId_t us1TaskHandle
- osThreadId_t us2TaskHandle
- osThreadId_t TxTaskHandle
- CAN_TxHeaderTypeDef TxHeader
- CAN_FilterTypeDef canfilterconfig
- uint32_t TxMailbox
- uint8_t TxData [8]
- uint8_t Distance_1 = 0

  Distance measured by sensor 1 in cm.
- uint8_t Distance_2 = 0

  Distance measured by sensor 2 in cm.

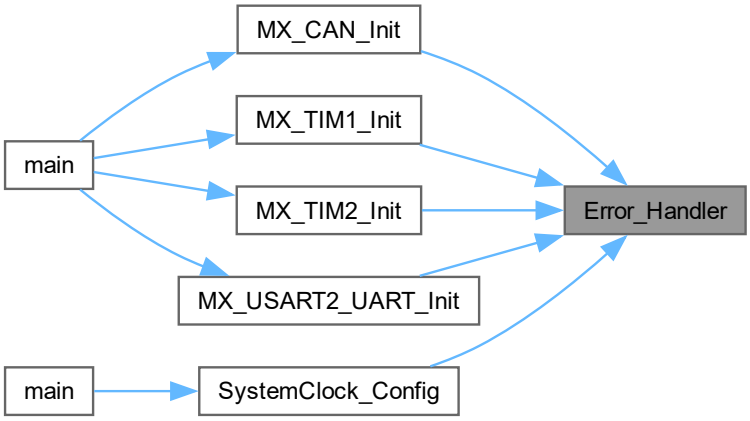
4.16.1 Function Documentation

Error_Handler()

```
void Error_Handler (
            void )
```

Error Handler.
Handles unrecoverable application errors.
Handles HAL errors.
Referenced by MX_CAN_Init(), MX_TIM1_Init(), MX_TIM2_Init(), MX_USART2_UART_Init(), and SystemClock_Config().

Here is the caller graph for this function:



HAL_TIM_IC_CaptureCallback()

```
void HAL_TIM_IC_CaptureCallback (
            TIM_HandleTypeDef * htim)
```

HAL Timer Input Capture callback.

Parameters

htim	Pointer to the timer handle

Note

Forwarded to ultrasonic sensor module

References USensor_TIM_IC_Callback().
Here is the call graph for this function:

main()

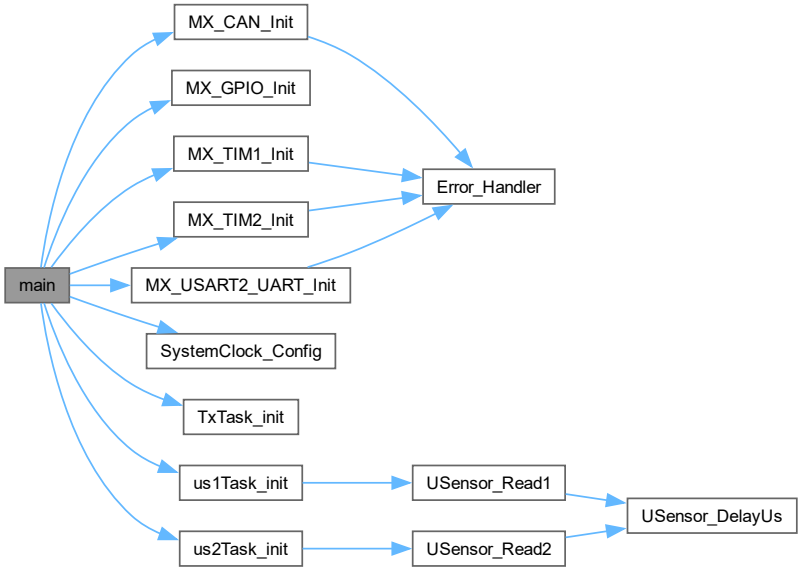
```
int main (
            void )
```

Main program entry point.

Return values

int	(not used, FreeRTOS scheduler takes over)

References hcan, htim1, htim2, MX_CAN_Init(), MX_GPIO_Init(), MX_TIM1_Init(), MX_TIM2_Init(), MX_USART2_UART_Init(), SystemClock_Config(), TxTask_init(), TxTaskHandle, us1Task_init(), us1TaskHandle, us2Task_init(), and us2TaskHandle.
Here is the call graph for this function:

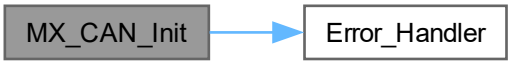


MX_CAN_Init()

```
void MX_CAN_Init (
            void )  [static]
```

CAN Initialization Function.
Configure CAN transmit header and filters
This block sets up the CAN message header for transmitting data and configures the filter to accept messages from the receiver node.
< Standard CAN frame
< Data frame
< Transmitter ID
< Filter bank index
< Accept messages from ID 0x104
< Mask to match ID 0x104
< ID mask mode
< 32-bit filter scale
< Only relevant for dual CAN controllers
< Apply filter configuration
References canfilterconfig, Error_Handler(), hcan, and TxHeader.
Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

MX_GPIO_Init()

```
void MX_GPIO_Init (
            void )  [static]
```

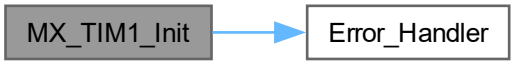
GPIO Initialization Function.
References TRIG1_Pin, and TRIG2_Pin.
Referenced by main().
Here is the caller graph for this function:

MX_TIM1_Init()

```
void MX_TIM1_Init (
            void )  [static]
```

TIM1 Initialization Function for input capture.
References Error_Handler(), and htim1.
Referenced by main().

Here is the call graph for this function:

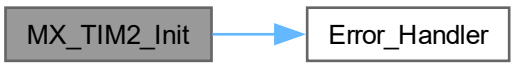


Here is the caller graph for this function:

MX_TIM2_Init()

```
void MX_TIM2_Init (
            void )  [static]
```

TIM2 Initialization Function for input capture.
References Error_Handler(), and htim2.
Referenced by main().
Here is the call graph for this function:



Here is the caller graph for this function:

MX_USART2_UART_Init()

```
void MX_USART2_UART_Init (
            void )  [static]
```

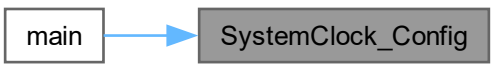
USART2 Initialization Function.
References Error_Handler(), and huart2.
Referenced by main().
Here is the call graph for this function:

Here is the caller graph for this function:

SystemClock_Config()

```
void SystemClock_Config (
            void )
```

System Clock Configuration.
Configures the system clock. Initializes the RCC Oscillators according to the specified parameters in the RCC_↩ OscInitTypeDef structure.
Initializes the CPU, AHB and APB buses clocks
Referenced by main().
Here is the caller graph for this function:



4.16.2 Variable Documentation

canfilterconfig

```
CAN_FilterTypeDef canfilterconfig
```

CAN filter config
Referenced by MX_CAN_Init(), and MX_CAN_Init().

Distance_1

```
uint8_t Distance_1 = 0
```
Distance measured by sensor 1 in cm.
Distance measured by sensor 1
Referenced by TxTask_init(), and USensor_TIM_IC_Callback().

Distance_2

```
uint8_t Distance_2 = 0
```
Distance measured by sensor 2 in cm.
Distance measured by sensor 2
Referenced by TxTask_init(), and USensor_TIM_IC_Callback().

hcan

```
CAN_HandleTypeDef hcan
```
CAN handle

htim1

```
TIM_HandleTypeDef htim1
```
External peripheral handles used in main.c.
Timer 1 handle
Referenced by main(), MX_TIM1_Init(), USensor_DelayUs(), USensor_Read1(), and USensor_TIM_IC_Callback().

htim2

```
TIM_HandleTypeDef htim2
```
Timer 2 handle
Referenced by main(), MX_TIM2_Init(), USensor_Read2(), and USensor_TIM_IC_Callback().

huart2

```
UART_HandleTypeDef huart2
```
UART2 handle

TxData

```
uint8_t TxData[8]
```
CAN transmit buffer
Referenced by TxTask_init().

TxHeader

```
CAN_TxHeaderTypeDef TxHeader
```
CAN transmit header

TxMailbox

```
uint32_t TxMailbox
```
CAN mailbox index

TxTaskHandle

`osThreadId_t TxTaskHandle`
CAN transmit task handle
Referenced by main().

us1TaskHandle

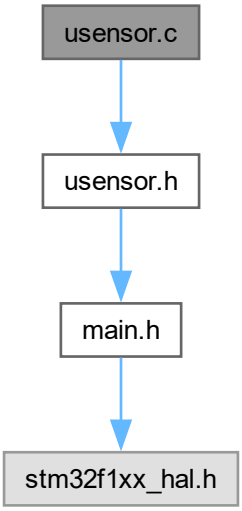
`osThreadId_t us1TaskHandle`
Ultrasonic sensor 1 task handle
Referenced by main().

us2TaskHandle

`osThreadId_t us2TaskHandle`
Ultrasonic sensor 2 task handle
Referenced by main().

4.17 usensor.c File Reference

Ultrasonic sensor measurement driver using STM32 timers.
Include dependency graph for usensor.c:



Functions

- void USensor_DelayUs (uint16_t us)

  Microsecond delay using TIM1.
- void USensor_Read1 (void)

  Trigger the first ultrasonic sensor.
- void USensor_Read2 (void)

  Trigger the second ultrasonic sensor.
- void USensor_TIM_IC_Callback (TIM_HandleTypeDef *htim)

  Input capture callback called from HAL_TIM_IC_CaptureCallback.

Variables

- static uint32_t IC1_Val1 = 0

  Internal variable for input capture of sensor 1.
- static uint32_t IC1_Val2 = 0

  Internal variable for input capture of sensor 1 (second reading).
- static uint8_t IC1_FirstCaptured = 0

  Flag to indicate first capture of sensor 1.
- static uint32_t IC1_Diff = 0

  Time difference captured for sensor 1.
- static uint32_t IC2_Val1 = 0

  Internal variable for input capture of sensor 2.
- static uint32_t IC2_Val2 = 0

  Internal variable for input capture of sensor 2 (second reading).
- static uint8_t IC2_FirstCaptured = 0

  Flag to indicate first capture of sensor 2.
- static uint32_t IC2_Diff = 0

  Time difference captured for sensor 2.

4.17.1 Detailed Description

Ultrasonic sensor measurement driver using STM32 timers.
This file implements:

- Trigger generation for two ultrasonic sensors
- Microsecond delay using a hardware timer
- Input Capture processing for echo pulse measurement

The module converts echo pulse duration into a distance in centimeters and updates the global variables Distance↩
_1 and Distance_2.

4.17.2 Function Documentation

USensor_DelayUs()

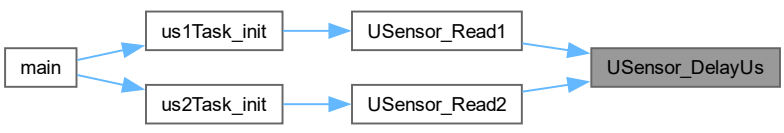
```
void USensor_DelayUs (
            uint16_t us)
```

Microsecond delay using TIM1.
Delay for a specified number of microseconds.

Parameters

us	Number of microseconds to delay

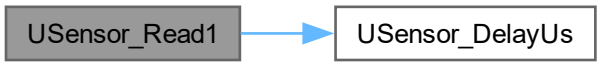
References htim1.
Referenced by USensor_Read1(), and USensor_Read2().
Here is the caller graph for this function:



USensor_Read1()

```
void USensor_Read1 (
            void )
```

Trigger the first ultrasonic sensor.
< 10us trigger pulse
References htim1, USENSOR1_TRIG_PIN, USensor_DelayUs(), and USENSOR_GPIO_PORT.
Referenced by us1Task_init().
Here is the call graph for this function:

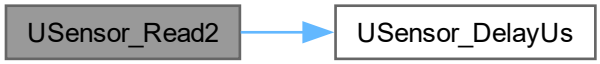


Here is the caller graph for this function:

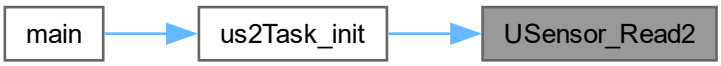
USensor_Read2()

```
void USensor_Read2 (
            void )
```

Trigger the second ultrasonic sensor.
References htim2, USENSOR2_TRIG_PIN, USensor_DelayUs(), and USENSOR_GPIO_PORT.
Referenced by us2Task_init().
Here is the call graph for this function:



Here is the caller graph for this function:



USensor_TIM_IC_Callback()

```
void USensor_TIM_IC_Callback (
            TIM_HandleTypeDef * htim)
```

Input capture callback called from HAL_TIM_IC_CaptureCallback.
Callback function for Timer Input Capture events.

Parameters

htim	Pointer to the TIM handle

Measures the pulse width for both ultrasonic sensors and calculates distance.
References Distance_1, Distance_2, htim1, htim2, IC1_Diff, IC1_FirstCaptured, IC1_Val1, IC1_Val2, IC2_Diff, IC2_FirstCaptured, IC2_Val1, and IC2_Val2.
Referenced by HAL_TIM_IC_CaptureCallback().
Here is the caller graph for this function:

4.17.3 Variable Documentation

IC1_Diff

```
uint32_t IC1_Diff = 0  [static]
```

Time difference captured for sensor 1.
Referenced by USensor_TIM_IC_Callback().

IC1_FirstCaptured

```
uint8_t IC1_FirstCaptured = 0  [static]
```

Flag to indicate first capture of sensor 1.
Referenced by USensor_TIM_IC_Callback().

IC1_Val1

```
uint32_t IC1_Val1 = 0  [static]
```

Internal variable for input capture of sensor 1.
Referenced by USensor_TIM_IC_Callback().

IC1_Val2

```
uint32_t IC1_Val2 = 0  [static]
```
Internal variable for input capture of sensor 1 (second reading).
Referenced by USensor_TIM_IC_Callback().

IC2_Diff

```
uint32_t IC2_Diff = 0  [static]
```
Time difference captured for sensor 2.
Referenced by USensor_TIM_IC_Callback().

IC2_FirstCaptured

```
uint8_t IC2_FirstCaptured = 0  [static]
```
Flag to indicate first capture of sensor 2.
Referenced by USensor_TIM_IC_Callback().

IC2_Val1

```
uint32_t IC2_Val1 = 0  [static]
```
Internal variable for input capture of sensor 2.
Referenced by USensor_TIM_IC_Callback().

IC2_Val2

```
uint32_t IC2_Val2 = 0  [static]
```
Internal variable for input capture of sensor 2 (second reading).
Referenced by USensor_TIM_IC_Callback().