sae_501_601_Cachan

v0.0

Generated by Doxygen 1.9.8

Chapter 1

Topic Index

1.1 Topics

Here is a list of all topics with brief descriptions:

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

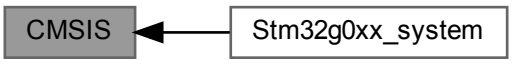
Here is a list of all files with brief descriptions:

Chapter 4

Topic Documentation

4.1 CMSIS

Collaboration diagram for CMSIS:



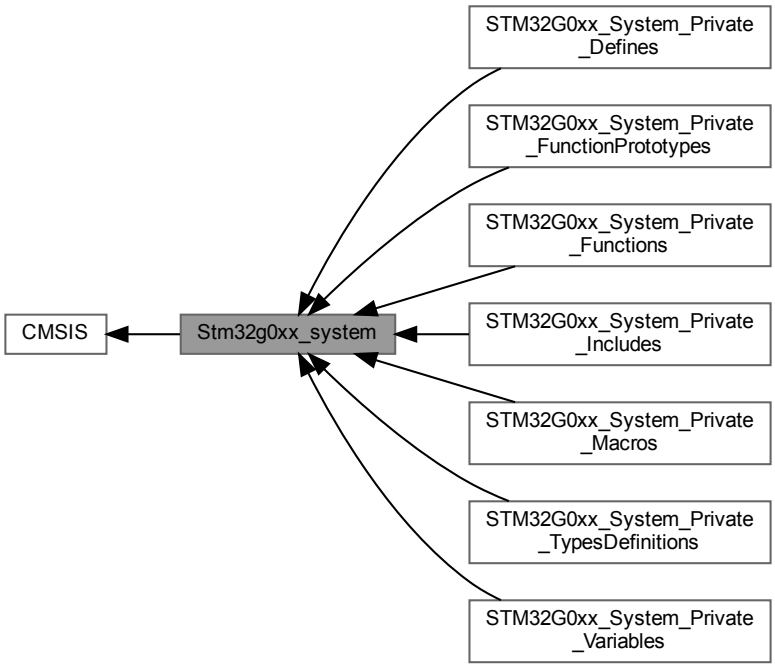
Modules

- Stm32g0xx_system

4.1.1 Detailed Description

4.1.2 Stm32g0xx_system

Collaboration diagram for Stm32g0xx_system:



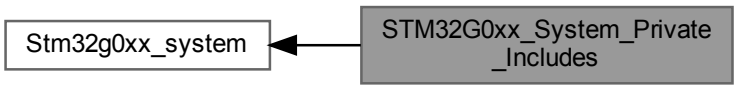
Modules

- STM32G0xx_System_Private_Includes
- STM32G0xx_System_Private_TypesDefinitions
- STM32G0xx_System_Private_Defines
- STM32G0xx_System_Private_Macros
- STM32G0xx_System_Private_Variables
- STM32G0xx_System_Private_FunctionPrototypes
- STM32G0xx_System_Private_Functions

4.1.2.1 Detailed Description

4.1.2.2 STM32G0xx_System_Private_Includes

Collaboration diagram for STM32G0xx_System_Private_Includes:



Macros

- #define HSE_VALUE (8000000UL)
- #define HSI_VALUE (16000000UL)
- #define LSI_VALUE (32000UL)
- #define LSE_VALUE (32768UL)

4.1.2.2.1 Detailed Description

4.1.2.2.2 Macro Definition Documentation

4.1.2.2.2.1 HSE_VALUE

```
#define HSE_VALUE (8000000UL)
```

Value of the External oscillator in Hz

Definition at line 80 of file system_stm32g0xx.c.

4.1.2.2.2.2 HSI_VALUE

```
#define HSI_VALUE (16000000UL)
```

Value of the Internal oscillator in Hz

Definition at line 84 of file system_stm32g0xx.c.

4.1.2.2.2.3 LSE_VALUE

```
#define LSE_VALUE (32768UL)
```

Value of LSE in Hz

Definition at line 92 of file system_stm32g0xx.c.

4.1.2.2.2.4 LSI_VALUE

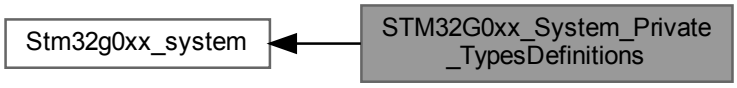
```
#define LSI_VALUE (32000UL)
```

Value of LSI in Hz

Definition at line 88 of file system_stm32g0xx.c.

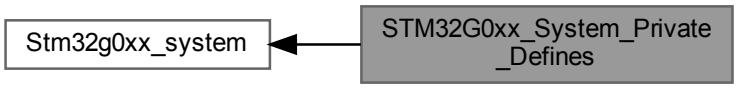
4.1.2.3 STM32G0xx_System_Private_TypesDefinitions

Collaboration diagram for STM32G0xx_System_Private_TypesDefinitions:



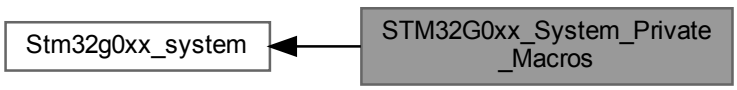
4.1.2.4 STM32G0xx_System_Private_Defines

Collaboration diagram for STM32G0xx_System_Private_Defines:



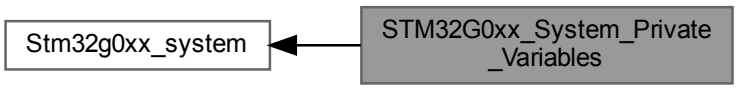
4.1.2.5 STM32G0xx_System_Private_Macros

Collaboration diagram for STM32G0xx_System_Private_Macros:



4.1.2.6 STM32G0xx_System_Private_Variables

Collaboration diagram for STM32G0xx_System_Private_Variables:



Variables

- uint32_t SystemCoreClock = 16000000UL
- const uint32_t AHBPrescTable [16UL] = {0UL, 0UL, 0UL, 0UL, 0UL, 0UL, 0UL, 0UL, 1UL, 2UL, 3UL, 4UL, 6UL, 7UL, 8UL, 9UL}
- const uint32_t APBPrescTable [8UL] = {0UL, 0UL, 0UL, 0UL, 1UL, 2UL, 3UL, 4UL}

4.1.2.6.1 Detailed Description

4.1.2.6.2 Variable Documentation

4.1.2.6.2.1 AHBPrescTable

```
const uint32_t AHBPrescTable[16UL] = {0UL, 0UL, 0UL, 0UL, 0UL, 0UL, 0UL, 0UL, 1UL, 2UL, 3UL,
4UL, 6UL, 7UL, 8UL, 9UL}
```

Definition at line 161 of file system_stm32g0xx.c.

4.1.2.6.2.2 APBPrescTable

```
const uint32_t APBPrescTable[8UL] = {0UL, 0UL, 0UL, 0UL, 1UL, 2UL, 3UL, 4UL}
```

Definition at line 162 of file system_stm32g0xx.c.

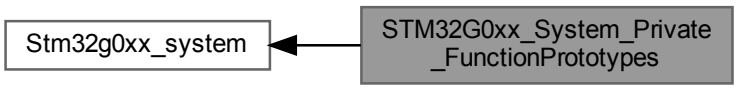
4.1.2.6.2.3 SystemCoreClock

```
uint32_t SystemCoreClock = 16000000UL
```

Definition at line 159 of file system_stm32g0xx.c.

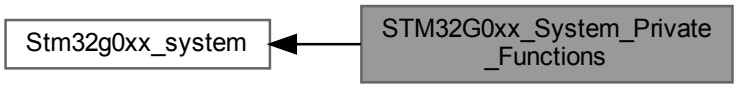
4.1.2.7 STM32G0xx_System_Private_FunctionPrototypes

Collaboration diagram for STM32G0xx_System_Private_FunctionPrototypes:



4.1.2.8 STM32G0xx_System_Private_Functions

Collaboration diagram for STM32G0xx_System_Private_Functions:



Functions

- void SystemInit (void)

  Setup the microcontroller system.
- void SystemCoreClockUpdate (void)

  Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

4.1.2.8.1 Detailed Description

4.1.2.8.2 Function Documentation

4.1.2.8.2.1 SystemCoreClockUpdate()

```
void SystemCoreClockUpdate (
            void  )
```

Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

Note

Each time the core clock (HCLK) changes, this function must be called to update SystemCoreClock variable value. Otherwise, any configuration based on this variable will be incorrect.

- The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:

- If SYSCLK source is HSI, SystemCoreClock will contain the HSI_VALUE(**) / HSI division factor
- If SYSCLK source is HSE, SystemCoreClock will contain the HSE_VALUE(***)
- If SYSCLK source is LSI, SystemCoreClock will contain the LSI_VALUE
- If SYSCLK source is LSE, SystemCoreClock will contain the LSE_VALUE
- If SYSCLK source is PLL, SystemCoreClock will contain the HSE_VALUE(***) or HSI_VALUE(*) multiplied/divided by the PLL factors.

(**) HSI_VALUE is a constant defined in stm32g0xx_hal_conf.h file (default value 16 MHz) but the real value may vary depending on the variations in voltage and temperature.

(***) HSE_VALUE is a constant defined in stm32g0xx_hal_conf.h file (default value 8 MHz), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.

- The result of this function could be not correct when using fractional value for HSE crystal.

Parameters

None	

Return values

None	

Definition at line 233 of file system_stm32g0xx.c.

4.1.2.8.2.2 SystemInit()

```
void SystemInit (
             void  )
```

Setup the microcontroller system.

Parameters

None	

Return values

None	

Definition at line 185 of file system_stm32g0xx.c.

Chapter 5

Data Structure Documentation

5.1 bmi088_cs_t Struct Reference

Structure de définition d'un Chip Select SPI.

`#include <driver_ins.h>`

Data Fields

- GPIO_TypeDef * port
  Port GPIO STM32.
- uint16_t pin
  Numéro de Pin GPIO.

5.1.1 Detailed Description

Structure de définition d'un Chip Select SPI.

Definition at line 52 of file driver_ins.h.

5.1.2 Field Documentation

5.1.2.1 pin

`uint16_t pin`

Numéro de Pin GPIO.

Definition at line 54 of file driver_ins.h.

5.1.2.2 port

```
GPIO_TypeDef* port
```

Port GPIO STM32.

Definition at line 53 of file driver_ins.h.

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

- Core/Inc/driver_ins.h

5.2 bmi088_data_t Struct Reference

Structure de données physiques unifiées pour l'application.

```
#include <driver_ins.h>
```

Data Fields

- float accel_x_mms2
  Accélération X (mm/s^2).
- float accel_y_mms2
  Accélération Y (mm/s^2).
- float accel_z_mms2
  Accélération Z (mm/s^2).
- float gyro_x_rads
  Vitesse angulaire X (rad/s).
- float gyro_y_rads
  Vitesse angulaire Y (rad/s).
- float gyro_z_rads
  Vitesse angulaire Z (rad/s).
- uint32_t timestamp_ms
  Date de la mesure (ms depuis le démarrage).

5.2.1 Detailed Description

Structure de données physiques unifiées pour l'application.

Definition at line 60 of file driver_ins.h.

5.2.2 Field Documentation

5.2.2.1 accel_x_mms2

```
float accel_x_mms2
```

Accélération X (mm/s^2).

Definition at line 61 of file driver_ins.h.

5.2.2.2 accel_y_mms2

```
float accel_y_mms2
```

Accélération Y (mm/s^2).

Definition at line 62 of file driver_ins.h.

5.2.2.3 accel_z_mms2

```
float accel_z_mms2
```

Accélération Z (mm/s^2).

Definition at line 63 of file driver_ins.h.

5.2.2.4 gyro_x_rads

```
float gyro_x_rads
```

Vitesse angulaire X (rad/s).

Definition at line 65 of file driver_ins.h.

5.2.2.5 gyro_y_rads

```
float gyro_y_rads
```

Vitesse angulaire Y (rad/s).

Definition at line 66 of file driver_ins.h.

5.2.2.6 gyro_z_rads

```
float gyro_z_rads
```

Vitesse angulaire Z (rad/s).

Definition at line 67 of file driver_ins.h.

5.2.2.7 timestamp_ms

```
uint32_t timestamp_ms
```

Date de la mesure (ms depuis le démarrage).

Definition at line 69 of file driver_ins.h.

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

- Core/Inc/driver_ins.h

5.3 Motor_Context_t Struct Reference

Structure de contexte interne pour la gestion des transitions.

`#include <driver_motor.h>`

Data Fields

- int16_t target_speed_mms
  Vitesse cible demandée en mm/s.
- uint8_t target_pwm
  Pourcentage PWM cible calculé (0-100).
- bool target_forward
  Direction cible (true=avant, false=arrière).
- uint32_t deadline_ms
  Echéance temporelle pour les états temporisés.

5.3.1 Detailed Description

Structure de contexte interne pour la gestion des transitions.

Definition at line 33 of file driver_motor.h.

5.3.2 Field Documentation

5.3.2.1 deadline_ms

```
uint32_t deadline_ms
```

Echéance temporelle pour les états temporisés.

Definition at line 37 of file driver_motor.h.

5.3.2.2 target_forward

```
bool target_forward
```

Direction cible (true=avant, false=arrière).

Definition at line 36 of file driver_motor.h.

5.3.2.3 target_pwm

```
uint8_t target_pwm
```

Pourcentage PWM cible calculé (0-100).

Definition at line 35 of file driver_motor.h.

5.3.2.4 target_speed_mms

```
int16_t target_speed_mms
```

Vitesse cible demandée en mm/s.

Definition at line 34 of file driver_motor.h.

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

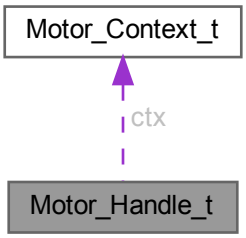
- Core/Inc/driver_motor.h

5.4 Motor_Handle_t Struct Reference

Handle principal de l'objet Moteur.

```
#include <driver_motor.h>
```

Collaboration diagram for Motor_Handle_t:



Data Fields

- TIM_HandleTypeDef * htim
  Pointeur vers le handle du Timer PWM (ex: &htim2).
- uint32_t channel
  Canal du Timer utilisé (ex: TIM_CHANNEL_1).
- uint16_t min_pulse_ticks
  Valeur du registre CCR pour 0% de PWM.
- uint16_t max_pulse_ticks
  Valeur du registre CCR pour 100% de PWM.
- int16_t max_speed_pos_mms
  Vitesse physique maximale en marche avant (mm/s).
- int16_t max_speed_neg_mms
  Vitesse physique maximale en marche arrière (mm/s, valeur négative).
- MotorState_t state
  État actuel de la machine à états.
- bool go_forward
  Direction actuelle appliquée.
- Motor_Context_t ctx
  Contexte de transition (cible).

5.4.1 Detailed Description

Handle principal de l'objet Moteur.

Contient la configuration matérielle, les limites physiques et l'état courant.

Definition at line 44 of file driver_motor.h.

5.4.2 Field Documentation

5.4.2.1 channel

```
uint32_t channel
```

Canal du Timer utilisé (ex: TIM_CHANNEL_1).

Definition at line 46 of file driver_motor.h.

5.4.2.2 ctx

```
Motor_Context_t ctx
```

Contexte de transition (cible).

Definition at line 55 of file driver_motor.h.

5.4.2.3 go_forward

```
bool go_forward
```

Direction actuelle appliquée.

Definition at line 54 of file driver_motor.h.

5.4.2.4 htim

```
TIM_HandleTypeDef* htim
```

Pointeur vers le handle du Timer PWM (ex: &htim2).

Definition at line 45 of file driver_motor.h.

5.4.2.5 max_pulse_ticks

```
uint16_t max_pulse_ticks
```

Valeur du registre CCR pour 100% de PWM.

Definition at line 48 of file driver_motor.h.

5.4.2.6 max_speed_neg_mms

```
int16_t max_speed_neg_mms
```

Vitesse physique maximale en marche arrière (mm/s, valeur négative).

Definition at line 51 of file driver_motor.h.

5.4.2.7 max_speed_pos_mms

```
int16_t max_speed_pos_mms
```

Vitesse physique maximale en marche avant (mm/s).

Definition at line 50 of file driver_motor.h.

5.4.2.8 min_pulse_ticks

```
uint16_t min_pulse_ticks
```

Valeur du registre CCR pour 0% de PWM.

Definition at line 47 of file driver_motor.h.

5.4.2.9 state

```
MotorState_t state
```

État actuel de la machine à états.

Definition at line 53 of file driver_motor.h.

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

- Core/Inc/driver_motor.h

5.5 Servo_Handle_t Struct Reference

Structure de configuration et de gestion du Servo.

```
#include <driver_servo.h>
```

Data Fields

- TIM_HandleTypeDef * htim
  Pointeur vers le handle du Timer (HAL).
- uint32_t channel
  Canal du Timer (ex: TIM_CHANNEL_1).
- uint16_t min_pulse_ticks
  Valeur registre CCR pour la position min (ex: 3200).
- uint16_t max_pulse_ticks
  Valeur registre CCR pour la position max (ex: 6400).

5.5.1 Detailed Description

Structure de configuration et de gestion du Servo.

Cette structure lie le matériel (Timer/Channel) aux bornes physiques du servomoteur (valeurs PWM min/max).

Definition at line 18 of file driver_servo.h.

5.5.2 Field Documentation

5.5.2.1 channel

```
uint32_t channel
```

Canal du Timer (ex: TIM_CHANNEL_1).

Definition at line 20 of file driver_servo.h.

5.5.2.2 htim

```
TIM_HandleTypeDef* htim
```

Pointeur vers le handle du Timer (HAL).

Definition at line 19 of file driver_servo.h.

5.5.2.3 max_pulse_ticks

```
uint16_t max_pulse_ticks
```

Valeur registre CCR pour la position max (ex: 6400).

Definition at line 22 of file driver_servo.h.

5.5.2.4 min_pulse_ticks

```
uint16_t min_pulse_ticks
```

Valeur registre CCR pour la position min (ex: 3200).

Definition at line 21 of file driver_servo.h.

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

- Core/Inc/driver_servo.h

5.6 Speedometer_Handle_t Struct Reference

Structure de gestion du tachymètre.

```
#include <driver_speedometer.h>
```

Data Fields

- TIM_HandleTypeDef * htim
  Pointeur vers le Timer utilisé en mode compteur.
- uint16_t last_counter_val
  Valeur du compteur lors de la dernière lecture.
- uint32_t last_process_time
  Timestamp (ms) de la dernière lecture.
- float current_speed_ms
  Vitesse actuelle calculée en m/s.

5.6.1 Detailed Description

Structure de gestion du tachymètre.

Stocke l'état précédent du compteur et du temps pour calculer la vitesse différentielle.

Definition at line 35 of file driver_speedometer.h.

5.6.2 Field Documentation

5.6.2.1 current_speed_ms

```
float current_speed_ms
```

Vitesse actuelle calculée en m/s.

Definition at line 39 of file driver_speedometer.h.

5.6.2.2 htim

```
TIM_HandleTypeDef* htim
```

Pointeur vers le Timer utilisé en mode compteur.

Definition at line 36 of file driver_speedometer.h.

5.6.2.3 last_counter_val

```
uint16_t last_counter_val
```

Valeur du compteur lors de la dernière lecture.

Definition at line 37 of file driver_speedometer.h.

5.6.2.4 last_process_time

```
uint32_t last_process_time
```

Timestamp (ms) de la dernière lecture.

Definition at line 38 of file driver_speedometer.h.

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

- Core/Inc/driver_speedometer.h

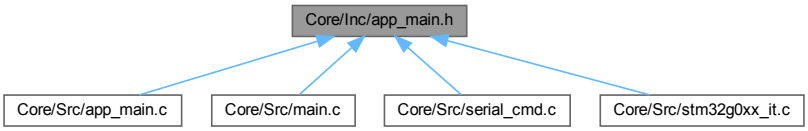
Chapter 6

File Documentation

6.1 Core/Inc/app_main.h File Reference

En-tête de l'application principale.

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



Functions

- void app_config (void)

  Configure l'ensemble de l'application (Hardware + Drivers).
- void app_loop (void)

  Exécute la boucle principale (Scheduler).

Variables

- float speed_speedo_data

  Variable globale de vitesse (m/s) partagée entre le Speedometer et la Télémétrie.
- volatile uint32_t tim3_overflow_cnt

  Compteur d'overflow pour l'extension 32-bits du Timer microseconde.

6.1.1 Detailed Description

En-tête de l'application principale.

Expose les points d'entrée (Setup/Loop) et les variables globales partagées nécessaires à la gestion du temps et de la vitesse.

Definition in file app_main.h.

6.1.2 Function Documentation

6.1.2.1 app_config()

```
void app_config (
            void  )
```

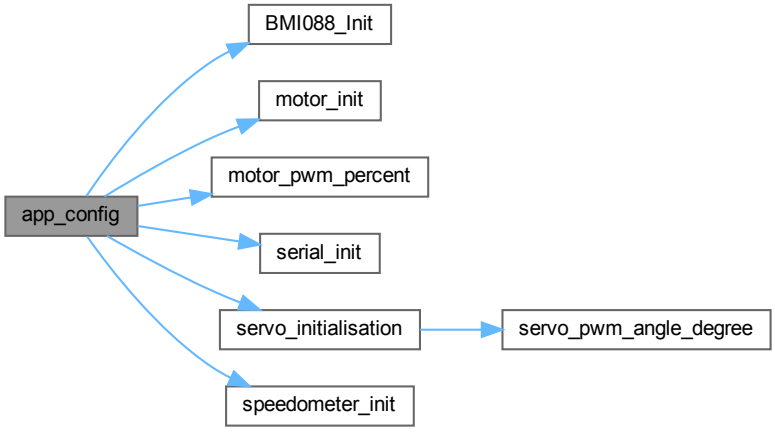
Configure l'ensemble de l'application (Hardware + Drivers).

Configure l'ensemble de l'application (Hardware + Drivers).

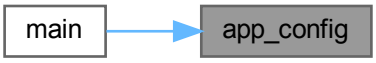
Configure les Timers, active les interruptions nécessaires, initialise les drivers (Série, BMI088, Servo, Moteur, Speedo) et cale les horloges.

Definition at line 193 of file app_main.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.2.2 app_loop()

```
void app_loop (
            void  )
```

Exécute la boucle principale (Scheduler).

Exécute la boucle principale (Scheduler).

Exécute séquentiellement :

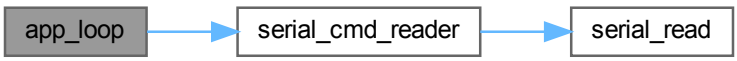
1. La lecture des données série (Polling).
2. Le traitement des commandes (si disponibles).
3. La vérification de sécurité.
4. L'ordonnancement des tâches périodiques basées sur `now_us`.

   Note

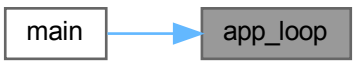
   Le compteur `now_us` boucle après environ 71 minutes.

Definition at line 221 of file app_main.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.3 Variable Documentation

6.1.3.1 speed_speedo_data

```
float speed_speedo_data  [extern]
```

Variable globale de vitesse (m/s) partagée entre le Speedometer et la Télémétrie.

Variable globale de vitesse (m/s) partagée entre le Speedometer et la Télémétrie.

Definition at line 81 of file app_main.c.

6.1.3.2 tim3_overflow_cnt

```
volatile uint32_t tim3_overflow_cnt  [extern]
```

Compteur d'overflow pour l'extension 32-bits du Timer microseconde.

Compteur d'overflow pour l'extension 32-bits du Timer microseconde.

Definition at line 84 of file app_main.c.

6.2 app_main.h

Go to the documentation of this file.

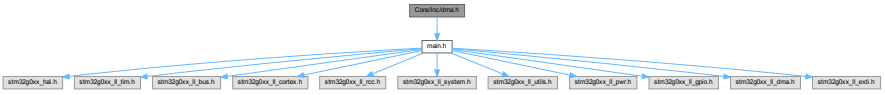
```
00001
00008 #ifndef INC_APP_MAIN_H_
00009 #define INC_APP_MAIN_H_
00010
00014 void app_config(void);
00015
00019 void app_loop(void);
00020
00022 extern float speed_speedo_data;
00023
00025 extern volatile uint32_t tim3_overflow_cnt;
00026
00027 #endif
```

6.3 Core/Inc/dma.h File Reference

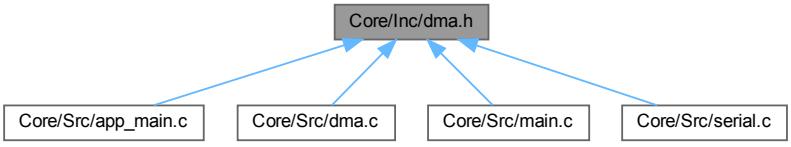
This file contains all the function prototypes for the dma.c file.

```
#include "main.h"
```

Include dependency graph for dma.h:



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



Functions

- void MX_DMA_Init (void)

6.3.1 Detailed Description

This file contains all the function prototypes for the dma.c file.

Attention

Definition in file dma.h.

6.3.2 Function Documentation

6.3.2.1 MX_DMA_Init()

```
void MX_DMA_Init (
             void  )
```

Enable DMA controller clock

Definition at line 39 of file dma.c.

Here is the caller graph for this function:

6.4 dma.h

Go to the documentation of this file.

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020 /* Define to prevent recursive inclusion -------------------------------------*/
00021 #ifndef __DMA_H__
00022 #define __DMA_H__
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif
00027
00028 /* Includes ------------------------------------------------------------------*/
00029 #include "main.h"
00030
00031 /* DMA memory to memory transfer handles -------------------------------------*/
00032
00033 /* USER CODE BEGIN Includes */
```

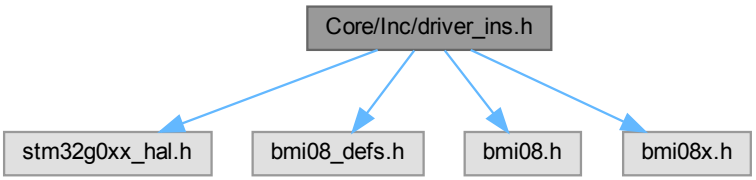
```
00034
00035 /* USER CODE END Includes */
00036
00037 /* USER CODE BEGIN Private defines */
00038
00039 /* USER CODE END Private defines */
00040
00041 void MX_DMA_Init(void);
00042
00043 /* USER CODE BEGIN Prototypes */
00044
00045 /* USER CODE END Prototypes */
00046
00047 #ifdef __cplusplus
00048 }
00049 #endif
00050
00051 #endif /* __DMA_H__ */
00052
```

6.5 Core/Inc/driver_ins.h File Reference

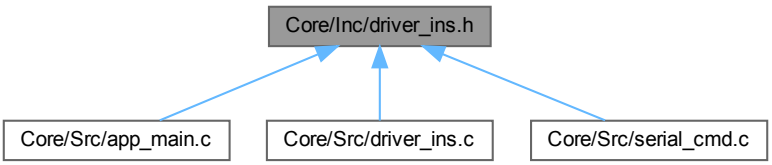
Définitions et prototypes pour le pilote BMI088.

```
#include "stm32g0xx_hal.h"
#include "bmi08_defs.h"
#include "bmi08.h"
#include "bmi08x.h"
```

Include dependency graph for driver_ins.h:



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



Data Structures

- struct bmi088_cs_t

  Structure de définition d'un Chip Select SPI.
- struct bmi088_data_t

  Structure de données physiques unifiées pour l'application.

Macros

- #define BMI088_CS_ACC_GPIO_Port GPIOB

  Port GPIO du Chip Select Accéléromètre (PA4/PB13 selon config).
- #define BMI088_CS_ACC_Pin GPIO_PIN_13

  Pin GPIO du Chip Select Accéléromètre.
- #define BMI088_CS_GYRO_GPIO_Port GPIOB

  Port GPIO du Chip Select Gyroscope (PA1/PB14 selon config).
- #define BMI088_CS_GYRO_Pin GPIO_PIN_14

  Pin GPIO du Chip Select Gyroscope.
- #define ACCEL_RANGE_3G_LSB 10922.67f

  Facteur d'échelle LSB/g pour la gamme +/- 3g.
- #define ACCEL_RANGE_6G_LSB 5461.33f

  Facteur d'échelle LSB/g pour la gamme +/- 6g.
- #define ACCEL_RANGE_12G_LSB 2730.67f

  Facteur d'échelle LSB/g pour la gamme +/- 12g.
- #define ACCEL_RANGE_24G_LSB 1365.33f

  Facteur d'échelle LSB/g pour la gamme +/- 24g.
- #define G_TO_MM_S2 9806.65f

  Constante de conversion de gravité g vers mm/s^2.
- #define GYRO_RANGE_125DPS_LSB 262.4f

  Facteur d'échelle LSB/dps pour la gamme +/- 125 dps.
- #define GYRO_RANGE_250DPS_LSB 131.2f

  Facteur d'échelle LSB/dps pour la gamme +/- 250 dps.
- #define GYRO_RANGE_500DPS_LSB 65.6f

  Facteur d'échelle LSB/dps pour la gamme +/- 500 dps.
- #define GYRO_RANGE_1000DPS_LSB 32.768f

  Facteur d'échelle LSB/dps pour la gamme +/- 1000 dps.
- #define GYRO_RANGE_2000DPS_LSB 16.4f

  Facteur d'échelle LSB/dps pour la gamme +/- 2000 dps.
- #define DEG_TO_RAD 0.017453292519943295f

  Constante de conversion degrés vers radians.

Functions

- int8_t BMI088_Init (SPI_HandleTypeDef ∗hspi)

  Initialise le driver BMI088.
- int8_t BMI088_Read_Accel_Raw (struct bmi08_sensor_data ∗accel_data)

  Lit les registres bruts de l'accéléromètre.
- int8_t BMI088_Read_Gyro_Raw (struct bmi08_sensor_data ∗gyro_data)

  Lit les registres bruts du gyroscope.
- int8_t BMI088_Read_All (bmi088_data_t ∗data)

  Effectue une lecture complète et convertit les données.
- void BMI088_Convert_Accel (struct bmi08_sensor_data ∗accel_raw, float ∗accel_mms2)

  Convertit un jeu de données brutes accéléromètre en mm/s^2.
- void BMI088_Convert_Gyro (struct bmi08_sensor_data ∗gyro_raw, float ∗gyro_rads)

  Convertit un jeu de données brutes gyroscope en rad/s.
- uint8_t BMI088_Test_Communication (uint8_t print_result)

  Vérifie la présence des capteurs sur le bus SPI.
- int8_t BMI088_Soft_Reset (void)

  Redémarre les capteurs via commande logicielle.

6.5.1 Detailed Description

Définitions et prototypes pour le pilote BMI088.

Contient les macros de configuration matérielle, les constantes de conversion physique et les structures de données publiques.

Definition in file driver_ins.h.

6.5.2 Macro Definition Documentation

6.5.2.1 ACCEL_RANGE_12G_LSB

```
#define ACCEL_RANGE_12G_LSB 2730.67f
```

Facteur d'échelle LSB/g pour la gamme +/- 12g.

Definition at line 30 of file driver_ins.h.

6.5.2.2 ACCEL_RANGE_24G_LSB

```
#define ACCEL_RANGE_24G_LSB 1365.33f
```

Facteur d'échelle LSB/g pour la gamme +/- 24g.

Definition at line 32 of file driver_ins.h.

6.5.2.3 ACCEL_RANGE_3G_LSB

```
#define ACCEL_RANGE_3G_LSB 10922.67f
```

Facteur d'échelle LSB/g pour la gamme +/- 3g.

Definition at line 26 of file driver_ins.h.

6.5.2.4 ACCEL_RANGE_6G_LSB

```
#define ACCEL_RANGE_6G_LSB 5461.33f
```

Facteur d'échelle LSB/g pour la gamme +/- 6g.

Definition at line 28 of file driver_ins.h.

6.5.2.5 BMI088_CS_ACC_GPIO_Port

```
#define BMI088_CS_ACC_GPIO_Port GPIOB
```

Port GPIO du Chip Select Accéléromètre (PA4/PB13 selon config).

Definition at line 17 of file driver_ins.h.

6.5.2.6 BMI088_CS_ACC_Pin

```
#define BMI088_CS_ACC_Pin GPIO_PIN_13
```

Pin GPIO du Chip Select Accéléromètre.

Definition at line 19 of file driver_ins.h.

6.5.2.7 BMI088_CS_GYRO_GPIO_Port

```
#define BMI088_CS_GYRO_GPIO_Port GPIOB
```

Port GPIO du Chip Select Gyroscope (PA1/PB14 selon config).

Definition at line 21 of file driver_ins.h.

6.5.2.8 BMI088_CS_GYRO_Pin

```
#define BMI088_CS_GYRO_Pin GPIO_PIN_14
```

Pin GPIO du Chip Select Gyroscope.

Definition at line 23 of file driver_ins.h.

6.5.2.9 DEG_TO_RAD

```
#define DEG_TO_RAD 0.017453292519943295f
```

Constante de conversion degrés vers radians.

Definition at line 47 of file driver_ins.h.

6.5.2.10 G_TO_MM_S2

```
#define G_TO_MM_S2 9806.65f
```

Constante de conversion de gravité g vers mm/s^2.

Definition at line 34 of file driver_ins.h.

6.5.2.11 GYRO_RANGE_1000DPS_LSB

```
#define GYRO_RANGE_1000DPS_LSB 32.768f
```

Facteur d'échelle LSB/dps pour la gamme +/- 1000 dps.

Definition at line 43 of file driver_ins.h.

6.5.2.12 GYRO_RANGE_125DPS_LSB

```
#define GYRO_RANGE_125DPS_LSB 262.4f
```

Facteur d'échelle LSB/dps pour la gamme +/- 125 dps.

Definition at line 37 of file driver_ins.h.

6.5.2.13 GYRO_RANGE_2000DPS_LSB

```
#define GYRO_RANGE_2000DPS_LSB 16.4f
```

Facteur d'échelle LSB/dps pour la gamme +/- 2000 dps.

Definition at line 45 of file driver_ins.h.

6.5.2.14 GYRO_RANGE_250DPS_LSB

```
#define GYRO_RANGE_250DPS_LSB 131.2f
```

Facteur d'échelle LSB/dps pour la gamme +/- 250 dps.

Definition at line 39 of file driver_ins.h.

6.5.2.15 GYRO_RANGE_500DPS_LSB

```
#define GYRO_RANGE_500DPS_LSB 65.6f
```

Facteur d'échelle LSB/dps pour la gamme +/- 500 dps.

Definition at line 41 of file driver_ins.h.

6.5.3 Function Documentation

6.5.3.1 BMI088_Convert_Accel()

```
void BMI088_Convert_Accel (
            struct bmi08_sensor_data * accel_raw,
            float * accel_mms2 )
```

Convertit un jeu de données brutes accéléromètre en mm/s^2.

Parameters

accel_raw	Données d'entrée brutes.
accel_mms2	Tableau de sortie [X, Y, Z].

Convertit un jeu de données brutes accéléromètre en mm/s^2.

Parameters

accel_raw	Pointeur vers les données brutes d'entrée.
accel_mms2	Pointeur vers le tableau de sortie (x, y, z) en mm/s^2.

Definition at line 261 of file driver_ins.c.

6.5.3.2 BMI088_Convert_Gyro()

```
void BMI088_Convert_Gyro (
            struct bmi08_sensor_data * gyro_raw,
            float * gyro_rads )
```

Convertit un jeu de données brutes gyroscope en rad/s.

Parameters

gyro_raw	Données d'entrée brutes.
gyro_rads	Tableau de sortie [X, Y, Z].

Convertit un jeu de données brutes gyroscope en rad/s.

Parameters

gyro_raw	Pointeur vers les données brutes d'entrée.
gyro_rads	Pointeur vers le tableau de sortie (x, y, z) en rad/s.

Definition at line 276 of file driver_ins.c.

6.5.3.3 BMI088_Init()

```
int8_t BMI088_Init (
            SPI_HandleTypeDef * hspi )
```

Initialise le driver BMI088.

Parameters

hspi	Pointeur vers le handle SPI utilisé.

Returns

BMI08_OK ou code d'erreur.

Initialise le driver BMI088.

Parameters

hspi	Pointeur vers le handle SPI STM32.

Returns

BMI08_OK si l'initialisation réussit, code d'erreur sinon.

Definition at line 133 of file driver_ins.c.

Here is the caller graph for this function:

6.5.3.4 BMI088_Read_Accel_Raw()

```
int8_t BMI088_Read_Accel_Raw (
            struct bmi08_sensor_data * accel_data )
```

Lit les registres bruts de l'accéléromètre.

Parameters

accel_data	Structure de sortie pour les données brutes.

Returns

BMI08_OK ou code d'erreur.

Lit les registres bruts de l'accéléromètre.

Parameters

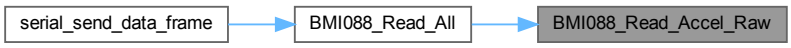
accel_data	Pointeur vers la structure de destination des données brutes.

Returns

BMI08_OK en cas de succès, ou code d'erreur.

Definition at line 200 of file driver_ins.c.

Here is the caller graph for this function:



6.5.3.5 BMI088_Read_All()

```
int8_t BMI088_Read_All (
            bmi088_data_t * data )
```

Effectue une lecture complète et convertit les données.

Parameters

data	Structure de sortie pour les données physiques.

Returns

BMI08_OK ou code d'erreur.

Effectue une lecture complète et convertit les données.

Parameters

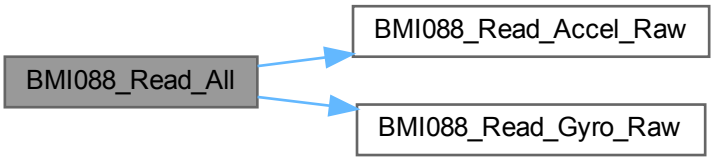
data	Pointeur vers la structure de données utilisateur (unités physiques).

Returns

BMI08_OK en cas de succès, ou code d'erreur.

Definition at line 226 of file driver_ins.c.

Here is the call graph for this function:



Here is the caller graph for this function:

6.5.3.6 BMI088_Read_Gyro_Raw()

```
int8_t BMI088_Read_Gyro_Raw (
            struct bmi08_sensor_data * gyro_data )
```

Lit les registres bruts du gyroscope.

Parameters

gyro_data	Structure de sortie pour les données brutes.

Returns

BMI08_OK ou code d'erreur.

Lit les registres bruts du gyroscope.

Parameters

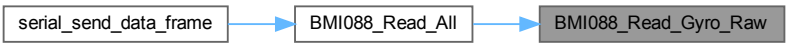
gyro_data	Pointeur vers la structure de destination des données brutes.

Returns

BMI08_OK en cas de succès, ou code d'erreur.

Definition at line 213 of file driver_ins.c.

Here is the caller graph for this function:



6.5.3.7 BMI088_Soft_Reset()

```
int8_t BMI088_Soft_Reset (
            void  )
```

Redémarre les capteurs via commande logicielle.

Returns

BMI08_OK ou code d'erreur.

Redémarre les capteurs via commande logicielle.

Returns

BMI08_OK en cas de succès, ou code d'erreur SPI.

Definition at line 314 of file driver_ins.c.

6.5.3.8 BMI088_Test_Communication()

```
uint8_t BMI088_Test_Communication (
            uint8_t print_result )
```

Vérifie la présence des capteurs sur le bus SPI.

Parameters

print_result	Active l'affichage debug via printf.

Returns

1 (Succès) ou 0 (Échec).

Vérifie la présence des capteurs sur le bus SPI.

Parameters

print_result	Si non nul, affiche le résultat sur la console de debug.

Returns

1 si la communication est établie avec succès, 0 sinon.

Definition at line 291 of file driver_ins.c.

6.6 driver_ins.h

Go to the documentation of this file.

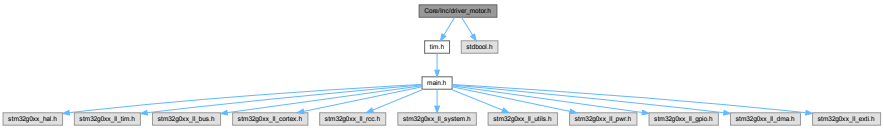
```
00001
00008 #ifndef BMI088_DRIVER_H
00009 #define BMI088_DRIVER_H
00010
00011 #include "stm32g0xx_hal.h"
00012 #include "bmi08_defs.h"
00013 #include "bmi08.h"
00014 #include "bmi08x.h"
00015
00017 #define BMI088_CS_ACC_GPIO_Port     GPIOB
00019 #define BMI088_CS_ACC_Pin           GPIO_PIN_13
00021 #define BMI088_CS_GYRO_GPIO_Port    GPIOB
00023 #define BMI088_CS_GYRO_Pin          GPIO_PIN_14
00024
00026 #define ACCEL_RANGE_3G_LSB          10922.67f
00028 #define ACCEL_RANGE_6G_LSB          5461.33f
00030 #define ACCEL_RANGE_12G_LSB         2730.67f
00032 #define ACCEL_RANGE_24G_LSB         1365.33f
00034 #define G_TO_MM_S2                  9806.65f
00035
00037 #define GYRO_RANGE_125DPS_LSB       262.4f
00039 #define GYRO_RANGE_250DPS_LSB       131.2f
00041 #define GYRO_RANGE_500DPS_LSB       65.6f
00043 #define GYRO_RANGE_1000DPS_LSB      32.768f
00045 #define GYRO_RANGE_2000DPS_LSB      16.4f
00047 #define DEG_TO_RAD                  0.017453292519943295f
00048
00052 typedef struct {
00053     GPIO_TypeDef *port;
00054     uint16_t pin;
00055 } bmi088_cs_t;
00056
00060 typedef struct {
00061     float accel_x_mms2;
00062     float accel_y_mms2;
00063     float accel_z_mms2;
00064
00065     float gyro_x_rads;
00066     float gyro_y_rads;
00067     float gyro_z_rads;
00068
00069     uint32_t timestamp_ms;
00070 } bmi088_data_t;
00071
00077 int8_t BMI088_Init(SPI_HandleTypeDef *hspi);
00078
00084 int8_t BMI088_Read_Accel_Raw(struct bmi08_sensor_data *accel_data);
00085
00091 int8_t BMI088_Read_Gyro_Raw(struct bmi08_sensor_data *gyro_data);
00092
00098 int8_t BMI088_Read_All(bmi088_data_t *data);
00099
00105 void BMI088_Convert_Accel(struct bmi08_sensor_data *accel_raw, float *accel_mms2);
00106
00112 void BMI088_Convert_Gyro(struct bmi08_sensor_data *gyro_raw, float *gyro_rads);
00113
00119 uint8_t BMI088_Test_Communication(uint8_t print_result);
00120
00125 int8_t BMI088_Soft_Reset(void);
00126
00127 #endif /* BMI088_DRIVER_H */
```

6.7 Core/Inc/driver_motor.h File Reference

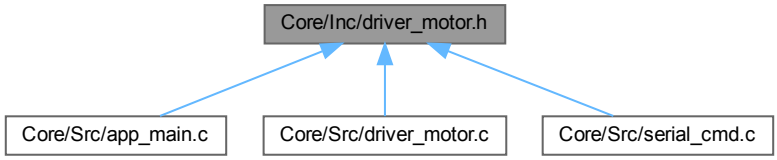
Fichier d'en-tête pour le pilote moteur (ESC).

```
#include "tim.h"
#include <stdbool.h>
```

Include dependency graph for driver_motor.h:



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



Data Structures

- struct Motor_Context_t

  Structure de contexte interne pour la gestion des transitions.
- struct Motor_Handle_t

  Handle principal de l'objet Moteur.

Enumerations

- enum MotorState_t {
  MOTOR_STATE_NEUTRAL = 0 , MOTOR_STATE_FORWARD_HOLD , MOTOR_STATE_FWD_BRAKE_TAP
  , MOTOR_STATE_FWD_NEUTRAL_GAP ,
  MOTOR_STATE_REVERSE_HOLD , MOTOR_STATE_REV_BRAKE_TAP , MOTOR_STATE_REV_NEUTRAL_GAP
  , MOTOR_STATE_NEUTRAL_TO_REVERSE_TAP ,
  MOTOR_STATE_NEUTRAL_TO_REVERSE_GAP }

  Énumération des états de la machine à états du moteur.

Functions

- void motor_init (Motor_Handle_t ∗hmotor)

  Initialise le driver moteur.
- void motor_pwm_percent (Motor_Handle_t ∗hmotor, uint8_t percent)

  Applique directement un pourcentage de PWM (Bypass FSM).
- void motor_set_speed_mms (Motor_Handle_t ∗hmotor, int16_t speed_mms)

  Définit la consigne de vitesse en mm/s.
- void motor_process_1ms (Motor_Handle_t ∗hmotor, uint32_t now_ms)

  Fonction de traitement cyclique (Machine à états).
- void app_config (void)

  Configuration globale de l'application (Callback ou Init).

6.7.1 Detailed Description

Fichier d'en-tête pour le pilote moteur (ESC).

Contient les définitions des états de la machine à états (FSM), les structures de configuration et les prototypes des fonctions de contrôle.

Definition in file driver_motor.h.

6.7.2 Enumeration Type Documentation

6.7.2.1 MotorState_t

```
enum MotorState_t
```

Énumération des états de la machine à états du moteur.

Gère les transitions complexes comme le freinage actif et les pauses au neutre.

Enumerator

MOTOR_STATE_NEUTRAL	Moteur au point mort (arrêt).
MOTOR_STATE_FORWARD_HOLD	Marche avant stable.
MOTOR_STATE_FWD_BRAKE_TAP	Séquence de freinage (depuis l'avant).
MOTOR_STATE_FWD_NEUTRAL_GAP	Pause au neutre après freinage avant.
MOTOR_STATE_REVERSE_HOLD	Marche arrière stable.
MOTOR_STATE_REV_BRAKE_TAP	Séquence de freinage (depuis l'arrière).
MOTOR_STATE_REV_NEUTRAL_GAP	Pause au neutre après freinage arrière.
MOTOR_STATE_NEUTRAL_TO_REVERSE_TAP	Coup de frein pour enclencher la marche arrière (spécifique ESC).
MOTOR_STATE_NEUTRAL_TO_REVERSE_GAP	Pause avant d'enclencher la marche arrière.

Definition at line 18 of file driver_motor.h.

6.7.3 Function Documentation

6.7.3.1 app_config()

```
void app_config (
            void  )
```

Configuration globale de l'application (Callback ou Init).

Configuration globale de l'application (Callback ou Init).

Configure l'ensemble de l'application (Hardware + Drivers).

Configure les Timers, active les interruptions nécessaires, initialise les drivers (Série, BMI088, Servo, Moteur, Speedo) et cale les horloges.

Definition at line 193 of file app_main.c.

6.7.3.2 motor_init()

```
void motor_init (
            Motor_Handle_t * hmotor )
```

Initialise le driver moteur.

Parameters

hmotor	Pointeur vers la structure de configuration.

Initialise le driver moteur.

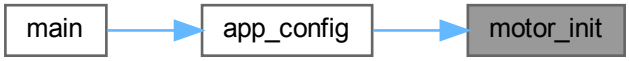
Place le moteur au neutre, réinitialise le contexte de commande et démarre la génération PWM hardware.

Parameters

hmotor	Pointeur vers le handle du moteur à initialiser.

Definition at line 88 of file driver_motor.c.

Here is the caller graph for this function:



6.7.3.3 motor_process_1ms()

```
void motor_process_1ms (
            Motor_Handle_t * hmotor,
            uint32_t now_ms )
```

Fonction de traitement cyclique (Machine à états).

Parameters

hmotor	Pointeur vers le handle moteur.
now_ms	Timestamp actuel en ms.

Fonction de traitement cyclique (Machine à états).

Doit être appelée périodiquement (ex: 1kHz). Gère la logique séquentielle :

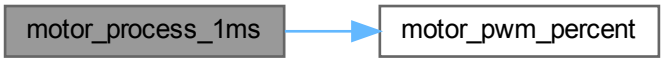
- Marche Avant -> Neutre -> Marche Arrière : Passage direct (géré par ESC en général) ou via frein.
- Inversion brusque : Application d'une séquence Frein -> Pause -> Nouveau sens.

Parameters

hmotor	Pointeur vers le handle du moteur.
now_ms	Temps système actuel en millisecondes.

Definition at line 140 of file driver_motor.c.

Here is the call graph for this function:



6.7.3.4 motor_pwm_percent()

```
void motor_pwm_percent (
            Motor_Handle_t * hmotor,
            uint8_t percent )
```

Applique directement un pourcentage de PWM (Bypass FSM).

Parameters

hmotor	Pointeur vers le handle moteur.
percent	Pourcentage du cycle (0 à 100).

Applique directement un pourcentage de PWM (Bypass FSM).

Note

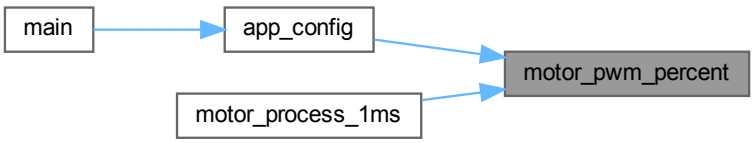
Utilisé principalement par la machine à états pour appliquer les commandes de freinage ou de neutre.

Parameters

hmotor	Pointeur vers le handle du moteur.
percent	Pourcentage PWM cible (0-100).

Definition at line 108 of file driver_motor.c.

Here is the caller graph for this function:



6.7.3.5 motor_set_speed_mms()

```
void motor_set_speed_mms (
            Motor_Handle_t * hmotor,
            int16_t speed_mms )
```

Définit la consigne de vitesse en mm/s.

Parameters

hmotor	Pointeur vers le handle moteur.
speed_mms	Vitesse souhaitée (Positive=Avant, Négative=Arrière).

Définit la consigne de vitesse en mm/s.

Met à jour le contexte cible. La machine à états (motor_process_1ms) se chargera d'atteindre cette cible en respectant les séquences de freinage.

Parameters

hmotor	Pointeur vers le handle du moteur.
speed_mms	Vitesse cible en mm/s (Positif = Avant, Négatif = Arrière).

Definition at line 120 of file driver_motor.c.

6.8 driver_motor.h

Go to the documentation of this file.

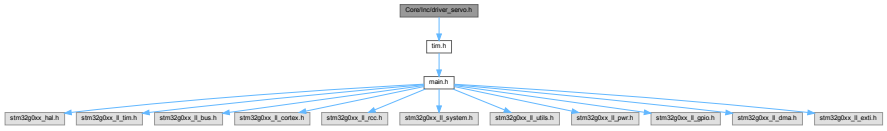
```
00001
00008 #ifndef INC_DRIVER_MOTOR_H_
00009 #define INC_DRIVER_MOTOR_H_
00010
00011 #include "tim.h"
00012 #include <stdbool.h>
00013
00018 typedef enum{
00019     MOTOR_STATE_NEUTRAL = 0,
00020     MOTOR_STATE_FORWARD_HOLD,
00021     MOTOR_STATE_FWD_BRAKE_TAP,
```

```
00022     MOTOR_STATE_FWD_NEUTRAL_GAP,
00023     MOTOR_STATE_REVERSE_HOLD,
00024     MOTOR_STATE_REV_BRAKE_TAP,
00025     MOTOR_STATE_REV_NEUTRAL_GAP,
00026     MOTOR_STATE_NEUTRAL_TO_REVERSE_TAP,
00027     MOTOR_STATE_NEUTRAL_TO_REVERSE_GAP
00028 } MotorState_t;
00029
00033 typedef struct{
00034     int16_t  target_speed_mms;
00035     uint8_t  target_pwm;
00036     bool     target_forward;
00037     uint32_t deadline_ms;
00038 } Motor_Context_t;
00039
00044 typedef struct{
00045     TIM_HandleTypeDef *htim;
00046     uint32_t channel;
00047     uint16_t min_pulse_ticks;
00048     uint16_t max_pulse_ticks;
00049
00050     int16_t  max_speed_pos_mms;
00051     int16_t  max_speed_neg_mms;
00052
00053     MotorState_t    state;
00054     bool            go_forward;
00055     Motor_Context_t ctx;
00056 } Motor_Handle_t;
00057
00062 void motor_init(Motor_Handle_t *hmotor);
00063
00069 void motor_pwm_percent(Motor_Handle_t *hmotor, uint8_t percent);
00070
00076 void motor_set_speed_mms(Motor_Handle_t *hmotor, int16_t speed_mms);
00077
00083 void motor_process_1ms(Motor_Handle_t *hmotor, uint32_t now_ms);
00084
00088 void app_config(void);
00089
00090 #endif
```

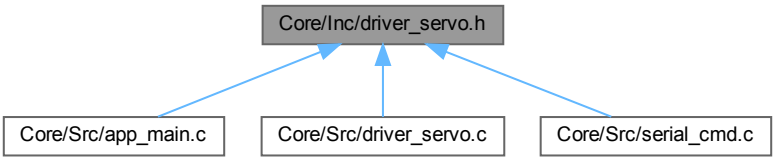
6.9 Core/Inc/driver_servo.h File Reference

Fichier d'en-tête pour le pilote du servomoteur.

`#include "tim.h"`
Include dependency graph for driver_servo.h:



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



Data Structures

- struct Servo_Handle_t

  Structure de configuration et de gestion du Servo.

Functions

- void servo_initialisation (Servo_Handle_t *hservo)

  Initialise le servo-moteur.
- void servo_pwm_percent (Servo_Handle_t *hservo, uint8_t percent)

  Commande le servo en pourcentage (0-100%).
- void servo_pwm_angle_degree (Servo_Handle_t *hservo, int8_t angle)

  Commande le servo en degrés.
- void servo_pwm_angle_abs_value (Servo_Handle_t *hservo, uint16_t abs_value)

  Commande le servo via une valeur absolue (Haute résolution).
- void app_test_config (void)

  Configure l'application de test (Fonction Debug).
- void app_test_loop (void)

  Boucle principale de l'application de test (Fonction Debug).

6.9.1 Detailed Description

Fichier d'en-tête pour le pilote du servomoteur.

Définit la structure de configuration (Handle) et les prototypes des fonctions de pilotage (Angle, Pourcentage, Valeur absolue).

Definition in file driver_servo.h.

6.9.2 Function Documentation

6.9.2.1 app_test_config()

```
void app_test_config (
            void  )
```

Configure l'application de test (Fonction Debug).

6.9.2.2 app_test_loop()

```
void app_test_loop (
            void  )
```

Boucle principale de l'application de test (Fonction Debug).

6.9.2.3 servo_initialisation()

```
void servo_initialisation (
            Servo_Handle_t * hservo )
```

Initialise le servo-moteur.

Parameters

hservo	Pointeur vers le handle du servo.

Initialise le servo-moteur.

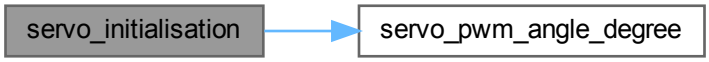
Positionne le servo à 0 degrés (neutre) et active le canal PWM.

Parameters

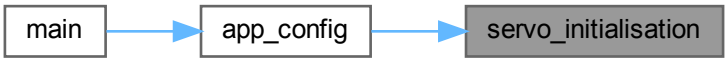
hservo	Pointeur vers le handle du servo.

Definition at line 122 of file driver_servo.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.9.2.4 servo_pwm_angle_abs_value()

```
void servo_pwm_angle_abs_value (
            Servo_Handle_t * hservo,
            uint16_t abs_value )
```

Commande le servo via une valeur absolue (Haute résolution).

Parameters

hservo	Pointeur vers le handle du servo.
abs_value	Valeur absolue normalisée (0 à 65535).

Commande le servo via une valeur absolue (Haute résolution).

Effectue une double conversion : Entrée -> Centi-degrés -> Ticks PWM. Gère également l'offset (Trim) et les limites physiques du timer.

Parameters

hservo	Pointeur vers le handle du servo.
abs_value	Valeur absolue normalisée (0 à 65535).

Definition at line 97 of file driver_servo.c.

6.9.2.5 servo_pwm_angle_degree()

```
void servo_pwm_angle_degree (
            Servo_Handle_t * hservo,
            int8_t angle )
```

Commande le servo en degrés.

Parameters

hservo	Pointeur vers le handle du servo.
angle	Angle cible en degrés (sera borné par le driver).

Commande le servo en degrés.

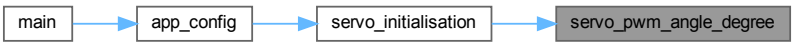
Applique un bornage de sécurité (CLAMP_MIN / CLAMP_MAX) puis convertit l'angle en pourcentage pour le PWM.

Parameters

hservo	Pointeur vers le handle du servo.
angle	Angle cible en degrés.

Definition at line 80 of file driver_servo.c.

Here is the caller graph for this function:



6.9.2.6 servo_pwm_percent()

```
void servo_pwm_percent (
            Servo_Handle_t * hservo,
            uint8_t percent )
```

Commande le servo en pourcentage (0-100%).

Parameters

hservo	Pointeur vers le handle du servo.
percent	Position cible en pourcentage.

Commande le servo en pourcentage (0-100%).

Parameters

hservo	Pointeur vers le handle du servo.
percent	Position cible en pourcentage.

Definition at line 68 of file driver_servo.c.

6.10 driver_servo.h

Go to the documentation of this file.

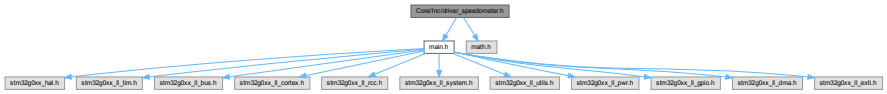
```
00001
00008 #ifndef INC_DRIVER_SERVO_H_
00009 #define INC_DRIVER_SERVO_H_
00010
00011 #include "tim.h"
00012
00018 typedef struct {
00019     TIM_HandleTypeDef *htim;
00020     uint32_t channel;
00021     uint16_t min_pulse_ticks;
00022     uint16_t max_pulse_ticks;
00023 } Servo_Handle_t;
00024
00029 void servo_initialisation(Servo_Handle_t *hservo);
00030
00036 void servo_pwm_percent(Servo_Handle_t *hservo, uint8_t percent);
00037
00043 void servo_pwm_angle_degree(Servo_Handle_t *hservo, int8_t angle);
00044
00050 void servo_pwm_angle_abs_value(Servo_Handle_t *hservo, uint16_t abs_value);
00051
00055 void app_test_config(void);
00056
00060 void app_test_loop(void);
00061
00062 #endif
```

6.11 Core/Inc/driver_speedometer.h File Reference

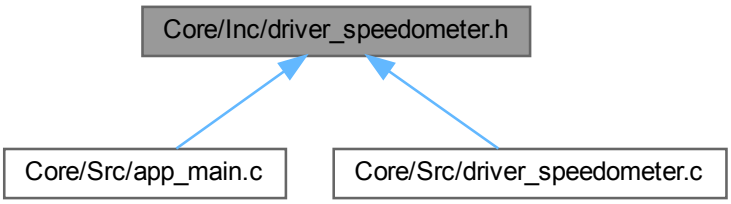
Fichier d'en-tête pour le capteur de vitesse (Tachymètre).

```
#include "main.h"
#include <math.h>
```

Include dependency graph for driver_speedometer.h:



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



Data Structures

- struct Speedometer_Handle_t

  Structure de gestion du tachymètre.

Macros

- #define WHEEL_DIAMETER_MM 68.0f

  Diamètre de la roue du véhicule en millimètres.
- #define NB_TOURS_TEST 10.0f

  Nombre de tours de roue effectués lors de la calibration.
- #define VALEUR_COMPTEUR_LUE 52.0f

  Nombre de ticks capteur relevés pour le nombre de tours de test.
- #define TICKS_PER_WHEEL_TURN (VALEUR_COMPTEUR_LUE / NB_TOURS_TEST)

  Nombre de ticks capteur correspondant à un tour complet de roue.
- #define PERIMETER_M ((WHEEL_DIAMETER_MM ∗ 3.14159f) / 1000.0f)

  Périmètre de la roue en mètres (Distance pour un tour).

Functions

- void speedometer_init (Speedometer_Handle_t ∗hSpeedo, TIM_HandleTypeDef ∗htim)

  Initialise l'objet tachymètre.
- float speedometer_solve_speed (Speedometer_Handle_t ∗hSpeedo)

  Calcule la vitesse actuelle basée sur les impulsions reçues.

6.11.1 Detailed Description

Fichier d'en-tête pour le capteur de vitesse (Tachymètre).

Contient les paramètres physiques du véhicule (diamètre roue), les constantes de calibration du capteur (ticks par tour) et la définition de la structure objet.

Definition in file driver_speedometer.h.

6.11.2 Macro Definition Documentation

6.11.2.1 NB_TOURS_TEST

```
#define NB_TOURS_TEST 10.0f
```

Nombre de tours de roue effectués lors de la calibration.

Definition at line 19 of file driver_speedometer.h.

6.11.2.2 PERIMETER_M

```
#define PERIMETER_M ((WHEEL_DIAMETER_MM * 3.14159f) / 1000.0f)
```

Périmètre de la roue en mètres (Distance pour un tour).

Definition at line 28 of file driver_speedometer.h.

6.11.2.3 TICKS_PER_WHEEL_TURN

```
#define TICKS_PER_WHEEL_TURN (VALEUR_COMPTEUR_LUE / NB_TOURS_TEST)
```

Nombre de ticks capteur correspondant à un tour complet de roue.

Definition at line 25 of file driver_speedometer.h.

6.11.2.4 VALEUR_COMPTEUR_LUE

```
#define VALEUR_COMPTEUR_LUE 52.0f
```

Nombre de ticks capteur relevés pour le nombre de tours de test.

Definition at line 22 of file driver_speedometer.h.

6.11.2.5 WHEEL_DIAMETER_MM

```
#define WHEEL_DIAMETER_MM 68.0f
```

Diamètre de la roue du véhicule en millimètres.

Definition at line 16 of file driver_speedometer.h.

6.11.3 Function Documentation

6.11.3.1 speedometer_init()

```
void speedometer_init (
            Speedometer_Handle_t * hSpeedo,
            TIM_HandleTypeDef * htim )
```

Initialise l'objet tachymètre.

Parameters

hSpeedo	Pointeur vers la structure à initialiser.
htim	Pointeur vers le handle du Timer matériel.

Initialise l'objet tachymètre.

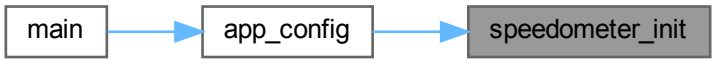
Associe le timer matériel à la structure, initialise les variables d'état (temps et compteur) et démarre le timer.

Parameters

hSpeedo	Pointeur vers la structure de gestion à initialiser.
htim	Pointeur vers le handle du Timer STM32 (HAL) utilisé.

Definition at line 48 of file driver_speedometer.c.

Here is the caller graph for this function:



6.11.3.2 speedometer_solve_speed()

```
float speedometer_solve_speed (
            Speedometer_Handle_t * hSpeedo )
```

Calcule la vitesse actuelle basée sur les impulsions reçues.

Parameters

hSpeedo	Pointeur vers la structure du tachymètre.

Returns

Vitesse en m/s.

Calcule la vitesse actuelle basée sur les impulsions reçues.

Cette fonction doit être appelée périodiquement. Elle calcule la différence de temps et de nombre d'impulsions (ticks) depuis le dernier appel.

Note

Gère implicitement le débordement (overflow) du compteur 16 bits via l'arithmétique non signée.

Parameters

hSpeedo	Pointeur vers la structure de gestion du tachymètre.

Returns

Vitesse calculée en mètres par seconde (m/s).

Definition at line 20 of file driver_speedometer.c.

6.12 driver_speedometer.h

Go to the documentation of this file.

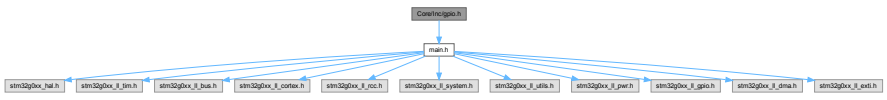
```
00001
00009 #ifndef DRIVER_SPEEDOMETER_H
00010 #define DRIVER_SPEEDOMETER_H
00011
00012 #include "main.h"
00013 #include <math.h>
00014
00016 #define WHEEL_DIAMETER_MM   68.0f
00017
00019 #define NB_TOURS_TEST       10.0f
00020
00022 #define VALEUR_COMPTEUR_LUE 52.0f
00023
00025 #define TICKS_PER_WHEEL_TURN    (VALEUR_COMPTEUR_LUE / NB_TOURS_TEST)
00026
00028 #define PERIMETER_M             ((WHEEL_DIAMETER_MM * 3.14159f) / 1000.0f)
00029
00035 typedef struct{
00036     TIM_HandleTypeDef *htim;
00037     uint16_t last_counter_val;
00038     uint32_t last_process_time;
00039     float current_speed_ms;
00040 } Speedometer_Handle_t;
00041
00047 void speedometer_init(Speedometer_Handle_t *hSpeedo, TIM_HandleTypeDef *htim);
00048
00054 float speedometer_solve_speed(Speedometer_Handle_t *hSpeedo);
00055
00056 #endif
```

6.13 Core/Inc/gpio.h File Reference

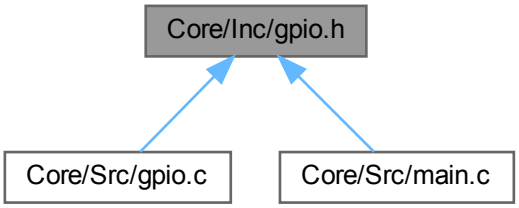
This file contains all the function prototypes for the gpio.c file.

```
#include "main.h"
```

Include dependency graph for gpio.h:



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



Functions

- void MX_GPIO_Init (void)

6.13.1 Detailed Description

This file contains all the function prototypes for the gpio.c file.

Attention

Definition in file gpio.h.

6.13.2 Function Documentation

6.13.2.1 MX_GPIO_Init()

```
void MX_GPIO_Init (
             void  )
```

Configure pins as Analog Input Output EVENT_OUT EXTI Free pins are configured automatically as Analog (this feature is enabled through the Code Generation settings)

Definition at line 44 of file gpio.c.

Here is the caller graph for this function:

6.14 gpio.h

Go to the documentation of this file.

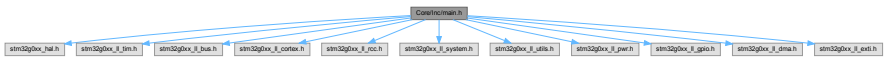
```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020 /* Define to prevent recursive inclusion -------------------------------------*/
00021 #ifndef __GPIO_H__
00022 #define __GPIO_H__
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif
00027
00028 /* Includes ------------------------------------------------------------------*/
00029 #include "main.h"
00030
00031 /* USER CODE BEGIN Includes */
00032
00033 /* USER CODE END Includes */
00034
00035 /* USER CODE BEGIN Private defines */
00036
00037 /* USER CODE END Private defines */
00038
00039 void MX_GPIO_Init(void);
00040
00041 /* USER CODE BEGIN Prototypes */
00042
00043 /* USER CODE END Prototypes */
00044
00045 #ifdef __cplusplus
00046 }
00047 #endif
00048 #endif /*__ GPIO_H__ */
00049
```

6.15 Core/Inc/main.h File Reference

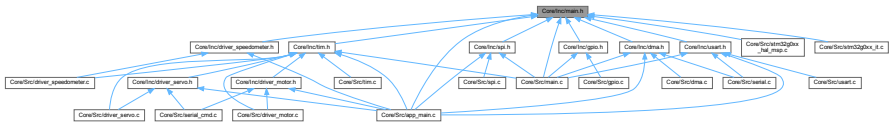
: Header for main.c file. This file contains the common defines of the application.

```
#include "stm32g0xx_hal.h"
#include "stm32g0xx_ll_tim.h"
#include "stm32g0xx_ll_bus.h"
#include "stm32g0xx_ll_cortex.h"
#include "stm32g0xx_ll_rcc.h"
#include "stm32g0xx_ll_system.h"
#include "stm32g0xx_ll_utils.h"
#include "stm32g0xx_ll_pwr.h"
#include "stm32g0xx_ll_gpio.h"
#include "stm32g0xx_ll_dma.h"
#include "stm32g0xx_ll_exti.h"
```

Include dependency graph for main.h:



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



Macros

- #define B1_Pin GPIO_PIN_13
- #define B1_GPIO_Port GPIOC
- #define MCO_Pin GPIO_PIN_0
- #define MCO_GPIO_Port GPIOF
- #define PWM_motor_Pin GPIO_PIN_0
- #define PWM_motor_GPIO_Port GPIOA
- #define USART2_TX_Pin GPIO_PIN_2
- #define USART2_TX_GPIO_Port GPIOA
- #define USART2_RX_Pin GPIO_PIN_3
- #define USART2_RX_GPIO_Port GPIOA
- #define LED_GREEN_Pin GPIO_PIN_5
- #define LED_GREEN_GPIO_Port GPIOA
- #define ACC_CS_Pin GPIO_PIN_13
- #define ACC_CS_GPIO_Port GPIOB
- #define GYR_CS_Pin GPIO_PIN_14
- #define GYR_CS_GPIO_Port GPIOB
- #define PWM_servo_Pin GPIO_PIN_8
- #define PWM_servo_GPIO_Port GPIOA
- #define TMS_Pin GPIO_PIN_13
- #define TMS_GPIO_Port GPIOA
- #define TCK_Pin GPIO_PIN_14
- #define TCK_GPIO_Port GPIOA

Functions

- void Error_Handler (void)

  This function is executed in case of error occurrence.

6.15.1 Detailed Description

: Header for main.c file. This file contains the common defines of the application.

Attention

Definition in file main.h.

6.15.2 Macro Definition Documentation

6.15.2.1 ACC_CS_GPIO_Port

```
#define ACC_CS_GPIO_Port GPIOB
```

Definition at line 85 of file main.h.

6.15.2.2 ACC_CS_Pin

```
#define ACC_CS_Pin GPIO_PIN_13
```

Definition at line 84 of file main.h.

6.15.2.3 B1_GPIO_Port

```
#define B1_GPIO_Port GPIOC
```

Definition at line 73 of file main.h.

6.15.2.4 B1_Pin

```
#define B1_Pin GPIO_PIN_13
```

Definition at line 72 of file main.h.

6.15.2.5 GYR_CS_GPIO_Port

```
#define GYR_CS_GPIO_Port GPIOB
```

Definition at line 87 of file main.h.

6.15.2.6 GYR_CS_Pin

```
#define GYR_CS_Pin GPIO_PIN_14
```

Definition at line 86 of file main.h.

6.15.2.7 LED_GREEN_GPIO_Port

```
#define LED_GREEN_GPIO_Port GPIOA
```

Definition at line 83 of file main.h.

6.15.2.8 LED_GREEN_Pin

```
#define LED_GREEN_Pin GPIO_PIN_5
```

Definition at line 82 of file main.h.

6.15.2.9 MCO_GPIO_Port

```
#define MCO_GPIO_Port GPIOF
```

Definition at line 75 of file main.h.

6.15.2.10 MCO_Pin

```
#define MCO_Pin GPIO_PIN_0
```

Definition at line 74 of file main.h.

6.15.2.11 PWM_motor_GPIO_Port

```
#define PWM_motor_GPIO_Port GPIOA
```

Definition at line 77 of file main.h.

6.15.2.12 PWM_motor_Pin

```
#define PWM_motor_Pin GPIO_PIN_0
```

Definition at line 76 of file main.h.

6.15.2.13 PWM_servo_GPIO_Port

```
#define PWM_servo_GPIO_Port GPIOA
```

Definition at line 89 of file main.h.

6.15.2.14 PWM_servo_Pin

```
#define PWM_servo_Pin GPIO_PIN_8
```

Definition at line 88 of file main.h.

6.15.2.15 TCK_GPIO_Port

```
#define TCK_GPIO_Port GPIOA
```

Definition at line 93 of file main.h.

6.15.2.16 TCK_Pin

```
#define TCK_Pin GPIO_PIN_14
```

Definition at line 92 of file main.h.

6.15.2.17 TMS_GPIO_Port

```
#define TMS_GPIO_Port GPIOA
```

Definition at line 91 of file main.h.

6.15.2.18 TMS_Pin

```
#define TMS_Pin GPIO_PIN_13
```

Definition at line 90 of file main.h.

6.15.2.19 USART2_RX_GPIO_Port

```
#define USART2_RX_GPIO_Port GPIOA
```

Definition at line 81 of file main.h.

6.15.2.20 USART2_RX_Pin

```
#define USART2_RX_Pin GPIO_PIN_3
```

Definition at line 80 of file main.h.

6.15.2.21 USART2_TX_GPIO_Port

```
#define USART2_TX_GPIO_Port GPIOA
```

Definition at line 79 of file main.h.

6.15.2.22 USART2_TX_Pin

```
#define USART2_TX_Pin GPIO_PIN_2
```

Definition at line 78 of file main.h.

6.15.3 Function Documentation

6.15.3.1 Error_Handler()

```
void Error_Handler (
            void  )
```

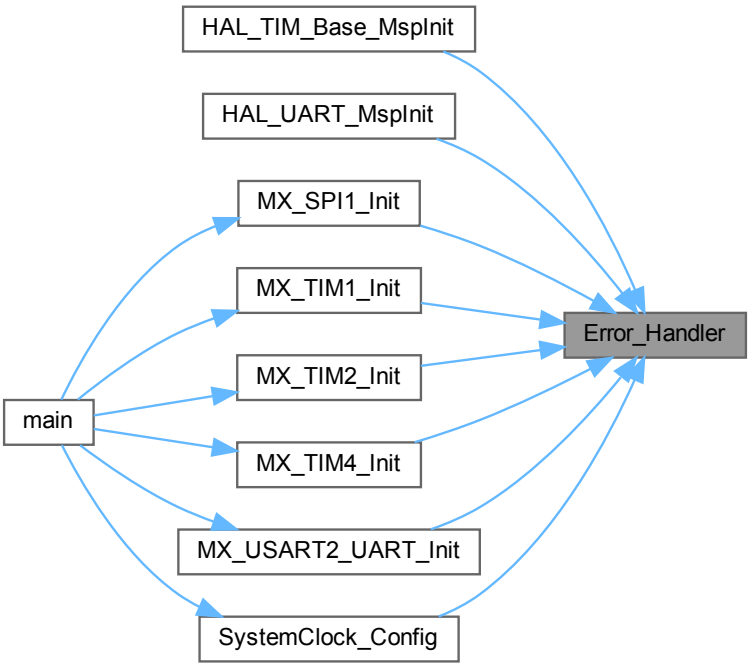
This function is executed in case of error occurrence.

Return values

None	

Definition at line 173 of file main.c.

Here is the caller graph for this function:



6.16 main.h

Go to the documentation of this file.

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020
00021 /* Define to prevent recursive inclusion -------------------------------------*/
00022 #ifndef __MAIN_H
00023 #define __MAIN_H
00024
00025 #ifdef __cplusplus
00026 extern "C" {
00027 #endif
00028
00029 /* Includes ------------------------------------------------------------------*/
00030 #include "stm32g0xx_hal.h"
00031
00032 #include "stm32g0xx_ll_tim.h"
00033 #include "stm32g0xx_ll_bus.h"
00034 #include "stm32g0xx_ll_cortex.h"
00035 #include "stm32g0xx_ll_rcc.h"
00036 #include "stm32g0xx_ll_system.h"
00037 #include "stm32g0xx_ll_utils.h"
00038 #include "stm32g0xx_ll_pwr.h"
00039 #include "stm32g0xx_ll_gpio.h"
00040 #include "stm32g0xx_ll_dma.h"
00041
00042 #include "stm32g0xx_ll_exti.h"
00043
00044 /* Private includes ----------------------------------------------------------*/
00045 /* USER CODE BEGIN Includes */
00046
00047 /* USER CODE END Includes */
00048
00049 /* Exported types ------------------------------------------------------------*/
00050 /* USER CODE BEGIN ET */
```

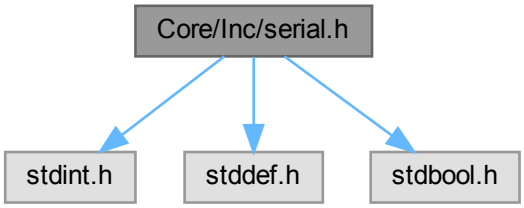
```
00051
00052 /* USER CODE END ET */
00053
00054 /* Exported constants --------------------------------------------------------*/
00055 /* USER CODE BEGIN EC */
00056
00057 /* USER CODE END EC */
00058
00059 /* Exported macro ------------------------------------------------------------*/
00060 /* USER CODE BEGIN EM */
00061
00062 /* USER CODE END EM */
00063
00064 /* Exported functions prototypes ---------------------------------------------*/
00065 void Error_Handler(void);
00066
00067 /* USER CODE BEGIN EFP */
00068
00069 /* USER CODE END EFP */
00070
00071 /* Private defines -----------------------------------------------------------*/
00072 #define B1_Pin GPIO_PIN_13
00073 #define B1_GPIO_Port GPIOC
00074 #define MCO_Pin GPIO_PIN_0
00075 #define MCO_GPIO_Port GPIOF
00076 #define PWM_motor_Pin GPIO_PIN_0
00077 #define PWM_motor_GPIO_Port GPIOA
00078 #define USART2_TX_Pin GPIO_PIN_2
00079 #define USART2_TX_GPIO_Port GPIOA
00080 #define USART2_RX_Pin GPIO_PIN_3
00081 #define USART2_RX_GPIO_Port GPIOA
00082 #define LED_GREEN_Pin GPIO_PIN_5
00083 #define LED_GREEN_GPIO_Port GPIOA
00084 #define ACC_CS_Pin GPIO_PIN_13
00085 #define ACC_CS_GPIO_Port GPIOB
00086 #define GYR_CS_Pin GPIO_PIN_14
00087 #define GYR_CS_GPIO_Port GPIOB
00088 #define PWM_servo_Pin GPIO_PIN_8
00089 #define PWM_servo_GPIO_Port GPIOA
00090 #define TMS_Pin GPIO_PIN_13
00091 #define TMS_GPIO_Port GPIOA
00092 #define TCK_Pin GPIO_PIN_14
00093 #define TCK_GPIO_Port GPIOA
00094
00095 /* USER CODE BEGIN Private defines */
00096
00097 /* USER CODE END Private defines */
00098
00099 #ifdef __cplusplus
00100 }
00101 #endif
00102
00103 #endif /* __MAIN_H */
```

6.17 Core/Inc/serial.h File Reference

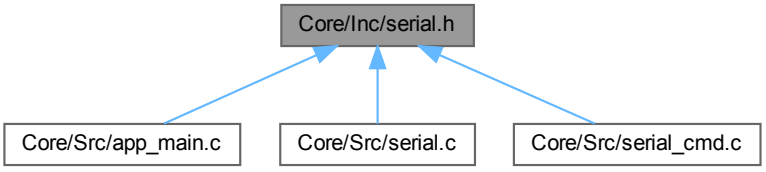
Interface du driver Série (UART + DMA) et définitions du protocole.

```
#include <stdint.h>
#include <stddef.h>
#include <stdbool.h>
```

Include dependency graph for serial.h:



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



Macros

- #define SERIAL_RX_RING_SIZE 1024u

  Taille du buffer circulaire logiciel de réception (Doit être une puissance de 2).
- #define SERIAL_RX_CHUNK_SIZE 256u

  Taille du buffer linéaire DMA pour la réception (Double buffer partiel).
- #define SERIAL_TX_CHUNK_MAX 255u

  Taille maximale d'un transfert DMA unique en émission.
- #define SERIAL_UART huart2

  Instance UART HAL utilisée (liaison avec usart.h).
- #define PROTO_HDR_RW_MASK 0x80u

  Masque pour extraire le bit R/W de l'entête.
- #define PROTO_HDR_ADDR_MASK 0x7Fu

  Masque pour extraire l'adresse du registre (0-127).
- #define PROTO_MAKE_HDR(rw, addr) ((uint8_t)(((rw) ? 0x80u : 0x00u) | ((addr) & 0x7Fu)))

  Macro de construction de l'octet d'entête (Header).
- #define PROTO_IS_READ(hdr) (((hdr) & 0x80u) != 0)

  Macro vérifiant si l'entête correspond à une lecture.
- #define PROTO_ADDR(hdr) ((hdr) & 0x7Fu)

  Macro extrayant l'adresse du registre depuis l'entête.

Functions

- void serial_init (void)

  Initialise la couche série (DMA + Buffers).
- int serial_write_nb (const uint8_t *data, uint16_t len)

  Écriture non-bloquante partielle.
- int serial_write_all_nb (const uint8_t *data, uint16_t len)

  Écriture non-bloquante atomique (Tout ou rien).
- int serial_write (const uint8_t *data, uint16_t len)

  Écriture bloquante (Wrapper).
- size_t serial_available (void)

  Retourne le nombre d'octets disponibles dans le buffer RX.
- size_t serial_read (uint8_t *dst, size_t max_len)

  Lit des données depuis le buffer RX.
- size_t serial_read_until (uint8_t *dst, size_t max_len, uint8_t delim)

  Lit des données jusqu'à un délimiteur spécifique.
- uint8_t serial_crc8_atm (const uint8_t *data, uint16_t len)

  Calcule le CRC8 (Polynôme ATM 0x07).
- int proto_send_write16 (uint8_t addr, int16_t value)

  Envoie une commande d'écriture 16 bits (Frame Write).
- int proto_send_read_burst (uint8_t addr, uint8_t count, uint8_t flags)

  Envoie une requête de lecture (Frame Read Request).

6.17.1 Detailed Description

Interface du driver Série (UART + DMA) et définitions du protocole.

Définit les tailles de buffers, les macros de manipulation du protocole binaire 4 octets et les prototypes des fonctions d'E/S.

Definition in file serial.h.

6.17.2 Macro Definition Documentation

6.17.2.1 PROTO_ADDR

```
#define PROTO_ADDR(
              hdr ) ( (hdr) & 0x7Fu )
```

Macro extrayant l'adresse du registre depuis l'entête.

Definition at line 47 of file serial.h.

6.17.2.2 PROTO_HDR_ADDR_MASK

```
#define PROTO_HDR_ADDR_MASK 0x7Fu
```

Masque pour extraire l'adresse du registre (0-127).

Definition at line 38 of file serial.h.

6.17.2.3 PROTO_HDR_RW_MASK

```
#define PROTO_HDR_RW_MASK 0x80u
```

Masque pour extraire le bit R/W de l'entête.

Definition at line 35 of file serial.h.

6.17.2.4 PROTO_IS_READ

```
#define PROTO_IS_READ(
            hdr ) ( ((hdr) & 0x80u) != 0 )
```

Macro vérifiant si l'entête correspond à une lecture.

Definition at line 44 of file serial.h.

6.17.2.5 PROTO_MAKE_HDR

```
#define PROTO_MAKE_HDR(
            rw,
            addr ) ( (uint8_t)(((rw) ?  0x80u :  0x00u) | ((addr) & 0x7Fu)) )
```

Macro de construction de l'octet d'entête (Header).

Definition at line 41 of file serial.h.

6.17.2.6 SERIAL_RX_CHUNK_SIZE

```
#define SERIAL_RX_CHUNK_SIZE 256u
```

Taille du buffer linéaire DMA pour la réception (Double buffer partiel).

Definition at line 22 of file serial.h.

6.17.2.7 SERIAL_RX_RING_SIZE

```
#define SERIAL_RX_RING_SIZE 1024u
```

Taille du buffer circulaire logiciel de réception (Doit être une puissance de 2).

Definition at line 19 of file serial.h.

6.17.2.8 SERIAL_TX_CHUNK_MAX

```
#define SERIAL_TX_CHUNK_MAX 255u
```

Taille maximale d'un transfert DMA unique en émission.

Definition at line 25 of file serial.h.

6.17.2.9 SERIAL_UART

```
#define SERIAL_UART huart2
```

Instance UART HAL utilisée (liaison avec usart.h).

Definition at line 28 of file serial.h.

6.17.3 Function Documentation

6.17.3.1 proto_send_read_burst()

```
int proto_send_read_burst (
            uint8_t addr,
            uint8_t count,
            uint8_t flags )
```

Envoie une requête de lecture (Frame Read Request).

Trame : [HDR(1,addr) | COUNT | FLAGS | CRC]

Parameters

addr	Adresse de départ.
count	Nombre d'octets/registres à lire.
flags	Flags spécifiques.

Returns

Résultat de l'envoi série.

Envoie une requête de lecture (Frame Read Request).

Parameters

addr	Adresse de départ.
count	Nombre de registres à lire.
flags	Flags optionnels.

Returns

Résultat de l'envoi.

Definition at line 372 of file serial.c.

6.17.3.2 proto_send_write16()

```
int proto_send_write16 (
            uint8_t addr,
            int16_t value )
```

Envoie une commande d'écriture 16 bits (Frame Write).

Trame : [HDR(0,addr) | VAL_LO | VAL_HI | CRC]

Parameters

addr	Adresse du registre.
value	Valeur à écrire.

Returns

Résultat de l'envoi série.

Envoie une commande d'écriture 16 bits (Frame Write).

Parameters

addr	Adresse du registre virtuel.
value	Valeur 16 bits à écrire.

Returns

Résultat de l'envoi.

Definition at line 359 of file serial.c.

6.17.3.3 serial_available()

```
size_t serial_available (
            void  )
```

Retourne le nombre d'octets disponibles dans le buffer RX.

Retourne le nombre d'octets disponibles dans le buffer RX.

Returns

Nombre d'octets dans le buffer RX.

Definition at line 204 of file serial.c.

6.17.3.4 serial_crc8_atm()

```
uint8_t serial_crc8_atm (
            const uint8_t * data,
            uint16_t len )
```

Calcule le CRC8 (Polynôme ATM 0x07).

Parameters

data	Pointeur vers les données.
len	Longueur.

Returns

Checksum calculé.

Calcule le CRC8 (Polynôme ATM 0x07).

Parameters

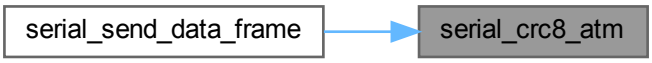
data	Pointeur vers les données.
len	Longueur des données.

Returns

Valeur du CRC calculé.

Definition at line 326 of file serial.c.

Here is the caller graph for this function:



6.17.3.5 serial_init()

```
void serial_init (
             void  )
```

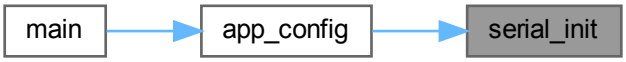
Initialise la couche série (DMA + Buffers).

Initialise la couche série (DMA + Buffers).

Lance la réception DMA en mode "ReceiveToIdle" pour détecter les fins de trames sans attendre que le buffer soit plein.

Definition at line 122 of file serial.c.

Here is the caller graph for this function:



6.17.3.6 serial_read()

```
size_t serial_read (
            uint8_t * dst,
            size_t max_len )
```

Lit des données depuis le buffer RX.

Parameters

dst	Buffer de destination.
max_len	Nombre max d'octets à lire.

Returns

Nombre d'octets lus.

Copie les données du buffer circulaire interne vers le buffer utilisateur. Gère le cas où les données sont à cheval sur la fin du buffer (wrap).

Parameters

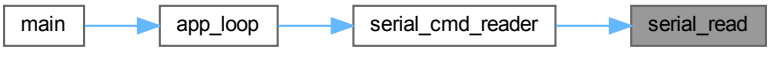
dst	Buffer de destination.
max_len	Nombre maximum d'octets à lire.

Returns

Nombre d'octets réellement lus.

Definition at line 214 of file serial.c.

Here is the caller graph for this function:



6.17.3.7 serial_read_until()

```
size_t serial_read_until (
            uint8_t * dst,
            size_t max_len,
            uint8_t delim )
```

Lit des données jusqu'à un délimiteur spécifique.

Parameters

dst	Buffer de destination.
max_len	Taille max du buffer.
delim	Caractère de fin (ex: '\n').

Returns

Nombre d'octets lus.

Lit des données jusqu'à un délimiteur spécifique.

Note

Fonction utile pour lire des lignes complètes (ex: jusqu'à '\n').

Parameters

dst	Buffer de destination.
max_len	Taille max du buffer destination.
delim	Caractère délimiteur recherché.

Returns

Nombre d'octets lus (incluant le délimiteur), ou 0 si non trouvé.

Definition at line 250 of file serial.c.

6.17.3.8 serial_write()

```
int serial_write (
            const uint8_t * data,
            uint16_t len )
```

Écriture bloquante (Wrapper).

Parameters

data	Pointeur vers les données.
len	Nombre d'octets.

Returns

0 si succès, -1 si erreur.

Écriture bloquante (Wrapper).

Note

En réalité non-bloquant ici, renvoie -1 en cas d'échec.

Parameters

data	Pointeur vers les données.
len	Longueur.

Returns

0 si succès, -1 si erreur.

Definition at line 182 of file serial.c.

Here is the call graph for this function:

6.17.3.9 serial_write_all_nb()

```
int serial_write_all_nb (
            const uint8_t * data,
            uint16_t len )
```

Écriture non-bloquante atomique (Tout ou rien).

Parameters

data	Pointeur vers les données.
len	Nombre d'octets.

Returns

len si succès, code erreur négatif si buffer plein.

Écriture non-bloquante atomique (Tout ou rien).

Parameters

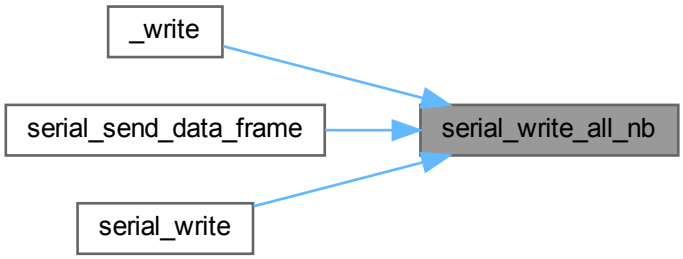
data	Pointeur vers les données.
len	Nombre d'octets.

Returns

len si succès, -EWOULDBLOCK si pas assez d'espace.

Definition at line 159 of file serial.c.

Here is the caller graph for this function:



6.17.3.10 serial_write_nb()

```
int serial_write_nb (
            const uint8_t * data,
            uint16_t len )
```

Écriture non-bloquante partielle.

Parameters

data	Pointeur vers les données.
len	Nombre d'octets souhaités.

Returns

Nombre d'octets écrits (peut être inférieur à len).

Écriture non-bloquante partielle.

Note

Copie autant de données que possible. S'arrête si le buffer est plein.

Parameters

data	Pointeur vers les données.
len	Nombre d'octets à écrire.

Returns

Nombre d'octets réellement écrits ou code d'erreur négatif.

Definition at line 134 of file serial.c.

6.18 serial.h

Go to the documentation of this file.

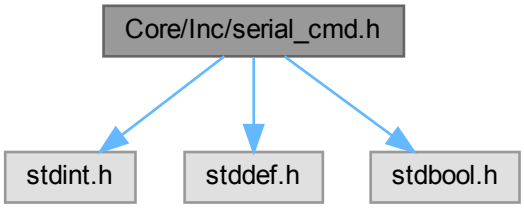
```
00001
00008 #ifndef INC_SERIAL_H_
00009 #define INC_SERIAL_H_
00010
00011 #pragma once
00012 #include <stdint.h>
00013 #include <stddef.h>
00014 #include <stdbool.h>
00015
00016 /* ---------- CONFIGURATION HARDWARE & BUFFERS ---------- */
00017
00019 #define SERIAL_RX_RING_SIZE   1024u
00020
00022 #define SERIAL_RX_CHUNK_SIZE  256u
00023
00025 #define SERIAL_TX_CHUNK_MAX   255u
00026
00028 #define SERIAL_UART           huart2
00029
00030 /* ---------- DÉFINITIONS DU PROTOCOLE (4 OCTETS) ---------- */
00031 /* Format trame : [HDR | D0 | D1 | CRC8] */
00032 /* HDR : bit7 = R(1)/W(0), bits6..0 = Adresse registre (0..127) */
00033
00035 #define PROTO_HDR_RW_MASK     0x80u
00036
00038 #define PROTO_HDR_ADDR_MASK   0x7Fu
00039
00041 #define PROTO_MAKE_HDR(rw, addr)    ( (uint8_t)(((rw) ? 0x80u : 0x00u) | ((addr) & 0x7Fu)) )
00042
00044 #define PROTO_IS_READ(hdr)          ( ((hdr) & 0x80u) != 0 )
00045
00047 #define PROTO_ADDR(hdr)             ( (hdr) & 0x7Fu )
00048
00049 /* ---------- FONCTIONS BAS NIVEAU (LINK LAYER) ---------- */
00050
00054 void     serial_init(void);
00055
00062 int      serial_write_nb(const uint8_t *data, uint16_t len);
00063
00070 int      serial_write_all_nb(const uint8_t *data, uint16_t len);
00071
00078 int      serial_write(const uint8_t *data, uint16_t len);
00079
00083 size_t   serial_available(void);
00084
00091 size_t   serial_read(uint8_t *dst, size_t max_len);
00092
00100 size_t   serial_read_until(uint8_t *dst, size_t max_len, uint8_t delim);
00101
00102 /* ---------- UTILITAIRES ---------- */
00103
00110 uint8_t  serial_crc8_atm(const uint8_t *data, uint16_t len);
00111
00112 /* ---------- FONCTIONS PROTOCOLE ---------- */
00113
00121 int      proto_send_write16(uint8_t addr, int16_t value);
00122
00131 int      proto_send_read_burst(uint8_t addr, uint8_t count, uint8_t flags);
00132
00140 static inline int proto_send_data16(uint8_t addr, int16_t value){
00141     return proto_send_write16(addr, value);
00142 }
00143
00144 #endif
```

6.19 Core/Inc/serial_cmd.h File Reference

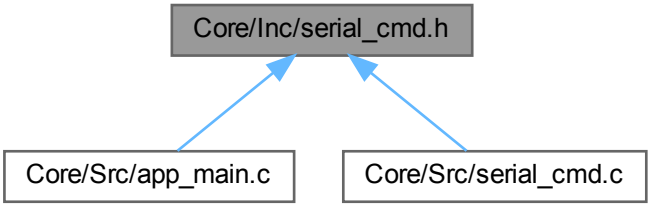
Définitions des commandes et structures du protocole applicatif.

```
#include <stdint.h>
#include <stddef.h>
#include <stdbool.h>
```

Include dependency graph for serial_cmd.h:



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



Macros

- #define REG_SERVO_CMD 0x00

  Adresse du registre virtuel pour la commande Servo (0-100%).
- #define REG_MOTOR_CMD 0x01

  Adresse du registre virtuel pour la commande Moteur (mm/s).
- #define REG_BMI 0x02

  Adresse du registre virtuel pour les commandes BMI088 (réservé).

Enumerations

- enum ParserSwitch {
  PARSER_IDLE , PARSER_SERVO_CMD , PARSER_MOTOR_CMD , PARSER_BMI_CMD ,
  PARSER_OTHERS }

  Indicateur de résultat du parsing pour la boucle principale.

Functions

- struct __attribute__ ((packed))

  Structure de la trame de télémétrie envoyée vers la Pi5.
- void serial_cmd_reader (void)

  Fonction de lecture périodique des commandes entrantes.
- void serial_send_data_frame (void)

  Capture les capteurs et envoie la trame de télémétrie.

Variables

- SerialImuFrame_t
- ParserSwitch parser_state

  État global du parseur, lu par le main pour appliquer les consignes.
- int8_t shadow_servo_cmd

  Dernière consigne reçue pour le servo (Shadow Register).
- int16_t shadow_motor_cmd

  Dernière consigne reçue pour le moteur (Shadow Register).

6.19.1 Detailed Description

Définitions des commandes et structures du protocole applicatif.

Contient les adresses des registres virtuels, la définition de la trame de télémétrie binaire et les états de notification du parseur.

Definition in file serial_cmd.h.

6.19.2 Macro Definition Documentation

6.19.2.1 REG_BMI

```
#define REG_BMI 0x02
```

Adresse du registre virtuel pour les commandes BMI088 (réservé).

Definition at line 20 of file serial_cmd.h.

6.19.2.2 REG_MOTOR_CMD

```
#define REG_MOTOR_CMD 0x01
```

Adresse du registre virtuel pour la commande Moteur (mm/s).

Definition at line 18 of file serial_cmd.h.

6.19.2.3 REG_SERVO_CMD

```
#define REG_SERVO_CMD 0x00
```

Adresse du registre virtuel pour la commande Servo (0-100%).

Definition at line 16 of file serial_cmd.h.

6.19.3 Enumeration Type Documentation

6.19.3.1 ParserSwitch

```
enum ParserSwitch
```

Indicateur de résultat du parsing pour la boucle principale.

Enumerator

PARSER_IDLE	Aucune nouvelle commande traitée.
PARSER_SERVO_CMD	Une commande Servo a été reçue et validée.
PARSER_MOTOR_CMD	Une commande Moteur a été reçue et validée.
PARSER_BMI_CMD	Une commande BMI a été reçue.
PARSER_OTHERS	Une autre commande a été reçue.

Definition at line 42 of file serial_cmd.h.

6.19.4 Function Documentation

6.19.4.1 __attribute__()

```
struct __attribute__ (
            (packed)  )
```

Structure de la trame de télémétrie envoyée vers la Pi5.

Note

Structure "packed" pour éviter le padding et garantir l'alignement binaire. Format total : 4 (Header/Meta) + 4 (Time) + 12 (Accel) + 12 (Gyro) + 4 (Speed) + 1 (CRC) = 37 octets.

< Octet de synchronisation 1 (0xAA).

< Octet de synchronisation 2 (0x55).

< Type de packet (0x01 pour Télémétrie).

< Longueur du payload (32 octets).

< Timestamp système (HAL_GetTick).

< Données Accéléromètre [X, Y, Z] en mm/s^2.

< Données Gyroscope [X, Y, Z] en rad/s.

< Vitesse linéaire du véhicule en m/s.

< Checksum CRC-8 pour validation de l'intégrité.

Definition at line 1 of file serial_cmd.h.

6.19.4.2 serial_cmd_reader()

```
void serial_cmd_reader (
            void  )
```

Fonction de lecture périodique des commandes entrantes.

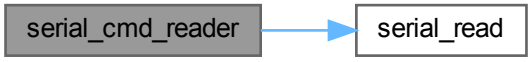
Lit le buffer circulaire RX et alimente la machine à états.

Fonction de lecture périodique des commandes entrantes.

Récupère les données brutes du buffer circulaire RX et les passe octet par octet à la machine à états.

Definition at line 138 of file serial_cmd.c.

Here is the call graph for this function:



Here is the caller graph for this function:

6.19.4.3 serial_send_data_frame()

```
void serial_send_data_frame (
            void  )
```

Capture les capteurs et envoie la trame de télémétrie.

Lit l'IMU et le compteur de vitesse, remplit SerialImuFrame_t, calcule le CRC et transmet via DMA.

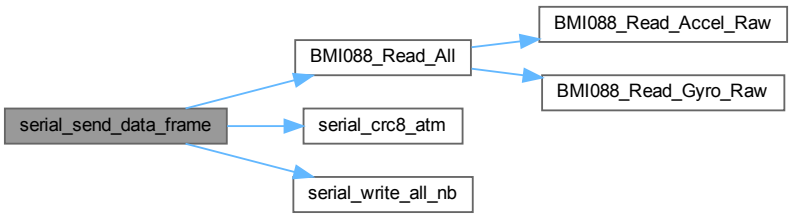
Capture les capteurs et envoie la trame de télémétrie.

1. Lit l'IMU (Accéléromètre + Gyroscope).
2. Récupère la vitesse (Speedometer).

3. Déduit le signe de la vitesse grâce à la commande moteur (Marche AR).
4. Formate le paquet binaire (SerialImuFrame_t) avec CRC.
5. Envoie le tout de manière non-bloquante via DMA.

Definition at line 156 of file serial_cmd.c.

Here is the call graph for this function:



6.19.5 Variable Documentation

6.19.5.1 parser_state

```
ParserSwitch parser_state  [extern]
```

État global du parseur, lu par le main pour appliquer les consignes.

État global du parseur, lu par le main pour appliquer les consignes.

Definition at line 18 of file serial_cmd.c.

6.19.5.2 SerialImuFrame_t

```
SerialImuFrame_t
```

Definition at line 37 of file serial_cmd.h.

6.19.5.3 shadow_motor_cmd

```
int16_t shadow_motor_cmd  [extern]
```

Dernière consigne reçue pour le moteur (Shadow Register).

Dernière consigne reçue pour le moteur (Shadow Register).

Definition at line 42 of file serial_cmd.c.

6.19.5.4 shadow_servo_cmd

```
int8_t shadow_servo_cmd  [extern]
```

Dernière consigne reçue pour le servo (Shadow Register).

Dernière consigne reçue pour le servo (Shadow Register).

Definition at line 40 of file serial_cmd.c.

6.20 serial_cmd.h

Go to the documentation of this file.

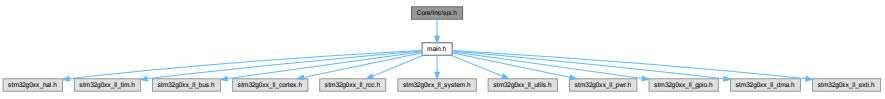
```
00001
00008 #ifndef INC_SERIAL_CMD_H_
00009 #define INC_SERIAL_CMD_H_
00010
00011 #include <stdint.h>
00012 #include <stddef.h>
00013 #include <stdbool.h>
00014
00016 #define REG_SERVO_CMD 0x00
00018 #define REG_MOTOR_CMD 0x01
00020 #define REG_BMI       0x02
00021
00027 typedef struct __attribute__((packed)) {
00028     uint8_t head1;
00029     uint8_t head2;
00030     uint8_t type;
00031     uint8_t len;
00032     uint32_t timestamp;
00033     float accel[3];
00034     float gyro[3];
00035     float speed;
00036     uint8_t crc;
00037 } SerialImuFrame_t;
00038
00042 typedef enum{
00043     PARSER_IDLE,
00044     PARSER_SERVO_CMD,
00045     PARSER_MOTOR_CMD,
00046     PARSER_BMI_CMD,
00047     PARSER_OTHERS
00048 } ParserSwitch;
00049
00051 extern ParserSwitch parser_state;
00052
00054 extern int8_t shadow_servo_cmd;
00055
00057 extern int16_t shadow_motor_cmd;
00058
00063 void serial_cmd_reader(void);
00064
00070 void serial_send_data_frame(void);
00071
00072 #endif
```

6.21 Core/Inc/spi.h File Reference

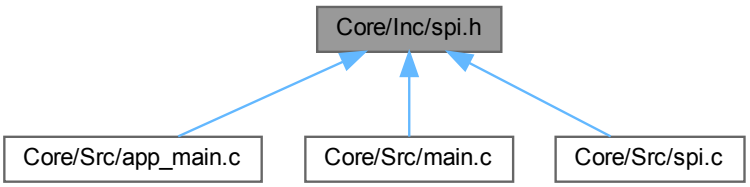
This file contains all the function prototypes for the spi.c file.

```
#include "main.h"
```

Include dependency graph for spi.h:



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



Functions

- void MX_SPI1_Init (void)

Variables

- SPI_HandleTypeDef hspi1

6.21.1 Detailed Description

This file contains all the function prototypes for the spi.c file.

Attention

Definition in file spi.h.

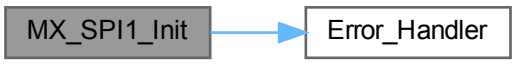
6.21.2 Function Documentation

6.21.2.1 MX_SPI1_Init()

```
void MX_SPI1_Init (
            void  )
```

Definition at line 30 of file spi.c.

Here is the call graph for this function:



Here is the caller graph for this function:

6.21.3 Variable Documentation

6.21.3.1 hspi1

```
SPI_HandleTypeDef hspi1  [extern]
```

Definition at line 27 of file spi.c.

6.22 spi.h

Go to the documentation of this file.

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020 /* Define to prevent recursive inclusion -------------------------------------*/
00021 #ifndef __SPI_H__
00022 #define __SPI_H__
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif
00027
00028 /* Includes ------------------------------------------------------------------*/
00029 #include "main.h"
00030
00031 /* USER CODE BEGIN Includes */
00032
00033 /* USER CODE END Includes */
00034
00035 extern SPI_HandleTypeDef hspi1;
00036
00037 /* USER CODE BEGIN Private defines */
00038
00039 /* USER CODE END Private defines */
00040
00041 void MX_SPI1_Init(void);
00042
```

```
00043 /* USER CODE BEGIN Prototypes */
00044
00045 /* USER CODE END Prototypes */
00046
00047 #ifdef __cplusplus
00048 }
00049 #endif
00050
00051 #endif /* __SPI_H__ */
00052
```

6.23 Core/Inc/stm32_assert.h File Reference

STM32 assert file.

Macros

- #define assert_param(expr) ((void)0U)

6.23.1 Detailed Description

STM32 assert file.

Author

MCD Application Team

Attention

Definition in file stm32_assert.h.

6.23.2 Macro Definition Documentation

6.23.2.1 assert_param

```
#define assert_param(
              expr ) ((void)0U)
```

Definition at line 45 of file stm32_assert.h.

6.24 stm32_assert.h

Go to the documentation of this file.

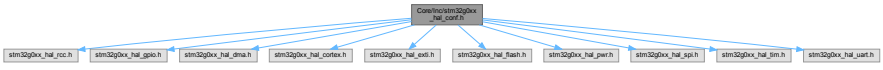
```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020 /* Define to prevent recursive inclusion -------------------------------------*/
00021 #ifndef __STM32_ASSERT_H
00022 #define __STM32_ASSERT_H
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif
00027
00028 /* Exported types ------------------------------------------------------------*/
00029 /* Exported constants --------------------------------------------------------*/
00030 /* Includes ------------------------------------------------------------------*/
00031 /* Exported macro ------------------------------------------------------------*/
00032 #ifdef  USE_FULL_ASSERT
00041 #define assert_param(expr) ((expr) ? (void)0U : assert_failed((uint8_t *)__FILE__, __LINE__))
00042 /* Exported functions ------------------------------------------------------- */
00043 void assert_failed(uint8_t *file, uint32_t line);
00044 #else
00045 #define assert_param(expr) ((void)0U)
00046 #endif /* USE_FULL_ASSERT */
00047
00048 #ifdef __cplusplus
00049 }
00050 #endif
00051
00052 #endif /* __STM32_ASSERT_H */
00053
```

6.25 Core/Inc/stm32g0xx_hal_conf.h File Reference

HAL configuration file.

```
#include "stm32g0xx_hal_rcc.h"
#include "stm32g0xx_hal_gpio.h"
#include "stm32g0xx_hal_dma.h"
#include "stm32g0xx_hal_cortex.h"
#include "stm32g0xx_hal_exti.h"
#include "stm32g0xx_hal_flash.h"
#include "stm32g0xx_hal_pwr.h"
#include "stm32g0xx_hal_spi.h"
#include "stm32g0xx_hal_tim.h"
#include "stm32g0xx_hal_uart.h"
```

Include dependency graph for stm32g0xx_hal_conf.h:



Macros

- #define HAL_MODULE_ENABLED

  This is the list of modules to be used in the HAL driver.
- #define HAL_SPI_MODULE_ENABLED
- #define HAL_TIM_MODULE_ENABLED
- #define HAL_UART_MODULE_ENABLED
- #define HAL_GPIO_MODULE_ENABLED
- #define HAL_EXTI_MODULE_ENABLED

- #define HAL_DMA_MODULE_ENABLED
- #define HAL_RCC_MODULE_ENABLED
- #define HAL_FLASH_MODULE_ENABLED
- #define HAL_PWR_MODULE_ENABLED
- #define HAL_CORTEX_MODULE_ENABLED
- #define USE_HAL_ADC_REGISTER_CALLBACKS 0u

  This is the list of modules where register callback can be used.
- #define USE_HAL_CEC_REGISTER_CALLBACKS 0u
- #define USE_HAL_COMP_REGISTER_CALLBACKS 0u
- #define USE_HAL_CRYP_REGISTER_CALLBACKS 0u
- #define USE_HAL_DAC_REGISTER_CALLBACKS 0u
- #define USE_HAL_FDCAN_REGISTER_CALLBACKS 0u
- #define USE_HAL_HCD_REGISTER_CALLBACKS 0u
- #define USE_HAL_I2C_REGISTER_CALLBACKS 0u
- #define USE_HAL_I2S_REGISTER_CALLBACKS 0u
- #define USE_HAL_IRDA_REGISTER_CALLBACKS 0u
- #define USE_HAL_LPTIM_REGISTER_CALLBACKS 0u
- #define USE_HAL_PCD_REGISTER_CALLBACKS 0u
- #define USE_HAL_RNG_REGISTER_CALLBACKS 0u
- #define USE_HAL_RTC_REGISTER_CALLBACKS 0u
- #define USE_HAL_SMBUS_REGISTER_CALLBACKS 0u
- #define USE_HAL_SPI_REGISTER_CALLBACKS 0u
- #define USE_HAL_TIM_REGISTER_CALLBACKS 0u
- #define USE_HAL_UART_REGISTER_CALLBACKS 0u
- #define USE_HAL_USART_REGISTER_CALLBACKS 0u
- #define USE_HAL_WWDG_REGISTER_CALLBACKS 0u
- #define HSE_VALUE (8000000UL)

  Adjust the value of External High Speed oscillator (HSE) used in your application. This value is used by the RCC HAL module to compute the system frequency (when HSE is used as system clock source, directly or through the PLL).
- #define HSE_STARTUP_TIMEOUT (100UL)
- #define HSI_VALUE (16000000UL)

  Internal High Speed oscillator (HSI) value. This value is used by the RCC HAL module to compute the system frequency (when HSI is used as system clock source, directly or through the PLL).
- #define LSI_VALUE (32000UL)

  Internal Low Speed oscillator (LSI) value.
- #define LSE_VALUE (32768UL)

  External Low Speed oscillator (LSE) value. This value is used by the UART, RTC HAL module to compute the system frequency.
- #define LSE_STARTUP_TIMEOUT (5000UL)
- #define EXTERNAL_I2S1_CLOCK_VALUE (12288000UL)

  External clock source for I2S1 peripheral This value is used by the RCC HAL module to compute the I2S1 clock source frequency.
- #define VDD_VALUE (3300UL)

  This is the HAL system configuration section.
- #define TICK_INT_PRIORITY 0U
- #define USE_RTOS 0U
- #define PREFETCH_ENABLE 1U
- #define INSTRUCTION_CACHE_ENABLE 1U
- #define USE_SPI_CRC 0U
- #define USE_HAL_CRYP_SUSPEND_RESUME 1U
- #define assert_param(expr) ((void)0U)

  Uncomment the line below to expanse the "assert_param" macro in the HAL drivers code.

6.25.1 Detailed Description

HAL configuration file.

Author

MCD Application Team

Attention

Definition in file stm32g0xx_hal_conf.h.

6.25.2 Macro Definition Documentation

6.25.2.1 assert_param

```
#define assert_param(
             expr ) ((void)0U)
```

Uncomment the line below to expanse the "assert_param" macro in the HAL drivers code.

Include modules header file

Definition at line 344 of file stm32g0xx_hal_conf.h.

6.25.2.2 EXTERNAL_I2S1_CLOCK_VALUE

```
#define EXTERNAL_I2S1_CLOCK_VALUE (12288000UL)
```

External clock source for I2S1 peripheral This value is used by the RCC HAL module to compute the I2S1 clock source frequency.

Value of the I2S1 External clock source in Hz

Definition at line 157 of file stm32g0xx_hal_conf.h.

6.25.2.3 HAL_CORTEX_MODULE_ENABLED

```
#define HAL_CORTEX_MODULE_ENABLED
```

Definition at line 67 of file stm32g0xx_hal_conf.h.

6.25.2.4 HAL_DMA_MODULE_ENABLED

```
#define HAL_DMA_MODULE_ENABLED
```

Definition at line 63 of file stm32g0xx_hal_conf.h.

6.25.2.5 HAL_EXTI_MODULE_ENABLED

```
#define HAL_EXTI_MODULE_ENABLED
```

Definition at line 62 of file stm32g0xx_hal_conf.h.

6.25.2.6 HAL_FLASH_MODULE_ENABLED

```
#define HAL_FLASH_MODULE_ENABLED
```

Definition at line 65 of file stm32g0xx_hal_conf.h.

6.25.2.7 HAL_GPIO_MODULE_ENABLED

```
#define HAL_GPIO_MODULE_ENABLED
```

Definition at line 61 of file stm32g0xx_hal_conf.h.

6.25.2.8 HAL_MODULE_ENABLED

```
#define HAL_MODULE_ENABLED
```

This is the list of modules to be used in the HAL driver.

Definition at line 36 of file stm32g0xx_hal_conf.h.

6.25.2.9 HAL_PWR_MODULE_ENABLED

```
#define HAL_PWR_MODULE_ENABLED
```

Definition at line 66 of file stm32g0xx_hal_conf.h.

6.25.2.10 HAL_RCC_MODULE_ENABLED

```
#define HAL_RCC_MODULE_ENABLED
```

Definition at line 64 of file stm32g0xx_hal_conf.h.

6.25.2.11 HAL_SPI_MODULE_ENABLED

```
#define HAL_SPI_MODULE_ENABLED
```

Definition at line 56 of file stm32g0xx_hal_conf.h.

6.25.2.12 HAL_TIM_MODULE_ENABLED

```
#define HAL_TIM_MODULE_ENABLED
```

Definition at line 57 of file stm32g0xx_hal_conf.h.

6.25.2.13 HAL_UART_MODULE_ENABLED

```
#define HAL_UART_MODULE_ENABLED
```

Definition at line 58 of file stm32g0xx_hal_conf.h.

6.25.2.14 HSE_STARTUP_TIMEOUT

```
#define HSE_STARTUP_TIMEOUT (100UL)
```

Time out for HSE start up, in ms

Definition at line 105 of file stm32g0xx_hal_conf.h.

6.25.2.15 HSE_VALUE

```
#define HSE_VALUE (8000000UL)
```

Adjust the value of External High Speed oscillator (HSE) used in your application. This value is used by the RCC HAL module to compute the system frequency (when HSE is used as system clock source, directly or through the PLL).

Value of the External oscillator in Hz

Definition at line 101 of file stm32g0xx_hal_conf.h.

6.25.2.16 HSI_VALUE

```
#define HSI_VALUE (16000000UL)
```

Internal High Speed oscillator (HSI) value. This value is used by the RCC HAL module to compute the system frequency (when HSI is used as system clock source, directly or through the PLL).

Value of the Internal oscillator in Hz

Definition at line 114 of file stm32g0xx_hal_conf.h.

6.25.2.17 INSTRUCTION_CACHE_ENABLE

```
#define INSTRUCTION_CACHE_ENABLE 1U
```

Definition at line 182 of file stm32g0xx_hal_conf.h.

6.25.2.18 LSE_STARTUP_TIMEOUT

```
#define LSE_STARTUP_TIMEOUT (5000UL)
```

Time out for LSE start up, in ms

Definition at line 148 of file stm32g0xx_hal_conf.h.

6.25.2.19 LSE_VALUE

```
#define LSE_VALUE (32768UL)
```

External Low Speed oscillator (LSE) value. This value is used by the UART, RTC HAL module to compute the system frequency.

< Value of the Internal Low Speed oscillator in Hz The real value may vary depending on the variations in voltage and temperature. Value of the External oscillator in Hz

Definition at line 144 of file stm32g0xx_hal_conf.h.

6.25.2.20 LSI_VALUE

```
#define LSI_VALUE (32000UL)
```

Internal Low Speed oscillator (LSI) value.

LSI Typical Value in Hz

Definition at line 135 of file stm32g0xx_hal_conf.h.

6.25.2.21 PREFETCH_ENABLE

```
#define PREFETCH_ENABLE 1U
```

Definition at line 181 of file stm32g0xx_hal_conf.h.

6.25.2.22 TICK_INT_PRIORITY

```
#define TICK_INT_PRIORITY 0U
```

tick interrupt priority

Definition at line 179 of file stm32g0xx_hal_conf.h.

6.25.2.23 USE_HAL_ADC_REGISTER_CALLBACKS

```
#define USE_HAL_ADC_REGISTER_CALLBACKS 0u
```

This is the list of modules where register callback can be used.

Definition at line 73 of file stm32g0xx_hal_conf.h.

6.25.2.24 USE_HAL_CEC_REGISTER_CALLBACKS

```
#define USE_HAL_CEC_REGISTER_CALLBACKS 0u
```

Definition at line 74 of file stm32g0xx_hal_conf.h.

6.25.2.25 USE_HAL_COMP_REGISTER_CALLBACKS

```
#define USE_HAL_COMP_REGISTER_CALLBACKS 0u
```

Definition at line 75 of file stm32g0xx_hal_conf.h.

6.25.2.26 USE_HAL_CRYP_REGISTER_CALLBACKS

```
#define USE_HAL_CRYP_REGISTER_CALLBACKS 0u
```

Definition at line 76 of file stm32g0xx_hal_conf.h.

6.25.2.27 USE_HAL_CRYP_SUSPEND_RESUME

```
#define USE_HAL_CRYP_SUSPEND_RESUME 1U
```

Definition at line 195 of file stm32g0xx_hal_conf.h.

6.25.2.28 USE_HAL_DAC_REGISTER_CALLBACKS

```
#define USE_HAL_DAC_REGISTER_CALLBACKS 0u
```

Definition at line 77 of file stm32g0xx_hal_conf.h.

6.25.2.29 USE_HAL_FDCAN_REGISTER_CALLBACKS

```
#define USE_HAL_FDCAN_REGISTER_CALLBACKS 0u
```

Definition at line 78 of file stm32g0xx_hal_conf.h.

6.25.2.30 USE_HAL_HCD_REGISTER_CALLBACKS

```
#define USE_HAL_HCD_REGISTER_CALLBACKS 0u
```

Definition at line 79 of file stm32g0xx_hal_conf.h.

6.25.2.31 USE_HAL_I2C_REGISTER_CALLBACKS

```
#define USE_HAL_I2C_REGISTER_CALLBACKS 0u
```

Definition at line 80 of file stm32g0xx_hal_conf.h.

6.25.2.32 USE_HAL_I2S_REGISTER_CALLBACKS

```
#define USE_HAL_I2S_REGISTER_CALLBACKS 0u
```

Definition at line 81 of file stm32g0xx_hal_conf.h.

6.25.2.33 USE_HAL_IRDA_REGISTER_CALLBACKS

```
#define USE_HAL_IRDA_REGISTER_CALLBACKS 0u
```

Definition at line 82 of file stm32g0xx_hal_conf.h.

6.25.2.34 USE_HAL_LPTIM_REGISTER_CALLBACKS

```
#define USE_HAL_LPTIM_REGISTER_CALLBACKS 0u
```

Definition at line 83 of file stm32g0xx_hal_conf.h.

6.25.2.35 USE_HAL_PCD_REGISTER_CALLBACKS

```
#define USE_HAL_PCD_REGISTER_CALLBACKS 0u
```

Definition at line 84 of file stm32g0xx_hal_conf.h.

6.25.2.36 USE_HAL_RNG_REGISTER_CALLBACKS

```
#define USE_HAL_RNG_REGISTER_CALLBACKS 0u
```

Definition at line 85 of file stm32g0xx_hal_conf.h.

6.25.2.37 USE_HAL_RTC_REGISTER_CALLBACKS

```
#define USE_HAL_RTC_REGISTER_CALLBACKS 0u
```

Definition at line 86 of file stm32g0xx_hal_conf.h.

6.25.2.38 USE_HAL_SMBUS_REGISTER_CALLBACKS

```
#define USE_HAL_SMBUS_REGISTER_CALLBACKS 0u
```

Definition at line 87 of file stm32g0xx_hal_conf.h.

6.25.2.39 USE_HAL_SPI_REGISTER_CALLBACKS

```
#define USE_HAL_SPI_REGISTER_CALLBACKS 0u
```

Definition at line 88 of file stm32g0xx_hal_conf.h.

6.25.2.40 USE_HAL_TIM_REGISTER_CALLBACKS

```
#define USE_HAL_TIM_REGISTER_CALLBACKS 0u
```

Definition at line 89 of file stm32g0xx_hal_conf.h.

6.25.2.41 USE_HAL_UART_REGISTER_CALLBACKS

```
#define USE_HAL_UART_REGISTER_CALLBACKS 0u
```

Definition at line 90 of file stm32g0xx_hal_conf.h.

6.25.2.42 USE_HAL_USART_REGISTER_CALLBACKS

```
#define USE_HAL_USART_REGISTER_CALLBACKS 0u
```

Definition at line 91 of file stm32g0xx_hal_conf.h.

6.25.2.43 USE_HAL_WWDG_REGISTER_CALLBACKS

```
#define USE_HAL_WWDG_REGISTER_CALLBACKS 0u
```

Definition at line 92 of file stm32g0xx_hal_conf.h.

6.25.2.44 USE_RTOS

```
#define USE_RTOS 0U
```

Definition at line 180 of file stm32g0xx_hal_conf.h.

6.25.2.45 USE_SPI_CRC

```
#define USE_SPI_CRC 0U
```

Definition at line 191 of file stm32g0xx_hal_conf.h.

6.25.2.46 VDD_VALUE

```
#define VDD_VALUE (3300UL)
```

This is the HAL system configuration section.

Value of VDD in mv

Definition at line 178 of file stm32g0xx_hal_conf.h.

6.26 stm32g0xx_hal_conf.h

Go to the documentation of this file.

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020
00021 /* Define to prevent recursive inclusion -------------------------------------*/
00022 #ifndef STM32G0xx_HAL_CONF_H
00023 #define STM32G0xx_HAL_CONF_H
00024
00025 #ifdef __cplusplus
00026 extern "C" {
00027 #endif
00028
00029 /* Exported types ------------------------------------------------------------*/
00030 /* Exported constants --------------------------------------------------------*/
00031
00032 /* ########################## Module Selection ############################## */
00036 #define HAL_MODULE_ENABLED
00037 /* #define HAL_ADC_MODULE_ENABLED   */
00038 /* #define HAL_CEC_MODULE_ENABLED   */
00039 /* #define HAL_COMP_MODULE_ENABLED   */
00040 /* #define HAL_CRC_MODULE_ENABLED   */
00041 /* #define HAL_CRYP_MODULE_ENABLED   */
00042 /* #define HAL_DAC_MODULE_ENABLED   */
00043 /* #define HAL_EXTI_MODULE_ENABLED   */
00044 /* #define HAL_FDCAN_MODULE_ENABLED   */
00045 /* #define HAL_HCD_MODULE_ENABLED   */
00046 /* #define HAL_I2C_MODULE_ENABLED   */
00047 /* #define HAL_I2S_MODULE_ENABLED   */
00048 /* #define HAL_IWDG_MODULE_ENABLED   */
00049 /* #define HAL_IRDA_MODULE_ENABLED   */
00050 /* #define HAL_LPTIM_MODULE_ENABLED   */
00051 /* #define HAL_PCD_MODULE_ENABLED   */
00052 /* #define HAL_RNG_MODULE_ENABLED   */
00053 /* #define HAL_RTC_MODULE_ENABLED   */
00054 /* #define HAL_SMARTCARD_MODULE_ENABLED   */
00055 /* #define HAL_SMBUS_MODULE_ENABLED   */
00056 #define HAL_SPI_MODULE_ENABLED
00057 #define HAL_TIM_MODULE_ENABLED
00058 #define HAL_UART_MODULE_ENABLED
00059 /* #define HAL_USART_MODULE_ENABLED   */
00060 /* #define HAL_WWDG_MODULE_ENABLED   */
00061 #define HAL_GPIO_MODULE_ENABLED
00062 #define HAL_EXTI_MODULE_ENABLED
00063 #define HAL_DMA_MODULE_ENABLED
00064 #define HAL_RCC_MODULE_ENABLED
00065 #define HAL_FLASH_MODULE_ENABLED
00066 #define HAL_PWR_MODULE_ENABLED
00067 #define HAL_CORTEX_MODULE_ENABLED
00068
00069 /* ########################## Register Callbacks selection ############################## */
00073 #define USE_HAL_ADC_REGISTER_CALLBACKS    0u
00074 #define USE_HAL_CEC_REGISTER_CALLBACKS    0u
00075 #define USE_HAL_COMP_REGISTER_CALLBACKS   0u
00076 #define USE_HAL_CRYP_REGISTER_CALLBACKS   0u
00077 #define USE_HAL_DAC_REGISTER_CALLBACKS    0u
00078 #define USE_HAL_FDCAN_REGISTER_CALLBACKS  0u
00079 #define USE_HAL_HCD_REGISTER_CALLBACKS    0u
00080 #define USE_HAL_I2C_REGISTER_CALLBACKS    0u
00081 #define USE_HAL_I2S_REGISTER_CALLBACKS    0u
00082 #define USE_HAL_IRDA_REGISTER_CALLBACKS   0u
00083 #define USE_HAL_LPTIM_REGISTER_CALLBACKS  0u
00084 #define USE_HAL_PCD_REGISTER_CALLBACKS    0u
00085 #define USE_HAL_RNG_REGISTER_CALLBACKS    0u
00086 #define USE_HAL_RTC_REGISTER_CALLBACKS    0u
```

```
00087 #define USE_HAL_SMBUS_REGISTER_CALLBACKS  0u
00088 #define USE_HAL_SPI_REGISTER_CALLBACKS    0u
00089 #define USE_HAL_TIM_REGISTER_CALLBACKS    0u
00090 #define USE_HAL_UART_REGISTER_CALLBACKS   0u
00091 #define USE_HAL_USART_REGISTER_CALLBACKS  0u
00092 #define USE_HAL_WWDG_REGISTER_CALLBACKS   0u
00093
00094 /* ########################## Oscillator Values adaptation ####################*/
00100 #if !defined  (HSE_VALUE)
00101 #define HSE_VALUE    (8000000UL)
00102 #endif /* HSE_VALUE */
00103
00104 #if !defined  (HSE_STARTUP_TIMEOUT)
00105 #define HSE_STARTUP_TIMEOUT    (100UL)
00106 #endif /* HSE_STARTUP_TIMEOUT */
00107
00113 #if !defined  (HSI_VALUE)
00114 #define HSI_VALUE    (16000000UL)
00115 #endif /* HSI_VALUE */
00116
00117 #if defined(STM32G0C1xx) || defined(STM32G0B1xx) || defined(STM32G0B0xx)
00125 #if !defined  (HSI48_VALUE)
00126   #define HSI48_VALUE   48000000U
00128 #endif /* HSI48_VALUE */
00129 #endif
00130
00134 #if !defined  (LSI_VALUE)
00135 #define LSI_VALUE  (32000UL)
00136 #endif /* LSI_VALUE */
00143 #if !defined  (LSE_VALUE)
00144 #define LSE_VALUE    (32768UL)
00145 #endif /* LSE_VALUE */
00146
00147 #if !defined  (LSE_STARTUP_TIMEOUT)
00148 #define LSE_STARTUP_TIMEOUT    (5000UL)
00149 #endif /* LSE_STARTUP_TIMEOUT */
00150
00156 #if !defined  (EXTERNAL_I2S1_CLOCK_VALUE)
00157 #define EXTERNAL_I2S1_CLOCK_VALUE    (12288000UL)
00158 #endif /* EXTERNAL_I2S1_CLOCK_VALUE */
00159
00160 #if defined(STM32G0C1xx) || defined(STM32G0B1xx) || defined(STM32G0B0xx)
00166 #if !defined  (EXTERNAL_I2S2_CLOCK_VALUE)
00167   #define EXTERNAL_I2S2_CLOCK_VALUE    12288000U
00168 #endif /* EXTERNAL_I2S2_CLOCK_VALUE */
00169 #endif
00170
00171 /* Tip: To avoid modifying this file each time you need to use different HSE,
00172    ===  you can define the HSE value in your toolchain compiler preprocessor. */
00173
00174 /* ########################### System Configuration ######################### */
00178 #define  VDD_VALUE                    (3300UL)
00179 #define  TICK_INT_PRIORITY            0U
00180 #define  USE_RTOS                     0U
00181 #define  PREFETCH_ENABLE              1U
00182 #define  INSTRUCTION_CACHE_ENABLE     1U
00183
00184 /* ################## SPI peripheral configuration ########################## */
00185
00186 /* CRC FEATURE: Use to activate CRC feature inside HAL SPI Driver
00187 * Activated: CRC code is present inside driver
00188 * Deactivated: CRC code cleaned from driver
00189 */
00190
00191 #define USE_SPI_CRC                    0U
00192
00193 /* ################## CRYP peripheral configuration ########################## */
00194
00195 #define USE_HAL_CRYP_SUSPEND_RESUME     1U
00196
00197 /* ########################## Assert Selection ############################## */
00202 /* #define USE_FULL_ASSERT    1U */
00203
00204 /* Includes ------------------------------------------------------------------*/
00209 #ifdef HAL_RCC_MODULE_ENABLED
00210 #include "stm32g0xx_hal_rcc.h"
00211 #endif /* HAL_RCC_MODULE_ENABLED */
00212
00213 #ifdef HAL_GPIO_MODULE_ENABLED
00214 #include "stm32g0xx_hal_gpio.h"
00215 #endif /* HAL_GPIO_MODULE_ENABLED */
00216
00217 #ifdef HAL_DMA_MODULE_ENABLED
00218 #include "stm32g0xx_hal_dma.h"
00219 #endif /* HAL_DMA_MODULE_ENABLED */
00220
00221 #ifdef HAL_CORTEX_MODULE_ENABLED
```

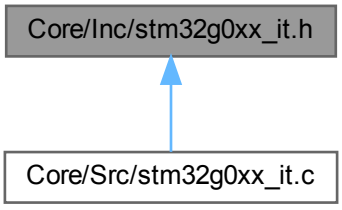
```
00222 #include "stm32g0xx_hal_cortex.h"
00223 #endif /* HAL_CORTEX_MODULE_ENABLED */
00224
00225 #ifdef HAL_ADC_MODULE_ENABLED
00226 #include "stm32g0xx_hal_adc.h"
00227 #include "stm32g0xx_hal_adc_ex.h"
00228 #endif /* HAL_ADC_MODULE_ENABLED */
00229
00230 #ifdef HAL_CEC_MODULE_ENABLED
00231 #include "stm32g0xx_hal_cec.h"
00232 #endif /* HAL_CEC_MODULE_ENABLED */
00233
00234 #ifdef HAL_COMP_MODULE_ENABLED
00235 #include "stm32g0xx_hal_comp.h"
00236 #endif /* HAL_COMP_MODULE_ENABLED */
00237
00238 #ifdef HAL_CRC_MODULE_ENABLED
00239 #include "stm32g0xx_hal_crc.h"
00240 #endif /* HAL_CRC_MODULE_ENABLED */
00241
00242 #ifdef HAL_CRYP_MODULE_ENABLED
00243 #include "stm32g0xx_hal_cryp.h"
00244 #endif /* HAL_CRYP_MODULE_ENABLED */
00245
00246 #ifdef HAL_DAC_MODULE_ENABLED
00247 #include "stm32g0xx_hal_dac.h"
00248 #endif /* HAL_DAC_MODULE_ENABLED */
00249
00250 #ifdef HAL_EXTI_MODULE_ENABLED
00251 #include "stm32g0xx_hal_exti.h"
00252 #endif /* HAL_EXTI_MODULE_ENABLED */
00253
00254 #ifdef HAL_FLASH_MODULE_ENABLED
00255 #include "stm32g0xx_hal_flash.h"
00256 #endif /* HAL_FLASH_MODULE_ENABLED */
00257
00258 #ifdef HAL_FDCAN_MODULE_ENABLED
00259 #include "stm32g0xx_hal_fdcan.h"
00260 #endif /* HAL_FDCAN_MODULE_ENABLED */
00261
00262 #ifdef HAL_HCD_MODULE_ENABLED
00263 #include "stm32g0xx_hal_hcd.h"
00264 #endif /* HAL_HCD_MODULE_ENABLED */
00265
00266 #ifdef HAL_I2C_MODULE_ENABLED
00267 #include "stm32g0xx_hal_i2c.h"
00268 #endif /* HAL_I2C_MODULE_ENABLED */
00269
00270 #ifdef HAL_I2S_MODULE_ENABLED
00271 #include "stm32g0xx_hal_i2s.h"
00272 #endif /* HAL_I2S_MODULE_ENABLED */
00273
00274 #ifdef HAL_IRDA_MODULE_ENABLED
00275 #include "stm32g0xx_hal_irda.h"
00276 #endif /* HAL_IRDA_MODULE_ENABLED */
00277
00278 #ifdef HAL_IWDG_MODULE_ENABLED
00279 #include "stm32g0xx_hal_iwdg.h"
00280 #endif /* HAL_IWDG_MODULE_ENABLED */
00281
00282 #ifdef HAL_LPTIM_MODULE_ENABLED
00283 #include "stm32g0xx_hal_lptim.h"
00284 #endif /* HAL_LPTIM_MODULE_ENABLED */
00285
00286 #ifdef HAL_PCD_MODULE_ENABLED
00287 #include "stm32g0xx_hal_pcd.h"
00288 #endif /* HAL_PCD_MODULE_ENABLED */
00289
00290 #ifdef HAL_PWR_MODULE_ENABLED
00291 #include "stm32g0xx_hal_pwr.h"
00292 #endif /* HAL_PWR_MODULE_ENABLED */
00293
00294 #ifdef HAL_RNG_MODULE_ENABLED
00295 #include "stm32g0xx_hal_rng.h"
00296 #endif /* HAL_RNG_MODULE_ENABLED */
00297
00298 #ifdef HAL_RTC_MODULE_ENABLED
00299 #include "stm32g0xx_hal_rtc.h"
00300 #endif /* HAL_RTC_MODULE_ENABLED */
00301
00302 #ifdef HAL_SMARTCARD_MODULE_ENABLED
00303 #include "stm32g0xx_hal_smartcard.h"
00304 #endif /* HAL_SMARTCARD_MODULE_ENABLED */
00305
00306 #ifdef HAL_SMBUS_MODULE_ENABLED
00307 #include "stm32g0xx_hal_smbus.h"
00308 #endif /* HAL_SMBUS_MODULE_ENABLED */
```

```
00309
00310 #ifdef HAL_SPI_MODULE_ENABLED
00311 #include "stm32g0xx_hal_spi.h"
00312 #endif /* HAL_SPI_MODULE_ENABLED */
00313
00314 #ifdef HAL_TIM_MODULE_ENABLED
00315 #include "stm32g0xx_hal_tim.h"
00316 #endif /* HAL_TIM_MODULE_ENABLED */
00317
00318 #ifdef HAL_UART_MODULE_ENABLED
00319 #include "stm32g0xx_hal_uart.h"
00320 #endif /* HAL_UART_MODULE_ENABLED */
00321
00322 #ifdef HAL_USART_MODULE_ENABLED
00323 #include "stm32g0xx_hal_usart.h"
00324 #endif /* HAL_USART_MODULE_ENABLED */
00325
00326 #ifdef HAL_WWDG_MODULE_ENABLED
00327 #include "stm32g0xx_hal_wwdg.h"
00328 #endif /* HAL_WWDG_MODULE_ENABLED */
00329
00330 /* Exported macro ------------------------------------------------------------*/
00331 #ifdef  USE_FULL_ASSERT
00340 #define assert_param(expr) ((expr) ? (void)0U : assert_failed((uint8_t *)__FILE__, __LINE__))
00341 /* Exported functions ------------------------------------------------------- */
00342 void assert_failed(uint8_t *file, uint32_t line);
00343 #else
00344 #define assert_param(expr) ((void)0U)
00345 #endif /* USE_FULL_ASSERT */
00346
00347 #ifdef __cplusplus
00348 }
00349 #endif
00350
00351 #endif /* STM32G0xx_HAL_CONF_H */
```

6.27 Core/Inc/stm32g0xx_it.h File Reference

This file contains the headers of the interrupt handlers.

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



Functions

- void NMI_Handler (void)

  This function handles Non maskable interrupt.
- void HardFault_Handler (void)

  This function handles Hard fault interrupt.
- void SVC_Handler (void)

  This function handles System service call via SWI instruction.

- void PendSV_Handler (void)

  This function handles Pendable request for system service.
- void SysTick_Handler (void)

  This function handles System tick timer.
- void DMA1_Channel1_IRQHandler (void)

  This function handles DMA1 channel 1 interrupt.
- void DMA1_Channel2_3_IRQHandler (void)

  This function handles DMA1 channel 2 and channel 3 interrupts.
- void TIM3_TIM4_IRQHandler (void)

  This function handles TIM3, TIM4 global Interrupt.
- void SPI1_IRQHandler (void)

  This function handles SPI1/I2S1 Interrupt.
- void USART2_LPUART2_IRQHandler (void)

  This function handles USART2 + LPUART2 Interrupt.

6.27.1 Detailed Description

This file contains the headers of the interrupt handlers.

Attention

Definition in file stm32g0xx_it.h.

6.27.2 Function Documentation

6.27.2.1 DMA1_Channel1_IRQHandler()

```
void DMA1_Channel1_IRQHandler (
            void  )
```

This function handles DMA1 channel 1 interrupt.

Definition at line 151 of file stm32g0xx_it.c.

6.27.2.2 DMA1_Channel2_3_IRQHandler()

```
void DMA1_Channel2_3_IRQHandler (
            void  )
```

This function handles DMA1 channel 2 and channel 3 interrupts.

Definition at line 165 of file stm32g0xx_it.c.

6.27.2.3 HardFault_Handler()

```
void HardFault_Handler (
             void  )
```

This function handles Hard fault interrupt.

Definition at line 89 of file stm32g0xx_it.c.

6.27.2.4 NMI_Handler()

```
void NMI_Handler (
             void  )
```

This function handles Non maskable interrupt.

Definition at line 74 of file stm32g0xx_it.c.

6.27.2.5 PendSV_Handler()

```
void PendSV_Handler (
             void  )
```

This function handles Pendable request for system service.

Definition at line 117 of file stm32g0xx_it.c.

6.27.2.6 SPI1_IRQHandler()

```
void SPI1_IRQHandler (
             void  )
```

This function handles SPI1/I2S1 Interrupt.

Definition at line 200 of file stm32g0xx_it.c.

6.27.2.7 SVC_Handler()

```
void SVC_Handler (
             void  )
```

This function handles System service call via SWI instruction.

Definition at line 104 of file stm32g0xx_it.c.

6.27.2.8 SysTick_Handler()

```
void SysTick_Handler (
             void  )
```

This function handles System tick timer.

Definition at line 130 of file stm32g0xx_it.c.

6.27.2.9 TIM3_TIM4_IRQHandler()

```
void TIM3_TIM4_IRQHandler (
            void  )
```

This function handles TIM3, TIM4 global Interrupt.

-

Definition at line 179 of file stm32g0xx_it.c.

6.27.2.10 USART2_LPUART2_IRQHandler()

```
void USART2_LPUART2_IRQHandler (
            void  )
```

This function handles USART2 + LPUART2 Interrupt.

Definition at line 214 of file stm32g0xx_it.c.

6.28 stm32g0xx_it.h

Go to the documentation of this file.

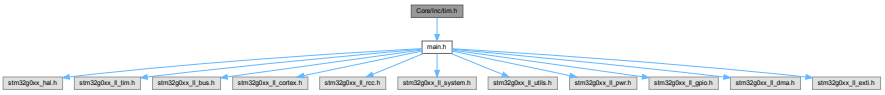
```
00001 /* USER CODE BEGIN Header */
00018 /* USER CODE END Header */
00019
00020 /* Define to prevent recursive inclusion -------------------------------------*/
00021 #ifndef __STM32G0xx_IT_H
00022 #define __STM32G0xx_IT_H
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif
00027
00028 /* Private includes ----------------------------------------------------------*/
00029 /* USER CODE BEGIN Includes */
00030
00031 /* USER CODE END Includes */
00032
00033 /* Exported types ------------------------------------------------------------*/
00034 /* USER CODE BEGIN ET */
00035
00036 /* USER CODE END ET */
00037
00038 /* Exported constants --------------------------------------------------------*/
00039 /* USER CODE BEGIN EC */
00040
00041 /* USER CODE END EC */
00042
00043 /* Exported macro ------------------------------------------------------------*/
00044 /* USER CODE BEGIN EM */
00045
00046 /* USER CODE END EM */
00047
00048 /* Exported functions prototypes ---------------------------------------------*/
00049 void NMI_Handler(void);
00050 void HardFault_Handler(void);
00051 void SVC_Handler(void);
00052 void PendSV_Handler(void);
00053 void SysTick_Handler(void);
00054 void DMA1_Channel1_IRQHandler(void);
00055 void DMA1_Channel2_3_IRQHandler(void);
00056 void TIM3_TIM4_IRQHandler(void);
00057 void SPI1_IRQHandler(void);
00058 void USART2_LPUART2_IRQHandler(void);
00059 /* USER CODE BEGIN EFP */
00060
00061 /* USER CODE END EFP */
00062
00063 #ifdef __cplusplus
00064 }
00065 #endif
00066
00067 #endif /* __STM32G0xx_IT_H */
```

6.29 Core/Inc/tim.h File Reference

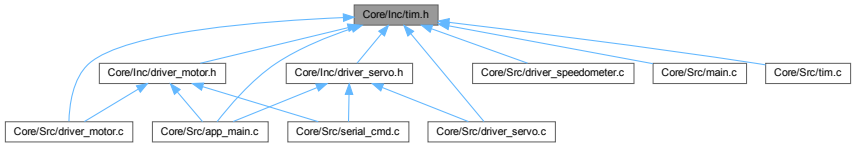
This file contains all the function prototypes for the tim.c file.

```
#include "main.h"
```

Include dependency graph for tim.h:



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



Functions

- void MX_TIM1_Init (void)
- void MX_TIM2_Init (void)
- void MX_TIM3_Init (void)
- void MX_TIM4_Init (void)
- void HAL_TIM_MspPostInit (TIM_HandleTypeDef ∗htim)

Variables

- TIM_HandleTypeDef htim1
- TIM_HandleTypeDef htim2
- TIM_HandleTypeDef htim4

6.29.1 Detailed Description

This file contains all the function prototypes for the tim.c file.

Attention

Definition in file tim.h.

6.29.2 Function Documentation

6.29.2.1 HAL_TIM_MspPostInit()

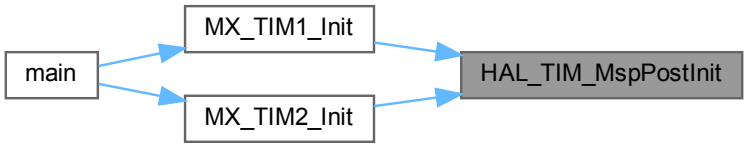
```
void HAL_TIM_MspPostInit (
            TIM_HandleTypeDef * htim )
```

TIM1 GPIO Configuration PA8 ------> TIM1_CH1

TIM2 GPIO Configuration PA0 ------> TIM2_CH1

Definition at line 306 of file tim.c.

Here is the caller graph for this function:

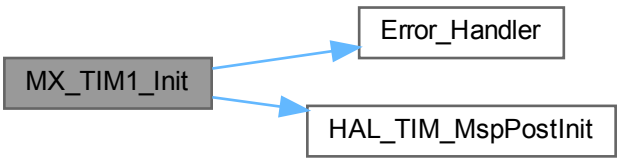


6.29.2.2 MX_TIM1_Init()

```
void MX_TIM1_Init (
            void  )
```

Definition at line 32 of file tim.c.

Here is the call graph for this function:



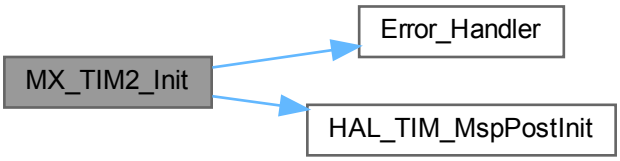
Here is the caller graph for this function:

6.29.2.3 MX_TIM2_Init()

```
void MX_TIM2_Init (
            void  )
```

Definition at line 109 of file tim.c.

Here is the call graph for this function:



Here is the caller graph for this function:

6.29.2.4 MX_TIM3_Init()

```
void MX_TIM3_Init (
            void  )
```

Definition at line 163 of file tim.c.

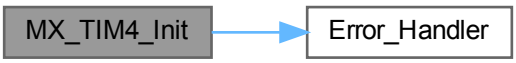
Here is the caller graph for this function:

6.29.2.5 MX_TIM4_Init()

```
void MX_TIM4_Init (
             void  )
```

Definition at line 197 of file tim.c.

Here is the call graph for this function:



Here is the caller graph for this function:

6.29.3 Variable Documentation

6.29.3.1 htim1

```
TIM_HandleTypeDef htim1  [extern]
```

Definition at line 27 of file tim.c.

6.29.3.2 htim2

```
TIM_HandleTypeDef htim2  [extern]
```

Definition at line 28 of file tim.c.

6.29.3.3 htim4

```
TIM_HandleTypeDef htim4  [extern]
```

Definition at line 29 of file tim.c.

6.30 tim.h

Go to the documentation of this file.

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020 /* Define to prevent recursive inclusion -------------------------------------*/
00021 #ifndef __TIM_H__
00022 #define __TIM_H__
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif
00027
00028 /* Includes ------------------------------------------------------------------*/
00029 #include "main.h"
00030
00031 /* USER CODE BEGIN Includes */
00032
00033 /* USER CODE END Includes */
00034
00035 extern TIM_HandleTypeDef htim1;
00036
00037 extern TIM_HandleTypeDef htim2;
00038
00039 extern TIM_HandleTypeDef htim4;
00040
00041 /* USER CODE BEGIN Private defines */
00042
00043 /* USER CODE END Private defines */
00044
00045 void MX_TIM1_Init(void);
00046 void MX_TIM2_Init(void);
00047 void MX_TIM3_Init(void);
00048 void MX_TIM4_Init(void);
00049
00050 void HAL_TIM_MspPostInit(TIM_HandleTypeDef *htim);
00051
00052 /* USER CODE BEGIN Prototypes */
00053
00054 /* USER CODE END Prototypes */
00055
00056 #ifdef __cplusplus
00057 }
00058 #endif
00059
00060 #endif /* __TIM_H__ */
00061
```

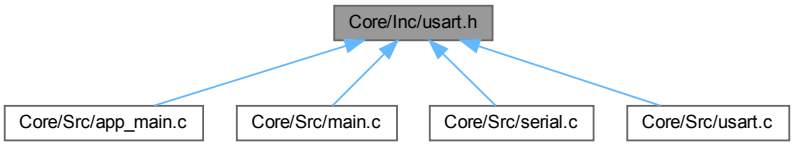
6.31 Core/Inc/usart.h File Reference

This file contains all the function prototypes for the usart.c file.

```
#include "main.h"
```

Include dependency graph for usart.h:

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



Functions

- void MX_USART2_UART_Init (void)

Variables

- UART_HandleTypeDef huart2

6.31.1 Detailed Description

This file contains all the function prototypes for the usart.c file.

Attention

Definition in file usart.h.

6.31.2 Function Documentation

6.31.2.1 MX_USART2_UART_Init()

```
void MX_USART2_UART_Init (
            void  )
```

Definition at line 33 of file usart.c.

Here is the call graph for this function:

Here is the caller graph for this function:

6.31.3 Variable Documentation

6.31.3.1 huart2

```
UART_HandleTypeDef huart2  [extern]
```

Definition at line 27 of file usart.c.

6.32 usart.h

Go to the documentation of this file.

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020 /* Define to prevent recursive inclusion -------------------------------------*/
00021 #ifndef __USART_H__
00022 #define __USART_H__
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif
00027
```

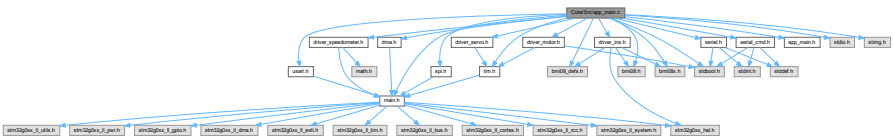
```
00028 /* Includes ------------------------------------------------------------------*/
00029 #include "main.h"
00030
00031 /* USER CODE BEGIN Includes */
00032
00033 /* USER CODE END Includes */
00034
00035 extern UART_HandleTypeDef huart2;
00036
00037 /* USER CODE BEGIN Private defines */
00038
00039 /* USER CODE END Private defines */
00040
00041 void MX_USART2_UART_Init(void);
00042
00043 /* USER CODE BEGIN Prototypes */
00044
00045 /* USER CODE END Prototypes */
00046
00047 #ifdef __cplusplus
00048 }
00049 #endif
00050
00051 #endif /* __USART_H__ */
00052
```

6.33 Core/Src/app_main.c File Reference

Point d'entrée principal de l'application (Main Loop & Scheduler).

```
#include "main.h"
#include "spi.h"
#include "tim.h"
#include "usart.h"
#include "app_main.h"
#include "driver_ins.h"
#include "bmi08_defs.h"
#include "bmi08.h"
#include "bmi08x.h"
#include "driver_servo.h"
#include "driver_motor.h"
#include "dma.h"
#include "serial.h"
#include "serial_cmd.h"
#include "driver_speedometer.h"
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
```

Include dependency graph for app_main.c:



Macros

- #define TASK_MOTOR_US 1000

  Période d'exécution de la tâche moteur (1 ms).
- #define TASK_TELEMETRY_US 10000

Période d'envoi de la télémétrie (10 ms).

- #define TASK_SPEED_US 100000

  Période de calcul de la vitesse (100 ms).
- #define FAILSAFE_TIMEOUT_MS 500

  Délai d'inactivité avant déclenchement du Failsafe (arrêt d'urgence).
- #define t_1_ms 3200

  Durée d'impulsion pour 1ms (référence PWM).
- #define t_2_ms 6400

  Durée d'impulsion pour 2ms (référence PWM).
- #define PWM_MIN_ESC 3200

  Borne PWM minimale pour l'ESC.
- #define PWM_MAX_ESC 6400

  Borne PWM maximale pour l'ESC.

Functions

- void app_config (void)

  Initialisation générale de l'application.
- void app_loop (void)

  Boucle principale de l'application (Super Loop).

Variables

- Speedometer_Handle_t hSpeedo

  Instance du capteur de vitesse (Tachymètre).
- float speed_speedo_data = 0.0f

  Variable globale stockant la vitesse actuelle (partagée avec serial_cmd).
- volatile uint32_t tim3_overflow_cnt = 0

  Compteur de débordements pour le Timer 3 (Extension 16-bit vers 32-bit).

6.33.1 Detailed Description

Point d'entrée principal de l'application (Main Loop & Scheduler).

Ce fichier contient la boucle principale, l'initialisation du système, et l'ordonnanceur coopératif pour les tâches périodiques :

- Gestion Moteur (1kHz)
- Télémétrie (100Hz)
- Lecture Vitesse (10Hz)
- Traitement des commandes et Sécurité Failsafe.

Definition in file app_main.c.

6.33.2 Macro Definition Documentation

6.33.2.1 FAILSAFE_TIMEOUT_MS

```
#define FAILSAFE_TIMEOUT_MS 500
```

Délai d'inactivité avant déclenchement du Failsafe (arrêt d'urgence).

Definition at line 38 of file app_main.c.

6.33.2.2 PWM_MAX_ESC

```
#define PWM_MAX_ESC 6400
```

Borne PWM maximale pour l'ESC.

Definition at line 48 of file app_main.c.

6.33.2.3 PWM_MIN_ESC

```
#define PWM_MIN_ESC 3200
```

Borne PWM minimale pour l'ESC.

Definition at line 46 of file app_main.c.

6.33.2.4 t_1_ms

```
#define t_1_ms 3200
```

Durée d'impulsion pour 1ms (référence PWM).

Definition at line 41 of file app_main.c.

6.33.2.5 t_2_ms

```
#define t_2_ms 6400
```

Durée d'impulsion pour 2ms (référence PWM).

Definition at line 43 of file app_main.c.

6.33.2.6 TASK_MOTOR_US

```
#define TASK_MOTOR_US 1000
```

Période d'exécution de la tâche moteur (1 ms).

Definition at line 32 of file app_main.c.

6.33.2.7 TASK_SPEED_US

```
#define TASK_SPEED_US 100000
```

Période de calcul de la vitesse (100 ms).

Definition at line 36 of file app_main.c.

6.33.2.8 TASK_TELEMETRY_US

```
#define TASK_TELEMETRY_US 10000
```

Période d'envoi de la télémétrie (10 ms).

Definition at line 34 of file app_main.c.

6.33.3 Function Documentation

6.33.3.1 app_config()

```
void app_config (
            void  )
```

Initialisation générale de l'application.

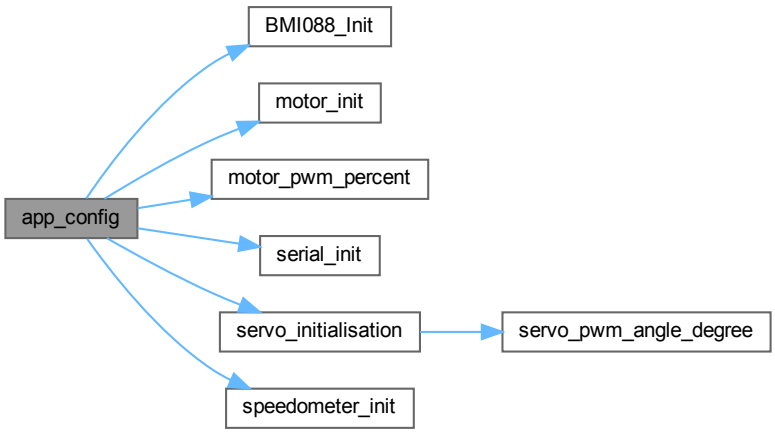
Configuration globale de l'application (Callback ou Init).

Configure l'ensemble de l'application (Hardware + Drivers).

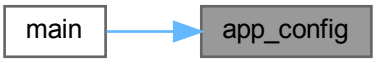
Configure les Timers, active les interruptions nécessaires, initialise les drivers (Série, BMI088, Servo, Moteur, Speedo) et cale les horloges.

Definition at line 193 of file app_main.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.33.3.2 app_loop()

```
void app_loop (
            void  )
```

Boucle principale de l'application (Super Loop).

Exécute la boucle principale (Scheduler).

Exécute séquentiellement :

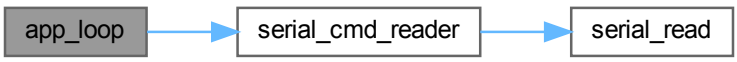
1. La lecture des données série (Polling).
2. Le traitement des commandes (si disponibles).
3. La vérification de sécurité.
4. L'ordonnancement des tâches périodiques basées sur `now_us`.

Note

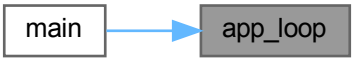
Le compteur `now_us` boucle après environ 71 minutes.

Definition at line 221 of file app_main.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.33.4 Variable Documentation

6.33.4.1 hSpeedo

```
Speedometer_Handle_t hSpeedo
```

Instance du capteur de vitesse (Tachymètre).

Definition at line 69 of file app_main.c.

6.33.4.2 speed_speedo_data

```
float speed_speedo_data = 0.0f
```

Variable globale stockant la vitesse actuelle (partagée avec serial_cmd).

Variable globale de vitesse (m/s) partagée entre le Speedometer et la Télémétrie.

Definition at line 81 of file app_main.c.

6.33.4.3 tim3_overflow_cnt

```
volatile uint32_t tim3_overflow_cnt = 0
```

Compteur de débordements pour le Timer 3 (Extension 16-bit vers 32-bit).

Compteur d'overflow pour l'extension 32-bits du Timer microseconde.

Definition at line 84 of file app_main.c.

6.34 app_main.c

Go to the documentation of this file.

```
00001
00012 #include "main.h"
00013 #include "spi.h"
00014 #include "tim.h"
00015 #include "usart.h"
00016 #include "app_main.h"
00017 #include "driver_ins.h"
00018 #include "bmi08_defs.h"
00019 #include "bmi08.h"
00020 #include "bmi08x.h"
00021 #include "driver_servo.h"
00022 #include "driver_motor.h"
00023 #include "dma.h"
00024 #include "serial.h"
00025 #include "serial_cmd.h"
00026 #include "driver_speedometer.h"
00027 #include <stdio.h>
00028 #include <string.h>
00029 #include <stdbool.h>
00030
00032 #define TASK_MOTOR_US       1000
00034 #define TASK_TELEMETRY_US   10000
00036 #define TASK_SPEED_US       100000
00038 #define FAILSAFE_TIMEOUT_MS 500
00039
00041 #define t_1_ms 3200
00043 #define t_2_ms 6400
00044
```

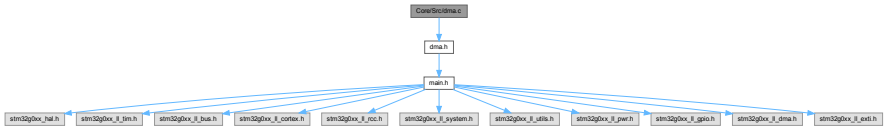
```
00046 #define PWM_MIN_ESC 3200
00048 #define PWM_MAX_ESC 6400
00049
00051 static Servo_Handle_t hServo1 = {
00052     .htim = &htim1,                    // Instance du Timer
00053     .channel = TIM_CHANNEL_1,          // Canal PWM
00054     .min_pulse_ticks = t_1_ms,         // Ticks pour 0% (ex: 1ms)
00055     .max_pulse_ticks = t_2_ms          // Ticks pour 100% (ex: 2ms)
00056 };
00057
00059 static Motor_Handle_t hMotor1 = {
00060     .htim = &htim2,                       // Instance du Timer
00061     .channel = TIM_CHANNEL_1,          // Canal PWM
00062     .min_pulse_ticks = PWM_MIN_ESC,     //
00063     .max_pulse_ticks = PWM_MAX_ESC,    //
00064     .max_speed_pos_mms = 1000,         // Limit max frwd
00065     .max_speed_neg_mms = -500          // Limit max bkwd
00066 };
00067
00069 Speedometer_Handle_t hSpeedo;
00070
00072 static uint32_t last_cmd_time_ms = 0;
00074 static uint32_t last_motor_us = 0;
00076 static uint32_t last_telemetry_us = 0;
00078 static uint32_t last_speed_us = 0;
00079
00081 float speed_speedo_data = 0.0f;
00082
00084 volatile uint32_t tim3_overflow_cnt = 0;
00085
00086 static uint32_t GetMicrosTotal(void);
00087 static void process_incoming_commands(void);
00088 static void check_failsafe_security(void);
00089 static void task_motor_update(uint32_t now_us);
00090 static void task_telemetry_update(uint32_t now_us);
00091 static void task_get_speed(uint32_t now_us);
00092
00099 static uint32_t GetMicrosTotal(void){
00100     uint32_t m_overflow;
00101     uint16_t m_counter;
00102
00103     do{
00104         m_overflow = tim3_overflow_cnt;
00105         m_counter = LL_TIM_GetCounter(TIM3);
00106     }
00107     while(m_overflow != tim3_overflow_cnt);
00108
00109     return (m_overflow « 16) + m_counter;
00110 }
00111
00118 static void process_incoming_commands(void){
00119     if(parser_state == PARSER_IDLE){
00120         return;
00121     }
00122
00123     last_cmd_time_ms = HAL_GetTick();
00124
00125     switch(parser_state){
00126         case PARSER_SERVO_CMD:
00127             servo_pwm_angle_degree(&hServo1, shadow_servo_cmd);
00128             break;
00129
00130         case PARSER_MOTOR_CMD:
00131             motor_set_speed_mms(&hMotor1, shadow_motor_cmd);
00132             break;
00133
00134         default:
00135             break;
00136     }
00137
00138     parser_state = PARSER_IDLE;
00139 }
00140
00146 static void check_failsafe_security(void){
00147     if((HAL_GetTick() - last_cmd_time_ms) > FAILSAFE_TIMEOUT_MS){
00148         motor_set_speed_mms(&hMotor1, 0);
00149     }
00150 }
00151
00157 static void task_motor_update(uint32_t now_us){
00158     if((uint32_t)(now_us - last_motor_us) >= TASK_MOTOR_US){
00159         last_motor_us = now_us;
00160         motor_process_1ms(&hMotor1, HAL_GetTick());
00161     }
00162 }
00163
00169 static void task_telemetry_update(uint32_t now_us){
```

```
00170     if((uint32_t)(now_us - last_telemetry_us) >= TASK_TELEMETRY_US){
00171         last_telemetry_us = now_us;
00172         serial_send_data_frame();
00173     }
00174 }
00175
00181 static void task_get_speed(uint32_t now_us){
00182     if((uint32_t)(now_us - last_speed_us) >= TASK_SPEED_US){
00183         last_speed_us = now_us;
00184         speed_speedo_data = speedometer_solve_speed(&hSpeedo);
00185     }
00186 }
00187
00193 void app_config(void){
00194     LL_TIM_EnableCounter(TIM3);
00195     LL_TIM_EnableIT_UPDATE(TIM3);
00196     HAL_TIM_Base_Start(&htim4);
00197
00198     serial_init();
00199     BMI088_Init(&hspi1);
00200     servo_initialisation(&hServo1);
00201     motor_init(&hMotor1);
00202     motor_pwm_percent(&hMotor1, 50);
00203
00204     last_cmd_time_ms  = HAL_GetTick();
00205     last_motor_us     = GetMicrosTotal();
00206     last_telemetry_us = GetMicrosTotal();
00207     last_speed_us     = GetMicrosTotal();
00208
00209     speedometer_init(&hSpeedo, &htim4);
00210 }
00211
00221 void app_loop(void){
00222     uint32_t now_us = GetMicrosTotal(); //70 minutes max
00223
00224     serial_cmd_reader();
00225
00226     process_incoming_commands();
00227     check_failsafe_security();
00228
00229     task_motor_update(now_us);
00230     task_get_speed(now_us);
00231     task_telemetry_update(now_us);
00232 }
```

6.35 Core/Src/dma.c File Reference

This file provides code for the configuration of all the requested memory to memory DMA transfers.

`#include "dma.h"`
Include dependency graph for dma.c:



Functions

- void MX_DMA_Init (void)

6.35.1 Detailed Description

This file provides code for the configuration of all the requested memory to memory DMA transfers.

Attention

Definition in file dma.c.

6.35.2 Function Documentation

6.35.2.1 MX_DMA_Init()

```
void MX_DMA_Init (
              void  )
```

Enable DMA controller clock

Definition at line 39 of file dma.c.

Here is the caller graph for this function:

6.36 dma.c

Go to the documentation of this file.

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020
00021 /* Includes ------------------------------------------------------------------*/
00022 #include "dma.h"
00023
00024 /* USER CODE BEGIN 0 */
00025
00026 /* USER CODE END 0 */
00027
00028 /*----------------------------------------------------------------------------*/
00029 /* Configure DMA                                                              */
00030 /*----------------------------------------------------------------------------*/
00031
00032 /* USER CODE BEGIN 1 */
00033
00034 /* USER CODE END 1 */
00035
00039 void MX_DMA_Init(void)
00040 {
00041
00042   /* DMA controller clock enable */
00043   __HAL_RCC_DMA1_CLK_ENABLE();
00044
```

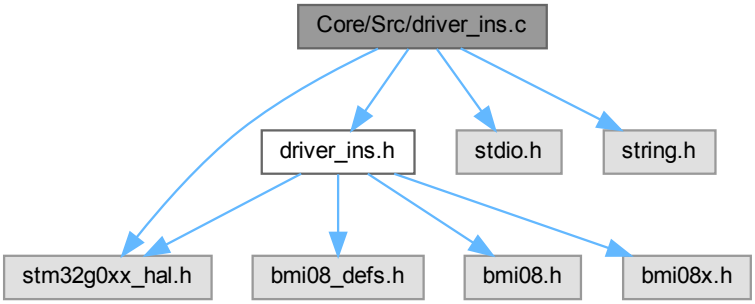
```
00045   /* DMA interrupt init */
00046   /* DMA1_Channel1_IRQn interrupt configuration */
00047   HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 0, 0);
00048   HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);
00049   /* DMA1_Channel2_3_IRQn interrupt configuration */
00050   HAL_NVIC_SetPriority(DMA1_Channel2_3_IRQn, 0, 0);
00051   HAL_NVIC_EnableIRQ(DMA1_Channel2_3_IRQn);
00052
00053 }
00054
00055 /* USER CODE BEGIN 2 */
00056
00057 /* USER CODE END 2 */
00058
```

6.37 Core/Src/driver_ins.c File Reference

Implémentation du pilote pour l'IMU BMI088 (Accéléromètre + Gyroscope).

```
#include "stm32g0xx_hal.h"
#include "driver_ins.h"
#include <stdio.h>
#include <string.h>
```

Include dependency graph for driver_ins.c:



Functions

- int8_t BMI088_Init (SPI_HandleTypeDef *hspi)

  Initialise le module BMI088 (Accéléromètre et Gyroscope).
- int8_t BMI088_Read_Accel_Raw (struct bmi08_sensor_data *accel_data)

  Lit les données brutes de l'accéléromètre.
- int8_t BMI088_Read_Gyro_Raw (struct bmi08_sensor_data *gyro_data)

  Lit les données brutes du gyroscope.
- int8_t BMI088_Read_All (bmi088_data_t *data)

  Lit et convertit l'ensemble des données IMU (Accel + Gyro).
- void BMI088_Convert_Accel (struct bmi08_sensor_data *accel_raw, float *accel_mms2)

  Convertit les données brutes d'accélération en unités physiques.
- void BMI088_Convert_Gyro (struct bmi08_sensor_data *gyro_raw, float *gyro_rads)

  Convertit les données brutes gyroscopiques en unités physiques.
- uint8_t BMI088_Test_Communication (uint8_t print_result)

  Teste la communication SPI en vérifiant les IDs des puces.
- int8_t BMI088_Soft_Reset (void)

  Effectue une réinitialisation logicielle (Soft Reset) des capteurs.

6.37.1 Detailed Description

Implémentation du pilote pour l'IMU BMI088 (Accéléromètre + Gyroscope).

Gère l'initialisation, la communication SPI et la conversion des données brutes en unités physiques via l'API Bosch SensorTec.

Definition in file driver_ins.c.

6.37.2 Function Documentation

6.37.2.1 BMI088_Convert_Accel()

```
void BMI088_Convert_Accel (
            struct bmi08_sensor_data * accel_raw,
            float * accel_mms2 )
```

Convertit les données brutes d'accélération en unités physiques.

Convertit un jeu de données brutes accéléromètre en mm/s^2.

Parameters

accel_raw	Pointeur vers les données brutes d'entrée.
accel_mms2	Pointeur vers le tableau de sortie (x, y, z) en mm/s^2.

Definition at line 261 of file driver_ins.c.

6.37.2.2 BMI088_Convert_Gyro()

```
void BMI088_Convert_Gyro (
            struct bmi08_sensor_data * gyro_raw,
            float * gyro_rads )
```

Convertit les données brutes gyroscopiques en unités physiques.

Convertit un jeu de données brutes gyroscope en rad/s.

Parameters

gyro_raw	Pointeur vers les données brutes d'entrée.
gyro_rads	Pointeur vers le tableau de sortie (x, y, z) en rad/s.

Definition at line 276 of file driver_ins.c.

6.37.2.3 BMI088_Init()

```
int8_t BMI088_Init (
            SPI_HandleTypeDef * hspi )
```

Initialise le module BMI088 (Accéléromètre et Gyroscope).

Initialise le driver BMI088.

Parameters

hspi	Pointeur vers le handle SPI STM32.

Returns

BMI08_OK si l'initialisation réussit, code d'erreur sinon.

Definition at line 133 of file driver_ins.c.

Here is the caller graph for this function:

6.37.2.4 BMI088_Read_Accel_Raw()

```
int8_t BMI088_Read_Accel_Raw (
            struct bmi08_sensor_data * accel_data )
```

Lit les données brutes de l'accéléromètre.

Lit les registres bruts de l'accéléromètre.

Parameters

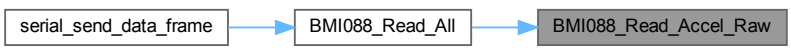
accel_data	Pointeur vers la structure de destination des données brutes.

Returns

BMI08_OK en cas de succès, ou code d'erreur.

Definition at line 200 of file driver_ins.c.

Here is the caller graph for this function:



6.37.2.5 BMI088_Read_All()

```
int8_t BMI088_Read_All (
            bmi088_data_t * data )
```

Lit et convertit l'ensemble des données IMU (Accel + Gyro).

Effectue une lecture complète et convertit les données.

Parameters

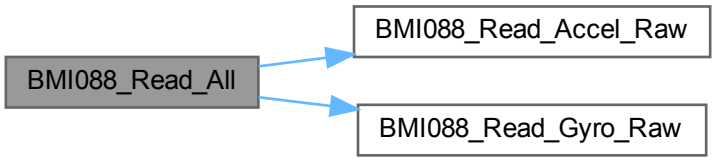
data	Pointeur vers la structure de données utilisateur (unités physiques).

Returns

BMI08_OK en cas de succès, ou code d'erreur.

Definition at line 226 of file driver_ins.c.

Here is the call graph for this function:



Here is the caller graph for this function:

6.37.2.6 BMI088_Read_Gyro_Raw()

```
int8_t BMI088_Read_Gyro_Raw (
            struct bmi08_sensor_data * gyro_data )
```

Lit les données brutes du gyroscope.

Lit les registres bruts du gyroscope.

Parameters

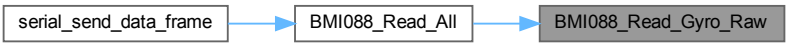
gyro_data	Pointeur vers la structure de destination des données brutes.

Returns

BMI08_OK en cas de succès, ou code d'erreur.

Definition at line 213 of file driver_ins.c.

Here is the caller graph for this function:



6.37.2.7 BMI088_Soft_Reset()

```
int8_t BMI088_Soft_Reset (
            void  )
```

Effectue une réinitialisation logicielle (Soft Reset) des capteurs.

Redémarre les capteurs via commande logicielle.

Returns

BMI08_OK en cas de succès, ou code d'erreur SPI.

Definition at line 314 of file driver_ins.c.

6.37.2.8 BMI088_Test_Communication()

```
uint8_t BMI088_Test_Communication (
            uint8_t print_result )
```

Teste la communication SPI en vérifiant les IDs des puces.

Vérifie la présence des capteurs sur le bus SPI.

Parameters

print_result	Si non nul, affiche le résultat sur la console de debug.

Returns

1 si la communication est établie avec succès, 0 sinon.

Definition at line 291 of file driver_ins.c.

6.38 driver_ins.c

Go to the documentation of this file.

```
00001
00008 #include "stm32g0xx_hal.h"
00009 #include "driver_ins.h"
00010 #include <stdio.h>
00011 #include <string.h>
00012
00014 static struct bmi08_dev bmi088_dev;
00015
00017 static SPI_HandleTypeDef *bmi088_hspi;
00018
00020 static bmi088_cs_t cs_accel = {
00021     .port = BMI088_CS_ACC_GPIO_Port,
00022     .pin  = BMI088_CS_ACC_Pin
00023 };
00024
00026 static bmi088_cs_t cs_gyro = {
00027     .port = BMI088_CS_GYRO_GPIO_Port,
00028     .pin  = BMI088_CS_GYRO_Pin
00029 };
00030
00041 static int8_t bmi088_spi_read(uint8_t reg_addr, uint8_t *reg_data, uint32_t len, void *intf_ptr){
00042     if (reg_data == NULL || intf_ptr == NULL) return BMI08_E_NULL_PTR;
00043     if (len == 0 || len > BMI08_MAX_LEN) return BMI08_E_RD_WR_LENGTH_INVALID;
00044
00045     bmi088_cs_t *cs = (bmi088_cs_t*)intf_ptr;
00046     HAL_StatusTypeDef status;
00047
00048     HAL_GPIO_WritePin(cs->port, cs->pin, GPIO_PIN_RESET);
00049
00050     if(cs->pin == BMI088_CS_ACC_Pin){
00051         uint8_t tx_buf[256] = {0};
00052         uint8_t rx_buf[256] = {0};
00053
00054         tx_buf[0] = reg_addr | 0x80;
00055         tx_buf[1] = 0x00;
00056
00057         status = HAL_SPI_TransmitReceive(bmi088_hspi, tx_buf, rx_buf, len + 2, HAL_MAX_DELAY);
00058
00059         /*
00060         for(uint32_t i = 0; i < len + 2; i++){
00061             printf("REG RX i : %d : %02X\r\n", i, rx_buf[i]);
00062         }
00063         */
00064
00065         HAL_GPIO_WritePin(cs->port, cs->pin, GPIO_PIN_SET);
00066
00067         if(status != HAL_OK) return BMI08_E_COM_FAIL;
00068
00069         for(uint32_t i = 0; i < len; i++){
00070             reg_data[i] = rx_buf[i + 1];
00071         }
00072     }
00073     else{
00074         uint8_t tx_buf[256] = {0};
00075         uint8_t rx_buf[256] = {0};
00076
00077         tx_buf[0] = reg_addr | 0x80;
00078
00079         status = HAL_SPI_TransmitReceive(bmi088_hspi, tx_buf, rx_buf, len + 1, HAL_MAX_DELAY);
00080
00081         HAL_GPIO_WritePin(cs->port, cs->pin, GPIO_PIN_SET);
00082
00083         if(status != HAL_OK) return BMI08_E_COM_FAIL;
00084
00085         for(uint32_t i = 0; i < len; i++){
00086             reg_data[i] = rx_buf[i + 1];
00087         }
00088     }
00089
00090     return BMI08_OK;
```

```
00091 }
00092
00101 static int8_t bmi088_spi_write(uint8_t reg_addr, const uint8_t *reg_data, uint32_t len, void
      *intf_ptr){
00102     bmi088_cs_t *cs = (bmi088_cs_t*)intf_ptr;
00103
00104     HAL_GPIO_WritePin(cs->port, cs->pin, GPIO_PIN_RESET);
00105
00106     uint8_t addr = reg_addr & 0x7F;
00107     HAL_SPI_Transmit(bmi088_hspi, &addr, 1, HAL_MAX_DELAY);
00108
00109     HAL_SPI_Transmit(bmi088_hspi, (uint8_t*)reg_data, len, HAL_MAX_DELAY);
00110
00111     HAL_GPIO_WritePin(cs->port, cs->pin, GPIO_PIN_SET);
00112
00113     return BMI08_OK;
00114 }
00115
00122 static void bmi088_delay_us(uint32_t period, void *intf_ptr){
00123     uint32_t delay_ms = period / 1000;
00124     if(delay_ms == 0) delay_ms = 1;
00125     HAL_Delay(delay_ms);
00126 }
00127
00133 int8_t BMI088_Init(SPI_HandleTypeDef *hspi){
00134     if(hspi == NULL){
00135         return BMI08_E_NULL_PTR;
00136     }
00137
00138     bmi088_hspi = hspi;
00139
00140     HAL_GPIO_WritePin(BMI088_CS_ACC_GPIO_Port, BMI088_CS_ACC_Pin, GPIO_PIN_SET);
00141     HAL_GPIO_WritePin(BMI088_CS_GYRO_GPIO_Port, BMI088_CS_GYRO_Pin, GPIO_PIN_SET);
00142
00143     bmi088_dev.intf = BMI08_SPI_INTF;
00144     bmi088_dev.read = bmi088_spi_read;
00145     bmi088_dev.write = bmi088_spi_write;
00146     bmi088_dev.delay_us = bmi088_delay_us;
00147     bmi088_dev.intf_ptr_accel = &cs_accel;
00148     bmi088_dev.intf_ptr_gyro = &cs_gyro;
00149
00150     int8_t rslt_accel = bmi08a_init(&bmi088_dev);
00151
00152 #ifdef DEBUG
00153     //printf("BMI088 Init Accel: %d (0=OK)\r\n", rslt_accel);
00154     //printf("Accel Chip ID: 0x%02X (attendu: 0x1E)\r\n", bmi088_dev.accel_chip_id);
00155 #endif
00156
00157     int8_t rslt_gyro = bmi08g_init(&bmi088_dev);
00158
00159 #ifdef DEBUG
00160     //printf("BMI088 Init Gyro: %d (0=OK)\r\n", rslt_gyro);
00161     //printf("Gyro Chip ID: 0x%02X (attendu: 0x0F)\r\n", bmi088_dev.gyro_chip_id);
00162 #endif
00163
00164     if(rslt_accel != BMI08_OK || rslt_gyro != BMI08_OK){
00165         return BMI08_E_DEV_NOT_FOUND;
00166     }
00167
00168     bmi088_dev.accel_cfg.odr   = BMI08_ACCEL_ODR_100_HZ;
00169     bmi088_dev.accel_cfg.range = BMI088_ACCEL_RANGE_6G;
00170     bmi088_dev.accel_cfg.bw    = BMI08_ACCEL_BW_NORMAL;
00171     bmi088_dev.accel_cfg.power = BMI08_ACCEL_PM_ACTIVE;
00172
00173     rslt_accel  = bmi08a_set_power_mode(&bmi088_dev);
00174     rslt_accel |= bmi08a_set_meas_conf(&bmi088_dev);
00175
00176     bmi088_dev.gyro_cfg.odr   = BMI08_GYRO_BW_23_ODR_200_HZ;
00177     bmi088_dev.gyro_cfg.range = BMI08_GYRO_RANGE_1000_DPS;
00178     bmi088_dev.gyro_cfg.bw    = BMI08_GYRO_BW_23_ODR_200_HZ;
00179     bmi088_dev.gyro_cfg.power = BMI08_GYRO_PM_NORMAL;
00180
00181     rslt_gyro  = bmi08g_set_power_mode(&bmi088_dev);
00182     rslt_gyro |= bmi08g_set_meas_conf(&bmi088_dev);
00183
00184     if(rslt_accel != BMI08_OK || rslt_gyro != BMI08_OK){
00185         return BMI08_E_COM_FAIL;
00186     }
00187
00188 #ifdef DEBUG
00189     //printf("BMI088 initialisé avec succès!\r\n");
00190 #endif
00191
00192     return BMI08_OK;
00193 }
00194
00200 int8_t BMI088_Read_Accel_Raw(struct bmi08_sensor_data *accel_data){
```

```
00201     if(accel_data == NULL){
00202         return BMI08_E_NULL_PTR;
00203     }
00204 
00205     return bmi08a_get_data(accel_data, &bmi088_dev);
00206 }
00207 
00213 int8_t BMI088_Read_Gyro_Raw(struct bmi08_sensor_data *gyro_data){
00214     if(gyro_data == NULL){
00215         return BMI08_E_NULL_PTR;
00216     }
00217 
00218     return bmi08g_get_data(gyro_data, &bmi088_dev);
00219 }
00220 
00226 int8_t BMI088_Read_All(bmi088_data_t *data){
00227     if(data == NULL){
00228         return BMI08_E_NULL_PTR;
00229     }
00230 
00231     struct bmi08_sensor_data accel_raw, gyro_raw;
00232 
00233     int8_t rslt = BMI088_Read_Accel_Raw(&accel_raw);
00234     if(rslt != BMI08_OK){
00235         return rslt;
00236     }
00237 
00238     rslt = BMI088_Read_Gyro_Raw(&gyro_raw);
00239     if(rslt != BMI08_OK){
00240         return rslt;
00241     }
00242 
00243     data->accel_x_mms2 = ((float)accel_raw.x / ACCEL_RANGE_6G_LSB) * G_TO_MM_S2;
00244     data->accel_y_mms2 = ((float)accel_raw.y / ACCEL_RANGE_6G_LSB) * G_TO_MM_S2;
00245     data->accel_z_mms2 = ((float)accel_raw.z / ACCEL_RANGE_6G_LSB) * G_TO_MM_S2;
00246 
00247     data->gyro_x_rads = ((float)gyro_raw.x / GYRO_RANGE_1000DPS_LSB) * DEG_TO_RAD;
00248     data->gyro_y_rads = ((float)gyro_raw.y / GYRO_RANGE_1000DPS_LSB) * DEG_TO_RAD;
00249     data->gyro_z_rads = ((float)gyro_raw.z / GYRO_RANGE_1000DPS_LSB) * DEG_TO_RAD;
00250 
00251     data->timestamp_ms = HAL_GetTick();
00252 
00253     return BMI08_OK;
00254 }
00255 
00261 void BMI088_Convert_Accel(struct bmi08_sensor_data *accel_raw, float *accel_mms2){
00262     if(accel_raw == NULL || accel_mms2 == NULL){
00263         return;
00264     }
00265 
00266     accel_mms2[0] = ((float)accel_raw->x / ACCEL_RANGE_6G_LSB) * G_TO_MM_S2;
00267     accel_mms2[1] = ((float)accel_raw->y / ACCEL_RANGE_6G_LSB) * G_TO_MM_S2;
00268     accel_mms2[2] = ((float)accel_raw->z / ACCEL_RANGE_6G_LSB) * G_TO_MM_S2;
00269 }
00270 
00276 void BMI088_Convert_Gyro(struct bmi08_sensor_data *gyro_raw, float *gyro_rads){
00277     if(gyro_raw == NULL || gyro_rads == NULL){
00278         return;
00279     }
00280 
00281     gyro_rads[0] = ((float)gyro_raw->x / GYRO_RANGE_1000DPS_LSB) * DEG_TO_RAD;
00282     gyro_rads[1] = ((float)gyro_raw->y / GYRO_RANGE_1000DPS_LSB) * DEG_TO_RAD;
00283     gyro_rads[2] = ((float)gyro_raw->z / GYRO_RANGE_1000DPS_LSB) * DEG_TO_RAD;
00284 }
00285 
00291 uint8_t BMI088_Test_Communication(uint8_t print_result){
00292     uint8_t accel_ok = (bmi088_dev.accel_chip_id == BMI088_ACCEL_CHIP_ID);
00293     uint8_t gyro_ok = (bmi088_dev.gyro_chip_id == BMI08_GYRO_CHIP_ID);
00294 
00295     if(print_result){
00296 
00297         /*
00298         printf("=== BMI088 Communication Test ===\r\n");
00299         printf("Accelerometer: %s (ID: 0x%02X)\r\n",
00300                accel_ok ? "OK" : "FAIL", bmi088_dev.accel_chip_id);
00301         printf("Gyroscope: %s (ID: 0x%02X)\r\n",
00302                gyro_ok ? "OK" : "FAIL", bmi088_dev.gyro_chip_id);
00303         printf("================================\r\n");
00304         */
00305     }
00306 
00307     return (accel_ok && gyro_ok);
00308 }
00309 
00314 int8_t BMI088_Soft_Reset(void){
00315     uint8_t soft_reset_cmd = BMI08_SOFT_RESET_CMD;
00316     int8_t rslt = bmi088_spi_write(BMI08_REG_ACCEL_SOFTRESET, &soft_reset_cmd, 1, &cs_accel);
```

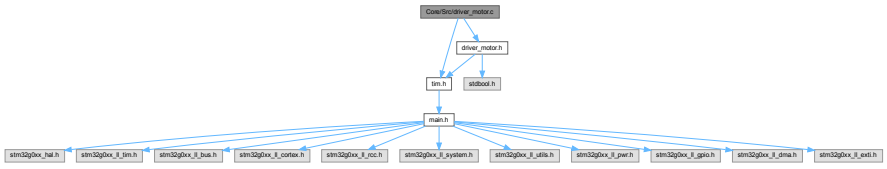
```
00317
00318     HAL_Delay(50);
00319
00320     rslt |= bmi088_spi_write(BMI08_REG_GYRO_SOFTRESET, &soft_reset_cmd, 1, &cs_gyro);
00321
00322     HAL_Delay(50);
00323
00324     return rslt;
00325 }
```

6.39 Core/Src/driver_motor.c File Reference

Pilote de moteur non-bloquant pour ESC (Electronic Speed Controller).

```
#include "driver_motor.h"
#include "tim.h"
```

Include dependency graph for driver_motor.c:



Macros

- #define T_BRAKE_MS 120

  Durée de l'état de freinage actif lors d'une inversion de sens (ms).
- #define T_NEUTRAL_GAP_MS 120

  Durée de pause au point mort après un freinage (ms).
- #define PWM_NEUTRAL 50u

  Rapport cyclique (%) correspondant au point mort (arrêt).
- #define PWM_BRAKE_REV 40u

  Rapport cyclique (%) pour le freinage en marche arrière.
- #define PWM_BRAKE_FWD 60u

  Rapport cyclique (%) pour le freinage en marche avant.

Functions

- void motor_init (Motor_Handle_t *hmotor)

  Initialise le moteur et sa machine à états.
- void motor_pwm_percent (Motor_Handle_t *hmotor, uint8_t percent)

  Force une commande PWM directe en pourcentage.
- void motor_set_speed_mms (Motor_Handle_t *hmotor, int16_t speed_mms)

  Définit la nouvelle consigne de vitesse cible.
- void motor_process_1ms (Motor_Handle_t *hmotor, uint32_t now_ms)

  Machine à états principale de gestion du moteur.

6.39.1 Detailed Description

Pilote de moteur non-bloquant pour ESC (Electronic Speed Controller).

Implémente une machine à états finis pour gérer les transitions de sécurité du moteur (Accélération, Freinage, Passage au neutre, Inversion de sens) sans bloquer le processeur.

Definition in file driver_motor.c.

6.39.2 Macro Definition Documentation

6.39.2.1 PWM_BRAKE_FWD

```
#define PWM_BRAKE_FWD 60u
```

Rapport cyclique (%) pour le freinage en marche avant.

Definition at line 23 of file driver_motor.c.

6.39.2.2 PWM_BRAKE_REV

```
#define PWM_BRAKE_REV 40u
```

Rapport cyclique (%) pour le freinage en marche arrière.

Definition at line 21 of file driver_motor.c.

6.39.2.3 PWM_NEUTRAL

```
#define PWM_NEUTRAL 50u
```

Rapport cyclique (%) correspondant au point mort (arrêt).

Definition at line 19 of file driver_motor.c.

6.39.2.4 T_BRAKE_MS

```
#define T_BRAKE_MS 120
```

Durée de l'état de freinage actif lors d'une inversion de sens (ms).

Definition at line 15 of file driver_motor.c.

6.39.2.5 T_NEUTRAL_GAP_MS

```
#define T_NEUTRAL_GAP_MS 120
```

Durée de pause au point mort après un freinage (ms).

Definition at line 17 of file driver_motor.c.

6.39.3 Function Documentation

6.39.3.1 motor_init()

```
void motor_init (
            Motor_Handle_t * hmotor )
```

Initialise le moteur et sa machine à états.

Initialise le driver moteur.

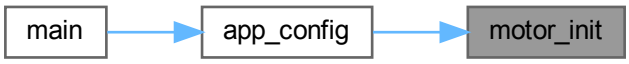
Place le moteur au neutre, réinitialise le contexte de commande et démarre la génération PWM hardware.

Parameters

hmotor	Pointeur vers le handle du moteur à initialiser.

Definition at line 88 of file driver_motor.c.

Here is the caller graph for this function:



6.39.3.2 motor_process_1ms()

```
void motor_process_1ms (
            Motor_Handle_t * hmotor,
            uint32_t now_ms )
```

Machine à états principale de gestion du moteur.

Fonction de traitement cyclique (Machine à états).

Doit être appelée périodiquement (ex: 1kHz). Gère la logique séquentielle :

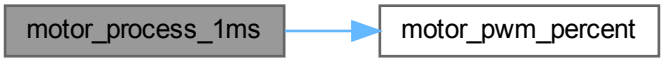
- Marche Avant -> Neutre -> Marche Arrière : Passage direct (géré par ESC en général) ou via frein.
- Inversion brusque : Application d'une séquence Frein -> Pause -> Nouveau sens.

Parameters

hmotor	Pointeur vers le handle du moteur.
now_ms	Temps système actuel en millisecondes.

Definition at line 140 of file driver_motor.c.

Here is the call graph for this function:



6.39.3.3 motor_pwm_percent()

```
void motor_pwm_percent (
            Motor_Handle_t * hmotor,
            uint8_t percent )
```

Force une commande PWM directe en pourcentage.

Applique directement un pourcentage de PWM (Bypass FSM).

Note

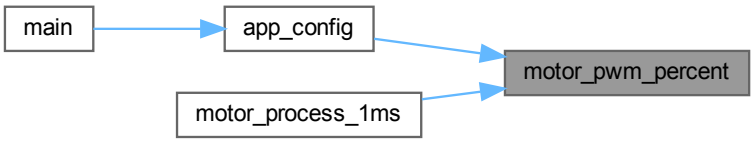
Utilisé principalement par la machine à états pour appliquer les commandes de freinage ou de neutre.

Parameters

hmotor	Pointeur vers le handle du moteur.
percent	Pourcentage PWM cible (0-100).

Definition at line 108 of file driver_motor.c.

Here is the caller graph for this function:



6.39.3.4 motor_set_speed_mms()

```
void motor_set_speed_mms (
            Motor_Handle_t * hmotor,
            int16_t speed_mms )
```

Définit la nouvelle consigne de vitesse cible.

Définit la consigne de vitesse en mm/s.

Met à jour le contexte cible. La machine à états (motor_process_1ms) se chargera d'atteindre cette cible en respectant les séquences de freinage.

Parameters

hmotor	Pointeur vers le handle du moteur.
speed_mms	Vitesse cible en mm/s (Positif = Avant, Négatif = Arrière).

Definition at line 120 of file driver_motor.c.

6.40 driver_motor.c

Go to the documentation of this file.

```
00001
00009 #include "driver_motor.h"
00010 #include "tim.h"
00011
00012 // 64Mhz - PSC=19 - ARR=63999 soit PWM{50Hz, duty=50%}
00013
00015 #define T_BRAKE_MS          120
00017 #define T_NEUTRAL_GAP_MS    120
00019 #define PWM_NEUTRAL         50u
00021 #define PWM_BRAKE_REV       40u
00023 #define PWM_BRAKE_FWD       60u
00024
00025 static uint8_t motor_speed_mms_to_pwm_percent(Motor_Handle_t *hmotor, int16_t value);
00026
00032 static inline void pwm_pulse(Motor_Handle_t *hmotor, uint16_t value){
00033     if (hmotor && hmotor->htim) {
00034         __HAL_TIM_SET_COMPARE(hmotor->htim, hmotor->channel, value);
00035     }
00036 }
00037
00044 static inline uint16_t motor_map_percent(Motor_Handle_t *hmotor, int16_t percent){
00045     if (percent < 0)   percent = 0;
00046     if (percent > 100) percent = 100;
00047
00048     const uint16_t min = hmotor->min_pulse_ticks;
00049     const uint16_t max = hmotor->max_pulse_ticks;
00050
00051     return (uint16_t)(min + ((uint32_t)(max - min) * (uint32_t)percent) / 100u);
00052 }
00053
00061 static uint8_t motor_speed_mms_to_pwm_percent(Motor_Handle_t *hmotor, int16_t value){
00062     if (value >= hmotor->max_speed_pos_mms) return 100;
00063     if (value <= hmotor->max_speed_neg_mms) return 0;
00064
00065     if (value >= 0)
00066         return (uint8_t)(50 + ((int32_t)value * 50) / hmotor->max_speed_pos_mms);
00067     else
00068         return (uint8_t)(50 + ((int32_t)value * 50) / (-hmotor->max_speed_neg_mms));
00069 }
00070
00078 static inline bool time_reached(uint32_t now, uint32_t deadline){
00079     return (int32_t)(now - deadline) >= 0;
00080 }
00081
00088 void motor_init(Motor_Handle_t *hmotor){
00089     if(hmotor && hmotor->htim){
00090         hmotor->state = MOTOR_STATE_NEUTRAL;
00091         hmotor->go_forward = true;
00092         hmotor->ctx.target_speed_mms = 0;
00093         hmotor->ctx.target_pwm = PWM_NEUTRAL;
00094         hmotor->ctx.target_forward = true;
00095         hmotor->ctx.deadline_ms = 0;
00096
00097         HAL_TIM_PWM_Start(hmotor->htim, hmotor->channel);
00098     }
00099 }
00100
00108 void motor_pwm_percent(Motor_Handle_t *hmotor, uint8_t percent){
00109     uint16_t value = motor_map_percent(hmotor, (int16_t)percent);
00110     pwm_pulse(hmotor, value);
00111 }
00112
00120 void motor_set_speed_mms(Motor_Handle_t *hmotor, int16_t speed_mms){
00121     hmotor->ctx.target_speed_mms = speed_mms;
00122
00123     if(speed_mms == 0){
00124         hmotor->ctx.target_pwm = PWM_NEUTRAL;
00125     }
00126     else{
00127         hmotor->ctx.target_pwm = motor_speed_mms_to_pwm_percent(hmotor, speed_mms);
00128         hmotor->ctx.target_forward = (speed_mms > 0);
00129     }
00130 }
00131
00140 void motor_process_1ms(Motor_Handle_t *hmotor, uint32_t now_ms){
00141     if (!hmotor) return;
```

```
00142
00143     const bool want_neutral = (hmotor->ctx.target_speed_mms == 0);
00144
00145     switch (hmotor->state){
00146         case MOTOR_STATE_NEUTRAL:
00147             motor_pwm_percent(hmotor, PWM_NEUTRAL);
00148
00149             if(!want_neutral){
00150                 if(hmotor->ctx.target_forward){
00151                     motor_pwm_percent(hmotor, hmotor->ctx.target_pwm);
00152                     hmotor->go_forward = true;
00153                     hmotor->state = MOTOR_STATE_FORWARD_HOLD;
00154                 }
00155                 else{
00156                     motor_pwm_percent(hmotor, PWM_BRAKE_REV);
00157                     hmotor->ctx.deadline_ms = now_ms + T_BRAKE_MS;
00158                     hmotor->state = MOTOR_STATE_NEUTRAL_TO_REVERSE_TAP;
00159                 }
00160             }
00161             break;
00162
00163         case MOTOR_STATE_NEUTRAL_TO_REVERSE_TAP:
00164             if (!time_reached(now_ms, hmotor->ctx.deadline_ms)) break;
00165             motor_pwm_percent(hmotor, PWM_NEUTRAL);
00166             hmotor->ctx.deadline_ms = now_ms + T_NEUTRAL_GAP_MS;
00167             hmotor->state = MOTOR_STATE_NEUTRAL_TO_REVERSE_GAP;
00168             break;
00169
00170         case MOTOR_STATE_NEUTRAL_TO_REVERSE_GAP:
00171             if(!time_reached(now_ms, hmotor->ctx.deadline_ms)) break;
00172             if(want_neutral){
00173                 motor_pwm_percent(hmotor, PWM_NEUTRAL);
00174                 hmotor->state = MOTOR_STATE_NEUTRAL;
00175             }
00176             else if (hmotor->ctx.target_forward){
00177                 motor_pwm_percent(hmotor, hmotor->ctx.target_pwm);
00178                 hmotor->go_forward = true;
00179                 hmotor->state = MOTOR_STATE_FORWARD_HOLD;
00180             }
00181             else{
00182                 motor_pwm_percent(hmotor, hmotor->ctx.target_pwm);
00183                 hmotor->go_forward = false;
00184                 hmotor->state = MOTOR_STATE_REVERSE_HOLD;
00185             }
00186             break;
00187
00188         case MOTOR_STATE_FORWARD_HOLD:
00189             if(want_neutral){
00190                 motor_pwm_percent(hmotor, PWM_NEUTRAL);
00191                 hmotor->state = MOTOR_STATE_NEUTRAL;
00192                 break;
00193             }
00194             if(hmotor->ctx.target_forward){
00195                 motor_pwm_percent(hmotor, hmotor->ctx.target_pwm);
00196             }
00197             else{
00198                 motor_pwm_percent(hmotor, PWM_BRAKE_REV);
00199                 hmotor->ctx.deadline_ms = now_ms + T_BRAKE_MS;
00200                 hmotor->state = MOTOR_STATE_FWD_BRAKE_TAP;
00201             }
00202             break;
00203
00204         case MOTOR_STATE_FWD_BRAKE_TAP:
00205             if(!time_reached(now_ms, hmotor->ctx.deadline_ms)) break;
00206             motor_pwm_percent(hmotor, PWM_NEUTRAL);
00207             hmotor->ctx.deadline_ms = now_ms + T_NEUTRAL_GAP_MS;
00208             hmotor->state = MOTOR_STATE_FWD_NEUTRAL_GAP;
00209             break;
00210
00211         case MOTOR_STATE_FWD_NEUTRAL_GAP:
00212             if(!time_reached(now_ms, hmotor->ctx.deadline_ms)) break;
00213             if(want_neutral){
00214                 motor_pwm_percent(hmotor, PWM_NEUTRAL);
00215                 hmotor->state = MOTOR_STATE_NEUTRAL;
00216             }
00217             else if(hmotor->ctx.target_forward){
00218                 motor_pwm_percent(hmotor, hmotor->ctx.target_pwm);
00219                 hmotor->go_forward = true;
00220                 hmotor->state = MOTOR_STATE_FORWARD_HOLD;
00221             }
00222             else{
00223                 motor_pwm_percent(hmotor, hmotor->ctx.target_pwm);
00224                 hmotor->go_forward = false;
00225                 hmotor->state = MOTOR_STATE_REVERSE_HOLD;
00226             }
00227             break;
00228
```

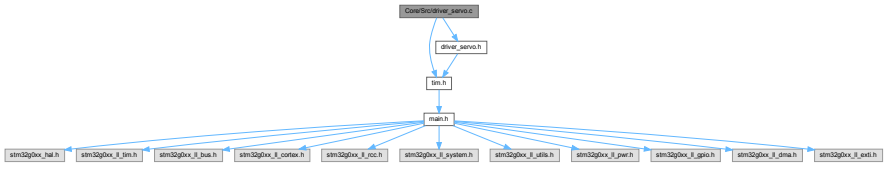
```
00229         case MOTOR_STATE_REVERSE_HOLD:
00230             if(want_neutral){
00231                 motor_pwm_percent(hmotor, PWM_NEUTRAL);
00232                 hmotor->state = MOTOR_STATE_NEUTRAL;
00233                 break;
00234             }
00235             if(!hmotor->ctx.target_forward){
00236                 motor_pwm_percent(hmotor, hmotor->ctx.target_pwm);
00237             }
00238             else{
00239                 motor_pwm_percent(hmotor, PWM_BRAKE_FWD);
00240                 hmotor->ctx.deadline_ms = now_ms + T_BRAKE_MS;
00241                 hmotor->state = MOTOR_STATE_REV_BRAKE_TAP;
00242             }
00243             break;
00244 
00245         case MOTOR_STATE_REV_BRAKE_TAP:
00246             if(!time_reached(now_ms, hmotor->ctx.deadline_ms)) break;
00247             motor_pwm_percent(hmotor, PWM_NEUTRAL);
00248             hmotor->ctx.deadline_ms = now_ms + T_NEUTRAL_GAP_MS;
00249             hmotor->state = MOTOR_STATE_REV_NEUTRAL_GAP;
00250             break;
00251 
00252         case MOTOR_STATE_REV_NEUTRAL_GAP:
00253             if(!time_reached(now_ms, hmotor->ctx.deadline_ms)) break;
00254             if(want_neutral){
00255                 motor_pwm_percent(hmotor, PWM_NEUTRAL);
00256                 hmotor->state = MOTOR_STATE_NEUTRAL;
00257             }
00258             else if(!hmotor->ctx.target_forward){
00259                 motor_pwm_percent(hmotor, hmotor->ctx.target_pwm);
00260                 hmotor->go_forward = false;
00261                 hmotor->state = MOTOR_STATE_REVERSE_HOLD;
00262             }
00263             else{
00264                 motor_pwm_percent(hmotor, hmotor->ctx.target_pwm);
00265                 hmotor->go_forward = true;
00266                 hmotor->state = MOTOR_STATE_FORWARD_HOLD;
00267             }
00268             break;
00269     }
00270 }
```

6.41 Core/Src/driver_servo.c File Reference

Pilote PWM pour le servomoteur de direction.

```
#include "tim.h"
#include "driver_servo.h"
```

Include dependency graph for driver_servo.c:



Macros

- #define SERVO_OFFSET_PERCENT 5

  Décalage (offset) en pourcentage appliqué à la commande (Trim).
- #define SERVO_CLAMP_MIN -20

  Angle minimum autorisé en degrés (Borne mécanique logicielle).
- #define SERVO_CLAMP_MAX 20

  Angle maximum autorisé en degrés (Borne mécanique logicielle).

Functions

- void servo_pwm_percent (Servo_Handle_t *hservo, uint8_t percent)

  Commande le servo via un pourcentage (0 à 100%).
- void servo_pwm_angle_degree (Servo_Handle_t *hservo, int8_t angle)

  Commande le servo via un angle en degrés.
- void servo_pwm_angle_abs_value (Servo_Handle_t *hservo, uint16_t abs_value)

  Commande le servo via une valeur absolue haute résolution (0-65535).
- void servo_initialisation (Servo_Handle_t *hservo)

  Initialise le driver Servo.

6.41.1 Detailed Description

Pilote PWM pour le servomoteur de direction.

Fournit les fonctions de conversion (Degrés/Pourcentage/Absolu vers PWM) et de gestion des limites mécaniques pour le servo.

Definition in file driver_servo.c.

6.41.2 Macro Definition Documentation

6.41.2.1 SERVO_CLAMP_MAX

```
#define SERVO_CLAMP_MAX 20
```

Angle maximum autorisé en degrés (Borne mécanique logicielle).

Definition at line 16 of file driver_servo.c.

6.41.2.2 SERVO_CLAMP_MIN

```
#define SERVO_CLAMP_MIN -20
```

Angle minimum autorisé en degrés (Borne mécanique logicielle).

Definition at line 14 of file driver_servo.c.

6.41.2.3 SERVO_OFFSET_PERCENT

```
#define SERVO_OFFSET_PERCENT 5
```

Décalage (offset) en pourcentage appliqué à la commande (Trim).

Definition at line 12 of file driver_servo.c.

6.41.3 Function Documentation

6.41.3.1 servo_initialisation()

```
void servo_initialisation (
            Servo_Handle_t * hservo )
```

Initialise le driver Servo.

Initialise le servo-moteur.

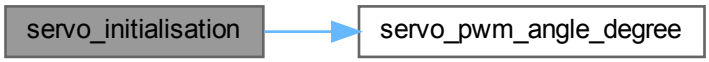
Positionne le servo à 0 degrés (neutre) et active le canal PWM.

Parameters

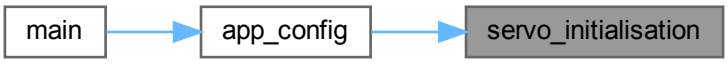
hservo	Pointeur vers le handle du servo.

Definition at line 122 of file driver_servo.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.41.3.2 servo_pwm_angle_abs_value()

```
void servo_pwm_angle_abs_value (
            Servo_Handle_t * hservo,
            uint16_t abs_value )
```

Commande le servo via une valeur absolue haute résolution (0-65535).

Commande le servo via une valeur absolue (Haute résolution).

Effectue une double conversion : Entrée -> Centi-degrés -> Ticks PWM. Gère également l'offset (Trim) et les limites physiques du timer.

Parameters

hservo	Pointeur vers le handle du servo.
abs_value	Valeur absolue normalisée (0 à 65535).

Definition at line 97 of file driver_servo.c.

6.41.3.3 servo_pwm_angle_degree()

```
void servo_pwm_angle_degree (
            Servo_Handle_t * hservo,
            int8_t angle )
```

Commande le servo via un angle en degrés.

Commande le servo en degrés.

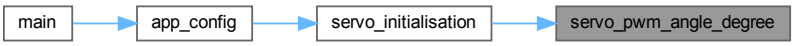
Applique un bornage de sécurité (CLAMP_MIN / CLAMP_MAX) puis convertit l'angle en pourcentage pour le PWM.

Parameters

hservo	Pointeur vers le handle du servo.
angle	Angle cible en degrés.

Definition at line 80 of file driver_servo.c.

Here is the caller graph for this function:



6.41.3.4 servo_pwm_percent()

```
void servo_pwm_percent (
            Servo_Handle_t * hservo,
            uint8_t percent )
```

Commande le servo via un pourcentage (0 à 100%).

Commande le servo en pourcentage (0-100%).

Parameters

hservo	Pointeur vers le handle du servo.
percent	Position cible en pourcentage.

Definition at line 68 of file driver_servo.c.

6.42 driver_servo.c

Go to the documentation of this file.

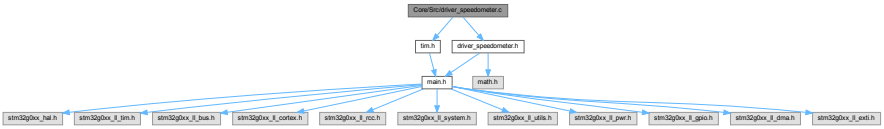
```
00001
00008 #include "tim.h"
00009 #include "driver_servo.h"
00010
00012 #define SERVO_OFFSET_PERCENT  5
00014 #define SERVO_CLAMP_MIN       -20
00016 #define SERVO_CLAMP_MAX       20
00017
00018 //static inline void pwm_pulse(Servo_Handle_t *hservo, uint16_t value);
00019 static int32_t map(int32_t x, int32_t in_min, int32_t in_max, int32_t out_min, int32_t out_max);
00020
00026 static inline void pwm_pulse(Servo_Handle_t *hservo, uint16_t value){
00027     if (hservo && hservo->htim) {
00028         __HAL_TIM_SET_COMPARE(hservo->htim, hservo->channel, value);
00029     }
00030 }
00031
00041 static int32_t map(int32_t x, int32_t in_min, int32_t in_max, int32_t out_min, int32_t out_max){
00042     return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
00043 }
00044
00052 static inline uint16_t servo_map_percent(Servo_Handle_t *hservo, int16_t percent){
00053     int16_t corrected = percent + SERVO_OFFSET_PERCENT;
00054
00055     if (corrected < 0)   corrected = 0;
00056     if (corrected > 100) corrected = 100;
00057
00058     uint16_t min = hservo->min_pulse_ticks;
00059     uint16_t max = hservo->max_pulse_ticks;
00060     return min + ((max - min) * corrected) / 100;
00061 }
00062
00068 void servo_pwm_percent(Servo_Handle_t *hservo, uint8_t percent){
00069     uint16_t value = servo_map_percent(hservo, percent);
00070     pwm_pulse(hservo, value);
00071 }
00072
00080 void servo_pwm_angle_degree(Servo_Handle_t *hservo, int8_t angle){
00081     if (angle < SERVO_CLAMP_MIN) angle = SERVO_CLAMP_MIN;
00082     if (angle > SERVO_CLAMP_MAX) angle = SERVO_CLAMP_MAX;
00083
00084     int16_t percent = ((angle + 35) * 100) / 70;
00085     uint16_t value = servo_map_percent(hservo, percent);
00086
00087     pwm_pulse(hservo, value);
00088 }
00089
00097 void servo_pwm_angle_abs_value(Servo_Handle_t *hservo, uint16_t abs_value){
00098     int32_t angle_centi = map(abs_value, 0, 65535, -4500, 4500);
00099
00100     if (angle_centi < -2000) angle_centi = -2000;
00101     if (angle_centi >  2000) angle_centi =  2000;
00102
00103     int32_t pwm_ticks = map(angle_centi, -3500, 3500, hservo->min_pulse_ticks,
    hservo->max_pulse_ticks);
00104
00105     int32_t range = hservo->max_pulse_ticks - hservo->min_pulse_ticks;
00106     int32_t offset_ticks = (range * SERVO_OFFSET_PERCENT) / 100;
00107
00108     pwm_ticks += offset_ticks;
00109
00110     // Sécurité bornes hardware
00111     if (pwm_ticks > hservo->max_pulse_ticks) pwm_ticks = hservo->max_pulse_ticks;
00112     if (pwm_ticks < hservo->min_pulse_ticks) pwm_ticks = hservo->min_pulse_ticks;
00113
00114     pwm_pulse(hservo, (uint16_t)pwm_ticks);
00115 }
00116
00122 void servo_initialisation(Servo_Handle_t *hservo){
00123     if(hservo && hservo->htim){
00124         servo_pwm_angle_degree(hservo, 0);
00125         HAL_TIM_PWM_Start(hservo->htim, hservo->channel);
00126     }
00127 }
```

6.43 Core/Src/driver_speedometer.c File Reference

Driver de calcul de vitesse basé sur un capteur à effet Hall.

```
#include "driver_speedometer.h"
#include "tim.h"
```

Include dependency graph for driver_speedometer.c:



Functions

- float speedometer_solve_speed (Speedometer_Handle_t *hSpeedo)

  Calcule la vitesse instantanée en m/s.
- void speedometer_init (Speedometer_Handle_t *hSpeedo, TIM_HandleTypeDef *htim)

  Initialise le driver tachymètre.

6.43.1 Detailed Description

Driver de calcul de vitesse basé sur un capteur à effet Hall.

Utilise un timer en mode compteur pour mesurer le nombre d'impulsions générées par la rotation de la roue et déduit la vitesse linéaire.

Definition in file driver_speedometer.c.

6.43.2 Function Documentation

6.43.2.1 speedometer_init()

```
void speedometer_init (
            Speedometer_Handle_t * hSpeedo,
            TIM_HandleTypeDef * htim )
```

Initialise le driver tachymètre.

Initialise l'objet tachymètre.

Associe le timer matériel à la structure, initialise les variables d'état (temps et compteur) et démarre le timer.

Parameters

hSpeedo	Pointeur vers la structure de gestion à initialiser.
htim	Pointeur vers le handle du Timer STM32 (HAL) utilisé.

Definition at line 48 of file driver_speedometer.c.

Here is the caller graph for this function:

6.43.2.2 speedometer_solve_speed()

```
float speedometer_solve_speed (
            Speedometer_Handle_t * hSpeedo )
```

Calcule la vitesse instantanée en m/s.

Calcule la vitesse actuelle basée sur les impulsions reçues.

Cette fonction doit être appelée périodiquement. Elle calcule la différence de temps et de nombre d'impulsions (ticks) depuis le dernier appel.

Note

Gère implicitement le débordement (overflow) du compteur 16 bits via l'arithmétique non signée.

Parameters

hSpeedo	Pointeur vers la structure de gestion du tachymètre.

Returns

Vitesse calculée en mètres par seconde (m/s).

Definition at line 20 of file driver_speedometer.c.

6.44 driver_speedometer.c

Go to the documentation of this file.

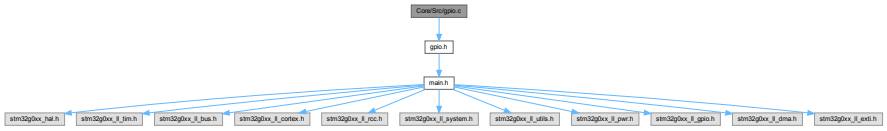
```
00001
00008 #include "driver_speedometer.h"
00009 #include "tim.h"
00010
00020 float speedometer_solve_speed(Speedometer_Handle_t *hSpeedo){
00021     uint32_t now = HAL_GetTick();
00022     uint32_t time_diff_ms = now - hSpeedo->last_process_time;
00023
00024     if(time_diff_ms == 0){
00025         return hSpeedo->current_speed_ms;
00026     }
00027
```

```
00028     uint16_t current_counter = (uint16_t)__HAL_TIM_GET_COUNTER(hSpeedo->htim);
00029     uint16_t pulses = current_counter - hSpeedo->last_counter_val;
00030     float wheel_turns = (float)pulses / TICKS_PER_WHEEL_TURN;
00031     float distance_m = wheel_turns * PERIMETER_M;
00032     float speed_ms = distance_m / ((float)time_diff_ms / 1000.0f);
00033
00034     hSpeedo->last_counter_val = current_counter;
00035     hSpeedo->last_process_time = now;
00036     hSpeedo->current_speed_ms = speed_ms;
00037
00038     return speed_ms;
00039 }
00040
00048 void speedometer_init(Speedometer_Handle_t *hSpeedo, TIM_HandleTypeDef *htim){
00049     hSpeedo->htim = htim;
00050     hSpeedo->last_counter_val = (uint16_t)__HAL_TIM_GET_COUNTER(hSpeedo->htim);
00051     hSpeedo->last_process_time = HAL_GetTick();
00052     hSpeedo->current_speed_ms = 0.0f;
00053
00054     HAL_TIM_Base_Start(hSpeedo->htim);
00055 }
```

6.45 Core/Src/gpio.c File Reference

This file provides code for the configuration of all used GPIO pins.

`#include "gpio.h"`
Include dependency graph for gpio.c:



Functions

- void MX_GPIO_Init (void)

6.45.1 Detailed Description

This file provides code for the configuration of all used GPIO pins.

Attention

Definition in file gpio.c.

6.45.2 Function Documentation

6.45.2.1 MX_GPIO_Init()

```
void MX_GPIO_Init (
             void  )
```

Configure pins as Analog Input Output EVENT_OUT EXTI Free pins are configured automatically as Analog (this feature is enabled through the Code Generation settings)

Definition at line 44 of file gpio.c.

Here is the caller graph for this function:

6.46 gpio.c

Go to the documentation of this file.

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020
00021 /* Includes ------------------------------------------------------------------*/
00022 #include "gpio.h"
00023
00024 /* USER CODE BEGIN 0 */
00025
00026 /* USER CODE END 0 */
00027
00028 /*----------------------------------------------------------------------------*/
00029 /* Configure GPIO                                                             */
00030 /*----------------------------------------------------------------------------*/
00031 /* USER CODE BEGIN 1 */
00032
00033 /* USER CODE END 1 */
00034
00044 void MX_GPIO_Init(void)
00045 {
00046
00047   GPIO_InitTypeDef GPIO_InitStruct = {0};
00048
00049   /* GPIO Ports Clock Enable */
00050   __HAL_RCC_GPIOC_CLK_ENABLE();
00051   __HAL_RCC_GPIOF_CLK_ENABLE();
00052   __HAL_RCC_GPIOA_CLK_ENABLE();
00053   __HAL_RCC_GPIOB_CLK_ENABLE();
00054   __HAL_RCC_GPIOD_CLK_ENABLE();
00055
00056   /*Configure GPIO pin Output Level */
00057   HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_RESET);
00058
00059   /*Configure GPIO pin Output Level */
00060   HAL_GPIO_WritePin(GPIOB, ACC_CS_Pin|GYR_CS_Pin, GPIO_PIN_SET);
00061
00062   /*Configure GPIO pins : PC11 PC12 PC0 PC1
00063                            PC2 PC3 PC4 PC5
00064                            PC6 PC7 PC8 PC9
00065                            PC10 */
00066   GPIO_InitStruct.Pin = GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_0|GPIO_PIN_1
00067                         |GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5
```

```
00068                           |GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_9
00069                           |GPIO_PIN_10;
00070   GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
00071   GPIO_InitStruct.Pull = GPIO_NOPULL;
00072   HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
00073
00074   /*Configure GPIO pin : B1_Pin */
00075   GPIO_InitStruct.Pin = B1_Pin;
00076   GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
00077   GPIO_InitStruct.Pull = GPIO_NOPULL;
00078   HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);
00079
00080   /*Configure GPIO pins : PF1 PF2 */
00081   GPIO_InitStruct.Pin = GPIO_PIN_1|GPIO_PIN_2;
00082   GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
00083   GPIO_InitStruct.Pull = GPIO_NOPULL;
00084   HAL_GPIO_Init(GPIOF, &GPIO_InitStruct);
00085
00086   /*Configure GPIO pins : PA4 PA9 PA10 PA11
00087                            PA12 PA15 */
00088   GPIO_InitStruct.Pin = GPIO_PIN_4|GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11
00089                           |GPIO_PIN_12|GPIO_PIN_15;
00090   GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
00091   GPIO_InitStruct.Pull = GPIO_NOPULL;
00092   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
00093
00094   /*Configure GPIO pin : LED_GREEN_Pin */
00095   GPIO_InitStruct.Pin = LED_GREEN_Pin;
00096   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
00097   GPIO_InitStruct.Pull = GPIO_NOPULL;
00098   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
00099   HAL_GPIO_Init(LED_GREEN_GPIO_Port, &GPIO_InitStruct);
00100
00101   /*Configure GPIO pins : PB0 PB1 PB2 PB10
00102                            PB11 PB12 PB15 PB3
00103                            PB4 PB5 PB7 PB8
00104                            PB9 */
00105   GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_10
00106                           |GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_15|GPIO_PIN_3
00107                           |GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_7|GPIO_PIN_8
00108                           |GPIO_PIN_9;
00109   GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
00110   GPIO_InitStruct.Pull = GPIO_NOPULL;
00111   HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
00112
00113   /*Configure GPIO pins : ACC_CS_Pin GYR_CS_Pin */
00114   GPIO_InitStruct.Pin = ACC_CS_Pin|GYR_CS_Pin;
00115   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
00116   GPIO_InitStruct.Pull = GPIO_NOPULL;
00117   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00118   HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
00119
00120   /*Configure GPIO pins : PD8 PD9 PD0 PD1
00121                            PD2 PD3 PD4 PD5
00122                            PD6 */
00123   GPIO_InitStruct.Pin = GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_0|GPIO_PIN_1
00124                           |GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5
00125                           |GPIO_PIN_6;
00126   GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
00127   GPIO_InitStruct.Pull = GPIO_NOPULL;
00128   HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
00129
00130 }
00131
00132 /* USER CODE BEGIN 2 */
00133
00134 /* USER CODE END 2 */
```

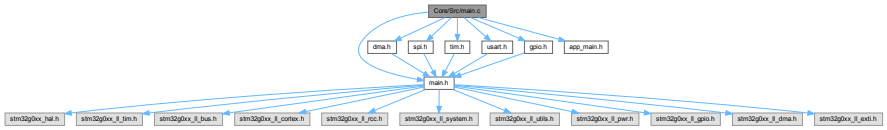
6.47 Core/Src/main.c File Reference

: Main program body

```
#include "main.h"
#include "dma.h"
#include "spi.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"
```

```
#include "app_main.h"
```

Include dependency graph for main.c:



Functions

- void SystemClock_Config (void)

  System Clock Configuration.
- int main (void)

  The application entry point.
- int __io_putchar (int ch)
- void Error_Handler (void)

  This function is executed in case of error occurrence.

6.47.1 Detailed Description

: Main program body

Attention

Definition in file main.c.

6.47.2 Function Documentation

6.47.2.1 __io_putchar()

```
int __io_putchar (
            int ch )
```

Definition at line 163 of file main.c.

6.47.2.2 Error_Handler()

```
void Error_Handler (
            void  )
```

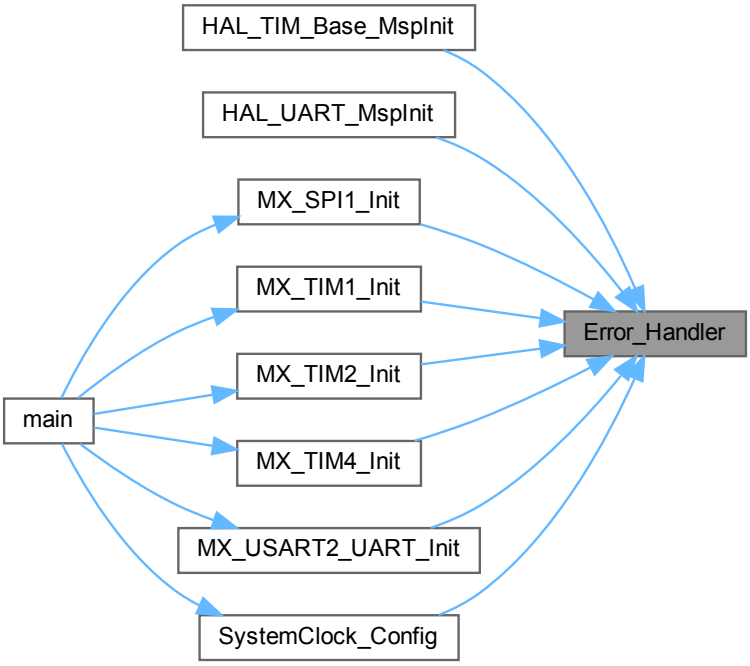
This function is executed in case of error occurrence.

Return values

None	

Definition at line 173 of file main.c.

Here is the caller graph for this function:



6.47.2.3 main()

```
int main (
              void  )
```

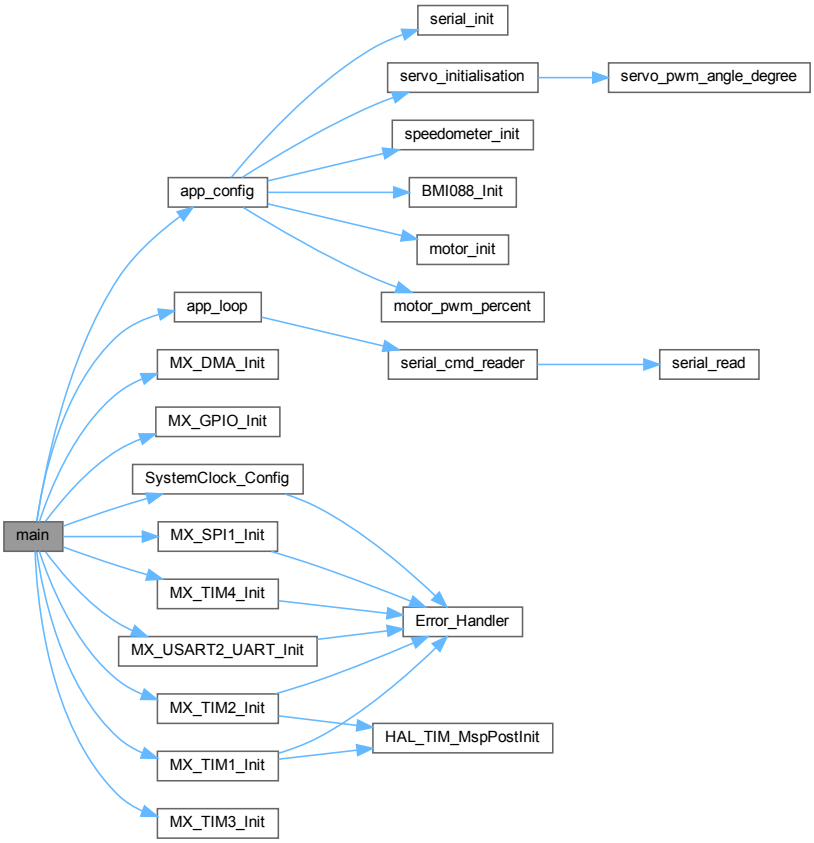
The application entry point.

Return values

int	

Definition at line 68 of file main.c.

Here is the call graph for this function:



6.47.2.4 SystemClock_Config()

```
void SystemClock_Config (
            void  )
```

System Clock Configuration.

Return values

None	

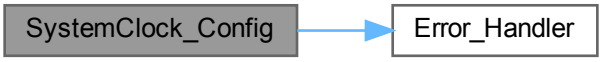
Configure the main internal regulator output voltage

Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef structure.

Initializes the CPU, AHB and APB buses clocks

Definition at line 120 of file main.c.

Here is the call graph for this function:



Here is the caller graph for this function:

6.48 main.c

Go to the documentation of this file.

```
00001 /* USER CODE BEGIN Header */
00018 /* USER CODE END Header */
00019 /* Includes ------------------------------------------------------------------*/
00020 #include "main.h"
00021 #include "dma.h"
00022 #include "spi.h"
00023 #include "tim.h"
00024 #include "usart.h"
00025 #include "gpio.h"
00026
00027 /* Private includes ----------------------------------------------------------*/
00028 /* USER CODE BEGIN Includes */
00029 #include "app_main.h"
00030 /* USER CODE END Includes */
00031
00032 /* Private typedef -----------------------------------------------------------*/
00033 /* USER CODE BEGIN PTD */
00034
00035 /* USER CODE END PTD */
00036
00037 /* Private define ------------------------------------------------------------*/
00038 /* USER CODE BEGIN PD */
00039
00040 /* USER CODE END PD */
00041
00042 /* Private macro -------------------------------------------------------------*/
00043 /* USER CODE BEGIN PM */
00044
00045 /* USER CODE END PM */
00046
00047 /* Private variables ---------------------------------------------------------*/
00048
00049 /* USER CODE BEGIN PV */
00050
00051 /* USER CODE END PV */
00052
00053 /* Private function prototypes -----------------------------------------------*/
00054 void SystemClock_Config(void);
00055 /* USER CODE BEGIN PFP */
00056
00057 /* USER CODE END PFP */
```

```
00058
00059 /* Private user code ---------------------------------------------------------*/
00060 /* USER CODE BEGIN 0 */
00061
00062 /* USER CODE END 0 */
00063
00068 int main(void)
00069 {
00070
00071   /* USER CODE BEGIN 1 */
00072
00073   /* USER CODE END 1 */
00074
00075   /* MCU Configuration--------------------------------------------------------*/
00076
00077   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
00078   HAL_Init();
00079
00080   /* USER CODE BEGIN Init */
00081
00082   /* USER CODE END Init */
00083
00084   /* Configure the system clock */
00085   SystemClock_Config();
00086
00087   /* USER CODE BEGIN SysInit */
00088
00089   /* USER CODE END SysInit */
00090
00091   /* Initialize all configured peripherals */
00092   MX_GPIO_Init();
00093   MX_DMA_Init();
00094   MX_USART2_UART_Init();
00095   MX_SPI1_Init();
00096   MX_TIM1_Init();
00097   MX_TIM2_Init();
00098   MX_TIM3_Init();
00099   MX_TIM4_Init();
00100   /* USER CODE BEGIN 2 */
00101   app_config();
00102   /* USER CODE END 2 */
00103
00104   /* Infinite loop */
00105   /* USER CODE BEGIN WHILE */
00106   while (1)
00107   {
00108     /* USER CODE END WHILE */
00109
00110     /* USER CODE BEGIN 3 */
00111       app_loop();
00112   }
00113   /* USER CODE END 3 */
00114 }
00115
00120 void SystemClock_Config(void)
00121 {
00122   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
00123   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
00124
00127   HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1);
00128
00132   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
00133   RCC_OscInitStruct.HSIState = RCC_HSI_ON;
00134   RCC_OscInitStruct.HSIDiv = RCC_HSI_DIV1;
00135   RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
00136   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
00137   RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
00138   RCC_OscInitStruct.PLL.PLLM = RCC_PLLM_DIV1;
00139   RCC_OscInitStruct.PLL.PLLN = 8;
00140   RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
00141   RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
00142   RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
00143   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
00144   {
00145     Error_Handler();
00146   }
00147
00150   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
00151                                 |RCC_CLOCKTYPE_PCLK1;
00152   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
00153   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
00154   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
00155
00156   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
00157   {
00158     Error_Handler();
00159   }
```

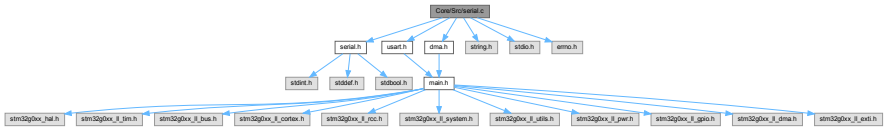
```
00160 }
00161
00162 /* USER CODE BEGIN 4 */
00163 int __io_putchar(int ch) {
00164     HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, HAL_MAX_DELAY);
00165     return ch;
00166 }
00167 /* USER CODE END 4 */
00168
00173 void Error_Handler(void)
00174 {
00175   /* USER CODE BEGIN Error_Handler_Debug */
00176   /* User can add his own implementation to report the HAL error return state */
00177   __disable_irq();
00178   while (1)
00179   {
00180   }
00181   /* USER CODE END Error_Handler_Debug */
00182 }
00183 #ifdef USE_FULL_ASSERT
00191 void assert_failed(uint8_t *file, uint32_t line)
00192 {
00193   /* USER CODE BEGIN 6 */
00194   /* User can add his own implementation to report the file name and line number,
00195      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
00196   /* USER CODE END 6 */
00197 }
00198 #endif /* USE_FULL_ASSERT */
```

6.49 Core/Src/serial.c File Reference

Gestionnaire de communication Série (UART) avec DMA et Buffers Circulaires.

```
#include "serial.h"
#include "usart.h"
#include "dma.h"
#include <string.h>
#include <stdio.h>
#include <errno.h>
```

Include dependency graph for serial.c:



Macros

- #define RING_MASK (SERIAL_RX_RING_SIZE-1u)

  Masque pour le calcul modulo du buffer RX (taille doit être puissance de 2).
- #define TX_RING_SIZE 1024u

  Taille du buffer circulaire d'émission.
- #define TX_RING_MASK (TX_RING_SIZE-1u)

  Masque pour le calcul modulo du buffer TX.

Functions

- void serial_init (void)
  Initialise le driver Série.
- int serial_write_nb (const uint8_t *data, uint16_t len)
  Écrit des données dans le buffer d'émission (Non-bloquant partiel).
- int serial_write_all_nb (const uint8_t *data, uint16_t len)
  Écrit des données seulement si tout le bloc peut entrer (Atomique).
- int serial_write (const uint8_t *data, uint16_t len)
  Wrapper bloquant pour l'écriture (simulé).
- int _write (int file, char *ptr, int len)
  Fonction système bas niveau pour rediriger printf().
- size_t serial_available (void)
  Vérifie le nombre d'octets disponibles à la lecture.
- size_t serial_read (uint8_t *dst, size_t max_len)
  Lit des données depuis le buffer RX.
- size_t serial_read_until (uint8_t *dst, size_t max_len, uint8_t delim)
  Lit des données jusqu'à rencontrer un délimiteur.
- void HAL_UART_TxCpltCallback (UART_HandleTypeDef *huart)
  Callback HAL appelé quand un transfert DMA TX est terminé.
- void HAL_UARTEx_RxEventCallback (UART_HandleTypeDef *huart, uint16_t Size)
  Callback HAL appelé lors d'un événement RX (Idle Line ou Transfer Complete).
- uint8_t serial_crc8_atm (const uint8_t *data, uint16_t len)
  Calcule un CRC-8 (Polynôme ATM : 0x07).
- int proto_send_write16 (uint8_t addr, int16_t value)
  Envoie une commande d'écriture 16 bits protocole.
- int proto_send_read_burst (uint8_t addr, uint8_t count, uint8_t flags)
  Envoie une demande de lecture multiple (Burst).

6.49.1 Detailed Description

Gestionnaire de communication Série (UART) avec DMA et Buffers Circulaires.

Ce fichier implémente un driver UART haute performance asynchrone.

- Réception : Utilise un buffer DMA linéaire couplé à un buffer circulaire logiciel.
- Transmission : Utilise un buffer circulaire logiciel vidé par DMA.
- Supporte les fonctions standard stdio (_write) pour printf.

Definition in file serial.c.

6.49.2 Macro Definition Documentation

6.49.2.1 RING_MASK

```
#define RING_MASK (SERIAL_RX_RING_SIZE-1u)
```

Masque pour le calcul modulo du buffer RX (taille doit être puissance de 2).

Definition at line 30 of file serial.c.

6.49.2.2 TX_RING_MASK

```
#define TX_RING_MASK (TX_RING_SIZE-1u)
```

Masque pour le calcul modulo du buffer TX.

Definition at line 57 of file serial.c.

6.49.2.3 TX_RING_SIZE

```
#define TX_RING_SIZE 1024u
```

Taille du buffer circulaire d'émission.

Definition at line 54 of file serial.c.

6.49.3 Function Documentation

6.49.3.1 _write()

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

Fonction système bas niveau pour rediriger printf().

Parameters

file	Descripteur de fichier (ignoré).
ptr	Données à écrire.
len	Longueur.

Returns

Nombre d'octets écrits ou -1.

Definition at line 194 of file serial.c.

Here is the call graph for this function:

6.49.3.2 HAL_UART_TxCpltCallback()

```
void HAL_UART_TxCpltCallback (
            UART_HandleTypeDef * huart )
```

Callback HAL appelé quand un transfert DMA TX est terminé.

Met à jour l'index de queue TX et relance une transmission si il reste des données dans le buffer.

Parameters

huart	Handle UART concerné.

Definition at line 277 of file serial.c.

6.49.3.3 HAL_UARTEx_RxEventCallback()

```
void HAL_UARTEx_RxEventCallback (
            UART_HandleTypeDef * huart,
            uint16_t Size )
```

Callback HAL appelé lors d'un événement RX (Idle Line ou Transfer Complete).

Transfère les données du buffer DMA linéaire (rx_chunk) vers le buffer circulaire logiciel (rx_ring).

Note

Gère la position précédente pour ne copier que les nouveaux octets reçus.

Parameters

huart	Handle UART concerné.
Size	Nombre total d'octets reçus dans le buffer DMA.

Definition at line 293 of file serial.c.

6.49.3.4 proto_send_read_burst()

```
int proto_send_read_burst (
            uint8_t addr,
            uint8_t count,
            uint8_t flags )
```

Envoie une demande de lecture multiple (Burst).

Envoie une requête de lecture (Frame Read Request).

Parameters

addr	Adresse de départ.
count	Nombre de registres à lire.
flags	Flags optionnels.

Returns

Résultat de l'envoi.

Definition at line 372 of file serial.c.

6.49.3.5 proto_send_write16()

```
int proto_send_write16 (
            uint8_t addr,
            int16_t value )
```

Envoie une commande d'écriture 16 bits protocole.

Envoie une commande d'écriture 16 bits (Frame Write).

Parameters

addr	Adresse du registre virtuel.
value	Valeur 16 bits à écrire.

Returns

Résultat de l'envoi.

Definition at line 359 of file serial.c.

6.49.3.6 serial_available()

```
size_t serial_available (
            void  )
```

Vérifie le nombre d'octets disponibles à la lecture.

Retourne le nombre d'octets disponibles dans le buffer RX.

Returns

Nombre d'octets dans le buffer RX.

Definition at line 204 of file serial.c.

6.49.3.7 serial_crc8_atm()

```
uint8_t serial_crc8_atm (
            const uint8_t * data,
            uint16_t len )
```

Calcule un CRC-8 (Polynôme ATM : 0x07).

Calcule le CRC8 (Polynôme ATM 0x07).

Parameters

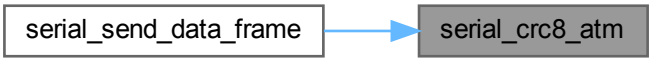
data	Pointeur vers les données.
len	Longueur des données.

Returns

Valeur du CRC calculé.

Definition at line 326 of file serial.c.

Here is the caller graph for this function:



6.49.3.8 serial_init()

```
void serial_init (
            void  )
```

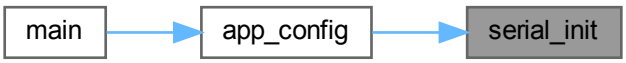
Initialise le driver Série.

Initialise la couche série (DMA + Buffers).

Lance la réception DMA en mode "ReceiveToIdle" pour détecter les fins de trames sans attendre que le buffer soit plein.

Definition at line 122 of file serial.c.

Here is the caller graph for this function:



6.49.3.9 serial_read()

```
size_t serial_read (
            uint8_t * dst,
            size_t max_len )
```

Lit des données depuis le buffer RX.

Copie les données du buffer circulaire interne vers le buffer utilisateur. Gère le cas où les données sont à cheval sur la fin du buffer (wrap).

Parameters

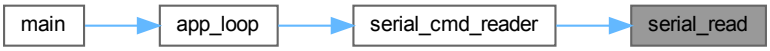
dst	Buffer de destination.
max_len	Nombre maximum d'octets à lire.

Returns

Nombre d'octets réellement lus.

Definition at line 214 of file serial.c.

Here is the caller graph for this function:



6.49.3.10 serial_read_until()

```
size_t serial_read_until (
            uint8_t * dst,
            size_t max_len,
            uint8_t delim )
```

Lit des données jusqu'à rencontrer un délimiteur.

Lit des données jusqu'à un délimiteur spécifique.

Note

Fonction utile pour lire des lignes complètes (ex: jusqu'à '\n').

Parameters

dst	Buffer de destination.
max_len	Taille max du buffer destination.
delim	Caractère délimiteur recherché.

Returns

Nombre d'octets lus (incluant le délimiteur), ou 0 si non trouvé.

Definition at line 250 of file serial.c.

6.49.3.11 serial_write()

```
int serial_write (
            const uint8_t * data,
            uint16_t len )
```

Wrapper bloquant pour l'écriture (simulé).

Écriture bloquante (Wrapper).

Note

En réalité non-bloquant ici, renvoie -1 en cas d'échec.

Parameters

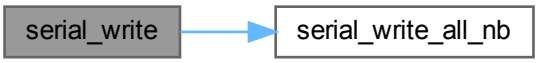
data	Pointeur vers les données.
len	Longueur.

Returns

0 si succès, -1 si erreur.

Definition at line 182 of file serial.c.

Here is the call graph for this function:



6.49.3.12 serial_write_all_nb()

```
int serial_write_all_nb (
            const uint8_t * data,
            uint16_t len )
```

Écrit des données seulement si tout le bloc peut entrer (Atomique).

Écriture non-bloquante atomique (Tout ou rien).

Parameters

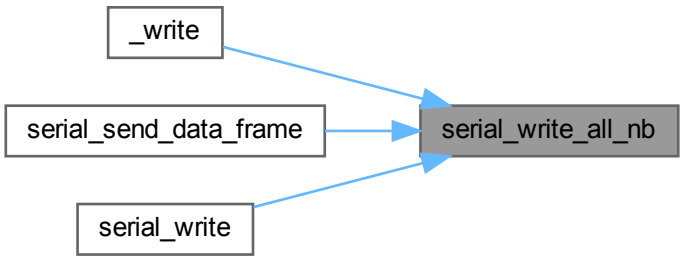
data	Pointeur vers les données.
len	Nombre d'octets.

Returns

len si succès, -EWOULDBLOCK si pas assez d'espace.

Definition at line 159 of file serial.c.

Here is the caller graph for this function:



6.49.3.13 serial_write_nb()

```
int serial_write_nb (
            const uint8_t * data,
            uint16_t len )
```

Écrit des données dans le buffer d'émission (Non-bloquant partiel).

Écriture non-bloquante partielle.

Note

Copie autant de données que possible. S'arrête si le buffer est plein.

Parameters

data	Pointeur vers les données.
len	Nombre d'octets à écrire.

Returns

Nombre d'octets réellement écrits ou code d'erreur négatif.

Definition at line 134 of file serial.c.

6.50 serial.c

Go to the documentation of this file.

```
00001
00010 #include "serial.h"
00011 #include "usart.h"
00012 #include "dma.h"
00013 #include <string.h>
00014 #include <stdio.h>
00015 #include <errno.h>
00016
00018 static uint8_t rx_chunk[SERIAL_RX_CHUNK_SIZE];
00019
00021 static uint8_t rx_ring[SERIAL_RX_RING_SIZE];
00022
00024 static volatile uint32_t rx_head=0;
00025
00027 static volatile uint32_t rx_tail=0;
00028
00030 #define RING_MASK (SERIAL_RX_RING_SIZE-1u)
00031
00032 #if (SERIAL_RX_RING_SIZE&(SERIAL_RX_RING_SIZE-1u))
00033 #error "SERIAL_RX_RING_SIZE must be a power of two"
00034 #endif
00035
00041 static inline void ring_push(uint8_t b){
00042     rx_ring[rx_head]=b;
00043     rx_head=(rx_head+1u)&RING_MASK;
00044     if(rx_head==rx_tail)rx_tail=(rx_tail+1u)&RING_MASK;
00045 }
00046
00051 static inline uint32_t ring_count(void){return (rx_head-rx_tail)&RING_MASK;}
00052
00054 #define TX_RING_SIZE 1024u
00055
00057 #define TX_RING_MASK (TX_RING_SIZE-1u)
00058
00059 #if (TX_RING_SIZE&(TX_RING_SIZE-1u))
00060 #error "TX_RING_SIZE must be a power of two"
00061 #endif
00062
00064 static uint8_t tx_ring[TX_RING_SIZE];
00065
00067 static volatile uint32_t tx_head=0;
00068
00070 static volatile uint32_t tx_tail=0;
00071
00073 static volatile uint8_t tx_busy=0;
00074
00079 static inline uint32_t tx_count(void){return (tx_head-tx_tail)&TX_RING_MASK;}
00080
00085 static inline uint32_t tx_space(void){return TX_RING_MASK-tx_count();}
00086
00093 static void serial_kick_tx(void){
00094     __disable_irq();
00095     if(tx_busy){
00096         __enable_irq();
00097         return;
00098     }
00099     uint32_t head=tx_head, tail=tx_tail;
00100     if(head==tail){
00101         __enable_irq();
00102         return;
00103     }
00104
00105     uint32_t linear=(head>=tail)?(head-tail):(TX_RING_SIZE-tail);
00106     uint16_t chunk=(linear>SERIAL_TX_CHUNK_MAX)?SERIAL_TX_CHUNK_MAX:(uint16_t)linear;
00107
00108     tx_busy=1;
00109
00110     if(HAL_UART_Transmit_DMA(&SERIAL_UART, &tx_ring[tail], chunk) != HAL_OK){
00111         tx_busy=0;
00112     }
00113
00114     __enable_irq();
00115 }
00116
00122 void serial_init(void){
00123     HAL_UARTEx_ReceiveToIdle_DMA(&SERIAL_UART,rx_chunk,SERIAL_RX_CHUNK_SIZE);
00124     __HAL_DMA_DISABLE_IT(SERIAL_UART.hdmarx,DMA_IT_HT);
00125 }
00126
00134 int serial_write_nb(const uint8_t *data,uint16_t len){
00135     uint16_t written=0;
00136     while(written<len){
00137         uint32_t space=tx_space();
00138         if(space==0)return(written>0)?(int)written:-EWOULDBLOCK;
00139         uint32_t head=tx_head;
00140         uint32_t
      room_linear=(head>=tx_tail)?(TX_RING_SIZE-head-((tx_tail==0)?1u:0u)):(tx_tail-head-1u);
```

```
00141         if(room_linear==0)return(written>0)?(int)written:-EWOULDBLOCK;
00142         uint32_t to_copy=len-written;
00143         if(to_copy>room_linear)to_copy=room_linear;
00144         if(to_copy>space)to_copy=space;
00145         memcpy(&tx_ring[head],&data[written],to_copy);
00146         tx_head=(head+to_copy)&TX_RING_MASK;
00147         written+=(uint16_t)to_copy;
00148     }
00149     serial_kick_tx();
00150     return(int)written;
00151 }
00152
00159 int serial_write_all_nb(const uint8_t *data,uint16_t len){
00160     if(tx_space()<len)return -EWOULDBLOCK;
00161     uint32_t head=tx_head;
00162     uint32_t first=(head>=tx_tail)?(TX_RING_SIZE-head-((tx_tail==0)?1u:0u)):(tx_tail-head-1u);
00163     if(first>len)first=len;
00164     memcpy(&tx_ring[head],data,first);
00165     tx_head=(head+first)&TX_RING_MASK;
00166     uint16_t remain=len-(uint16_t)first;
00167     if(remain){
00168         memcpy(&tx_ring[tx_head],data+first,remain);
00169         tx_head=(tx_head+remain)&TX_RING_MASK;
00170     }
00171     serial_kick_tx();
00172     return(int)len;
00173 }
00174
00182 int serial_write(const uint8_t *data,uint16_t len){
00183     int r=serial_write_all_nb(data,len);
00184     return(r>=0)?0:-1;
00185 }
00186
00194 int _write(int file,char *ptr,int len){
00195     (void)file;
00196     int r=serial_write_all_nb((const uint8_t*)ptr,(uint16_t)len);
00197     return(r>=0)?r:-1;
00198 }
00199
00204 size_t serial_available(void){return ring_count();}
00205
00214 size_t serial_read(uint8_t *dst, size_t max_len){
00215     __disable_irq();
00216
00217     size_t avail = ring_count();
00218     if(!avail || !max_len){
00219         __enable_irq();
00220         return 0;
00221     }
00222
00223     size_t to_copy = (avail < max_len) ? avail : max_len;
00224     size_t first = to_copy;
00225
00226     if(rx_tail + first > SERIAL_RX_RING_SIZE) {
00227         first = SERIAL_RX_RING_SIZE - rx_tail;
00228     }
00229
00230     memcpy(dst, &rx_ring[rx_tail], first);
00231     if(first < to_copy) {
00232         memcpy(dst + first, &rx_ring[0], to_copy - first);
00233     }
00234
00235     rx_tail = (rx_tail + to_copy) & RING_MASK;
00236
00237     __enable_irq();
00238
00239     return to_copy;
00240 }
00241
00250 size_t serial_read_until(uint8_t *dst,size_t max_len,uint8_t delim){
00251     if(!max_len)return 0;
00252     uint32_t head=rx_head,tail=rx_tail;
00253     if(head==tail)return 0;
00254     uint32_t i=tail;
00255     while(i!=head){
00256         if(rx_ring[i]==delim){
00257             size_t msg_len=(i>=tail)?(i-tail+1u):(SERIAL_RX_RING_SIZE-tail+i+1u);
00258             if(msg_len>max_len)return 0;
00259             size_t first=msg_len;
00260             if(tail+first>SERIAL_RX_RING_SIZE)first=SERIAL_RX_RING_SIZE-tail;
00261             memcpy(dst,&rx_ring[tail],first);
00262             if(first<msg_len)memcpy(dst+first,&rx_ring[0],msg_len-first);
00263             rx_tail=(tail+msg_len)&RING_MASK;
00264             return msg_len;
00265         }
00266         i=(i+1u)&RING_MASK;
00267     }
```

```
00268     return 0;
00269 }
00270
00277 void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart){
00278     if(huart!=&SERIAL_UART)return;
00279     uint16_t sent=huart->TxXferSize;
00280     tx_tail=(tx_tail+sent)&TX_RING_MASK;
00281     tx_busy=0;
00282     serial_kick_tx();
00283 }
00284
00293 void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size){
00294     if(huart != &SERIAL_UART) return;
00295
00296     static uint16_t old_pos = 0;
00297
00298     if(Size != old_pos){
00299         if(Size > old_pos){
00300             for(uint16_t k = old_pos; k < Size; k++){
00301                 ring_push(rx_chunk[k]);
00302             }
00303         }
00304         else{
00305             for(uint16_t k = old_pos; k < SERIAL_RX_CHUNK_SIZE; k++){
00306                 ring_push(rx_chunk[k]);
00307             }
00308             for(uint16_t k = 0; k < Size; k++){
00309                 ring_push(rx_chunk[k]);
00310             }
00311         }
00312         old_pos = Size;
00313     }
00314
00315     HAL_UARTEx_ReceiveToIdle_DMA(huart, rx_chunk, SERIAL_RX_CHUNK_SIZE);
00316
00317     __HAL_DMA_DISABLE_IT(huart->hdmarx, DMA_IT_HT);
00318 }
00319
00326 uint8_t serial_crc8_atm(const uint8_t *data,uint16_t len){
00327     uint8_t crc=0x00u;
00328     for(uint16_t i=0;i<len;i++){
00329         crc^=data[i];
00330         for(uint8_t b=0;b<8;b++){
00331             crc=(crc&0x80u)?(uint8_t)((crc<<1)^0x07u):(uint8_t)(crc<<1);
00332         }
00333     }
00334     return crc;
00335 }
00336
00344 static int proto_send_frame3(uint8_t hdr,uint8_t d0,uint8_t d1){
00345     uint8_t buf[4];
00346     buf[0]=hdr;
00347     buf[1]=d0;
00348     buf[2]=d1;
00349     buf[3]=serial_crc8_atm(buf,3);
00350     return serial_write_all_nb(buf,4);
00351 }
00352
00359 int proto_send_write16(uint8_t addr,int16_t value){
00360     uint8_t hdr=PROTO_MAKE_HDR(0,addr);
00361     uint16_t u=(uint16_t)value;
00362     return proto_send_frame3(hdr,(uint8_t)(u&0xFFu),(uint8_t)(u>>8));
00363 }
00364
00372 int proto_send_read_burst(uint8_t addr,uint8_t count,uint8_t flags){
00373     uint8_t hdr=PROTO_MAKE_HDR(1,addr);
00374     return proto_send_frame3(hdr,count,flags);
00375 }
```

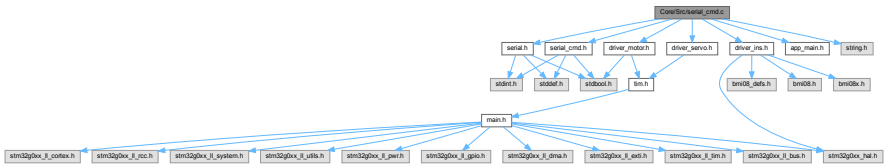
6.51 Core/Src/serial_cmd.c File Reference

Gestionnaire de commandes et télémétrie (Protocole Application).

```
#include "serial_cmd.h"
#include "serial.h"
#include "driver_motor.h"
#include "driver_servo.h"
#include "driver_ins.h"
```

```
#include "app_main.h"
#include <string.h>
```

Include dependency graph for serial_cmd.c:



Enumerations

- enum ParseState { S_HDR =0 , S_D0 , S_D1 , S_CRC }

  États de la machine à états de réception (Protocole 4 octets).

Functions

- void serial_cmd_reader (void)

  Fonction principale de lecture (Polling).
- void serial_send_data_frame (void)

  Construit et envoie la trame de télémétrie complète.

Variables

- ParserSwitch parser_state = PARSER_IDLE

  État courant du parseur, exposé à l'application principale pour déclencher les actions.
- int8_t shadow_servo_cmd = 0

  Copie locale de la dernière commande servo reçue.
- int16_t shadow_motor_cmd = 0

  Copie locale de la dernière commande moteur reçue.

6.51.1 Detailed Description

Gestionnaire de commandes et télémétrie (Protocole Application).

Ce fichier gère :

1. Le parsing des commandes entrantes (Pilotage Moteur/Servo) via une machine à états.
2. L'envoi périodique de la télémétrie (IMU + Vitesse) vers l'ordinateur de bord (Pi5).

Definition in file serial_cmd.c.

6.51.2 Enumeration Type Documentation

6.51.2.1 ParseState

```
enum ParseState
```

États de la machine à états de réception (Protocole 4 octets).

Enumerator

S_HDR	Attente de l'entête (Header).
S_D0	Attente de l'octet de donnée 0 (LSB).
S_D1	Attente de l'octet de donnée 1 (MSB).
S_CRC	Attente du CRC8 de validation.

Definition at line 23 of file serial_cmd.c.

6.51.3 Function Documentation

6.51.3.1 serial_cmd_reader()

```
void serial_cmd_reader (
            void  )
```

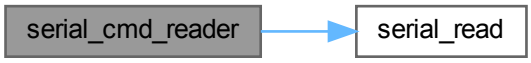
Fonction principale de lecture (Polling).

Fonction de lecture périodique des commandes entrantes.

Récupère les données brutes du buffer circulaire RX et les passe octet par octet à la machine à états.

Definition at line 138 of file serial_cmd.c.

Here is the call graph for this function:



Here is the caller graph for this function:

6.51.3.2 serial_send_data_frame()

```
void serial_send_data_frame (
            void  )
```

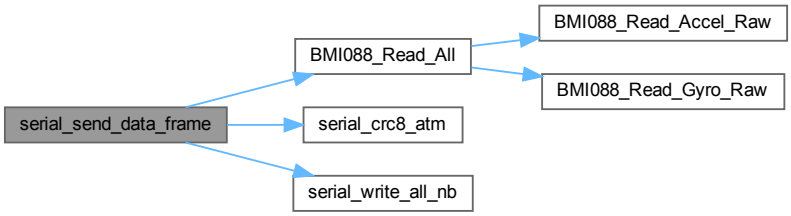
Construit et envoie la trame de télémétrie complète.

Capture les capteurs et envoie la trame de télémétrie.

1. Lit l'IMU (Accéléromètre + Gyroscope).
2. Récupère la vitesse (Speedometer).
3. Déduit le signe de la vitesse grâce à la commande moteur (Marche AR).
4. Formate le paquet binaire (SerialImuFrame_t) avec CRC.
5. Envoie le tout de manière non-bloquante via DMA.

Definition at line 156 of file serial_cmd.c.

Here is the call graph for this function:



6.51.4 Variable Documentation

6.51.4.1 parser_state

```
ParserSwitch parser_state = PARSER_IDLE
```

État courant du parseur, exposé à l'application principale pour déclencher les actions.

État global du parseur, lu par le main pour appliquer les consignes.

Definition at line 18 of file serial_cmd.c.

6.51.4.2 shadow_motor_cmd

```
int16_t shadow_motor_cmd = 0
```

Copie locale de la dernière commande moteur reçue.

Dernière consigne reçue pour le moteur (Shadow Register).

Definition at line 42 of file serial_cmd.c.

6.51.4.3 shadow_servo_cmd

```
int8_t shadow_servo_cmd = 0
```

Copie locale de la dernière commande servo reçue.

Dernière consigne reçue pour le servo (Shadow Register).

Definition at line 40 of file serial_cmd.c.

6.52 serial_cmd.c

Go to the documentation of this file.

```
00001
00009 #include "serial_cmd.h"
00010 #include "serial.h"
00011 #include "driver_motor.h"
00012 #include "driver_servo.h"
00013 #include "driver_ins.h"
00014 #include "app_main.h"
00015 #include <string.h>
00016
00018 ParserSwitch parser_state = PARSER_IDLE;
00019
00023 typedef enum{
00024     S_HDR=0,
00025     S_D0,
00026     S_D1,
00027     S_CRC
00028 } ParseState;
00029
00031 static ParseState st=S_HDR;
00033 static uint8_t hdr = 0;
00035 static uint8_t d0  = 0;
00037 static uint8_t d1  = 0;
00038
00040 int8_t  shadow_servo_cmd = 0;
00042 int16_t shadow_motor_cmd = 0;
00043
00045 static inline uint16_t to_u16(uint8_t lo,uint8_t hi){return(uint16_t)lo|((uint16_t)hi<<8);}
00047 static inline int16_t to_i16(uint8_t lo,uint8_t hi){return(int16_t)to_u16(lo,hi);}
00048
00055 static int16_t read_reg16(uint8_t addr){
00056     switch(addr){
00057         case REG_SERVO_CMD:return shadow_servo_cmd;
00058         case REG_MOTOR_CMD:return shadow_motor_cmd;
00059         case REG_BMI:return 0;
00060         default:return 0;
00061     }
00062 }
00063
00073 static void handle_frame(uint8_t hdr_b,uint8_t d0_b,uint8_t d1_b){
00074     const uint8_t  is_read = PROTO_IS_READ(hdr_b)?1u:0u;
00075     const uint8_t  addr    = PROTO_ADDR(hdr_b);
00076     const uint16_t udata16 = to_u16(d0_b,d1_b);
00077     const int16_t  data16  = to_i16(d0_b,d1_b);
00078
00079     (void)udata16;
00080
00081     if(is_read){
00082         uint8_t count=d0_b;
00083         for(uint8_t i=0; i<count; i++){
00084             uint8_t a = (uint8_t)((addr+i)&PROTO_HDR_ADDR_MASK);
00085             int16_t v = read_reg16(a);
00086             (void)proto_send_data16(a,v);
00087         }
00088         return;
00089     }
00090
00091     switch(addr){
00092         case REG_SERVO_CMD:
00093             parser_state = PARSER_SERVO_CMD;
00094             shadow_servo_cmd = data16;
00095         break;
00096
00097         case REG_MOTOR_CMD:
```

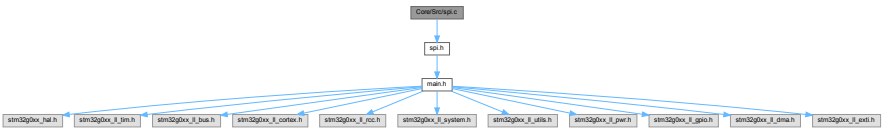
```
00098             parser_state = PARSER_MOTOR_CMD;
00099             shadow_motor_cmd = data16;
00100         break;
00101
00102         case REG_BMI:
00103             parser_state = PARSER_BMI_CMD;
00104         break;
00105
00106         default:
00107             parser_state = PARSER_OTHERS;
00108         break;
00109     }
00110 }
00111
00117 static void parse_byte(uint8_t b){
00118     switch(st){
00119         case S_HDR:hdr=b;st=S_D0;break;
00120         case S_D0:d0=b;st=S_D1;break;
00121         case S_D1:d1=b;st=S_CRC;break;
00122         case S_CRC:{
00123             uint8_t buf[3]={hdr,d0,d1};
00124             uint8_t crc=serial_crc8_atm(buf,3);
00125             if(crc==b)handle_frame(hdr,d0,d1);
00126             st=S_HDR;
00127         }
00128         break;
00129         default:st=S_HDR;break;
00130     }
00131 }
00132
00138 void serial_cmd_reader(void){
00139     uint8_t tmp[64];
00140     size_t n = serial_read(tmp,sizeof(tmp));
00141     if(!n)return;
00142     for(size_t i=0;i<n;i++){
00143         parse_byte(tmp[i]);
00144     }
00145 }
00146
00156 void serial_send_data_frame(void) {
00157     bmi088_data_t imu_data;
00158
00159     if (BMI088_Read_All(&imu_data) != BMI08_OK) {
00160         return;
00161     }
00162
00163     static SerialImuFrame_t frame;
00164
00165     frame.head1 = 0xAA;
00166     frame.head2 = 0x55;
00167     frame.type  = 0x01;
00168     /* payload: timestamp(4) + accel(12) + gyro(12) + speed(4) = 32 */
00169     frame.len   = 32;
00170     frame.timestamp = HAL_GetTick();
00171
00172     frame.accel[0] = imu_data.accel_x_mms2;
00173     frame.accel[1] = imu_data.accel_y_mms2;
00174     frame.accel[2] = imu_data.accel_z_mms2;
00175
00176     frame.gyro[0]  = imu_data.gyro_x_rads;
00177     frame.gyro[1]  = imu_data.gyro_y_rads;
00178     frame.gyro[2]  = imu_data.gyro_z_rads;
00179
00180     frame.speed = speed_speedo_data;
00181     frame.speed = (shadow_motor_cmd < 0) ? frame.speed * -1 : frame.speed; // Prise en compte de la
      commande pour le sens de rotation
00182
00183     uint8_t *raw_bytes = (uint8_t*)&frame;
00184
00185     frame.crc = serial_crc8_atm(raw_bytes, sizeof(SerialImuFrame_t) - 1);
00186
00187     serial_write_all_nb(raw_bytes, sizeof(SerialImuFrame_t));
00188 }
```

6.53 Core/Src/spi.c File Reference

This file provides code for the configuration of the SPI instances.

```
#include "spi.h"
```

Include dependency graph for spi.c:



Functions

- void MX_SPI1_Init (void)
- void HAL_SPI_MspInit (SPI_HandleTypeDef ∗spiHandle)
- void HAL_SPI_MspDeInit (SPI_HandleTypeDef ∗spiHandle)

Variables

- SPI_HandleTypeDef hspi1

6.53.1 Detailed Description

This file provides code for the configuration of the SPI instances.

Attention

Definition in file spi.c.

6.53.2 Function Documentation

6.53.2.1 HAL_SPI_MspDeInit()

```
void HAL_SPI_MspDeInit (
            SPI_HandleTypeDef * spiHandle )
```

SPI1 GPIO Configuration PA1 ------> SPI1_SCK PA6 ------> SPI1_MISO PA7 ------> SPI1_MOSI

Definition at line 98 of file spi.c.

6.53.2.2 HAL_SPI_MspInit()

```
void HAL_SPI_MspInit (
            SPI_HandleTypeDef * spiHandle )
```

SPI1 GPIO Configuration PA1 ----—> SPI1_SCK PA6 ----—> SPI1_MISO PA7 ----—> SPI1_MOSI

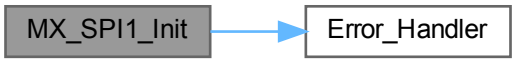
Definition at line 64 of file spi.c.

6.53.2.3 MX_SPI1_Init()

```
void MX_SPI1_Init (
            void  )
```

Definition at line 30 of file spi.c.

Here is the call graph for this function:



Here is the caller graph for this function:

6.53.3 Variable Documentation

6.53.3.1 hspi1

```
SPI_HandleTypeDef hspi1
```

Definition at line 27 of file spi.c.

6.54 spi.c

Go to the documentation of this file.

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020 /* Includes ------------------------------------------------------------------*/
00021 #include "spi.h"
00022
00023 /* USER CODE BEGIN 0 */
00024
00025 /* USER CODE END 0 */
00026
00027 SPI_HandleTypeDef hspi1;
00028
00029 /* SPI1 init function */
00030 void MX_SPI1_Init(void)
00031 {
00032
00033   /* USER CODE BEGIN SPI1_Init 0 */
00034
00035   /* USER CODE END SPI1_Init 0 */
00036
00037   /* USER CODE BEGIN SPI1_Init 1 */
00038
00039   /* USER CODE END SPI1_Init 1 */
00040   hspi1.Instance = SPI1;
00041   hspi1.Init.Mode = SPI_MODE_MASTER;
00042   hspi1.Init.Direction = SPI_DIRECTION_2LINES;
00043   hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
00044   hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
00045   hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
00046   hspi1.Init.NSS = SPI_NSS_SOFT;
00047   hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_32;
00048   hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
00049   hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
00050   hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
00051   hspi1.Init.CRCPolynomial = 7;
00052   hspi1.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
00053   hspi1.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
00054   if (HAL_SPI_Init(&hspi1) != HAL_OK)
00055   {
00056     Error_Handler();
00057   }
00058   /* USER CODE BEGIN SPI1_Init 2 */
00059
00060   /* USER CODE END SPI1_Init 2 */
00061
00062 }
00063
00064 void HAL_SPI_MspInit(SPI_HandleTypeDef* spiHandle)
00065 {
00066
00067   GPIO_InitTypeDef GPIO_InitStruct = {0};
00068   if(spiHandle->Instance==SPI1)
00069   {
00070   /* USER CODE BEGIN SPI1_MspInit 0 */
00071
00072   /* USER CODE END SPI1_MspInit 0 */
00073     /* SPI1 clock enable */
00074     __HAL_RCC_SPI1_CLK_ENABLE();
00075
00076     __HAL_RCC_GPIOA_CLK_ENABLE();
00082     GPIO_InitStruct.Pin = GPIO_PIN_1|GPIO_PIN_6|GPIO_PIN_7;
00083     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
00084     GPIO_InitStruct.Pull = GPIO_NOPULL;
00085     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00086     GPIO_InitStruct.Alternate = GPIO_AF0_SPI1;
00087     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
00088
00089     /* SPI1 interrupt Init */
00090     HAL_NVIC_SetPriority(SPI1_IRQn, 0, 0);
00091     HAL_NVIC_EnableIRQ(SPI1_IRQn);
00092   /* USER CODE BEGIN SPI1_MspInit 1 */
00093
00094   /* USER CODE END SPI1_MspInit 1 */
00095   }
00096 }
00097
00098 void HAL_SPI_MspDeInit(SPI_HandleTypeDef* spiHandle)
00099 {
00100
00101   if(spiHandle->Instance==SPI1)
00102   {
00103   /* USER CODE BEGIN SPI1_MspDeInit 0 */
00104
```

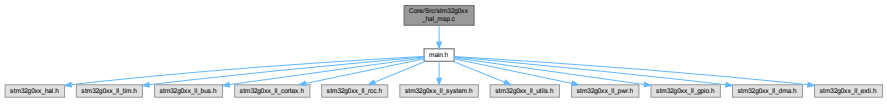
```
00105   /* USER CODE END SPI1_MspDeInit 0 */
00106     /* Peripheral clock disable */
00107     __HAL_RCC_SPI1_CLK_DISABLE();
00108
00114     HAL_GPIO_DeInit(GPIOA, GPIO_PIN_1|GPIO_PIN_6|GPIO_PIN_7);
00115
00116     /* SPI1 interrupt Deinit */
00117     HAL_NVIC_DisableIRQ(SPI1_IRQn);
00118   /* USER CODE BEGIN SPI1_MspDeInit 1 */
00119
00120   /* USER CODE END SPI1_MspDeInit 1 */
00121   }
00122 }
00123
00124 /* USER CODE BEGIN 1 */
00125
00126 /* USER CODE END 1 */
```

6.55 Core/Src/stm32g0xx_hal_msp.c File Reference

This file provides code for the MSP Initialization and de-Initialization codes.

```
#include "main.h"
```

Include dependency graph for stm32g0xx_hal_msp.c:



Functions

- void HAL_MspInit (void)

6.55.1 Detailed Description

This file provides code for the MSP Initialization and de-Initialization codes.

Attention

Definition in file stm32g0xx_hal_msp.c.

6.55.2 Function Documentation

6.55.2.1 HAL_MspInit()

```
void HAL_MspInit (
            void  )
```

Initializes the Global MSP. Disable the internal Pull-Up in Dead Battery pins of UCPD peripheral

Definition at line 63 of file stm32g0xx_hal_msp.c.

6.56 stm32g0xx_hal_msp.c

Go to the documentation of this file.

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020
00021 /* Includes ------------------------------------------------------------------*/
00022 #include "main.h"
00023 /* USER CODE BEGIN Includes */
00024
00025 /* USER CODE END Includes */
00026
00027 /* Private typedef -----------------------------------------------------------*/
00028 /* USER CODE BEGIN TD */
00029
00030 /* USER CODE END TD */
00031
00032 /* Private define ------------------------------------------------------------*/
00033 /* USER CODE BEGIN Define */
00034
00035 /* USER CODE END Define */
00036
00037 /* Private macro -------------------------------------------------------------*/
00038 /* USER CODE BEGIN Macro */
00039
00040 /* USER CODE END Macro */
00041
00042 /* Private variables ---------------------------------------------------------*/
00043 /* USER CODE BEGIN PV */
00044
00045 /* USER CODE END PV */
00046
00047 /* Private function prototypes -----------------------------------------------*/
00048 /* USER CODE BEGIN PFP */
00049
00050 /* USER CODE END PFP */
00051
00052 /* External functions --------------------------------------------------------*/
00053 /* USER CODE BEGIN ExternalFunctions */
00054
00055 /* USER CODE END ExternalFunctions */
00056
00057 /* USER CODE BEGIN 0 */
00058
00059 /* USER CODE END 0 */
00063 void HAL_MspInit(void)
00064 {
00065
00066   /* USER CODE BEGIN MspInit 0 */
00067
00068   /* USER CODE END MspInit 0 */
00069
00070   __HAL_RCC_SYSCFG_CLK_ENABLE();
00071   __HAL_RCC_PWR_CLK_ENABLE();
00072
00073   /* System interrupt init*/
00074
00077   HAL_SYSCFG_StrobeDBattpinsConfig(SYSCFG_CFGR1_UCPD1_STROBE | SYSCFG_CFGR1_UCPD2_STROBE);
00078
00079   /* USER CODE BEGIN MspInit 1 */
00080
00081   /* USER CODE END MspInit 1 */
00082 }
00083
00084 /* USER CODE BEGIN 1 */
00085
00086 /* USER CODE END 1 */
```

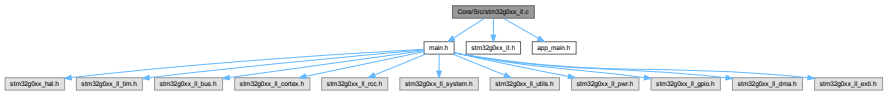
6.57 Core/Src/stm32g0xx_it.c File Reference

Interrupt Service Routines.

```
#include "main.h"
#include "stm32g0xx_it.h"
```

```
#include "app_main.h"
```

Include dependency graph for stm32g0xx_it.c:



Functions

- void NMI_Handler (void)

  This function handles Non maskable interrupt.
- void HardFault_Handler (void)

  This function handles Hard fault interrupt.
- void SVC_Handler (void)

  This function handles System service call via SWI instruction.
- void PendSV_Handler (void)

  This function handles Pendable request for system service.
- void SysTick_Handler (void)

  This function handles System tick timer.
- void DMA1_Channel1_IRQHandler (void)

  This function handles DMA1 channel 1 interrupt.
- void DMA1_Channel2_3_IRQHandler (void)

  This function handles DMA1 channel 2 and channel 3 interrupts.
- void TIM3_TIM4_IRQHandler (void)

  This function handles TIM3, TIM4 global Interrupt.
- void SPI1_IRQHandler (void)

  This function handles SPI1/I2S1 Interrupt.
- void USART2_LPUART2_IRQHandler (void)

  This function handles USART2 + LPUART2 Interrupt.

Variables

- SPI_HandleTypeDef hspi1
- TIM_HandleTypeDef htim4
- DMA_HandleTypeDef hdma_usart2_rx
- DMA_HandleTypeDef hdma_usart2_tx
- UART_HandleTypeDef huart2

6.57.1 Detailed Description

Interrupt Service Routines.

Attention

Definition in file stm32g0xx_it.c.

6.57.2 Function Documentation

6.57.2.1 DMA1_Channel1_IRQHandler()

```
void DMA1_Channel1_IRQHandler (
            void  )
```

This function handles DMA1 channel 1 interrupt.

Definition at line 151 of file stm32g0xx_it.c.

6.57.2.2 DMA1_Channel2_3_IRQHandler()

```
void DMA1_Channel2_3_IRQHandler (
            void  )
```

This function handles DMA1 channel 2 and channel 3 interrupts.

Definition at line 165 of file stm32g0xx_it.c.

6.57.2.3 HardFault_Handler()

```
void HardFault_Handler (
            void  )
```

This function handles Hard fault interrupt.

Definition at line 89 of file stm32g0xx_it.c.

6.57.2.4 NMI_Handler()

```
void NMI_Handler (
            void  )
```

This function handles Non maskable interrupt.

Definition at line 74 of file stm32g0xx_it.c.

6.57.2.5 PendSV_Handler()

```
void PendSV_Handler (
            void  )
```

This function handles Pendable request for system service.

Definition at line 117 of file stm32g0xx_it.c.

6.57.2.6 SPI1_IRQHandler()

```
void SPI1_IRQHandler (
             void  )
```

This function handles SPI1/I2S1 Interrupt.

Definition at line 200 of file stm32g0xx_it.c.

6.57.2.7 SVC_Handler()

```
void SVC_Handler (
             void  )
```

This function handles System service call via SWI instruction.

Definition at line 104 of file stm32g0xx_it.c.

6.57.2.8 SysTick_Handler()

```
void SysTick_Handler (
             void  )
```

This function handles System tick timer.

Definition at line 130 of file stm32g0xx_it.c.

6.57.2.9 TIM3_TIM4_IRQHandler()

```
void TIM3_TIM4_IRQHandler (
             void  )
```

This function handles TIM3, TIM4 global Interrupt.

-

Definition at line 179 of file stm32g0xx_it.c.

6.57.2.10 USART2_LPUART2_IRQHandler()

```
void USART2_LPUART2_IRQHandler (
             void  )
```

This function handles USART2 + LPUART2 Interrupt.

Definition at line 214 of file stm32g0xx_it.c.

6.57.3 Variable Documentation

6.57.3.1 hdma_usart2_rx

```
DMA_HandleTypeDef hdma_usart2_rx  [extern]
```

Definition at line 28 of file usart.c.

6.57.3.2 hdma_usart2_tx

```
DMA_HandleTypeDef hdma_usart2_tx  [extern]
```

Definition at line 29 of file usart.c.

6.57.3.3 hspi1

```
SPI_HandleTypeDef hspi1  [extern]
```

Definition at line 27 of file spi.c.

6.57.3.4 htim4

```
TIM_HandleTypeDef htim4  [extern]
```

Definition at line 29 of file tim.c.

6.57.3.5 huart2

```
UART_HandleTypeDef huart2  [extern]
```

Definition at line 27 of file usart.c.

6.58 stm32g0xx_it.c

Go to the documentation of this file.

```
00001 /* USER CODE BEGIN Header */
00018 /* USER CODE END Header */
00019
00020 /* Includes ------------------------------------------------------------------*/
00021 #include "main.h"
00022 #include "stm32g0xx_it.h"
00023 /* Private includes ----------------------------------------------------------*/
00024 /* USER CODE BEGIN Includes */
00025 #include "app_main.h"
00026 /* USER CODE END Includes */
00027
00028 /* Private typedef -----------------------------------------------------------*/
00029 /* USER CODE BEGIN TD */
00030
00031 /* USER CODE END TD */
00032
00033 /* Private define ------------------------------------------------------------*/
00034 /* USER CODE BEGIN PD */
00035
00036 /* USER CODE END PD */
00037
00038 /* Private macro -------------------------------------------------------------*/
00039 /* USER CODE BEGIN PM */
00040
00041 /* USER CODE END PM */
00042
00043 /* Private variables ---------------------------------------------------------*/
00044 /* USER CODE BEGIN PV */
00045
00046 /* USER CODE END PV */
00047
00048 /* Private function prototypes -----------------------------------------------*/
00049 /* USER CODE BEGIN PFP */
00050
00051 /* USER CODE END PFP */
00052
00053 /* Private user code ---------------------------------------------------------*/
00054 /* USER CODE BEGIN 0 */
00055
00056 /* USER CODE END 0 */
00057
00058 /* External variables --------------------------------------------------------*/
00059 extern SPI_HandleTypeDef hspi1;
00060 extern TIM_HandleTypeDef htim4;
00061 extern DMA_HandleTypeDef hdma_usart2_rx;
00062 extern DMA_HandleTypeDef hdma_usart2_tx;
00063 extern UART_HandleTypeDef huart2;
00064 /* USER CODE BEGIN EV */
00065
00066 /* USER CODE END EV */
00067
00068 /******************************************************************************/
00069 /*           Cortex-M0+ Processor Interruption and Exception Handlers          */
00070 /******************************************************************************/
00074 void NMI_Handler(void)
00075 {
00076   /* USER CODE BEGIN NonMaskableInt_IRQn 0 */
00077
00078   /* USER CODE END NonMaskableInt_IRQn 0 */
00079   /* USER CODE BEGIN NonMaskableInt_IRQn 1 */
00080    while (1)
00081   {
00082   }
00083   /* USER CODE END NonMaskableInt_IRQn 1 */
00084 }
00085
00089 void HardFault_Handler(void)
00090 {
00091   /* USER CODE BEGIN HardFault_IRQn 0 */
00092
00093   /* USER CODE END HardFault_IRQn 0 */
00094   while (1)
00095   {
00096     /* USER CODE BEGIN W1_HardFault_IRQn 0 */
00097     /* USER CODE END W1_HardFault_IRQn 0 */
00098   }
00099 }
00100
00104 void SVC_Handler(void)
00105 {
00106   /* USER CODE BEGIN SVC_IRQn 0 */
00107
```

```
00108   /* USER CODE END SVC_IRQn 0 */
00109   /* USER CODE BEGIN SVC_IRQn 1 */
00110
00111   /* USER CODE END SVC_IRQn 1 */
00112 }
00113
00117 void PendSV_Handler(void)
00118 {
00119   /* USER CODE BEGIN PendSV_IRQn 0 */
00120
00121   /* USER CODE END PendSV_IRQn 0 */
00122   /* USER CODE BEGIN PendSV_IRQn 1 */
00123
00124   /* USER CODE END PendSV_IRQn 1 */
00125 }
00126
00130 void SysTick_Handler(void)
00131 {
00132   /* USER CODE BEGIN SysTick_IRQn 0 */
00133
00134   /* USER CODE END SysTick_IRQn 0 */
00135   HAL_IncTick();
00136   /* USER CODE BEGIN SysTick_IRQn 1 */
00137
00138   /* USER CODE END SysTick_IRQn 1 */
00139 }
00140
00141 /******************************************************************************/
00142 /* STM32G0xx Peripheral Interrupt Handlers                                    */
00143 /* Add here the Interrupt Handlers for the used peripherals.                  */
00144 /* For the available peripheral interrupt handler names,                      */
00145 /* please refer to the startup file (startup_stm32g0xx.s).                    */
00146 /******************************************************************************/
00147
00151 void DMA1_Channel1_IRQHandler(void)
00152 {
00153   /* USER CODE BEGIN DMA1_Channel1_IRQn 0 */
00154
00155   /* USER CODE END DMA1_Channel1_IRQn 0 */
00156   HAL_DMA_IRQHandler(&hdma_usart2_rx);
00157   /* USER CODE BEGIN DMA1_Channel1_IRQn 1 */
00158
00159   /* USER CODE END DMA1_Channel1_IRQn 1 */
00160 }
00161
00165 void DMA1_Channel2_3_IRQHandler(void)
00166 {
00167   /* USER CODE BEGIN DMA1_Channel2_3_IRQn 0 */
00168
00169   /* USER CODE END DMA1_Channel2_3_IRQn 0 */
00170   HAL_DMA_IRQHandler(&hdma_usart2_tx);
00171   /* USER CODE BEGIN DMA1_Channel2_3_IRQn 1 */
00172
00173   /* USER CODE END DMA1_Channel2_3_IRQn 1 */
00174 }
00175
00179 void TIM3_TIM4_IRQHandler(void)
00180 {
00181   /* USER CODE BEGIN TIM3_TIM4_IRQn 0 */
00182
00184     if(LL_TIM_IsActiveFlag_UPDATE(TIM3)){
00185         LL_TIM_ClearFlag_UPDATE(TIM3);
00186         tim3_overflow_cnt++;
00187      }
00188      //*/
00189
00190   /* USER CODE END TIM3_TIM4_IRQn 0 */
00191   HAL_TIM_IRQHandler(&htim4);
00192   /* USER CODE BEGIN TIM3_TIM4_IRQn 1 */
00193
00194   /* USER CODE END TIM3_TIM4_IRQn 1 */
00195 }
00196
00200 void SPI1_IRQHandler(void)
00201 {
00202   /* USER CODE BEGIN SPI1_IRQn 0 */
00203
00204   /* USER CODE END SPI1_IRQn 0 */
00205   HAL_SPI_IRQHandler(&hspi1);
00206   /* USER CODE BEGIN SPI1_IRQn 1 */
00207
00208   /* USER CODE END SPI1_IRQn 1 */
00209 }
00210
00214 void USART2_LPUART2_IRQHandler(void)
00215 {
00216   /* USER CODE BEGIN USART2_LPUART2_IRQn 0 */
```

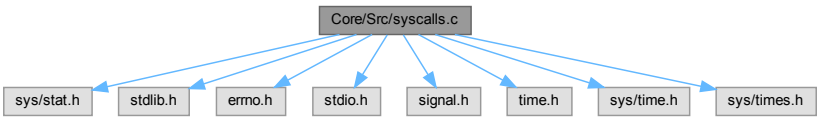
```
00217
00218   /* USER CODE END USART2_LPUART2_IRQn 0 */
00219   HAL_UART_IRQHandler(&huart2);
00220   /* USER CODE BEGIN USART2_LPUART2_IRQn 1 */
00221
00222   /* USER CODE END USART2_LPUART2_IRQn 1 */
00223 }
00224
00225 /* USER CODE BEGIN 1 */
00226
00227 /* USER CODE END 1 */
```

6.59 Core/Src/syscalls.c File Reference

STM32CubeIDE Minimal System calls file.

```
#include <sys/stat.h>
#include <stdlib.h>
#include <errno.h>
#include <stdio.h>
#include <signal.h>
#include <time.h>
#include <sys/time.h>
#include <sys/times.h>
```

Include dependency graph for syscalls.c:



Functions

- int __io_putchar (int ch) __attribute__((weak))
- int __io_getchar (void)
- void initialise_monitor_handles ()
- int _getpid (void)
- int _kill (int pid, int sig)
- void _exit (int status)
- __attribute__ ((weak))
- int _close (int file)
- int _fstat (int file, struct stat *st)
- int _isatty (int file)
- int _lseek (int file, int ptr, int dir)
- int _open (char *path, int flags,...)
- int _wait (int *status)
- int _unlink (char *name)
- int _times (struct tms *buf)
- int _stat (char *file, struct stat *st)
- int _link (char *old, char *new)
- int _fork (void)
- int _execve (char *name, char **argv, char **env)

Variables

- char ** environ = __env

6.59.1 Detailed Description

STM32CubeIDE Minimal System calls file.

Author

Auto-generated by STM32CubeIDE

```
For more information about which c-functions
need which of these lowlevel functions
please consult the Newlib libc-manual
```

Attention

Definition in file syscalls.c.

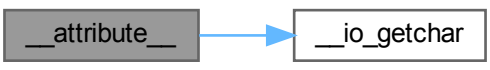
6.59.2 Function Documentation

6.59.2.1 __attribute__()

```
__attribute__ (
             (weak)  )
```

Definition at line 67 of file syscalls.c.

Here is the call graph for this function:

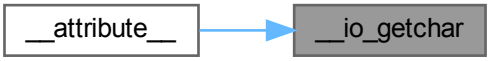


6.59.2.2 __io_getchar()

```
int __io_getchar (
            void  )  [extern]
```

Definition at line 36 of file syscalls.c.

Here is the caller graph for this function:



6.59.2.3 __io_putchar()

```
int __io_putchar (
            int ch )  [extern]
```

Definition at line 163 of file main.c.

6.59.2.4 _close()

```
int _close (
            int file )
```

Definition at line 92 of file syscalls.c.

6.59.2.5 _execve()

```
int _execve (
            char * name,
            char ** argv,
            char ** env )
```

Definition at line 169 of file syscalls.c.

6.59.2.6 _exit()

```
void _exit (
            int status )
```

Definition at line 61 of file syscalls.c.

Here is the call graph for this function:

6.59.2.7 _fork()

```
int _fork (
            void  )
```

Definition at line 163 of file syscalls.c.

6.59.2.8 _fstat()

```
int _fstat (
            int file,
            struct stat * st )
```

Definition at line 99 of file syscalls.c.

6.59.2.9 _getpid()

```
int _getpid (
            void  )
```

Definition at line 48 of file syscalls.c.

6.59.2.10 _isatty()

```
int _isatty (
            int file )
```

Definition at line 106 of file syscalls.c.

6.59.2.11 _kill()

```
int _kill (
              int pid,
              int sig )
```

Definition at line 53 of file syscalls.c.

Here is the caller graph for this function:

6.59.2.12 _link()

```
int _link (
              char * old,
              char * new )
```

Definition at line 155 of file syscalls.c.

6.59.2.13 _lseek()

```
int _lseek (
              int file,
              int ptr,
              int dir )
```

Definition at line 112 of file syscalls.c.

6.59.2.14 _open()

```
int _open (
              char * path,
              int flags,
               ...  )
```

Definition at line 120 of file syscalls.c.

6.59.2.15 _stat()

```
int _stat (
              char * file,
              struct stat * st )
```

Definition at line 148 of file syscalls.c.

6.59.2.16 _times()

```
int _times (
              struct tms * buf )
```

Definition at line 142 of file syscalls.c.

6.59.2.17 _unlink()

```
int _unlink (
              char * name )
```

Definition at line 135 of file syscalls.c.

6.59.2.18 _wait()

```
int _wait (
              int * status )
```

Definition at line 128 of file syscalls.c.

6.59.2.19 initialise_monitor_handles()

```
void initialise_monitor_handles ( )
```

Definition at line 44 of file syscalls.c.

6.59.3 Variable Documentation

6.59.3.1 environ

```
char** environ = __env
```

Definition at line 40 of file syscalls.c.

6.60 syscalls.c

Go to the documentation of this file.

```
00001
00023 /* Includes */
00024 #include <sys/stat.h>
00025 #include <stdlib.h>
00026 #include <errno.h>
00027 #include <stdio.h>
00028 #include <signal.h>
00029 #include <time.h>
00030 #include <sys/time.h>
00031 #include <sys/times.h>
00032
00033
00034 /* Variables */
00035 extern int __io_putchar(int ch) __attribute__((weak));
00036 extern int __io_getchar(void) __attribute__((weak));
00037
00038
00039 char *__env[1] = { 0 };
00040 char **environ = __env;
00041
00042
00043 /* Functions */
00044 void initialise_monitor_handles()
00045 {
00046 }
00047
00048 int _getpid(void)
00049 {
00050   return 1;
00051 }
00052
00053 int _kill(int pid, int sig)
00054 {
00055   (void)pid;
00056   (void)sig;
00057   errno = EINVAL;
00058   return -1;
00059 }
00060
00061 void _exit (int status)
00062 {
00063   _kill(status, -1);
00064   while (1) {}    /* Make sure we hang here */
00065 }
00066
00067 __attribute__((weak)) int _read(int file, char *ptr, int len)
00068 {
00069   (void)file;
00070   int DataIdx;
00071
00072   for (DataIdx = 0; DataIdx < len; DataIdx++)
00073   {
00074     *ptr++ = __io_getchar();
00075   }
00076
00077   return len;
00078 }
00079
00080 __attribute__((weak)) int _write(int file, char *ptr, int len)
00081 {
00082   (void)file;
00083   int DataIdx;
00084
00085   for (DataIdx = 0; DataIdx < len; DataIdx++)
00086   {
00087     __io_putchar(*ptr++);
00088   }
00089   return len;
00090 }
00091
00092 int _close(int file)
00093 {
00094   (void)file;
00095   return -1;
00096 }
00097
00098
00099 int _fstat(int file, struct stat *st)
00100 {
00101   (void)file;
00102   st->st_mode = S_IFCHR;
00103   return 0;
```

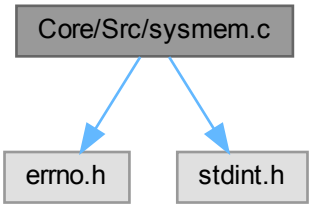
```
00104 }
00105
00106 int _isatty(int file)
00107 {
00108   (void)file;
00109   return 1;
00110 }
00111
00112 int _lseek(int file, int ptr, int dir)
00113 {
00114   (void)file;
00115   (void)ptr;
00116   (void)dir;
00117   return 0;
00118 }
00119
00120 int _open(char *path, int flags, ...)
00121 {
00122   (void)path;
00123   (void)flags;
00124   /* Pretend like we always fail */
00125   return -1;
00126 }
00127
00128 int _wait(int *status)
00129 {
00130   (void)status;
00131   errno = ECHILD;
00132   return -1;
00133 }
00134
00135 int _unlink(char *name)
00136 {
00137   (void)name;
00138   errno = ENOENT;
00139   return -1;
00140 }
00141
00142 int _times(struct tms *buf)
00143 {
00144   (void)buf;
00145   return -1;
00146 }
00147
00148 int _stat(char *file, struct stat *st)
00149 {
00150   (void)file;
00151   st->st_mode = S_IFCHR;
00152   return 0;
00153 }
00154
00155 int _link(char *old, char *new)
00156 {
00157   (void)old;
00158   (void)new;
00159   errno = EMLINK;
00160   return -1;
00161 }
00162
00163 int _fork(void)
00164 {
00165   errno = EAGAIN;
00166   return -1;
00167 }
00168
00169 int _execve(char *name, char **argv, char **env)
00170 {
00171   (void)name;
00172   (void)argv;
00173   (void)env;
00174   errno = ENOMEM;
00175   return -1;
00176 }
```

6.61 Core/Src/sysmem.c File Reference

STM32CubeIDE System Memory calls file.

```
#include <errno.h>
#include <stdint.h>
```

Include dependency graph for sysmem.c:



Functions

- void * _sbrk (ptrdiff_t incr)

  _sbrk() allocates memory to the newlib heap and is used by malloc and others from the C library

6.61.1 Detailed Description

STM32CubeIDE System Memory calls file.

Author

Generated by STM32CubeIDE

```
For more information about which C functions
need which of these lowlevel functions
please consult the newlib libc manual
```

Attention

Definition in file sysmem.c.

6.61.2 Function Documentation

6.61.2.1 _sbrk()

```
void * _sbrk (
             ptrdiff_t incr )
```

_sbrk() allocates memory to the newlib heap and is used by malloc and others from the C library

```
* ##########################################################################
* #  .data  #  .bss  #       newlib heap       #          MSP stack          #
* #         #        #                         # Reserved by _Min_Stack_Size #
* ##########################################################################
* ^-- RAM start      ^-- _end                             _estack, RAM end --^
*
```

This implementation starts allocating at the '_end' linker symbol The '_Min_Stack_Size' linker symbol reserves a memory for the MSP stack The implementation considers '_estack' linker symbol to be RAM end NOTE: If the MSP stack, at any point during execution, grows larger than the reserved size, please increase the '_Min_Stack_Size'.

Parameters

incr	Memory size

Returns

Pointer to allocated memory

Definition at line 53 of file sysmem.c.

6.62 sysmem.c

Go to the documentation of this file.

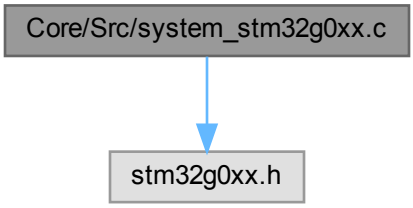
```
00001
00023 /* Includes */
00024 #include <errno.h>
00025 #include <stdint.h>
00026
00030 static uint8_t *__sbrk_heap_end = NULL;
00031
00053 void *_sbrk(ptrdiff_t incr)
00054 {
00055   extern uint8_t _end; /* Symbol defined in the linker script */
00056   extern uint8_t _estack; /* Symbol defined in the linker script */
00057   extern uint32_t _Min_Stack_Size; /* Symbol defined in the linker script */
00058   const uint32_t stack_limit = (uint32_t)&_estack - (uint32_t)&_Min_Stack_Size;
00059   const uint8_t *max_heap = (uint8_t *)stack_limit;
00060   uint8_t *prev_heap_end;
00061
00062   /* Initialize heap end at first call */
00063   if (NULL == __sbrk_heap_end)
00064   {
00065     __sbrk_heap_end = &_end;
00066   }
00067
00068   /* Protect heap from growing into the reserved MSP stack */
00069   if (__sbrk_heap_end + incr > max_heap)
00070   {
00071     errno = ENOMEM;
00072     return (void *)-1;
00073   }
00074
00075   prev_heap_end = __sbrk_heap_end;
00076   __sbrk_heap_end += incr;
00077
00078   return (void *)prev_heap_end;
00079 }
```

6.63 Core/Src/system_stm32g0xx.c File Reference

CMSIS Cortex-M0+ Device Peripheral Access Layer System Source File.

```
#include "stm32g0xx.h"
```

Include dependency graph for system_stm32g0xx.c:



Macros

- #define HSE_VALUE (8000000UL)
- #define HSI_VALUE (16000000UL)
- #define LSI_VALUE (32000UL)
- #define LSE_VALUE (32768UL)

Functions

- void SystemInit (void)

  Setup the microcontroller system.
- void SystemCoreClockUpdate (void)

  Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

Variables

- uint32_t SystemCoreClock = 16000000UL
- const uint32_t AHBPrescTable [16UL] = {0UL, 0UL, 0UL, 0UL, 0UL, 0UL, 0UL, 0UL, 1UL, 2UL, 3UL, 4UL, 6UL, 7UL, 8UL, 9UL}
- const uint32_t APBPrescTable [8UL] = {0UL, 0UL, 0UL, 0UL, 1UL, 2UL, 3UL, 4UL}

6.63.1 Detailed Description

CMSIS Cortex-M0+ Device Peripheral Access Layer System Source File.

Author

MCD Application Team

This file provides two functions and one global variable to be called from user application:

- SystemInit(): This function is called at startup just after reset and before branch to main program. This call is made inside the "startup_stm32g0xx.s" file.
- SystemCoreClock variable: Contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.
- SystemCoreClockUpdate(): Updates the variable SystemCoreClock and must be called whenever the core clock is changed during program execution.

After each device reset the HSI (8 MHz then 16 MHz) is used as system clock source. Then SystemInit() function is called, in "startup_stm32g0xx.s" file, to configure the system clock before to branch to main program.

6.63.2 This file configures the system clock as follows:

6.63.2.1 System Clock source | HSI

6.63.2.2 SYSCLK(Hz) | 16000000

6.63.2.3 HCLK(Hz) | 16000000

6.63.2.4 AHB Prescaler | 1

6.63.2.5 APB Prescaler | 1

6.63.2.6 HSI Division factor | 1

6.63.2.7 PLL_M | 1

6.63.2.8 PLL_N | 8

6.63.2.9 PLL_P | 7

6.63.2.10 PLL_Q | 2

6.63.2.11 PLL_R | 2

6.63.2.12 Require 48MHz for RNG | Disabled

==========

Attention

Definition in file system_stm32g0xx.c.

6.64 system_stm32g0xx.c

Go to the documentation of this file.

```
00001
00077 #include "stm32g0xx.h"
00078
00079 #if !defined  (HSE_VALUE)
00080 #define HSE_VALUE    (8000000UL)
00081 #endif /* HSE_VALUE */
00082
00083 #if !defined  (HSI_VALUE)
00084   #define HSI_VALUE  (16000000UL)
00085 #endif /* HSI_VALUE */
00086
00087 #if !defined  (LSI_VALUE)
00088  #define LSI_VALUE   (32000UL)
00089 #endif /* LSI_VALUE */
00090
00091 #if !defined  (LSE_VALUE)
00092   #define LSE_VALUE  (32768UL)
00093 #endif /* LSE_VALUE */
00094
00111 /************************ Miscellaneous Configuration ************************/
00112 /* Note: Following vector table addresses must be defined in line with linker
00113          configuration. */
00117 /* #define USER_VECT_TAB_ADDRESS */
00118
00119 #if defined(USER_VECT_TAB_ADDRESS)
00122 /* #define VECT_TAB_SRAM */
00123 #if defined(VECT_TAB_SRAM)
00124 #define VECT_TAB_BASE_ADDRESS   SRAM_BASE
00126 #define VECT_TAB_OFFSET         0x00000000U
00128 #else
00129 #define VECT_TAB_BASE_ADDRESS   FLASH_BASE
00131 #define VECT_TAB_OFFSET         0x00000000U
00133 #endif /* VECT_TAB_SRAM */
00134 #endif /* USER_VECT_TAB_ADDRESS */
00135 /******************************************************************************/
00151   /* The SystemCoreClock variable is updated in three ways:
00152       1) by calling CMSIS function SystemCoreClockUpdate()
```

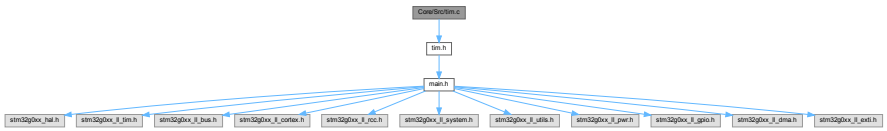
```
00153       2) by calling HAL API function HAL_RCC_GetHCLKFreq()
00154       3) each time HAL_RCC_ClockConfig() is called to configure the system clock frequency
00155          Note: If you use this function to configure the system clock; then there
00156                is no need to call the 2 first functions listed above, since SystemCoreClock
00157                variable is updated automatically.
00158   */
00159   uint32_t SystemCoreClock = 16000000UL;
00160
00161   const uint32_t AHBPrescTable[16UL] = {0UL, 0UL, 0UL, 0UL, 0UL, 0UL, 0UL, 0UL, 1UL, 2UL, 3UL, 4UL,
      6UL, 7UL, 8UL, 9UL};
00162   const uint32_t APBPrescTable[8UL] =  {0UL, 0UL, 0UL, 0UL, 1UL, 2UL, 3UL, 4UL};
00163
00185 void SystemInit(void)
00186 {
00187   /* Configure the Vector Table location ----------------------------------*/
00188 #if defined(USER_VECT_TAB_ADDRESS)
00189   SCB->VTOR = VECT_TAB_BASE_ADDRESS | VECT_TAB_OFFSET; /* Vector Table Relocation */
00190 #endif /* USER_VECT_TAB_ADDRESS */
00191 }
00192
00233 void SystemCoreClockUpdate(void)
00234 {
00235   uint32_t tmp;
00236   uint32_t pllvco;
00237   uint32_t pllr;
00238   uint32_t pllsource;
00239   uint32_t pllm;
00240   uint32_t hsidiv;
00241
00242   /* Get SYSCLK source -------------------------------------------------------*/
00243   switch (RCC->CFGR & RCC_CFGR_SWS)
00244   {
00245     case RCC_CFGR_SWS_0:                /* HSE used as system clock */
00246       SystemCoreClock = HSE_VALUE;
00247       break;
00248
00249     case (RCC_CFGR_SWS_1 | RCC_CFGR_SWS_0):  /* LSI used as system clock */
00250       SystemCoreClock = LSI_VALUE;
00251       break;
00252
00253     case RCC_CFGR_SWS_2:                /* LSE used as system clock */
00254       SystemCoreClock = LSE_VALUE;
00255       break;
00256
00257     case RCC_CFGR_SWS_1:  /* PLL used as system clock */
00258       /* PLL_VCO = (HSE_VALUE or HSI_VALUE / PLLM) * PLLN
00259          SYSCLK = PLL_VCO / PLLR
00260          */
00261       pllsource = (RCC->PLLCFGR & RCC_PLLCFGR_PLLSRC);
00262       pllm = ((RCC->PLLCFGR & RCC_PLLCFGR_PLLM) >> RCC_PLLCFGR_PLLM_Pos) + 1UL;
00263
00264       if(pllsource == 0x03UL)           /* HSE used as PLL clock source */
00265       {
00266         pllvco = (HSE_VALUE / pllm);
00267       }
00268       else                              /* HSI used as PLL clock source */
00269       {
00270           pllvco = (HSI_VALUE / pllm);
00271       }
00272       pllvco = pllvco * ((RCC->PLLCFGR & RCC_PLLCFGR_PLLN) >> RCC_PLLCFGR_PLLN_Pos);
00273       pllr = (((RCC->PLLCFGR & RCC_PLLCFGR_PLLR) >> RCC_PLLCFGR_PLLR_Pos) + 1UL);
00274
00275       SystemCoreClock = pllvco/pllr;
00276       break;
00277
00278     case 0x00000000U:                   /* HSI used as system clock */
00279     default:                            /* HSI used as system clock */
00280       hsidiv = (1UL << ((READ_BIT(RCC->CR, RCC_CR_HSIDIV))>> RCC_CR_HSIDIV_Pos));
00281       SystemCoreClock = (HSI_VALUE/hsidiv);
00282       break;
00283   }
00284   /* Compute HCLK clock frequency --------------------------------------------*/
00285   /* Get HCLK prescaler */
00286   tmp = AHBPrescTable[((RCC->CFGR & RCC_CFGR_HPRE) >> RCC_CFGR_HPRE_Pos)];
00287   /* HCLK clock frequency */
00288   SystemCoreClock >>= tmp;
00289 }
00290
00291
```

6.65 Core/Src/tim.c File Reference

This file provides code for the configuration of the TIM instances.

```
#include "tim.h"
```

Include dependency graph for tim.c:



Functions

- void MX_TIM1_Init (void)
- void MX_TIM2_Init (void)
- void MX_TIM3_Init (void)
- void MX_TIM4_Init (void)
- void HAL_TIM_Base_MspInit (TIM_HandleTypeDef *tim_baseHandle)
- void HAL_TIM_MspPostInit (TIM_HandleTypeDef *timHandle)
- void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef *tim_baseHandle)

Variables

- TIM_HandleTypeDef htim1
- TIM_HandleTypeDef htim2
- TIM_HandleTypeDef htim4

6.65.1 Detailed Description

This file provides code for the configuration of the TIM instances.

Attention

Definition in file tim.c.

6.65.2 Function Documentation

6.65.2.1 HAL_TIM_Base_MspDeInit()

```
void HAL_TIM_Base_MspDeInit (
            TIM_HandleTypeDef * tim_baseHandle )
```

TIM4 GPIO Configuration PB6 ------> TIM4_CH1
Uncomment the line below to disable the "TIM3_TIM4_IRQn" interrupt Be aware, disabling shared interrupt may affect other IPs
Definition at line 354 of file tim.c.

6.65.2.2 HAL_TIM_Base_MspInit()

```
void HAL_TIM_Base_MspInit (
            TIM_HandleTypeDef * tim_baseHandle )
```

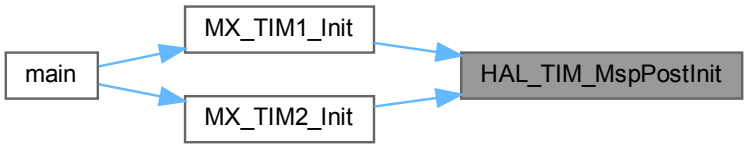
Initializes the peripherals clocks
TIM4 GPIO Configuration PB6 ------> TIM4_CH1
Definition at line 240 of file tim.c.

Here is the call graph for this function:

6.65.2.3 HAL_TIM_MspPostInit()

```
void HAL_TIM_MspPostInit (
            TIM_HandleTypeDef * timHandle )
```

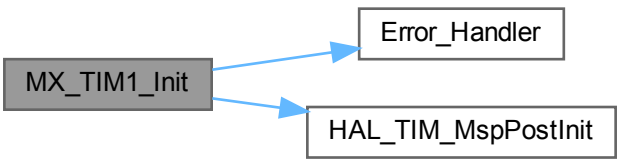
TIM1 GPIO Configuration PA8 ------> TIM1_CH1
TIM2 GPIO Configuration PA0 ------> TIM2_CH1
Definition at line 306 of file tim.c.
Here is the caller graph for this function:



6.65.2.4 MX_TIM1_Init()

```
void MX_TIM1_Init (
            void  )
```

Definition at line 32 of file tim.c.
Here is the call graph for this function:

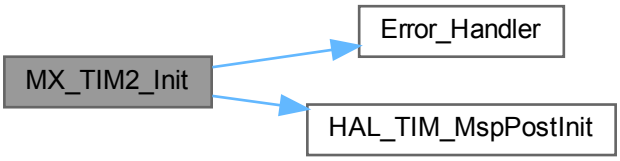


Here is the caller graph for this function:

6.65.2.5 MX_TIM2_Init()

```
void MX_TIM2_Init (
            void  )
```

Definition at line 109 of file tim.c.
Here is the call graph for this function:



Here is the caller graph for this function:

6.65.2.6 MX_TIM3_Init()

```
void MX_TIM3_Init (
            void  )
```

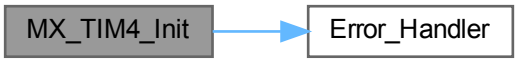
Definition at line 163 of file tim.c.

Here is the caller graph for this function:

6.65.2.7 MX_TIM4_Init()

```
void MX_TIM4_Init (
            void  )
```

Definition at line 197 of file tim.c.
Here is the call graph for this function:



Here is the caller graph for this function:

6.65.3 Variable Documentation

6.65.3.1 htim1

```
TIM_HandleTypeDef htim1
```

Definition at line 27 of file tim.c.

6.65.3.2 htim2

```
TIM_HandleTypeDef htim2
```

Definition at line 28 of file tim.c.

6.65.3.3 htim4

```
TIM_HandleTypeDef htim4
```

Definition at line 29 of file tim.c.

6.66 tim.c

Go to the documentation of this file.

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020 /* Includes ------------------------------------------------------------------*/
00021 #include "tim.h"
00022
00023 /* USER CODE BEGIN 0 */
00024
00025 /* USER CODE END 0 */
00026
00027 TIM_HandleTypeDef htim1;
00028 TIM_HandleTypeDef htim2;
00029 TIM_HandleTypeDef htim4;
00030
00031 /* TIM1 init function */
00032 void MX_TIM1_Init(void)
00033 {
00034
00035   /* USER CODE BEGIN TIM1_Init 0 */
00036
00037   /* USER CODE END TIM1_Init 0 */
00038
00039   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
00040   TIM_MasterConfigTypeDef sMasterConfig = {0};
00041   TIM_OC_InitTypeDef sConfigOC = {0};
00042   TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};
00043
00044   /* USER CODE BEGIN TIM1_Init 1 */
00045
00046   /* USER CODE END TIM1_Init 1 */
00047   htim1.Instance = TIM1;
00048   htim1.Init.Prescaler = 19;
00049   htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
00050   htim1.Init.Period = 63999;
00051   htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
00052   htim1.Init.RepetitionCounter = 0;
00053   htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
00054   if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
00055   {
00056     Error_Handler();
00057   }
00058   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
00059   if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
00060   {
00061     Error_Handler();
00062   }
00063   if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
00064   {
00065     Error_Handler();
00066   }
00067   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
00068   sMasterConfig.MasterOutputTrigger2 = TIM_TRGO2_RESET;
00069   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
00070   if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
00071   {
00072     Error_Handler();
00073   }
00074   sConfigOC.OCMode = TIM_OCMODE_PWM1;
00075   sConfigOC.Pulse = 30000;
00076   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
00077   sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
00078   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
00079   sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
00080   sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
00081   if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
00082   {
00083     Error_Handler();
00084   }
00085   sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
00086   sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
00087   sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
00088   sBreakDeadTimeConfig.DeadTime = 0;
00089   sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
00090   sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
00091   sBreakDeadTimeConfig.BreakFilter = 0;
00092   sBreakDeadTimeConfig.BreakAFMode = TIM_BREAK_AFMODE_INPUT;
00093   sBreakDeadTimeConfig.Break2State = TIM_BREAK2_DISABLE;
00094   sBreakDeadTimeConfig.Break2Polarity = TIM_BREAK2POLARITY_HIGH;
00095   sBreakDeadTimeConfig.Break2Filter = 0;
00096   sBreakDeadTimeConfig.Break2AFMode = TIM_BREAK_AFMODE_INPUT;
00097   sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
00098   if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
00099   {
00100     Error_Handler();
```

```
00101   }
00102   /* USER CODE BEGIN TIM1_Init 2 */
00103
00104   /* USER CODE END TIM1_Init 2 */
00105   HAL_TIM_MspPostInit(&htim1);
00106
00107 }
00108 /* TIM2 init function */
00109 void MX_TIM2_Init(void)
00110 {
00111
00112   /* USER CODE BEGIN TIM2_Init 0 */
00113
00114   /* USER CODE END TIM2_Init 0 */
00115
00116   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
00117   TIM_MasterConfigTypeDef sMasterConfig = {0};
00118   TIM_OC_InitTypeDef sConfigOC = {0};
00119
00120   /* USER CODE BEGIN TIM2_Init 1 */
00121
00122   /* USER CODE END TIM2_Init 1 */
00123   htim2.Instance = TIM2;
00124   htim2.Init.Prescaler = 19;
00125   htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
00126   htim2.Init.Period = 63999;
00127   htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
00128   htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
00129   if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
00130   {
00131     Error_Handler();
00132   }
00133   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
00134   if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
00135   {
00136     Error_Handler();
00137   }
00138   if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
00139   {
00140     Error_Handler();
00141   }
00142   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
00143   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
00144   if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
00145   {
00146     Error_Handler();
00147   }
00148   sConfigOC.OCMode = TIM_OCMODE_PWM1;
00149   sConfigOC.Pulse = 0;
00150   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
00151   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
00152   if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
00153   {
00154     Error_Handler();
00155   }
00156   /* USER CODE BEGIN TIM2_Init 2 */
00157
00158   /* USER CODE END TIM2_Init 2 */
00159   HAL_TIM_MspPostInit(&htim2);
00160
00161 }
00162 /* TIM3 init function */
00163 void MX_TIM3_Init(void)
00164 {
00165
00166   /* USER CODE BEGIN TIM3_Init 0 */
00167
00168   /* USER CODE END TIM3_Init 0 */
00169
00170   LL_TIM_InitTypeDef TIM_InitStruct = {0};
00171
00172   /* Peripheral clock enable */
00173   LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_TIM3);
00174
00175   /* TIM3 interrupt Init */
00176   NVIC_SetPriority(TIM3_TIM4_IRQn, 0);
00177   NVIC_EnableIRQ(TIM3_TIM4_IRQn);
00178
00179   /* USER CODE BEGIN TIM3_Init 1 */
00180
00181   /* USER CODE END TIM3_Init 1 */
00182   TIM_InitStruct.Prescaler = 63;
00183   TIM_InitStruct.CounterMode = LL_TIM_COUNTERMODE_UP;
00184   TIM_InitStruct.Autoreload = 65535;
00185   TIM_InitStruct.ClockDivision = LL_TIM_CLOCKDIVISION_DIV1;
00186   LL_TIM_Init(TIM3, &TIM_InitStruct);
00187   LL_TIM_DisableARRPreload(TIM3);
```

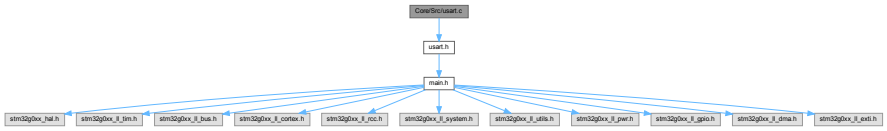
```
00188   LL_TIM_SetClockSource(TIM3, LL_TIM_CLOCKSOURCE_INTERNAL);
00189   LL_TIM_SetTriggerOutput(TIM3, LL_TIM_TRGO_RESET);
00190   LL_TIM_DisableMasterSlaveMode(TIM3);
00191   /* USER CODE BEGIN TIM3_Init 2 */
00192
00193   /* USER CODE END TIM3_Init 2 */
00194
00195 }
00196 /* TIM4 init function */
00197 void MX_TIM4_Init(void)
00198 {
00199
00200   /* USER CODE BEGIN TIM4_Init 0 */
00201
00202   /* USER CODE END TIM4_Init 0 */
00203
00204   TIM_SlaveConfigTypeDef sSlaveConfig = {0};
00205   TIM_MasterConfigTypeDef sMasterConfig = {0};
00206
00207   /* USER CODE BEGIN TIM4_Init 1 */
00208
00209   /* USER CODE END TIM4_Init 1 */
00210   htim4.Instance = TIM4;
00211   htim4.Init.Prescaler = 0;
00212   htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
00213   htim4.Init.Period = 65535;
00214   htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
00215   htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
00216   if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
00217   {
00218     Error_Handler();
00219   }
00220   sSlaveConfig.SlaveMode = TIM_SLAVEMODE_EXTERNAL1;
00221   sSlaveConfig.InputTrigger = TIM_TS_TI1FP1;
00222   sSlaveConfig.TriggerPolarity = TIM_TRIGGERPOLARITY_RISING;
00223   sSlaveConfig.TriggerFilter = 0;
00224   if (HAL_TIM_SlaveConfigSynchro(&htim4, &sSlaveConfig) != HAL_OK)
00225   {
00226     Error_Handler();
00227   }
00228   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
00229   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
00230   if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
00231   {
00232     Error_Handler();
00233   }
00234   /* USER CODE BEGIN TIM4_Init 2 */
00235
00236   /* USER CODE END TIM4_Init 2 */
00237
00238 }
00239
00240 void HAL_TIM_Base_MspInit(TIM_HandleTypeDef* tim_baseHandle)
00241 {
00242
00243   GPIO_InitTypeDef GPIO_InitStruct = {0};
00244   RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
00245   if(tim_baseHandle->Instance==TIM1)
00246   {
00247   /* USER CODE BEGIN TIM1_MspInit 0 */
00248
00249   /* USER CODE END TIM1_MspInit 0 */
00250
00253     PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_TIM1;
00254     PeriphClkInit.Tim1ClockSelection = RCC_TIM1CLKSOURCE_PCLK1;
00255     if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
00256     {
00257       Error_Handler();
00258     }
00259
00260     /* TIM1 clock enable */
00261     __HAL_RCC_TIM1_CLK_ENABLE();
00262   /* USER CODE BEGIN TIM1_MspInit 1 */
00263
00264   /* USER CODE END TIM1_MspInit 1 */
00265   }
00266   else if(tim_baseHandle->Instance==TIM2)
00267   {
00268   /* USER CODE BEGIN TIM2_MspInit 0 */
00269
00270   /* USER CODE END TIM2_MspInit 0 */
00271
00272     /* TIM2 clock enable */
00273     __HAL_RCC_TIM2_CLK_ENABLE();
00274   /* USER CODE BEGIN TIM2_MspInit 1 */
00275
00276   /* USER CODE END TIM2_MspInit 1 */
```

```
00277   }
00278   else if(tim_baseHandle->Instance==TIM4)
00279   {
00280   /* USER CODE BEGIN TIM4_MspInit 0 */
00281
00282   /* USER CODE END TIM4_MspInit 0 */
00283
00284     /* TIM4 clock enable */
00285     __HAL_RCC_TIM4_CLK_ENABLE();
00286
00287     __HAL_RCC_GPIOB_CLK_ENABLE();
00291     GPIO_InitStruct.Pin = GPIO_PIN_6;
00292     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
00293     GPIO_InitStruct.Pull = GPIO_PULLUP;
00294     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00295     GPIO_InitStruct.Alternate = GPIO_AF9_TIM4;
00296     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
00297
00298     /* TIM4 interrupt Init */
00299     HAL_NVIC_SetPriority(TIM3_TIM4_IRQn, 0, 0);
00300     HAL_NVIC_EnableIRQ(TIM3_TIM4_IRQn);
00301   /* USER CODE BEGIN TIM4_MspInit 1 */
00302
00303   /* USER CODE END TIM4_MspInit 1 */
00304   }
00305 }
00306 void HAL_TIM_MspPostInit(TIM_HandleTypeDef* timHandle)
00307 {
00308
00309   GPIO_InitTypeDef GPIO_InitStruct = {0};
00310   if(timHandle->Instance==TIM1)
00311   {
00312   /* USER CODE BEGIN TIM1_MspPostInit 0 */
00313
00314   /* USER CODE END TIM1_MspPostInit 0 */
00315     __HAL_RCC_GPIOA_CLK_ENABLE();
00319     GPIO_InitStruct.Pin = PWM_servo_Pin;
00320     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
00321     GPIO_InitStruct.Pull = GPIO_NOPULL;
00322     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00323     GPIO_InitStruct.Alternate = GPIO_AF2_TIM1;
00324     HAL_GPIO_Init(PWM_servo_GPIO_Port, &GPIO_InitStruct);
00325
00326   /* USER CODE BEGIN TIM1_MspPostInit 1 */
00327
00328   /* USER CODE END TIM1_MspPostInit 1 */
00329   }
00330   else if(timHandle->Instance==TIM2)
00331   {
00332   /* USER CODE BEGIN TIM2_MspPostInit 0 */
00333
00334   /* USER CODE END TIM2_MspPostInit 0 */
00335
00336     __HAL_RCC_GPIOA_CLK_ENABLE();
00340     GPIO_InitStruct.Pin = PWM_motor_Pin;
00341     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
00342     GPIO_InitStruct.Pull = GPIO_NOPULL;
00343     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00344     GPIO_InitStruct.Alternate = GPIO_AF2_TIM2;
00345     HAL_GPIO_Init(PWM_motor_GPIO_Port, &GPIO_InitStruct);
00346
00347   /* USER CODE BEGIN TIM2_MspPostInit 1 */
00348
00349   /* USER CODE END TIM2_MspPostInit 1 */
00350   }
00351
00352 }
00353
00354 void HAL_TIM_Base_MspDeInit(TIM_HandleTypeDef* tim_baseHandle)
00355 {
00356
00357   if(tim_baseHandle->Instance==TIM1)
00358   {
00359   /* USER CODE BEGIN TIM1_MspDeInit 0 */
00360
00361   /* USER CODE END TIM1_MspDeInit 0 */
00362     /* Peripheral clock disable */
00363     __HAL_RCC_TIM1_CLK_DISABLE();
00364   /* USER CODE BEGIN TIM1_MspDeInit 1 */
00365
00366   /* USER CODE END TIM1_MspDeInit 1 */
00367   }
00368   else if(tim_baseHandle->Instance==TIM2)
00369   {
00370   /* USER CODE BEGIN TIM2_MspDeInit 0 */
00371
00372   /* USER CODE END TIM2_MspDeInit 0 */
```

```
00373     /* Peripheral clock disable */
00374     __HAL_RCC_TIM2_CLK_DISABLE();
00375   /* USER CODE BEGIN TIM2_MspDeInit 1 */
00376
00377   /* USER CODE END TIM2_MspDeInit 1 */
00378   }
00379   else if(tim_baseHandle->Instance==TIM4)
00380   {
00381   /* USER CODE BEGIN TIM4_MspDeInit 0 */
00382
00383   /* USER CODE END TIM4_MspDeInit 0 */
00384     /* Peripheral clock disable */
00385     __HAL_RCC_TIM4_CLK_DISABLE();
00386
00390     HAL_GPIO_DeInit(GPIOB, GPIO_PIN_6);
00391
00392     /* TIM4 interrupt Deinit */
00393   /* USER CODE BEGIN TIM4:TIM3_TIM4_IRQn disable */
00398     /* HAL_NVIC_DisableIRQ(TIM3_TIM4_IRQn); */
00399   /* USER CODE END TIM4:TIM3_TIM4_IRQn disable */
00400
00401   /* USER CODE BEGIN TIM4_MspDeInit 1 */
00402
00403   /* USER CODE END TIM4_MspDeInit 1 */
00404   }
00405 }
00406
00407 /* USER CODE BEGIN 1 */
00408
00409 /* USER CODE END 1 */
```

6.67 Core/Src/usart.c File Reference

This file provides code for the configuration of the USART instances.
`#include "usart.h"`
Include dependency graph for usart.c:



Functions

- void MX_USART2_UART_Init (void)
- void HAL_UART_MspInit (UART_HandleTypeDef ∗uartHandle)
- void HAL_UART_MspDeInit (UART_HandleTypeDef ∗uartHandle)

Variables

- UART_HandleTypeDef huart2
- DMA_HandleTypeDef hdma_usart2_rx
- DMA_HandleTypeDef hdma_usart2_tx

6.67.1 Detailed Description

This file provides code for the configuration of the USART instances.

Attention

Definition in file usart.c.

6.67.2 Function Documentation

6.67.2.1 HAL_UART_MspDeInit()

```
void HAL_UART_MspDeInit (
            UART_HandleTypeDef * uartHandle )
```

USART2 GPIO Configuration PA2 ------> USART2_TX PA3 ------> USART2_RX
Definition at line 155 of file usart.c.

6.67.2.2 HAL_UART_MspInit()

```
void HAL_UART_MspInit (
            UART_HandleTypeDef * uartHandle )
```

Initializes the peripherals clocks
USART2 GPIO Configuration PA2 ------> USART2_TX PA3 ------> USART2_RX
Definition at line 76 of file usart.c.
Here is the call graph for this function:

6.67.2.3 MX_USART2_UART_Init()

```
void MX_USART2_UART_Init (
            void  )
```

Definition at line 33 of file usart.c.
Here is the call graph for this function:

Here is the caller graph for this function:

6.67.3 Variable Documentation

6.67.3.1 hdma_usart2_rx

`DMA_HandleTypeDef hdma_usart2_rx`
Definition at line 28 of file usart.c.

6.67.3.2 hdma_usart2_tx

`DMA_HandleTypeDef hdma_usart2_tx`
Definition at line 29 of file usart.c.

6.67.3.3 huart2

`UART_HandleTypeDef huart2`
Definition at line 27 of file usart.c.

6.68 usart.c

Go to the documentation of this file.

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020 /* Includes ------------------------------------------------------------------*/
00021 #include "usart.h"
00022
00023 /* USER CODE BEGIN 0 */
00024
00025 /* USER CODE END 0 */
00026
00027 UART_HandleTypeDef huart2;
00028 DMA_HandleTypeDef hdma_usart2_rx;
00029 DMA_HandleTypeDef hdma_usart2_tx;
00030
00031 /* USART2 init function */
00032
00033 void MX_USART2_UART_Init(void)
00034 {
00035
00036   /* USER CODE BEGIN USART2_Init 0 */
00037
00038   /* USER CODE END USART2_Init 0 */
00039
00040   /* USER CODE BEGIN USART2_Init 1 */
00041
00042   /* USER CODE END USART2_Init 1 */
00043   huart2.Instance = USART2;
00044   huart2.Init.BaudRate = 115200;
00045   huart2.Init.WordLength = UART_WORDLENGTH_8B;
00046   huart2.Init.StopBits = UART_STOPBITS_1;
00047   huart2.Init.Parity = UART_PARITY_NONE;
00048   huart2.Init.Mode = UART_MODE_TX_RX;
00049   huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
00050   huart2.Init.OverSampling = UART_OVERSAMPLING_16;
00051   huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
00052   huart2.Init.ClockPrescaler = UART_PRESCALER_DIV1;
00053   huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
00054   if (HAL_UART_Init(&huart2) != HAL_OK)
00055   {
00056     Error_Handler();
00057   }
00058   if (HAL_UARTEx_SetTxFifoThreshold(&huart2, UART_TXFIFO_THRESHOLD_1_8) != HAL_OK)
00059   {
00060     Error_Handler();
00061   }
00062   if (HAL_UARTEx_SetRxFifoThreshold(&huart2, UART_RXFIFO_THRESHOLD_1_8) != HAL_OK)
00063   {
00064     Error_Handler();
00065   }
00066   if (HAL_UARTEx_DisableFifoMode(&huart2) != HAL_OK)
00067   {
00068     Error_Handler();
00069   }
00070   /* USER CODE BEGIN USART2_Init 2 */
00071
00072   /* USER CODE END USART2_Init 2 */
00073
00074 }
00075
```

```
00076 void HAL_UART_MspInit(UART_HandleTypeDef* uartHandle)
00077 {
00078
00079   GPIO_InitTypeDef GPIO_InitStruct = {0};
00080   RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
00081   if(uartHandle->Instance==USART2)
00082   {
00083   /* USER CODE BEGIN USART2_MspInit 0 */
00084
00085   /* USER CODE END USART2_MspInit 0 */
00086
00089     PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART2;
00090     PeriphClkInit.Usart2ClockSelection = RCC_USART2CLKSOURCE_PCLK1;
00091     if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
00092     {
00093       Error_Handler();
00094     }
00095
00096     /* USART2 clock enable */
00097     __HAL_RCC_USART2_CLK_ENABLE();
00098
00099     __HAL_RCC_GPIOA_CLK_ENABLE();
00104     GPIO_InitStruct.Pin = USART2_TX_Pin|USART2_RX_Pin;
00105     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
00106     GPIO_InitStruct.Pull = GPIO_NOPULL;
00107     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00108     GPIO_InitStruct.Alternate = GPIO_AF1_USART2;
00109     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
00110
00111     /* USART2 DMA Init */
00112     /* USART2_RX Init */
00113     hdma_usart2_rx.Instance = DMA1_Channel1;
00114     hdma_usart2_rx.Init.Request = DMA_REQUEST_USART2_RX;
00115     hdma_usart2_rx.Init.Direction = DMA_PERIPH_TO_MEMORY;
00116     hdma_usart2_rx.Init.PeriphInc = DMA_PINC_DISABLE;
00117     hdma_usart2_rx.Init.MemInc = DMA_MINC_ENABLE;
00118     hdma_usart2_rx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
00119     hdma_usart2_rx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
00120     hdma_usart2_rx.Init.Mode = DMA_CIRCULAR;
00121     hdma_usart2_rx.Init.Priority = DMA_PRIORITY_HIGH;
00122     if (HAL_DMA_Init(&hdma_usart2_rx) != HAL_OK)
00123     {
00124       Error_Handler();
00125     }
00126
00127     __HAL_LINKDMA(uartHandle,hdmarx,hdma_usart2_rx);
00128
00129     /* USART2_TX Init */
00130     hdma_usart2_tx.Instance = DMA1_Channel2;
00131     hdma_usart2_tx.Init.Request = DMA_REQUEST_USART2_TX;
00132     hdma_usart2_tx.Init.Direction = DMA_MEMORY_TO_PERIPH;
00133     hdma_usart2_tx.Init.PeriphInc = DMA_PINC_DISABLE;
00134     hdma_usart2_tx.Init.MemInc = DMA_MINC_ENABLE;
00135     hdma_usart2_tx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
00136     hdma_usart2_tx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
00137     hdma_usart2_tx.Init.Mode = DMA_NORMAL;
00138     hdma_usart2_tx.Init.Priority = DMA_PRIORITY_MEDIUM;
00139     if (HAL_DMA_Init(&hdma_usart2_tx) != HAL_OK)
00140     {
00141       Error_Handler();
00142     }
00143
00144     __HAL_LINKDMA(uartHandle,hdmatx,hdma_usart2_tx);
00145
00146     /* USART2 interrupt Init */
00147     HAL_NVIC_SetPriority(USART2_LPUART2_IRQn, 0, 0);
00148     HAL_NVIC_EnableIRQ(USART2_LPUART2_IRQn);
00149   /* USER CODE BEGIN USART2_MspInit 1 */
00150
00151   /* USER CODE END USART2_MspInit 1 */
00152   }
00153 }
00154
00155 void HAL_UART_MspDeInit(UART_HandleTypeDef* uartHandle)
00156 {
00157
00158   if(uartHandle->Instance==USART2)
00159   {
00160   /* USER CODE BEGIN USART2_MspDeInit 0 */
00161
00162   /* USER CODE END USART2_MspDeInit 0 */
00163     /* Peripheral clock disable */
00164     __HAL_RCC_USART2_CLK_DISABLE();
00165
00170     HAL_GPIO_DeInit(GPIOA, USART2_TX_Pin|USART2_RX_Pin);
00171
00172     /* USART2 DMA DeInit */
```

```
00173     HAL_DMA_DeInit(uartHandle->hdmarx);
00174     HAL_DMA_DeInit(uartHandle->hdmatx);
00175
00176     /* USART2 interrupt Deinit */
00177     HAL_NVIC_DisableIRQ(USART2_LPUART2_IRQn);
00178   /* USER CODE BEGIN USART2_MspDeInit 1 */
00179
00180   /* USER CODE END USART2_MspDeInit 1 */
00181   }
00182 }
00183
00184 /* USER CODE BEGIN 1 */
00185
00186 /* USER CODE END 1 */
```

Index