ME 507 Term Project: Myoelectric Hand Control System

Generated by Doxygen 1.10.0

Chapter 1

Prosthetic Hand Control through Myoelectric and Pressure Sensors

Authors

Julia Fay & Jack Foxcroft

1.1 Project Description

Our project is centered around the control of an electromechanically actuated prosthetic hand manufactured by Australian orthopedic and prosthetic design company Ottobock donated to the EMPOWER Student Association. The goal is to create a proof-of-concept control system to actuate the prosthetic hand via myoelectric sensor input. Our design implements two myoelectric sensors, two brushed DC motors, and one rotary encoder. The hand can rotate about its central axis and open/close, controlled by two myoelectric sensors. The design also includes an emergency stop via radio transmitter for safety purposes. The selected MCU was the STM32L476RGT6 due to our familiarity with this chip from previous quarters and its number of ADC's and timer channels. The final product of this project did not meet all the goals outlined for the project due to issues with the PCB and hardware. During testing, our custom PCB's power rail broke down, so a NUCLEO development board was used. Additionally, the myoelectric sensors stopped functioning correctly during testing and no longer produced a consistently readable output. Thus, instead of using muscle actuation to move the motors, the gain on the sensors were manually increased and decreased by hand to force the expected output to be interpreted by our program. Lastly, the motor on the prosthetic hand initially selected to use for this project was nonfunctional, so an alternative hand without an encoder was used, affecting the controllability of the hand movement. An image of the final design is shown below.

The pressure sensor we selected is a simple and cost-effective part from amazon. It is a thin film sensor that will easily stick onto the thumb of our gripping device. The sensor only has two wire connections. One wire is the input, which will be supplied and set to 1V with the use of a resistor. The second wire is the output, which will be an input to one of the ADC's on the microcontroller.

The DC motor we selected is also a simple and cost-effective part. It is a 12V DC motor with an encoder with a top speed of 100 RPM. A convenient feature of this motor is the shaft output is perpendicular to the body of the motor allowing for easy attachment to our design. The motor requires a 3.3-5V power supply.

Overall, our design meets all the rules and requirements of the project. We have two actuators, and two unique sensors including the spin motor encoder and the myo electric sensors. All our hardware was purchased, or 3D printed.Our design is safe for users and bystanders as it will be a mainly stationary device that rotates and moves within a confined area as it will be attached to a steady base. We used the remote controller as a emergency shut-off switch for our design.

Chapter 2

Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Topic Documentation

5.1 CMSIS

Topics

- Stm32l4xx_system

5.1.1 Detailed Description

5.1.2 Stm32l4xx_system

Topics

- STM32L4xx_System_Private_Includes
- STM32L4xx_System_Private_TypesDefinitions
- STM32L4xx_System_Private_Defines
- STM32L4xx_System_Private_Macros
- STM32L4xx_System_Private_Variables
- STM32L4xx_System_Private_FunctionPrototypes
- STM32L4xx_System_Private_Functions

5.1.2.1 Detailed Description

5.1.2.2 STM32L4xx_System_Private_Includes

5.1.2.3 STM32L4xx_System_Private_TypesDefinitions

5.1.2.4 STM32L4xx_System_Private_Defines

Macros

- #define HSE_VALUE 8000000U
- #define MSI_VALUE 4000000U
- #define HSI_VALUE 16000000U

5.1.2.4.1 Detailed Description

5.1.2.4.2 Macro Definition Documentation

5.1.2.4.2.1 HSE_VALUE

```
#define HSE_VALUE 8000000U
```

Value of the External oscillator in Hz

5.1.2.4.2.2 HSI_VALUE

```
#define HSI_VALUE 16000000U
```

Value of the Internal oscillator in Hz

5.1.2.4.2.3 MSI_VALUE

```
#define MSI_VALUE 4000000U
```

Value of the Internal oscillator in Hz

5.1.2.5 STM32L4xx_System_Private_Macros

5.1.2.6 STM32L4xx_System_Private_Variables

Variables

- uint32_t SystemCoreClock = 4000000U
- const uint8_t AHBPrescTable [16] = {0U, 0U, 0U, 0U, 0U, 0U, 0U, 0U, 1U, 2U, 3U, 4U, 6U, 7U, 8U, 9U}
- const uint8_t APBPrescTable [8] = {0U, 0U, 0U, 0U, 1U, 2U, 3U, 4U}
- const uint32_t MSIRangeTable [12]

5.1.2.6.1 Detailed Description

5.1.2.6.2 Variable Documentation

5.1.2.6.2.1 AHBPrescTable

```
const uint8_t AHBPrescTable[16] = {0U, 0U, 0U, 0U, 0U, 0U, 0U, 0U, 1U, 2U, 3U, 4U, 6U, 7U, 8U,
9U}
```

5.1.2.6.2.2 APBPrescTable

```
const uint8_t APBPrescTable[8] = {0U, 0U, 0U, 0U, 1U, 2U, 3U, 4U}
```

5.1.2.6.2.3 MSIRangeTable

```
const uint32_t MSIRangeTable[12]
```

Initial value:

```
= {100000U,   200000U,   400000U,   800000U,  1000000U,  2000000U,
                                    4000000U, 8000000U, 16000000U, 24000000U, 32000000U, 48000000U}
```

5.1.2.6.2.4 SystemCoreClock

```
uint32_t SystemCoreClock = 4000000U
```

5.1.2.7 STM32L4xx_System_Private_FunctionPrototypes

5.1.2.8 STM32L4xx_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.

5.1.2.8.1 Detailed Description

5.1.2.8.2 Function Documentation

5.1.2.8.2.1 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 MSI, SystemCoreClock will contain the MSI_VALUE(∗)
- If SYSCLK source is HSI, SystemCoreClock will contain the HSI_VALUE(∗∗)
- If SYSCLK source is HSE, SystemCoreClock will contain the HSE_VALUE(∗∗∗)
- If SYSCLK source is PLL, SystemCoreClock will contain the HSE_VALUE(∗∗∗) or HSI_VALUE(∗) or MSI_VALUE(∗) multiplied/divided by the PLL factors.

(∗) MSI_VALUE is a constant defined in stm32l4xx_hal.h file (default value 4 MHz) but the real value may vary depending on the variations in voltage and temperature.

(∗∗) HSI_VALUE is a constant defined in stm32l4xx_hal.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 stm32l4xx_hal.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.

Return values

None	

5.1.2.8.2.2 SystemInit()

```
void SystemInit (
             void  )
```

Setup the microcontroller system.

Return values

None	

Chapter 6

Class Documentation

6.1 calibrate_t Struct Reference

Represents a calibration object that has a myoelectric sensor and an array length to perform the find_average function on.

`#include <calibrate.h>`

Public Attributes

- uint32_t data_pts
- myo_t * p_myo

6.1.1 Detailed Description

Represents a calibration object that has a myoelectric sensor and an array length to perform the find_average function on.

6.1.2 Member Data Documentation

6.1.2.1 data_pts

```
uint32_t calibrate_t::data_pts
```

The number of data points to be read.

6.1.2.2 p_myo

```
myo_t* calibrate_t::p_myo
```

The the myo electric sensor to read from.

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

- Core/Inc/calibrate.h

6.2 controller_t Struct Reference

Represents a controller object that controls a motor based on the encoder reading.

`#include <controller.h>`

Public Attributes

- motor_t * p_mot
- encoder_t * p_enc
- int32_t gain
- int32_t setpoint

6.2.1 Detailed Description

Represents a controller object that controls a motor based on the encoder reading.

6.2.2 Member Data Documentation

6.2.2.1 gain

`int32_t controller_t::gain`

The desired control loop gain.

6.2.2.2 p_enc

`encoder_t* controller_t::p_enc`

The encoder to be read from.

6.2.2.3 p_mot

`motor_t* controller_t::p_mot`

The motor object to be controlled.

6.2.2.4 setpoint

`int32_t controller_t::setpoint`

The desired set point for the motor.

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

- Core/Inc/controller.h

6.3 encoder_t Struct Reference

Represents a encoder object that has a timer with two channels, and an encoder count.

`#include <encoder_reader.h>`

Public Attributes

- uint32_t channel1
- uint32_t channel2
- TIM_HandleTypeDef * hal_tim
- int32_t mot_pos
- int32_t curr_count
- int32_t prev_count
- int32_t delta

6.3.1 Detailed Description

Represents a encoder object that has a timer with two channels, and an encoder count.

6.3.2 Member Data Documentation

6.3.2.1 channel1

`uint32_t encoder_t::channel1`

The timer channel for the first encoder output.

6.3.2.2 channel2

`uint32_t encoder_t::channel2`

The timer channel for the second encoder output.

6.3.2.3 curr_count

`int32_t encoder_t::curr_count`

The current encoder count.

6.3.2.4 delta

`int32_t encoder_t::delta`

The difference between the previous and current encoder count.

6.3.2.5 hal_tim

```
TIM_HandleTypeDef* encoder_t::hal_tim
```

The timer object both channels are from.

6.3.2.6 mot_pos

```
int32_t encoder_t::mot_pos
```

The motor position.

6.3.2.7 prev_count

```
int32_t encoder_t::prev_count
```

The previous encoder count.

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

- Core/Inc/encoder_reader.h

6.4 motor_t Struct Reference

Represents a motor objects with two PWM channels in a timer and a duty cycle.

```
#include <motor_driver.h>
```

Public Attributes

- int32_t pwm_val
- uint32_t channel1
- uint32_t channel2
- TIM_HandleTypeDef * hal_tim

6.4.1 Detailed Description

Represents a motor objects with two PWM channels in a timer and a duty cycle.

6.4.2 Member Data Documentation

6.4.2.1 channel1

```
uint32_t motor_t::channel1
```

Timer channel 1 used to generate a PWM signal that is sent to the motor driver.

6.4.2.2 channel2

```
uint32_t motor_t::channel2
```

Timer channel 2 used to generate a PWM signal that is sent to the motor driver.

6.4.2.3 hal_tim

```
TIM_HandleTypeDef* motor_t::hal_tim
```

The handle to the HAL timer object used for PWM generation.

6.4.2.4 pwm_val

```
int32_t motor_t::pwm_val
```

The CCR pwm value used to set the duty cycle of the motor.

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

- Core/Inc/motor_driver.h

6.5 myo_t Struct Reference

Represents a myoelectric sensor object that has an ADC object and a current sensor value.

```
#include <myo.h>
```

Public Attributes

- ADC_HandleTypeDef * hal_adc
- int16_t current_value

6.5.1 Detailed Description

Represents a myoelectric sensor object that has an ADC object and a current sensor value.

6.5.2 Member Data Documentation

6.5.2.1 current_value

```
int16_t myo_t::current_value
```

The current ADC value.

6.5.2.2 hal_adc

```
ADC_HandleTypeDef* myo_t::hal_adc
```

The ADC object.

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

- Core/Inc/myo.h

Chapter 7

File Documentation

7.1 Core/Inc/calibrate.h File Reference

Defines a function to calibrate the myoelectric sensor values by calculating an average from a specified number of data points.

```
#include <stdio.h>
#include "stm32l4xx_hal.h"
#include <stdint.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include "myo.h"
```

Classes

- struct calibrate_t

  Represents a calibration object that has a myoelectric sensor and an array length to perform the find_average function on.

Functions

- uint32_t find_average (calibrate_t *p_cali)

  A function to find the average myoelectric sensor value for a specified number of data points.

7.1.1 Detailed Description

Defines a function to calibrate the myoelectric sensor values by calculating an average from a specified number of data points.

7.1.2 Function Documentation

7.1.2.1 find_average()

```
uint32_t find_average (
            calibrate_t * p_cali )
```

A function to find the average myoelectric sensor value for a specified number of data points.

Parameters

p_cali	The calibration object to perform the function on.

7.2 calibrate.h

Go to the documentation of this file.

```
00001
00007 #ifndef SRC_CALIBRATE_H_
00008 #define SRC_CALIBRATE_H_
00009
00010 #include <stdio.h>
00011 #include "stm32l4xx_hal.h"
00012 #include <stdint.h>
00013 #include <string.h>
00014 #include <ctype.h>
00015 #include <stdlib.h>
00016 #include "myo.h"
00017
00022 struct{
00023
00024
00025         uint32_t data_pts;
00026         myo_t* p_myo;
00029 } typedef calibrate_t;
00030
00036 uint32_t find_average(calibrate_t*p_cali);
00037
00038
00039
00040 #endif /* SRC_CALIBRATE_H_ */
```

7.3 Core/Inc/controller.h File Reference

Defines the controller struct and its methods.

```
#include "motor_driver.h"
#include "encoder_reader.h"
```

Classes

- struct controller_t

  Represents a controller object that controls a motor based on the encoder reading.

Functions

- void controller_init (controller_t ∗p_cont)

  A function to initialize all of the timer channels.
- void controller_deinit (controller_t ∗p_cont)

  A function to de-initialize all of the timer channels.
- int32_t move (controller_t ∗p_cont, int32_t gain)

  A function to move the controlled motor to the desired position. The function calculates the PWM signal that will be sent to the motor by taking into account the setpoint of the motor, and the current position of the motor. These values are subtracted to find the error which is then multiplied by Kp. If this value exceeds -3,999 or 3,999, the value is saturated to either -3,999 or 3,999. The function also sets a minimum threshold for when a PWM signal is generated. Anything below 10 times the gain value is considered to be a PWM signal of zero. Then the set_duty function imported from the motor_driver class is run. The calculated PWM signal is returned at the end of the function.
- void set_setpoint (controller_t ∗p_cont, int32_t new_setpoint)

  A function to set the new controller set point.
- void set_K (controller_t ∗p_cont, int32_t new_gain)

  A function to update the control loop gain.

7.3.1 Detailed Description

Defines the controller struct and its methods.

7.3.2 Function Documentation

7.3.2.1 controller_deinit()

```
void controller_deinit (
            controller_t * p_cont )
```

A function to de-initialize all of the timer channels.

Parameters

p_cont	The controller object to perform the function on.

7.3.2.2 controller_init()

```
void controller_init (
            controller_t * p_cont )
```

A function to initialize all of the timer channels.

Parameters

p_cont	The controller object to perform the function on.

A function to initialize all of the timer channels.

Parameters

p_cont	The controller object to perform the function on.

A function to initialize all of the timer channels.

Parameters

p_cont	The controller object to perform the function on.

7.3.2.3 move()

```
int32_t move (
            controller_t * p_cont,
            int32_t gain )
```

A function to move the controlled motor to the desired position. The function calculates the PWM signal that will be sent to the motor by taking into account the setpoint of the motor, and the current position of the motor. These values are subtracted to find the error which is then multiplied by Kp. If this value exceeds -3,999 or 3,999, the value is saturated to either -3,999 or 3,999. The function also sets a minimum threshold for when a PWM signal is generated. Anything below 10 times the gain value is considered to be a PWM signal of zero. Then the set_duty function imported from the motor_driver class is run. The calculated PWM signal is returned at the end of the function.

Parameters

p_cont	The controller object to perform the function on.

Returns

pwm_sig The pwm value calculated from the closed loop proportional control.

A function to move the controlled motor to the desired position. The function calculates the PWM signal that will be sent to the motor by taking into account the setpoint of the motor, and the current position of the motor. These values are subtracted to find the error which is then multiplied by Kp. If this value exceeds -3,999 or 3,999, the value is saturated to either -3,999 or 3,999. The function also sets a minimum threshold for when a PWM signal is generated. Anything below 10 times the gain value is considered to be a PWM signal of zero. Then the set_duty function imported from the motor_driver class is run. The calculated PWM signal is returned at the end of the function.

Parameters

p_cont	The controller object to perform the function on.

7.3.2.4 set_K()

```
void set_K (
            controller_t * p_cont,
            int32_t new_gain )
```

A function to update the control loop gain.

Parameters

p_cont	The controller object to perform the function on.
new_gain	The new set point for the controller object.

7.3.2.5 set_setpoint()

```
void set_setpoint (
            controller_t * p_cont,
            int32_t new_setpoint )
```

A function to set the new controller set point.

Parameters

p_cont	The controller object to perform the function on.
new_setpoint	The new set point for the controller object.

7.4 controller.h

Go to the documentation of this file.

```
00001
00006 #ifndef INC_CONTROLLER_H_
00007 #define INC_CONTROLLER_H_
00008 #include "motor_driver.h"
00009 #include "encoder_reader.h"
00010
00014 struct {
00015
00016     motor_t* p_mot;
00017     encoder_t* p_enc;
00019     int32_t gain;
00020     int32_t setpoint;
00023 } typedef controller_t;
00024
00030 void controller_init(controller_t* p_cont);
00031
00037 void controller_deinit(controller_t* p_cont);
00038
00054 int32_t move(controller_t* p_cont, int32_t gain);
00055
00062 void set_setpoint(controller_t* p_cont, int32_t new_setpoint);
00063
00070 void set_K(controller_t* p_cont, int32_t new_gain );
00071
00072
00073 #endif /* INC_CONTROLLER_H_ */
```

7.5 Core/Inc/encoder_reader.h File Reference

Defines the encoder reader struct and its methods. This file is used to read the position of a motor using the attached motor encoder. This is accomplished by creating an Encoder class with several functions defined to aid in the task of reading the encoder including init, read, zero and loop.

```
#include <stdio.h>
#include <stdint.h>
#include "stm32l4xx_hal.h"
```

Classes

- struct encoder_t

  Represents a encoder object that has a timer with two channels, and an encoder count.

Functions

- void init_channels (encoder_t *p_enc)

  A function to initialize the channels for reading the encoder signals.
- void deinit_channels (encoder_t *p_enc)

  A function to stop the channels from reading the encoder signals.
- void zero (encoder_t *p_enc)

  A function to zero the encoder count.
- int32_t get_pos (encoder_t *p_enc)

  A function to read and return the encoder count.

7.5.1 Detailed Description

Defines the encoder reader struct and its methods. This file is used to read the position of a motor using the attached motor encoder. This is accomplished by creating an Encoder class with several functions defined to aid in the task of reading the encoder including init, read, zero and loop.

7.5.2 Function Documentation

7.5.2.1 deinit_channels()

```
void deinit_channels (
            encoder_t * p_enc )
```

A function to stop the channels from reading the encoder signals.

Parameters

p_enc	The encoder object to perform the function on.

7.5.2.2 get_pos()

```
int32_t get_pos (
            encoder_t * p_enc )
```

A function to read and return the encoder count.

Parameters

p_enc	The encoder object to perform the function on.

Returns

count The encoder count to be returned.

7.5.2.3 init_channels()

```
void init_channels (
            encoder_t * p_enc )
```

A function to initialize the channels for reading the encoder signals.

Parameters

p_enc	The encoder object to perform the function on.

7.5.2.4 zero()

```
void zero (
            encoder_t * p_enc )
```

A function to zero the encoder count.

Parameters

p_enc	The encoder object to perform the function on.

7.6 encoder_reader.h

Go to the documentation of this file.

```
00001
00009 #ifndef INC_ENCODER_READER_H_
00010 #define INC_ENCODER_READER_H_
00011 #include <stdio.h>
00012 #include <stdint.h>
00013 #include "stm32l4xx_hal.h"
00017 struct{
00018
00019
00020         uint32_t channel1;
00021         uint32_t channel2;
00023         TIM_HandleTypeDef* hal_tim;
00025         int32_t mot_pos;
00026         int32_t curr_count;
00027         int32_t prev_count;
00028         int32_t delta;
00032 } typedef encoder_t;
00033
00039 void init_channels(encoder_t *p_enc);
00040
00046 void deinit_channels(encoder_t *p_enc);
00047
00054 void zero(encoder_t* p_enc);
00055
00064 int32_t get_pos(encoder_t *p_enc);
00065
00066 #endif /* INC_ENCODER_READER_H_ */
```

7.7 Core/Inc/main.h File Reference

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

```
#include "stm32l4xx_hal.h"
```

Macros

- #define SMYO_Pin GPIO_PIN_0
- #define SMYO_GPIO_Port GPIOC
- #define HMYO_Pin GPIO_PIN_1
- #define HMYO_GPIO_Port GPIOC
- #define SPIN_PWMA_Pin GPIO_PIN_0
- #define SPIN_PWMA_GPIO_Port GPIOA
- #define SPIN_PWMB_Pin GPIO_PIN_1

- #define SPIN_PWMB_GPIO_Port GPIOA
- #define SPIN_ENCA_Pin GPIO_PIN_6
- #define SPIN_ENCA_GPIO_Port GPIOA
- #define SPIN_ENCB_Pin GPIO_PIN_7
- #define SPIN_ENCB_GPIO_Port GPIOA
- #define HAND_PWMA_Pin GPIO_PIN_10
- #define HAND_PWMA_GPIO_Port GPIOB
- #define HAND_PWMB_Pin GPIO_PIN_11
- #define HAND_PWMB_GPIO_Port GPIOB
- #define RADIO_Pin GPIO_PIN_8
- #define RADIO_GPIO_Port GPIOA
- #define HAND_ENCA_Pin GPIO_PIN_6
- #define HAND_ENCA_GPIO_Port GPIOB
- #define HAND_ENCB_Pin GPIO_PIN_7
- #define HAND_ENCB_GPIO_Port GPIOB

Functions

- void HAL_TIM_MspPostInit (TIM_HandleTypeDef *htim)
- void Error_Handler (void)

  This function is executed in case of error occurrence.

7.7.1 Detailed Description

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

Attention

7.7.2 Macro Definition Documentation

7.7.2.1 HAND_ENCA_GPIO_Port

```
#define HAND_ENCA_GPIO_Port GPIOB
```

7.7.2.2 HAND_ENCA_Pin

```
#define HAND_ENCA_Pin GPIO_PIN_6
```

7.7.2.3 HAND_ENCB_GPIO_Port

```
#define HAND_ENCB_GPIO_Port GPIOB
```

7.7.2.4 HAND_ENCB_Pin

```
#define HAND_ENCB_Pin GPIO_PIN_7
```

7.7.2.5 HAND_PWMA_GPIO_Port

```
#define HAND_PWMA_GPIO_Port GPIOB
```

7.7.2.6 HAND_PWMA_Pin

```
#define HAND_PWMA_Pin GPIO_PIN_10
```

7.7.2.7 HAND_PWMB_GPIO_Port

```
#define HAND_PWMB_GPIO_Port GPIOB
```

7.7.2.8 HAND_PWMB_Pin

```
#define HAND_PWMB_Pin GPIO_PIN_11
```

7.7.2.9 HMYO_GPIO_Port

```
#define HMYO_GPIO_Port GPIOC
```

7.7.2.10 HMYO_Pin

```
#define HMYO_Pin GPIO_PIN_1
```

7.7.2.11 RADIO_GPIO_Port

```
#define RADIO_GPIO_Port GPIOA
```

7.7.2.12 RADIO_Pin

```
#define RADIO_Pin GPIO_PIN_8
```

7.7.2.13 SMYO_GPIO_Port

```
#define SMYO_GPIO_Port GPIOC
```

7.7.2.14 SMYO_Pin

```
#define SMYO_Pin GPIO_PIN_0
```

7.7.2.15 SPIN_ENCA_GPIO_Port

```
#define SPIN_ENCA_GPIO_Port GPIOA
```

7.7.2.16 SPIN_ENCA_Pin

```
#define SPIN_ENCA_Pin GPIO_PIN_6
```

7.7.2.17 SPIN_ENCB_GPIO_Port

```
#define SPIN_ENCB_GPIO_Port GPIOA
```

7.7.2.18 SPIN_ENCB_Pin

```
#define SPIN_ENCB_Pin GPIO_PIN_7
```

7.7.2.19 SPIN_PWMA_GPIO_Port

```
#define SPIN_PWMA_GPIO_Port GPIOA
```

7.7.2.20 SPIN_PWMA_Pin

```
#define SPIN_PWMA_Pin GPIO_PIN_0
```

7.7.2.21 SPIN_PWMB_GPIO_Port

```
#define SPIN_PWMB_GPIO_Port GPIOA
```

7.7.2.22 SPIN_PWMB_Pin

```
#define SPIN_PWMB_Pin GPIO_PIN_1
```

7.7.3 Function Documentation

7.7.3.1 Error_Handler()

```
void Error_Handler (
             void  )
```

This function is executed in case of error occurrence.

Return values

None	

7.7.3.2 HAL_TIM_MspPostInit()

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

TIM2 GPIO Configuration PA0 ------> TIM2_CH1 PA1 ------> TIM2_CH2 PB10 ------> TIM2_CH3 PB11 ------> TIM2_CH4

TIM2 GPIO Configuration PA0 ------> TIM2_CH1 PA1 ------> TIM2_CH2 PB10 ------> TIM2_CH3 PB11 ------> TIM2_CH4

7.8 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 "stm32l4xx_hal.h"
00031
00032 /* Private includes ----------------------------------------------------------*/
00033 /* USER CODE BEGIN Includes */
00034
00035 /* USER CODE END Includes */
00036
00037 /* Exported types ------------------------------------------------------------*/
00038 /* USER CODE BEGIN ET */
00039
00040 /* USER CODE END ET */
00041
00042 /* Exported constants --------------------------------------------------------*/
00043 /* USER CODE BEGIN EC */
00044
00045 /* USER CODE END EC */
00046
00047 /* Exported macro ------------------------------------------------------------*/
00048 /* USER CODE BEGIN EM */
00049
00050 /* USER CODE END EM */
00051
00052 void HAL_TIM_MspPostInit(TIM_HandleTypeDef *htim);
00053
00054 /* Exported functions prototypes ---------------------------------------------*/
00055 void Error_Handler(void);
00056
00057 /* USER CODE BEGIN EFP */
00058
00059 /* USER CODE END EFP */
00060
00061 /* Private defines -----------------------------------------------------------*/
00062 #define SMYO_Pin GPIO_PIN_0
00063 #define SMYO_GPIO_Port GPIOC
00064 #define HMYO_Pin GPIO_PIN_1
00065 #define HMYO_GPIO_Port GPIOC
00066 #define SPIN_PWMA_Pin GPIO_PIN_0
00067 #define SPIN_PWMA_GPIO_Port GPIOA
00068 #define SPIN_PWMB_Pin GPIO_PIN_1
00069 #define SPIN_PWMB_GPIO_Port GPIOA
```

```
00070 #define SPIN_ENCA_Pin GPIO_PIN_6
00071 #define SPIN_ENCA_GPIO_Port GPIOA
00072 #define SPIN_ENCB_Pin GPIO_PIN_7
00073 #define SPIN_ENCB_GPIO_Port GPIOA
00074 #define HAND_PWMA_Pin GPIO_PIN_10
00075 #define HAND_PWMA_GPIO_Port GPIOB
00076 #define HAND_PWMB_Pin GPIO_PIN_11
00077 #define HAND_PWMB_GPIO_Port GPIOB
00078 #define RADIO_Pin GPIO_PIN_8
00079 #define RADIO_GPIO_Port GPIOA
00080 #define HAND_ENCA_Pin GPIO_PIN_6
00081 #define HAND_ENCA_GPIO_Port GPIOB
00082 #define HAND_ENCB_Pin GPIO_PIN_7
00083 #define HAND_ENCB_GPIO_Port GPIOB
00084
00085 /* USER CODE BEGIN Private defines */
00086
00087 /* USER CODE END Private defines */
00088
00089 #ifdef __cplusplus
00090 }
00091 #endif
00092
00093 #endif /* __MAIN_H */
```

7.9 Core/Inc/mainpage.h File Reference

7.10 mainpage.h

Go to the documentation of this file.

```
00001 /*
00002  * mainpage.h
00003  *
00004  *  Created on: May 22, 2024
00005  *      Author: julia
00006  */
00007
00008 #ifndef SRC_MAINPAGE_H_
00009 #define SRC_MAINPAGE_H_
00010
00053 #endif /* SRC_MAINPAGE_H_ */
```

7.11 Core/Inc/motor_driver.h File Reference

```
#include <stdio.h>
#include <stdint.h>
#include "stm32l4xx_hal.h"
```

Classes

- struct motor_t

  Represents a motor objects with two PWM channels in a timer and a duty cycle.

Functions

- void start_PWM (motor_t *p_mot)

  A function to enable the motor driver channels.
- void stop_PWM (motor_t *p_mot)

  A function to disable the motor driver channels.
- void set_duty (motor_t *p_mot, int32_t duty)

  A function to set the duty cycle for the motor.

7.11.1 Function Documentation

7.11.1.1 set_duty()

```
void set_duty (
            motor_t * p_mot,
            int32_t duty )
```

A function to set the duty cycle for the motor.

Parameters

p_mot	The motor object to perform the function on.
duty	The CCR value used to set the duty cycle of the motor.

7.11.1.2 start_PWM()

```
void start_PWM (
            motor_t * p_mot )
```

A function to enable the motor driver channels.

Parameters

p_mot	The motor object to perform the function on.

7.11.1.3 stop_PWM()

```
void stop_PWM (
            motor_t * p_mot )
```

A function to disable the motor driver channels.

Parameters

p_mot	The motor object to perform the function on.

7.12 motor_driver.h

Go to the documentation of this file.

```
00001 /*
00002  *@file           :  motor_driver.h
00003  *
00004  *@brief          : This is the motor driver library used to control the PWM signals going to the
      motors to control direction and speed.
00005  *                  Functions include starting and stopping the PWM timer channels, and updating the
      duty cycle for the motor.
00006  *
00007  *  Created on: Apr 18, 2024
```

```
00008  *      Author: Julia Fay
00009  */
00010
00011 #ifndef INC_MOTOR_DRIVER_H_
00012 #define INC_MOTOR_DRIVER_H_
00013 #include <stdio.h>
00014 #include <stdint.h>
00015 #include "stm32l4xx_hal.h"
00016
00020 struct {
00021
00022     int32_t  pwm_val;
00024     //two channels for each motor
00025     uint32_t channel1;
00026     uint32_t channel2;
00028     //The handle to the HAL timer object used for PWM generation. Include * so its a pointer to the
      object
00029     TIM_HandleTypeDef* hal_tim;
00031 } typedef motor_t;
00032
00033
00040 void start_PWM(motor_t* p_mot);
00041
00048 void stop_PWM(motor_t* p_mot);
00049
00056 void set_duty(motor_t* p_mot, int32_t duty);
00057
00058
00059 #endif /* INC_MOTOR_DRIVER_H_ */
```

7.13 Core/Inc/myo.h File Reference

Defines the myoelectric sensor struct and its methods. This file is used to read and interpret the output form the myoelectric sensor to be sent to the controller to determine the desired motor position.

```
#include <stdio.h>
#include <stdint.h>
#include "stm32l4xx_hal.h"
```

Classes

- struct myo_t

  Represents a myoelectric sensor object that has an ADC object and a current sensor value.

Functions

- uint16_t read_current (myo_t *p_myo)

  A function to get the ADC value for the myoelectric sensor.

7.13.1 Detailed Description

Defines the myoelectric sensor struct and its methods. This file is used to read and interpret the output form the myoelectric sensor to be sent to the controller to determine the desired motor position.

7.13.2 Function Documentation

7.13.2.1 read_current()

```
uint16_t read_current (
            myo_t * p_myo )
```

A function to get the ADC value for the myoelectric sensor.

Parameters

p_myo	The myoelectric sensor object to perform the function on.

Returns

current_value The current value of the sensor read from the ADC.

Parameters

p_myo	The myoelectric sensor object to perform the function on.

7.14 myo.h

Go to the documentation of this file.

```
00001
00007 #ifndef INC_MYO_H_
00008 #define INC_MYO_H_
00009
00010 #include <stdio.h>
00011 #include <stdint.h>
00012 #include "stm32l4xx_hal.h"
00016 struct{
00017
00018         ADC_HandleTypeDef* hal_adc;
00019         int16_t current_value;
00021 } typedef myo_t;
00022
00030 uint16_t read_current(myo_t* p_myo);
00031
00032
00033
00034 #endif /* INC_MYO_H_ */
```

7.15 Core/Inc/radio.h File Reference

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

Functions

- int check_delta (int16_t pulse_width)

  Checks the pulse width value from the radio transmitter and returns a 1 if the signal is larger or smaller than 1.5 ms.

7.15.1 Function Documentation

7.15.1.1 check_delta()

```
int check_delta (
            int16_t pulse_width )
```

Checks the pulse width value from the radio transmitter and returns a 1 if the signal is larger or smaller than 1.5 ms.

Parameters

pulse_width	The pusle width value in ms calculated in the interrupt callback function to be interpreted by the check_delta function.

Returns

valid Returns either a 0 or 1 based on whether the signal is at the expected value of 1.5 ms, in which case a 0 is returned, or a 1 ,when the signal deviates from 1.5 ms in either direction.

7.16 radio.h

Go to the documentation of this file.

```
00001 /*
00002  *@file           :  radio.h
00003  *
00004  *@brief          : This is the radio library that interprets the pulse width signal from the radio
      transmitter.
00005  *
00006  *  Created on: Apr 18, 2024
00007  *      Author: Julia Fay
00008  */
00009
00010 #ifndef INC_RADIO_H_
00011 #define INC_RADIO_H_
00012
00013 #include <stdint.h>
00014 #include <stdio.h>
00015
00016
00028 int check_delta(int16_t pulse_width);
00029
00030
00031 #endif /* INC_RADIO_H_ */
```

7.17 Core/Inc/stm32l4xx_hal_conf.h File Reference

HAL configuration template file. This file should be copied to the application folder and renamed to stm32l4xx_hal_conf.h.

```
#include "stm32l4xx_hal_rcc.h"
#include "stm32l4xx_hal_gpio.h"
#include "stm32l4xx_hal_dma.h"
#include "stm32l4xx_hal_cortex.h"
#include "stm32l4xx_hal_adc.h"
#include "stm32l4xx_hal_exti.h"
#include "stm32l4xx_hal_flash.h"
#include "stm32l4xx_hal_pwr.h"
#include "stm32l4xx_hal_tim.h"
#include "stm32l4xx_hal_uart.h"
```

Macros

- #define HAL_MODULE_ENABLED

  This is the list of modules to be used in the HAL driver.
- #define HAL_ADC_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 HSE_VALUE ((uint32_t)8000000U)

  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 ((uint32_t)100U)
- #define MSI_VALUE ((uint32_t)4000000U)

  Internal Multiple Speed oscillator (MSI) default value. This value is the default MSI range value after Reset.
- #define HSI_VALUE ((uint32_t)16000000U)

  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 HSI48_VALUE ((uint32_t)48000000U)

  Internal High Speed oscillator (HSI48) value for USB FS, SDMMC and RNG. This internal oscillator is mainly dedicated to provide a high precision clock to the USB peripheral by means of a special Clock Recovery System (CRS) circuitry. When the CRS is not used, the HSI48 RC oscillator runs on it default frequency which is subject to manufacturing process variations.
- #define LSI_VALUE 32000U

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

  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 5000U
- #define EXTERNAL_SAI1_CLOCK_VALUE 2097000U

  External clock source for SAI1 peripheral This value is used by the RCC HAL module to compute the SAI1 & SAI2 clock source frequency.
- #define EXTERNAL_SAI2_CLOCK_VALUE 2097000U

  External clock source for SAI2 peripheral This value is used by the RCC HAL module to compute the SAI1 & SAI2 clock source frequency.
- #define VDD_VALUE 3300U

  This is the HAL system configuration section.
- #define TICK_INT_PRIORITY 15U
- #define USE_RTOS 0U
- #define PREFETCH_ENABLE 0U
- #define INSTRUCTION_CACHE_ENABLE 1U
- #define DATA_CACHE_ENABLE 1U
- #define USE_HAL_ADC_REGISTER_CALLBACKS 0U

  Uncomment the line below to expanse the "assert_param" macro in the HAL drivers code.
- #define USE_HAL_CAN_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_DCMI_REGISTER_CALLBACKS 0U

- #define USE_HAL_DFSDM_REGISTER_CALLBACKS 0U
- #define USE_HAL_DMA2D_REGISTER_CALLBACKS 0U
- #define USE_HAL_DSI_REGISTER_CALLBACKS 0U
- #define USE_HAL_GFXMMU_REGISTER_CALLBACKS 0U
- #define USE_HAL_HASH_REGISTER_CALLBACKS 0U
- #define USE_HAL_HCD_REGISTER_CALLBACKS 0U
- #define USE_HAL_I2C_REGISTER_CALLBACKS 0U
- #define USE_HAL_IRDA_REGISTER_CALLBACKS 0U
- #define USE_HAL_LPTIM_REGISTER_CALLBACKS 0U
- #define USE_HAL_LTDC_REGISTER_CALLBACKS 0U
- #define USE_HAL_MMC_REGISTER_CALLBACKS 0U
- #define USE_HAL_OPAMP_REGISTER_CALLBACKS 0U
- #define USE_HAL_OSPI_REGISTER_CALLBACKS 0U
- #define USE_HAL_PCD_REGISTER_CALLBACKS 0U
- #define USE_HAL_QSPI_REGISTER_CALLBACKS 0U
- #define USE_HAL_RNG_REGISTER_CALLBACKS 0U
- #define USE_HAL_RTC_REGISTER_CALLBACKS 0U
- #define USE_HAL_SAI_REGISTER_CALLBACKS 0U
- #define USE_HAL_SD_REGISTER_CALLBACKS 0U
- #define USE_HAL_SMARTCARD_REGISTER_CALLBACKS 0U
- #define USE_HAL_SMBUS_REGISTER_CALLBACKS 0U
- #define USE_HAL_SPI_REGISTER_CALLBACKS 0U
- #define USE_HAL_SWPMI_REGISTER_CALLBACKS 0U
- #define USE_HAL_TIM_REGISTER_CALLBACKS 0U
- #define USE_HAL_TSC_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 USE_SPI_CRC 0U
- #define assert_param(expr) ((void)0U)

  Include module's header file.

7.17.1 Detailed Description

HAL configuration template file. This file should be copied to the application folder and renamed to stm32l4xx_hal_conf.h.

Author

MCD Application Team

Attention

7.17.2 Macro Definition Documentation

7.17.2.1 assert_param

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

Include module's header file.

7.17.2.2 DATA_CACHE_ENABLE

```
#define DATA_CACHE_ENABLE 1U
```

7.17.2.3 EXTERNAL_SAI1_CLOCK_VALUE

```
#define EXTERNAL_SAI1_CLOCK_VALUE 2097000U
```

External clock source for SAI1 peripheral This value is used by the RCC HAL module to compute the SAI1 & SAI2 clock source frequency.

Value of the SAI1 External clock source in Hz

7.17.2.4 EXTERNAL_SAI2_CLOCK_VALUE

```
#define EXTERNAL_SAI2_CLOCK_VALUE 2097000U
```

External clock source for SAI2 peripheral This value is used by the RCC HAL module to compute the SAI1 & SAI2 clock source frequency.

Value of the SAI2 External clock source in Hz

7.17.2.5 HAL_ADC_MODULE_ENABLED

```
#define HAL_ADC_MODULE_ENABLED
```

7.17.2.6 HAL_CORTEX_MODULE_ENABLED

```
#define HAL_CORTEX_MODULE_ENABLED
```

7.17.2.7 HAL_DMA_MODULE_ENABLED

```
#define HAL_DMA_MODULE_ENABLED
```

7.17.2.8 HAL_EXTI_MODULE_ENABLED

```
#define HAL_EXTI_MODULE_ENABLED
```

7.17.2.9 HAL_FLASH_MODULE_ENABLED

```
#define HAL_FLASH_MODULE_ENABLED
```

7.17.2.10 HAL_GPIO_MODULE_ENABLED

```
#define HAL_GPIO_MODULE_ENABLED
```

7.17.2.11 HAL_MODULE_ENABLED

```
#define HAL_MODULE_ENABLED
```

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

7.17.2.12 HAL_PWR_MODULE_ENABLED

```
#define HAL_PWR_MODULE_ENABLED
```

7.17.2.13 HAL_RCC_MODULE_ENABLED

```
#define HAL_RCC_MODULE_ENABLED
```

7.17.2.14 HAL_TIM_MODULE_ENABLED

```
#define HAL_TIM_MODULE_ENABLED
```

7.17.2.15 HAL_UART_MODULE_ENABLED

```
#define HAL_UART_MODULE_ENABLED
```

7.17.2.16 HSE_STARTUP_TIMEOUT

```
#define HSE_STARTUP_TIMEOUT ((uint32_t)100U)
```

Time out for HSE start up, in ms

7.17.2.17 HSE_VALUE

```
#define HSE_VALUE ((uint32_t)8000000U)
```

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

7.17.2.18 HSI48_VALUE

```
#define HSI48_VALUE ((uint32_t)48000000U)
```

Internal High Speed oscillator (HSI48) value for USB FS, SDMMC and RNG. This internal oscillator is mainly dedicated to provide a high precision clock to the USB peripheral by means of a special Clock Recovery System (CRS) circuitry. When the CRS is not used, the HSI48 RC oscillator runs on it default frequency which is subject to manufacturing process variations.

Value of the Internal High Speed oscillator for USB FS/SDMMC/RNG in Hz. The real value my vary depending on manufacturing process variations.

7.17.2.19 HSI_VALUE

```
#define HSI_VALUE ((uint32_t)16000000U)
```

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

7.17.2.20 INSTRUCTION_CACHE_ENABLE

```
#define INSTRUCTION_CACHE_ENABLE 1U
```

7.17.2.21 LSE_STARTUP_TIMEOUT

```
#define LSE_STARTUP_TIMEOUT 5000U
```

Time out for LSE start up, in ms

7.17.2.22 LSE_VALUE

```
#define LSE_VALUE 32768U
```

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

7.17.2.23 LSI_VALUE

```
#define LSI_VALUE 32000U
```

Internal Low Speed oscillator (LSI) value.

LSI Typical Value in Hz

7.17.2.24 MSI_VALUE

```
#define MSI_VALUE ((uint32_t)4000000U)
```

Internal Multiple Speed oscillator (MSI) default value. This value is the default MSI range value after Reset.

Value of the Internal oscillator in Hz

7.17.2.25 PREFETCH_ENABLE

```
#define PREFETCH_ENABLE 0U
```

7.17.2.26 TICK_INT_PRIORITY

```
#define TICK_INT_PRIORITY 15U
```

tick interrupt priority

7.17.2.27 USE_HAL_ADC_REGISTER_CALLBACKS

```
#define USE_HAL_ADC_REGISTER_CALLBACKS 0U
```

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

Set below the peripheral configuration to "1U" to add the support of HAL callback registration/deregistration feature for the HAL driver(s). This allows user application to provide specific callback functions thanks to HAL_PPP_↩ RegisterCallback() rather than overwriting the default weak callback functions (see each stm32l4xx_hal_ppp.h file for possible callback identifiers defined in HAL_PPP_CallbackIDTypeDef for each PPP peripheral).

7.17.2.28 USE_HAL_CAN_REGISTER_CALLBACKS

```
#define USE_HAL_CAN_REGISTER_CALLBACKS 0U
```

7.17.2.29 USE_HAL_COMP_REGISTER_CALLBACKS

```
#define USE_HAL_COMP_REGISTER_CALLBACKS 0U
```

7.17.2.30 USE_HAL_CRYP_REGISTER_CALLBACKS

```
#define USE_HAL_CRYP_REGISTER_CALLBACKS 0U
```

7.17.2.31 USE_HAL_DAC_REGISTER_CALLBACKS

```
#define USE_HAL_DAC_REGISTER_CALLBACKS 0U
```

7.17.2.32 USE_HAL_DCMI_REGISTER_CALLBACKS

```
#define USE_HAL_DCMI_REGISTER_CALLBACKS 0U
```

7.17.2.33 USE_HAL_DFSDM_REGISTER_CALLBACKS

```
#define USE_HAL_DFSDM_REGISTER_CALLBACKS 0U
```

7.17.2.34 USE_HAL_DMA2D_REGISTER_CALLBACKS

```
#define USE_HAL_DMA2D_REGISTER_CALLBACKS 0U
```

7.17.2.35 USE_HAL_DSI_REGISTER_CALLBACKS

```
#define USE_HAL_DSI_REGISTER_CALLBACKS 0U
```

7.17.2.36 USE_HAL_GFXMMU_REGISTER_CALLBACKS

```
#define USE_HAL_GFXMMU_REGISTER_CALLBACKS 0U
```

7.17.2.37 USE_HAL_HASH_REGISTER_CALLBACKS

```
#define USE_HAL_HASH_REGISTER_CALLBACKS 0U
```

7.17.2.38 USE_HAL_HCD_REGISTER_CALLBACKS

```
#define USE_HAL_HCD_REGISTER_CALLBACKS 0U
```

7.17.2.39 USE_HAL_I2C_REGISTER_CALLBACKS

```
#define USE_HAL_I2C_REGISTER_CALLBACKS 0U
```

7.17.2.40 USE_HAL_IRDA_REGISTER_CALLBACKS

```
#define USE_HAL_IRDA_REGISTER_CALLBACKS 0U
```

7.17.2.41 USE_HAL_LPTIM_REGISTER_CALLBACKS

```
#define USE_HAL_LPTIM_REGISTER_CALLBACKS 0U
```

7.17.2.42 USE_HAL_LTDC_REGISTER_CALLBACKS

```
#define USE_HAL_LTDC_REGISTER_CALLBACKS 0U
```

7.17.2.43 USE_HAL_MMC_REGISTER_CALLBACKS

```
#define USE_HAL_MMC_REGISTER_CALLBACKS 0U
```

7.17.2.44 USE_HAL_OPAMP_REGISTER_CALLBACKS

```
#define USE_HAL_OPAMP_REGISTER_CALLBACKS 0U
```

7.17.2.45 USE_HAL_OSPI_REGISTER_CALLBACKS

```
#define USE_HAL_OSPI_REGISTER_CALLBACKS 0U
```

7.17.2.46 USE_HAL_PCD_REGISTER_CALLBACKS

```
#define USE_HAL_PCD_REGISTER_CALLBACKS 0U
```

7.17.2.47 USE_HAL_QSPI_REGISTER_CALLBACKS

```
#define USE_HAL_QSPI_REGISTER_CALLBACKS 0U
```

7.17.2.48 USE_HAL_RNG_REGISTER_CALLBACKS

```
#define USE_HAL_RNG_REGISTER_CALLBACKS 0U
```

7.17.2.49 USE_HAL_RTC_REGISTER_CALLBACKS

```
#define USE_HAL_RTC_REGISTER_CALLBACKS 0U
```

7.17.2.50 USE_HAL_SAI_REGISTER_CALLBACKS

```
#define USE_HAL_SAI_REGISTER_CALLBACKS 0U
```

7.17.2.51 USE_HAL_SD_REGISTER_CALLBACKS

```
#define USE_HAL_SD_REGISTER_CALLBACKS 0U
```

7.17.2.52 USE_HAL_SMARTCARD_REGISTER_CALLBACKS

```
#define USE_HAL_SMARTCARD_REGISTER_CALLBACKS 0U
```

7.17.2.53 USE_HAL_SMBUS_REGISTER_CALLBACKS

```
#define USE_HAL_SMBUS_REGISTER_CALLBACKS 0U
```

7.17.2.54 USE_HAL_SPI_REGISTER_CALLBACKS

```
#define USE_HAL_SPI_REGISTER_CALLBACKS 0U
```

7.17.2.55 USE_HAL_SWPMI_REGISTER_CALLBACKS

```
#define USE_HAL_SWPMI_REGISTER_CALLBACKS 0U
```

7.17.2.56 USE_HAL_TIM_REGISTER_CALLBACKS

```
#define USE_HAL_TIM_REGISTER_CALLBACKS 0U
```

7.17.2.57 USE_HAL_TSC_REGISTER_CALLBACKS

```
#define USE_HAL_TSC_REGISTER_CALLBACKS 0U
```

7.17.2.58 USE_HAL_UART_REGISTER_CALLBACKS

```
#define USE_HAL_UART_REGISTER_CALLBACKS 0U
```

7.17.2.59 USE_HAL_USART_REGISTER_CALLBACKS

```
#define USE_HAL_USART_REGISTER_CALLBACKS 0U
```

7.17.2.60 USE_HAL_WWDG_REGISTER_CALLBACKS

```
#define USE_HAL_WWDG_REGISTER_CALLBACKS 0U
```

7.17.2.61 USE_RTOS

```
#define USE_RTOS 0U
```

7.17.2.62 USE_SPI_CRC

```
#define USE_SPI_CRC 0U
```

7.17.2.63 VDD_VALUE

```
#define VDD_VALUE 3300U
```

This is the HAL system configuration section.

Value of VDD in mv

7.18 stm32l4xx_hal_conf.h

Go to the documentation of this file.

```
00001 /* USER CODE BEGIN Header */
00021 /* USER CODE END Header */
00022
00023 /* Define to prevent recursive inclusion -------------------------------------*/
00024 #ifndef STM32L4xx_HAL_CONF_H
00025 #define STM32L4xx_HAL_CONF_H
00026
00027 #ifdef __cplusplus
00028  extern "C" {
00029 #endif
00030
00031 /* Exported types ------------------------------------------------------------*/
00032 /* Exported constants --------------------------------------------------------*/
00033
00034 /* ########################## Module Selection ############################## */
00038 #define HAL_MODULE_ENABLED
00039 #define HAL_ADC_MODULE_ENABLED
00040 /*#define HAL_CRYP_MODULE_ENABLED   */
00041 /*#define HAL_CAN_MODULE_ENABLED   */
00042 /*#define HAL_COMP_MODULE_ENABLED   */
00043 /*#define HAL_I2C_MODULE_ENABLED   */
00044 /*#define HAL_CRC_MODULE_ENABLED   */
00045 /*#define HAL_CRYP_MODULE_ENABLED   */
00046 /*#define HAL_DAC_MODULE_ENABLED   */
00047 /*#define HAL_DCMI_MODULE_ENABLED   */
00048 /*#define HAL_DMA2D_MODULE_ENABLED   */
00049 /*#define HAL_DFSDM_MODULE_ENABLED   */
00050 /*#define HAL_DSI_MODULE_ENABLED   */
00051 /*#define HAL_FIREWALL_MODULE_ENABLED   */
00052 /*#define HAL_GFXMMU_MODULE_ENABLED   */
00053 /*#define HAL_HCD_MODULE_ENABLED   */
00054 /*#define HAL_HASH_MODULE_ENABLED   */
00055 /*#define HAL_I2S_MODULE_ENABLED   */
00056 /*#define HAL_IRDA_MODULE_ENABLED   */
00057 /*#define HAL_IWDG_MODULE_ENABLED   */
00058 /*#define HAL_LTDC_MODULE_ENABLED   */
00059 /*#define HAL_LCD_MODULE_ENABLED   */
00060 /*#define HAL_LPTIM_MODULE_ENABLED   */
00061 /*#define HAL_MMC_MODULE_ENABLED   */
00062 /*#define HAL_NAND_MODULE_ENABLED   */
00063 /*#define HAL_NOR_MODULE_ENABLED   */
00064 /*#define HAL_OPAMP_MODULE_ENABLED   */
00065 /*#define HAL_OSPI_MODULE_ENABLED   */
00066 /*#define HAL_OSPI_MODULE_ENABLED   */
00067 /*#define HAL_PCD_MODULE_ENABLED   */
00068 /*#define HAL_PKA_MODULE_ENABLED   */
00069 /*#define HAL_QSPI_MODULE_ENABLED   */
00070 /*#define HAL_QSPI_MODULE_ENABLED   */
00071 /*#define HAL_RNG_MODULE_ENABLED   */
00072 /*#define HAL_RTC_MODULE_ENABLED   */
00073 /*#define HAL_SAI_MODULE_ENABLED   */
00074 /*#define HAL_SD_MODULE_ENABLED   */
00075 /*#define HAL_SMBUS_MODULE_ENABLED   */
00076 /*#define HAL_SMARTCARD_MODULE_ENABLED   */
00077 /*#define HAL_SPI_MODULE_ENABLED   */
00078 /*#define HAL_SRAM_MODULE_ENABLED   */
00079 /*#define HAL_SWPMI_MODULE_ENABLED   */
00080 #define HAL_TIM_MODULE_ENABLED
```

```
00081 /*#define HAL_TSC_MODULE_ENABLED   */
00082 #define HAL_UART_MODULE_ENABLED
00083 /*#define HAL_USART_MODULE_ENABLED   */
00084 /*#define HAL_WWDG_MODULE_ENABLED   */
00085 /*#define HAL_EXTI_MODULE_ENABLED   */
00086 /*#define HAL_PSSI_MODULE_ENABLED   */
00087 #define HAL_GPIO_MODULE_ENABLED
00088 #define HAL_EXTI_MODULE_ENABLED
00089 #define HAL_DMA_MODULE_ENABLED
00090 #define HAL_RCC_MODULE_ENABLED
00091 #define HAL_FLASH_MODULE_ENABLED
00092 #define HAL_PWR_MODULE_ENABLED
00093 #define HAL_CORTEX_MODULE_ENABLED
00094
00095 /* ########################## Oscillator Values adaptation ####################*/
00101 #if !defined  (HSE_VALUE)
00102   #define HSE_VALUE    ((uint32_t)8000000U)
00103 #endif /* HSE_VALUE */
00104
00105 #if !defined  (HSE_STARTUP_TIMEOUT)
00106   #define HSE_STARTUP_TIMEOUT    ((uint32_t)100U)
00107 #endif /* HSE_STARTUP_TIMEOUT */
00108
00113 #if !defined  (MSI_VALUE)
00114   #define MSI_VALUE    ((uint32_t)4000000U)
00115 #endif /* MSI_VALUE */
00121 #if !defined  (HSI_VALUE)
00122   #define HSI_VALUE    ((uint32_t)16000000U)
00123 #endif /* HSI_VALUE */
00124
00132 #if !defined  (HSI48_VALUE)
00133  #define HSI48_VALUE   ((uint32_t)48000000U)
00135 #endif /* HSI48_VALUE */
00136
00140 #if !defined  (LSI_VALUE)
00141  #define LSI_VALUE  32000U
00142 #endif /* LSI_VALUE */
00150 #if !defined  (LSE_VALUE)
00151   #define LSE_VALUE    32768U
00152 #endif /* LSE_VALUE */
00153
00154 #if !defined  (LSE_STARTUP_TIMEOUT)
00155   #define LSE_STARTUP_TIMEOUT    5000U
00156 #endif /* HSE_STARTUP_TIMEOUT */
00157
00163 #if !defined  (EXTERNAL_SAI1_CLOCK_VALUE)
00164   #define EXTERNAL_SAI1_CLOCK_VALUE    2097000U
00165 #endif /* EXTERNAL_SAI1_CLOCK_VALUE */
00166
00172 #if !defined  (EXTERNAL_SAI2_CLOCK_VALUE)
00173   #define EXTERNAL_SAI2_CLOCK_VALUE    2097000U
00174 #endif /* EXTERNAL_SAI2_CLOCK_VALUE */
00175
00176 /* Tip: To avoid modifying this file each time you need to use different HSE,
00177    ===  you can define the HSE value in your toolchain compiler preprocessor. */
00178
00179 /* ########################### System Configuration ######################### */
00184 #define  VDD_VALUE                    3300U
00185 #define  TICK_INT_PRIORITY            15U
00186 #define  USE_RTOS                     0U
00187 #define  PREFETCH_ENABLE              0U
00188 #define  INSTRUCTION_CACHE_ENABLE     1U
00189 #define  DATA_CACHE_ENABLE            1U
00190
00191 /* ########################## Assert Selection ############################## */
00196 /* #define USE_FULL_ASSERT    1U */
00197
00198 /* ################### Register callback feature configuration ############### */
00208 #define USE_HAL_ADC_REGISTER_CALLBACKS        0U
00209 #define USE_HAL_CAN_REGISTER_CALLBACKS        0U
00210 #define USE_HAL_COMP_REGISTER_CALLBACKS       0U
00211 #define USE_HAL_CRYP_REGISTER_CALLBACKS       0U
00212 #define USE_HAL_DAC_REGISTER_CALLBACKS        0U
00213 #define USE_HAL_DCMI_REGISTER_CALLBACKS       0U
00214 #define USE_HAL_DFSDM_REGISTER_CALLBACKS      0U
00215 #define USE_HAL_DMA2D_REGISTER_CALLBACKS      0U
00216 #define USE_HAL_DSI_REGISTER_CALLBACKS        0U
00217 #define USE_HAL_GFXMMU_REGISTER_CALLBACKS     0U
00218 #define USE_HAL_HASH_REGISTER_CALLBACKS       0U
00219 #define USE_HAL_HCD_REGISTER_CALLBACKS        0U
00220 #define USE_HAL_I2C_REGISTER_CALLBACKS        0U
00221 #define USE_HAL_IRDA_REGISTER_CALLBACKS       0U
00222 #define USE_HAL_LPTIM_REGISTER_CALLBACKS      0U
00223 #define USE_HAL_LTDC_REGISTER_CALLBACKS       0U
00224 #define USE_HAL_MMC_REGISTER_CALLBACKS        0U
00225 #define USE_HAL_OPAMP_REGISTER_CALLBACKS      0U
00226 #define USE_HAL_OSPI_REGISTER_CALLBACKS       0U
```

```
00227 #define USE_HAL_PCD_REGISTER_CALLBACKS        0U
00228 #define USE_HAL_QSPI_REGISTER_CALLBACKS       0U
00229 #define USE_HAL_RNG_REGISTER_CALLBACKS        0U
00230 #define USE_HAL_RTC_REGISTER_CALLBACKS        0U
00231 #define USE_HAL_SAI_REGISTER_CALLBACKS        0U
00232 #define USE_HAL_SD_REGISTER_CALLBACKS         0U
00233 #define USE_HAL_SMARTCARD_REGISTER_CALLBACKS  0U
00234 #define USE_HAL_SMBUS_REGISTER_CALLBACKS      0U
00235 #define USE_HAL_SPI_REGISTER_CALLBACKS        0U
00236 #define USE_HAL_SWPMI_REGISTER_CALLBACKS      0U
00237 #define USE_HAL_TIM_REGISTER_CALLBACKS        0U
00238 #define USE_HAL_TSC_REGISTER_CALLBACKS        0U
00239 #define USE_HAL_UART_REGISTER_CALLBACKS       0U
00240 #define USE_HAL_USART_REGISTER_CALLBACKS      0U
00241 #define USE_HAL_WWDG_REGISTER_CALLBACKS       0U
00242
00243 /* ################## SPI peripheral configuration ########################## */
00244
00245 /* CRC FEATURE: Use to activate CRC feature inside HAL SPI Driver
00246  * Activated: CRC code is present inside driver
00247  * Deactivated: CRC code cleaned from driver
00248  */
00249
00250 #define USE_SPI_CRC                   0U
00251
00252 /* Includes ------------------------------------------------------------------*/
00257 #ifdef HAL_RCC_MODULE_ENABLED
00258   #include "stm32l4xx_hal_rcc.h"
00259 #endif /* HAL_RCC_MODULE_ENABLED */
00260
00261 #ifdef HAL_GPIO_MODULE_ENABLED
00262   #include "stm32l4xx_hal_gpio.h"
00263 #endif /* HAL_GPIO_MODULE_ENABLED */
00264
00265 #ifdef HAL_DMA_MODULE_ENABLED
00266   #include "stm32l4xx_hal_dma.h"
00267 #endif /* HAL_DMA_MODULE_ENABLED */
00268
00269 #ifdef HAL_DFSDM_MODULE_ENABLED
00270   #include "stm32l4xx_hal_dfsdm.h"
00271 #endif /* HAL_DFSDM_MODULE_ENABLED */
00272
00273 #ifdef HAL_CORTEX_MODULE_ENABLED
00274   #include "stm32l4xx_hal_cortex.h"
00275 #endif /* HAL_CORTEX_MODULE_ENABLED */
00276
00277 #ifdef HAL_ADC_MODULE_ENABLED
00278   #include "stm32l4xx_hal_adc.h"
00279 #endif /* HAL_ADC_MODULE_ENABLED */
00280
00281 #ifdef HAL_CAN_MODULE_ENABLED
00282   #include "stm32l4xx_hal_can.h"
00283 #endif /* HAL_CAN_MODULE_ENABLED */
00284
00285 #ifdef HAL_CAN_LEGACY_MODULE_ENABLED
00286   #include "Legacy/stm32l4xx_hal_can_legacy.h"
00287 #endif /* HAL_CAN_LEGACY_MODULE_ENABLED */
00288
00289 #ifdef HAL_COMP_MODULE_ENABLED
00290   #include "stm32l4xx_hal_comp.h"
00291 #endif /* HAL_COMP_MODULE_ENABLED */
00292
00293 #ifdef HAL_CRC_MODULE_ENABLED
00294   #include "stm32l4xx_hal_crc.h"
00295 #endif /* HAL_CRC_MODULE_ENABLED */
00296
00297 #ifdef HAL_CRYP_MODULE_ENABLED
00298   #include "stm32l4xx_hal_cryp.h"
00299 #endif /* HAL_CRYP_MODULE_ENABLED */
00300
00301 #ifdef HAL_DAC_MODULE_ENABLED
00302   #include "stm32l4xx_hal_dac.h"
00303 #endif /* HAL_DAC_MODULE_ENABLED */
00304
00305 #ifdef HAL_DCMI_MODULE_ENABLED
00306   #include "stm32l4xx_hal_dcmi.h"
00307 #endif /* HAL_DCMI_MODULE_ENABLED */
00308
00309 #ifdef HAL_DMA2D_MODULE_ENABLED
00310   #include "stm32l4xx_hal_dma2d.h"
00311 #endif /* HAL_DMA2D_MODULE_ENABLED */
00312
00313 #ifdef HAL_DSI_MODULE_ENABLED
00314   #include "stm32l4xx_hal_dsi.h"
00315 #endif /* HAL_DSI_MODULE_ENABLED */
00316
00317 #ifdef HAL_EXTI_MODULE_ENABLED
```

```
00318   #include "stm32l4xx_hal_exti.h"
00319 #endif /* HAL_EXTI_MODULE_ENABLED */
00320
00321 #ifdef HAL_GFXMMU_MODULE_ENABLED
00322   #include "stm32l4xx_hal_gfxmmu.h"
00323 #endif /* HAL_GFXMMU_MODULE_ENABLED */
00324
00325 #ifdef HAL_FIREWALL_MODULE_ENABLED
00326   #include "stm32l4xx_hal_firewall.h"
00327 #endif /* HAL_FIREWALL_MODULE_ENABLED */
00328
00329 #ifdef HAL_FLASH_MODULE_ENABLED
00330   #include "stm32l4xx_hal_flash.h"
00331 #endif /* HAL_FLASH_MODULE_ENABLED */
00332
00333 #ifdef HAL_HASH_MODULE_ENABLED
00334   #include "stm32l4xx_hal_hash.h"
00335 #endif /* HAL_HASH_MODULE_ENABLED */
00336
00337 #ifdef HAL_HCD_MODULE_ENABLED
00338   #include "stm32l4xx_hal_hcd.h"
00339 #endif /* HAL_HCD_MODULE_ENABLED */
00340
00341 #ifdef HAL_I2C_MODULE_ENABLED
00342   #include "stm32l4xx_hal_i2c.h"
00343 #endif /* HAL_I2C_MODULE_ENABLED */
00344
00345 #ifdef HAL_IRDA_MODULE_ENABLED
00346   #include "stm32l4xx_hal_irda.h"
00347 #endif /* HAL_IRDA_MODULE_ENABLED */
00348
00349 #ifdef HAL_IWDG_MODULE_ENABLED
00350   #include "stm32l4xx_hal_iwdg.h"
00351 #endif /* HAL_IWDG_MODULE_ENABLED */
00352
00353 #ifdef HAL_LCD_MODULE_ENABLED
00354   #include "stm32l4xx_hal_lcd.h"
00355 #endif /* HAL_LCD_MODULE_ENABLED */
00356
00357 #ifdef HAL_LPTIM_MODULE_ENABLED
00358   #include "stm32l4xx_hal_lptim.h"
00359 #endif /* HAL_LPTIM_MODULE_ENABLED */
00360
00361 #ifdef HAL_LTDC_MODULE_ENABLED
00362   #include "stm32l4xx_hal_ltdc.h"
00363 #endif /* HAL_LTDC_MODULE_ENABLED */
00364
00365 #ifdef HAL_MMC_MODULE_ENABLED
00366   #include "stm32l4xx_hal_mmc.h"
00367 #endif /* HAL_MMC_MODULE_ENABLED */
00368
00369 #ifdef HAL_NAND_MODULE_ENABLED
00370   #include "stm32l4xx_hal_nand.h"
00371 #endif /* HAL_NAND_MODULE_ENABLED */
00372
00373 #ifdef HAL_NOR_MODULE_ENABLED
00374   #include "stm32l4xx_hal_nor.h"
00375 #endif /* HAL_NOR_MODULE_ENABLED */
00376
00377 #ifdef HAL_OPAMP_MODULE_ENABLED
00378   #include "stm32l4xx_hal_opamp.h"
00379 #endif /* HAL_OPAMP_MODULE_ENABLED */
00380
00381 #ifdef HAL_OSPI_MODULE_ENABLED
00382   #include "stm32l4xx_hal_ospi.h"
00383 #endif /* HAL_OSPI_MODULE_ENABLED */
00384
00385 #ifdef HAL_PCD_MODULE_ENABLED
00386   #include "stm32l4xx_hal_pcd.h"
00387 #endif /* HAL_PCD_MODULE_ENABLED */
00388
00389 #ifdef HAL_PKA_MODULE_ENABLED
00390   #include "stm32l4xx_hal_pka.h"
00391 #endif /* HAL_PKA_MODULE_ENABLED */
00392
00393 #ifdef HAL_PSSI_MODULE_ENABLED
00394   #include "stm32l4xx_hal_pssi.h"
00395 #endif /* HAL_PSSI_MODULE_ENABLED */
00396
00397 #ifdef HAL_PWR_MODULE_ENABLED
00398   #include "stm32l4xx_hal_pwr.h"
00399 #endif /* HAL_PWR_MODULE_ENABLED */
00400
00401 #ifdef HAL_QSPI_MODULE_ENABLED
00402   #include "stm32l4xx_hal_qspi.h"
00403 #endif /* HAL_QSPI_MODULE_ENABLED */
00404
```

```
00405 #ifdef HAL_RNG_MODULE_ENABLED
00406   #include "stm32l4xx_hal_rng.h"
00407 #endif /* HAL_RNG_MODULE_ENABLED */
00408
00409 #ifdef HAL_RTC_MODULE_ENABLED
00410   #include "stm32l4xx_hal_rtc.h"
00411 #endif /* HAL_RTC_MODULE_ENABLED */
00412
00413 #ifdef HAL_SAI_MODULE_ENABLED
00414   #include "stm32l4xx_hal_sai.h"
00415 #endif /* HAL_SAI_MODULE_ENABLED */
00416
00417 #ifdef HAL_SD_MODULE_ENABLED
00418   #include "stm32l4xx_hal_sd.h"
00419 #endif /* HAL_SD_MODULE_ENABLED */
00420
00421 #ifdef HAL_SMARTCARD_MODULE_ENABLED
00422   #include "stm32l4xx_hal_smartcard.h"
00423 #endif /* HAL_SMARTCARD_MODULE_ENABLED */
00424
00425 #ifdef HAL_SMBUS_MODULE_ENABLED
00426   #include "stm32l4xx_hal_smbus.h"
00427 #endif /* HAL_SMBUS_MODULE_ENABLED */
00428
00429 #ifdef HAL_SPI_MODULE_ENABLED
00430   #include "stm32l4xx_hal_spi.h"
00431 #endif /* HAL_SPI_MODULE_ENABLED */
00432
00433 #ifdef HAL_SRAM_MODULE_ENABLED
00434   #include "stm32l4xx_hal_sram.h"
00435 #endif /* HAL_SRAM_MODULE_ENABLED */
00436
00437 #ifdef HAL_SWPMI_MODULE_ENABLED
00438   #include "stm32l4xx_hal_swpmi.h"
00439 #endif /* HAL_SWPMI_MODULE_ENABLED */
00440
00441 #ifdef HAL_TIM_MODULE_ENABLED
00442   #include "stm32l4xx_hal_tim.h"
00443 #endif /* HAL_TIM_MODULE_ENABLED */
00444
00445 #ifdef HAL_TSC_MODULE_ENABLED
00446   #include "stm32l4xx_hal_tsc.h"
00447 #endif /* HAL_TSC_MODULE_ENABLED */
00448
00449 #ifdef HAL_UART_MODULE_ENABLED
00450   #include "stm32l4xx_hal_uart.h"
00451 #endif /* HAL_UART_MODULE_ENABLED */
00452
00453 #ifdef HAL_USART_MODULE_ENABLED
00454   #include "stm32l4xx_hal_usart.h"
00455 #endif /* HAL_USART_MODULE_ENABLED */
00456
00457 #ifdef HAL_WWDG_MODULE_ENABLED
00458   #include "stm32l4xx_hal_wwdg.h"
00459 #endif /* HAL_WWDG_MODULE_ENABLED */
00460
00461 /* Exported macro ------------------------------------------------------------*/
00462 #ifdef  USE_FULL_ASSERT
00471   #define assert_param(expr) ((expr) ? (void)0U : assert_failed((uint8_t *)__FILE__, __LINE__))
00472 /* Exported functions ------------------------------------------------------- */
00473   void assert_failed(uint8_t *file, uint32_t line);
00474 #else
00475   #define assert_param(expr) ((void)0U)
00476 #endif /* USE_FULL_ASSERT */
00477
00478 #ifdef __cplusplus
00479 }
00480 #endif
00481
00482 #endif /* STM32L4xx_HAL_CONF_H */
```

7.19 Core/Inc/stm32l4xx_it.h File Reference

This file contains the headers of the interrupt handlers.

Functions

- void NMI_Handler (void)

This function handles Non maskable interrupt.

- void HardFault_Handler (void)

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

  This function handles Memory management fault.
- void BusFault_Handler (void)

  This function handles Prefetch fault, memory access fault.
- void UsageFault_Handler (void)

  This function handles Undefined instruction or illegal state.
- void SVC_Handler (void)

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

  This function handles Debug monitor.
- void PendSV_Handler (void)

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

  This function handles System tick timer.
- void TIM1_CC_IRQHandler (void)

  This function handles TIM1 capture compare interrupt.

7.19.1 Detailed Description

This file contains the headers of the interrupt handlers.

Attention

7.19.2 Function Documentation

7.19.2.1 BusFault_Handler()

```
void BusFault_Handler (
            void  )
```

This function handles Prefetch fault, memory access fault.

7.19.2.2 DebugMon_Handler()

```
void DebugMon_Handler (
            void  )
```

This function handles Debug monitor.

7.19.2.3 HardFault_Handler()

```
void HardFault_Handler (
            void  )
```

This function handles Hard fault interrupt.

7.19.2.4 MemManage_Handler()

```
void MemManage_Handler (
            void  )
```

This function handles Memory management fault.

7.19.2.5 NMI_Handler()

```
void NMI_Handler (
            void  )
```

This function handles Non maskable interrupt.

7.19.2.6 PendSV_Handler()

```
void PendSV_Handler (
            void  )
```

This function handles Pendable request for system service.

7.19.2.7 SVC_Handler()

```
void SVC_Handler (
            void  )
```

This function handles System service call via SWI instruction.

7.19.2.8 SysTick_Handler()

```
void SysTick_Handler (
            void  )
```

This function handles System tick timer.

7.19.2.9 TIM1_CC_IRQHandler()

```
void TIM1_CC_IRQHandler (
            void  )
```

This function handles TIM1 capture compare interrupt.

7.19.2.10 UsageFault_Handler()

```
void UsageFault_Handler (
             void  )
```

This function handles Undefined instruction or illegal state.

7.20 stm32l4xx_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 __STM32L4xx_IT_H
00022 #define __STM32L4xx_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 MemManage_Handler(void);
00052 void BusFault_Handler(void);
00053 void UsageFault_Handler(void);
00054 void SVC_Handler(void);
00055 void DebugMon_Handler(void);
00056 void PendSV_Handler(void);
00057 void SysTick_Handler(void);
00058 void TIM1_CC_IRQHandler(void);
00059 /* USER CODE BEGIN EFP */
00060
00061 /* USER CODE END EFP */
00062
00063 #ifdef __cplusplus
00064 }
00065 #endif
00066
00067 #endif /* __STM32L4xx_IT_H */
```

7.21 Core/Src/calibrate.c File Reference

```
#include "calibrate.h"
```

Functions

- uint32_t find_average (calibrate_t *p_cali)

  A function to find the average myoelectric sensor value for a specified number of data points.

7.21.1 Function Documentation

7.21.1.1 find_average()

```
uint32_t find_average (
            calibrate_t * p_cali )
```

A function to find the average myoelectric sensor value for a specified number of data points.

Parameters

p_cali	The calibration object to perform the function on.

7.22 Core/Src/controller.c File Reference

```
#include "controller.h"
#include "motor_driver.h"
#include "encoder_reader.h"
```

Functions

- void controller_init (controller_t *p_cont)

  A function to move the controlled motor to the desired position. The run function in the P_Control class calculates the PWM signal that will be sent to the motor by taking into account the setpoint of the motor, and the current position of the motor. These values are subtracted to find the error which is then multiplied by Kp. If this value exceeds -100 or 100, the value is saturated to either -100 or 100. Then the set_duty_cycle function imported from the motor_driver class is run. The calculated PWM signal is returned at the end of the function.
- void controller_deinit (controller_t *p_cont)

  A function to de-initialize all of the timer channels.
- int32_t move (controller_t *p_cont, int32_t gain)

  A function to move the controlled motor to the desired position.
- void set_setpoint (controller_t *p_cont, int32_t new_setpoint)

  A function to set the new controller set point.
- void set_K (controller_t *p_cont, int32_t new_gain)

  A function to update the control loop gain.

7.22.1 Function Documentation

7.22.1.1 controller_deinit()

```
void controller_deinit (
            controller_t * p_cont )
```

A function to de-initialize all of the timer channels.

Parameters

p_cont	The controller object to perform the function on.

7.22.1.2 controller_init()

```
void controller_init (
            controller_t * p_cont )
```

A function to move the controlled motor to the desired position. The run function in the P_Control class calculates the PWM signal that will be sent to the motor by taking into account the setpoint of the motor, and the current position of the motor. These values are subtracted to find the error which is then multiplied by Kp. If this value exceeds -100 or 100, the value is saturated to either -100 or 100. Then the set_duty_cycle function imported from the motor_driver class is run. The calculated PWM signal is returned at the end of the function.

A function to initialize all of the timer channels.

Parameters

p_cont	The controller object to perform the function on.

A function to initialize all of the timer channels.

Parameters

p_cont	The controller object to perform the function on.

7.22.1.3 move()

```
int32_t move (
            controller_t * p_cont,
            int32_t gain )
```

A function to move the controlled motor to the desired position.

A function to move the controlled motor to the desired position. The function calculates the PWM signal that will be sent to the motor by taking into account the setpoint of the motor, and the current position of the motor. These values are subtracted to find the error which is then multiplied by Kp. If this value exceeds -3,999 or 3,999, the value is saturated to either -3,999 or 3,999. The function also sets a minimum threshold for when a PWM signal is generated. Anything below 10 times the gain value is considered to be a PWM signal of zero. Then the set_duty function imported from the motor_driver class is run. The calculated PWM signal is returned at the end of the function.

Parameters

p_cont	The controller object to perform the function on.

7.22.1.4 set_K()

```
void set_K (
            controller_t * p_cont,
            int32_t new_gain )
```

A function to update the control loop gain.

Parameters

p_cont	The controller object to perform the function on.
new_gain	The new set point for the controller object.

7.22.1.5 set_setpoint()

```
void set_setpoint (
            controller_t * p_cont,
            int32_t new_setpoint )
```

A function to set the new controller set point.

Parameters

p_cont	The controller object to perform the function on.
new_setpoint	The new set point for the controller object.

7.23 Core/Src/encoder_reader.c File Reference

```
#include "encoder_reader.h"
```

Functions

- void init_channels (encoder_t *p_enc)

  A function to initialize the channels for reading the encoder signals.
- void deinit_channels (encoder_t *p_enc)

  A function to stop the channels from reading the encoder signals.
- void zero (encoder_t *p_enc)

  A function to zero the encoder count.
- int32_t get_pos (encoder_t *p_enc)

  A function to read and return the encoder count.

7.23.1 Function Documentation

7.23.1.1 deinit_channels()

```
void deinit_channels (
            encoder_t * p_enc )
```

A function to stop the channels from reading the encoder signals.

Parameters

p_enc	The encoder object to perform the function on.

7.23.1.2 get_pos()

```
int32_t get_pos (
            encoder_t * p_enc )
```

A function to read and return the encoder count.

Parameters

p_enc	The encoder object to perform the function on.

Returns

count The encoder count to be returned.

7.23.1.3 init_channels()

```
void init_channels (
            encoder_t * p_enc )
```

A function to initialize the channels for reading the encoder signals.

Parameters

p_enc	The encoder object to perform the function on.

7.23.1.4 zero()

```
void zero (
            encoder_t * p_enc )
```

A function to zero the encoder count.

Parameters

p_enc	The encoder object to perform the function on.

7.24 Core/Src/main.c File Reference

: Main program body for the ME 507 term project. The main contents of this file is the finite state machine that controls the prosthetic hand operation. This includes the tasks to open and close the hand and spin the hand from

left to right. Lastly, there is a task to check for an emergency stop signal from a radio transmitter device.

```
#include "main.h"
#include "encoder_reader.h"
#include "motor_driver.h"
#include "controller.h"
#include "myo.h"
#include "calibrate.h"
#include "radio.h"
#include <stdio.h>
#include "stm32l4xx_hal.h"
#include <stdint.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
```

Functions

- void SystemClock_Config (void)

  System Clock Configuration.
- void PeriphCommonClock_Config (void)

  Peripherals Common Clock Configuration.
- static void MX_GPIO_Init (void)

  GPIO Initialization Function.
- static void MX_TIM1_Init (void)

  TIM1 Initialization Function.
- static void MX_TIM2_Init (void)

  TIM2 Initialization Function.
- static void MX_USART2_UART_Init (void)

  USART2 Initialization Function.
- static void MX_ADC1_Init (void)

  ADC1 Initialization Function.
- static void MX_TIM3_Init (void)

  TIM3 Initialization Function.
- static void MX_TIM4_Init (void)

  TIM4 Initialization Function.
- static void MX_ADC2_Init (void)

  ADC2 Initialization Function.
- void task1 (void)

  Task 1 - SPIN TASK.
- void task2 (void)

  Task 2 - HAND TASK.
- void task3 (void)

  Task 3 - WIRELESS E STOP TASK.
- int main (void)

  The application entry point.
- void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef ∗htim)

  Input Capture callback in non-blocking mode. This callback routine calculates the radio transmitted pulse width in ms.
- void Error_Handler (void)

  This function is executed in case of error occurrence.

Variables

- ADC_HandleTypeDef hadc1
- ADC_HandleTypeDef hadc2
- TIM_HandleTypeDef htim1
- TIM_HandleTypeDef htim2
- TIM_HandleTypeDef htim3
- TIM_HandleTypeDef htim4
- UART_HandleTypeDef huart2
- encoder_t spin_enc
- motor_t spin_mot
- motor_t hand_mot
- controller_t spin_cont
- myo_t smyo
- myo_t hmyo
- calibrate_t scali
- calibrate_t hcali
- uint32_t smyo_tst = 0
- uint32_t hand_count_tst = 0
- int32_t spos_tst = 0
- int32_t smyo_av = 0
- uint16_t ch1_val
- uint16_t ch2_val
- uint16_t ch1_p
- uint16_t ch2_p
- uint16_t radio_pulse = 1500
- char tst_buff [150]
- int m

7.24.1 Detailed Description

: Main program body for the ME 507 term project. The main contents of this file is the finite state machine that controls the prosthetic hand operation. This includes the tasks to open and close the hand and spin the hand from left to right. Lastly, there is a task to check for an emergency stop signal from a radio transmitter device.

Authors

: Julia Fay & Jack Foxcroft

Attention

7.24.2 Function Documentation

7.24.2.1 Error_Handler()

```
void Error_Handler (
            void  )
```

This function is executed in case of error occurrence.

Return values

None	

7.24.2.2 HAL_TIM_IC_CaptureCallback()

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

Input Capture callback in non-blocking mode. This callback routine calculates the radio transmitted pulse width in ms.

Parameters

htim	TIM IC handle

Return values

None	

7.24.2.3 main()

```
int main (
            void  )
```

The application entry point.

Return values

int	

7.24.2.4 MX_ADC1_Init()

```
static void MX_ADC1_Init (
            void  )  [static]
```

ADC1 Initialization Function.

Parameters

None	

Return values

None	

Common config

Configure the ADC multi-mode

Configure Regular Channel

7.24.2.5 MX_ADC2_Init()

```
static void MX_ADC2_Init (
             void  )  [static]
```

ADC2 Initialization Function.

Parameters

None	

Return values

None	

Common config

Configure Regular Channel

7.24.2.6 MX_GPIO_Init()

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

GPIO Initialization Function.

Parameters

None	

Return values

None	

7.24.2.7 MX_TIM1_Init()

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

TIM1 Initialization Function.

Parameters

None	

Return values

None	

7.24.2.8 MX_TIM2_Init()

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

TIM2 Initialization Function.

Parameters

None	

Return values

None	

7.24.2.9 MX_TIM3_Init()

```
static void MX_TIM3_Init (
            void  )  [static]
```

TIM3 Initialization Function.

Parameters

None	

Return values

None	

7.24.2.10 MX_TIM4_Init()

```
static void MX_TIM4_Init (
            void  )  [static]
```

TIM4 Initialization Function.

Parameters

None	

Return values

None	

7.24.2.11 MX_USART2_UART_Init()

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

USART2 Initialization Function.

Parameters

None	

Return values

None	

7.24.2.12 PeriphCommonClock_Config()

```
void PeriphCommonClock_Config (
            void  )
```

Peripherals Common Clock Configuration.

Return values

None	

Initializes the peripherals clock

7.24.2.13 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

7.24.2.14 task1()

```
void task1 (
            void  )
```

Task 1 - SPIN TASK.

This function implements a state machine to control a motor based on input from a myoelectric sensor. The state machine has three states:

- State 0: Initialization Initializes the motor driver, PWM channel, encoder channels, and zeroes the encoder value. Also, calibrates the myoelectric sensor to find an average value to be used as a threshold for determining motor direction.
- State 1: Interpret Myoelectric Sensor Reads the current value from the myoelectric sensor and determines the direction of motor movement based on the sensor's value relative to a predefined threshold. Transitions to State 2 if a significant change in the sensor value is detected.
- State 2: Move Motor Sets the motor's position based on the interpreted direction from State 1. If the sensor indicates a forward direction, the motor's setpoint is adjusted accordingly, and the motor is moved to the new setpoint. If the sensor indicates a backward direction, the motor is moved in the opposite direction. Once the motor reaches the setpoint, the state machine transitions back to State 1.

7.24.2.15 task2()

```
void task2 (
            void  )
```

Task 2 - HAND TASK.

This function implements a state machine to control a hand motor based on input from a myoelectric sensor. The state machine has three states:

- State 0: Initialization
  - Initializes the motor driver PWM channel.
  - Sets the motor to be at rest.
  - Calibrates the myoelectric sensor by finding an average value to use as a threshold.
- State 1: Interpret Myoelectric Sensor
  - Reads the current value from the myoelectric sensor.
  - Determines the direction of hand movement (open or close) based on the sensor's value relative to predefined thresholds.
  - Transitions to State 2 if a significant change in the sensor value is detected.
- State 2: Move Hand
  - Moves the hand to the open or closed position based on the interpreted direction from State 1.
  - If the sensor indicates to open the hand and the hand is currently closed, the hand is gradually opened by setting a positive duty cycle to the motor. The hand position is updated, and a counter ensures the hand moves incrementally.
  - If the sensor indicates to close the hand and the hand is currently open, the hand is gradually closed by setting a negative duty cycle to the motor. The hand position is updated, and a counter ensures the hand moves incrementally.
  - Once the hand reaches the desired position (fully open or fully closed), the state machine transitions back to State 1.

7.24.2.16 task3()

```
void task3 (
            void  )
```

Task 3 - WIRELESS E STOP TASK.

This function implements a state machine to handle a wireless emergency stop (E-stop) signal. The state machine has three states:

- State 0: Initialization
  - Starts the timer input capture interrupt for channels 1 and 2.
  - Transitions to State 1 to wait for the E-stop signal.
- State 1: Wait for Signal
  - Continuously monitors for an E-stop signal by checking the radio pulse.
  - If a valid E-stop signal is detected, it transitions to State 2 to perform the emergency stop.
- State 2: Emergency Stop
  - Outputs an emergency stop message.
  - Stops the motors by setting their duty cycles to zero.
  - Deinitializes the motor controller and stops the PWM signals.
  - Remains in this state after performing the emergency stop.

7.24.3 Variable Documentation

7.24.3.1 ch1_p

```
uint16_t ch1_p
```

7.24.3.2 ch1_val

```
uint16_t ch1_val
```

7.24.3.3 ch2_p

```
uint16_t ch2_p
```

7.24.3.4 ch2_val

```
uint16_t ch2_val
```

7.24.3.5 hadc1

```
ADC_HandleTypeDef hadc1
```

7.24.3.6 hadc2

```
ADC_HandleTypeDef hadc2
```

7.24.3.7 hand_count_tst

```
uint32_t hand_count_tst = 0
```

7.24.3.8 hand_mot

```
motor_t hand_mot
```

Initial value:

```
= {.pwm_val = 0,
                    .channel1 = TIM_CHANNEL_3,
                    .channel2 = TIM_CHANNEL_4,
                    .hal_tim = &htim2
                    }
```

7.24.3.9 hcali

```
calibrate_t hcali
```

Initial value:

```
= {.data_pts = 1000000,
                            .p_myo = &hmyo
                        }
```

7.24.3.10 hmyo

```
myo_t hmyo
```

Initial value:

```
= {.hal_adc = &hadc2,
                 .current_value = 0}
```

7.24.3.11 htim1

```
TIM_HandleTypeDef htim1
```

7.24.3.12 htim2

```
TIM_HandleTypeDef htim2
```

7.24.3.13 htim3

```
TIM_HandleTypeDef htim3
```

7.24.3.14 htim4

```
TIM_HandleTypeDef htim4
```

7.24.3.15 huart2

```
UART_HandleTypeDef huart2
```

7.24.3.16 m

```
int m
```

7.24.3.17 radio_pulse

```
uint16_t radio_pulse = 1500
```

7.24.3.18 scali

```
calibrate_t scali
```

Initial value:

```
= {.data_pts = 1000000,
                        .p_myo = &smyo
                    }
```

7.24.3.19 smyo

```
myo_t smyo
```

Initial value:

```
= {.hal_adc = &hadc1,
                  .current_value = 0}
```

7.24.3.20 smyo_av

```
int32_t smyo_av = 0
```

7.24.3.21 smyo_tst

```
uint32_t smyo_tst = 0
```

7.24.3.22 spin_cont

```
controller_t spin_cont
```

Initial value:

```
= {.p_mot = &spin_mot,
                              .p_enc = &spin_enc,
                              .gain = 300,
                              .setpoint = 0}
```

7.24.3.23 spin_enc

```
encoder_t spin_enc
```

Initial value:

```
= {.channel1 = TIM_CHANNEL_1,
                          .channel2 = TIM_CHANNEL_2,
                          .hal_tim = &htim3,
                          .mot_pos = 0,
                          .curr_count = 0,
                          .prev_count = 0,
                          .delta = 0}
```

7.24.3.24 spin_mot

```
motor_t spin_mot
```

Initial value:

```
= {.pwm_val = 0,
                .channel1 = TIM_CHANNEL_1,
                .channel2 = TIM_CHANNEL_2,
                .hal_tim = &htim2
                }
```

7.24.3.25 spos_tst

```
int32_t spos_tst = 0
```

7.24.3.26 tst_buff

```
char tst_buff[150]
```

7.25 Core/Src/motor_driver.c File Reference

```
#include "motor_driver.h"
```

Functions

- void start_PWM (motor_t *p_mot)

  A function to enable the motor driver channels.
- void stop_PWM (motor_t *p_mot)

  A function to disable the motor driver channels.
- void set_duty (motor_t *p_mot, int32_t pwm_sig)

  A function to set the duty cycle for the motor.

7.25.1 Function Documentation

7.25.1.1 set_duty()

```
void set_duty (
            motor_t * p_mot,
            int32_t duty )
```

A function to set the duty cycle for the motor.

Parameters

p_mot	The motor object to perform the function on.
duty	The CCR value used to set the duty cycle of the motor.

7.25.1.2 start_PWM()

```
void start_PWM (
            motor_t * p_mot )
```

A function to enable the motor driver channels.

Parameters

p_mot	The motor object to perform the function on.

7.25.1.3 stop_PWM()

```
void stop_PWM (
            motor_t * p_mot )
```

A function to disable the motor driver channels.

Parameters

p_mot	The motor object to perform the function on.

7.26 Core/Src/myo.c File Reference

```
#include "myo.h"
```

Functions

- uint16_t read_current (myo_t *p_myo)

  A function to get the ADC value for the myoelectric sensor.

7.26.1 Function Documentation

7.26.1.1 read_current()

```
uint16_t read_current (
            myo_t * p_myo )
```

A function to get the ADC value for the myoelectric sensor.

Parameters

p_myo	The myoelectric sensor object to perform the function on.

7.27 Core/Src/radio.c File Reference

```
#include "radio.h"
```

Functions

- int check_delta (int16_t pulse_width)

  Checks the pulse width value from the radio transmitter and returns a 1 if the signal is larger or smaller than 1.5 ms.

7.27.1 Function Documentation

7.27.1.1 check_delta()

```
int check_delta (
            int16_t pulse_width )
```

Checks the pulse width value from the radio transmitter and returns a 1 if the signal is larger or smaller than 1.5 ms.

Parameters

pulse_width	The pusle width value in ms calculated in the interrupt callback function to be interpreted by the check_delta function.

Returns

valid Returns either a 0 or 1 based on whether the signal is at the expected value of 1.5 ms, in which case a 0 is returned, or a 1 ,when the signal deviates from 1.5 ms in either direction.

7.28 Core/Src/stm32l4xx_hal_msp.c File Reference

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

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

Functions

- void HAL_TIM_MspPostInit (TIM_HandleTypeDef *htim)
- void HAL_MspInit (void)

- void HAL_ADC_MspInit (ADC_HandleTypeDef ∗hadc)

  ADC MSP Initialization This function configures the hardware resources used in this example.
- void HAL_ADC_MspDeInit (ADC_HandleTypeDef ∗hadc)

  ADC MSP De-Initialization This function freeze the hardware resources used in this example.
- void HAL_TIM_IC_MspInit (TIM_HandleTypeDef ∗htim_ic)

  TIM_IC MSP Initialization This function configures the hardware resources used in this example.
- void HAL_TIM_PWM_MspInit (TIM_HandleTypeDef ∗htim_pwm)

  TIM_PWM MSP Initialization This function configures the hardware resources used in this example.
- void HAL_TIM_Encoder_MspInit (TIM_HandleTypeDef ∗htim_encoder)

  TIM_Encoder MSP Initialization This function configures the hardware resources used in this example.
- void HAL_TIM_IC_MspDeInit (TIM_HandleTypeDef ∗htim_ic)

  TIM_IC MSP De-Initialization This function freeze the hardware resources used in this example.
- void HAL_TIM_PWM_MspDeInit (TIM_HandleTypeDef ∗htim_pwm)

  TIM_PWM MSP De-Initialization This function freeze the hardware resources used in this example.
- void HAL_TIM_Encoder_MspDeInit (TIM_HandleTypeDef ∗htim_encoder)

  TIM_Encoder MSP De-Initialization This function freeze the hardware resources used in this example.
- void HAL_UART_MspInit (UART_HandleTypeDef ∗huart)

  UART MSP Initialization This function configures the hardware resources used in this example.
- void HAL_UART_MspDeInit (UART_HandleTypeDef ∗huart)

  UART MSP De-Initialization This function freeze the hardware resources used in this example.

Variables

- static uint32_t HAL_RCC_ADC_CLK_ENABLED =0

7.28.1 Detailed Description

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

Attention

7.28.2 Function Documentation

7.28.2.1 HAL_ADC_MspDeInit()

```
void HAL_ADC_MspDeInit (
            ADC_HandleTypeDef * hadc )
```

ADC MSP De-Initialization This function freeze the hardware resources used in this example.

Parameters

hadc	ADC handle pointer

Return values

None	

ADC1 GPIO Configuration PC0 ------> ADC1_IN1

ADC2 GPIO Configuration PC1 ------> ADC2_IN2

7.28.2.2 HAL_ADC_MspInit()

```
void HAL_ADC_MspInit (
             ADC_HandleTypeDef * hadc )
```

ADC MSP Initialization This function configures the hardware resources used in this example.

Parameters

hadc	ADC handle pointer

Return values

None	

ADC1 GPIO Configuration PC0 ------> ADC1_IN1

ADC2 GPIO Configuration PC1 ------> ADC2_IN2

7.28.2.3 HAL_MspInit()

```
void HAL_MspInit (
             void  )
```

Initializes the Global MSP.

7.28.2.4 HAL_TIM_Encoder_MspDeInit()

```
void HAL_TIM_Encoder_MspDeInit (
             TIM_HandleTypeDef * htim_encoder )
```

TIM_Encoder MSP De-Initialization This function freeze the hardware resources used in this example.

Parameters

htim_encoder	TIM_Encoder handle pointer

Return values

None	

TIM3 GPIO Configuration PA6 ------> TIM3_CH1 PA7 ------> TIM3_CH2

TIM4 GPIO Configuration PB6 ------> TIM4_CH1 PB7 ------> TIM4_CH2

7.28.2.5 HAL_TIM_Encoder_MspInit()

```
void HAL_TIM_Encoder_MspInit (
            TIM_HandleTypeDef * htim_encoder )
```

TIM_Encoder MSP Initialization This function configures the hardware resources used in this example.

Parameters

htim_encoder	TIM_Encoder handle pointer

Return values

None	

TIM3 GPIO Configuration PA6 ------> TIM3_CH1 PA7 ------> TIM3_CH2

TIM4 GPIO Configuration PB6 ------> TIM4_CH1 PB7 ------> TIM4_CH2

7.28.2.6 HAL_TIM_IC_MspDeInit()

```
void HAL_TIM_IC_MspDeInit (
            TIM_HandleTypeDef * htim_ic )
```

TIM_IC MSP De-Initialization This function freeze the hardware resources used in this example.

Parameters

htim↩ _ic	TIM_IC handle pointer

Return values

None	

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

7.28.2.7 HAL_TIM_IC_MspInit()

```
void HAL_TIM_IC_MspInit (
```

```
            TIM_HandleTypeDef * htim_ic )
```

TIM_IC MSP Initialization This function configures the hardware resources used in this example.

Parameters

htim_ic	TIM_IC handle pointer

Return values

None	

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

7.28.2.8 HAL_TIM_MspPostInit()

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

TIM2 GPIO Configuration PA0 ------> TIM2_CH1 PA1 ------> TIM2_CH2 PB10 ------> TIM2_CH3 PB11 ------> TIM2_CH4

7.28.2.9 HAL_TIM_PWM_MspDeInit()

```
void HAL_TIM_PWM_MspDeInit (
            TIM_HandleTypeDef * htim_pwm )
```

TIM_PWM MSP De-Initialization This function freeze the hardware resources used in this example.

Parameters

htim_pwm	TIM_PWM handle pointer

Return values

None	

7.28.2.10 HAL_TIM_PWM_MspInit()

```
void HAL_TIM_PWM_MspInit (
            TIM_HandleTypeDef * htim_pwm )
```

TIM_PWM MSP Initialization This function configures the hardware resources used in this example.

Parameters

htim_pwm	TIM_PWM handle pointer

Return values

None	

7.28.2.11 HAL_UART_MspDeInit()

```
void HAL_UART_MspDeInit (
             UART_HandleTypeDef * huart )
```

UART MSP De-Initialization This function freeze the hardware resources used in this example.

Parameters

huart	UART handle pointer

Return values

None	

USART2 GPIO Configuration PA2 ------> USART2_TX PA3 ------> USART2_RX

7.28.2.12 HAL_UART_MspInit()

```
void HAL_UART_MspInit (
             UART_HandleTypeDef * huart )
```

UART MSP Initialization This function configures the hardware resources used in this example.

Parameters

huart	UART handle pointer

Return values

None	

Initializes the peripherals clock

USART2 GPIO Configuration PA2 ------> USART2_TX PA3 ------> USART2_RX

7.28.3 Variable Documentation

7.28.3.1 HAL_RCC_ADC_CLK_ENABLED

```
uint32_t HAL_RCC_ADC_CLK_ENABLED =0  [static]
```

7.29 Core/Src/stm32l4xx_it.c File Reference

Interrupt Service Routines.

```
#include "main.h"
#include "stm32l4xx_it.h"
```

Functions

- void NMI_Handler (void)

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

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

  This function handles Memory management fault.
- void BusFault_Handler (void)

  This function handles Prefetch fault, memory access fault.
- void UsageFault_Handler (void)

  This function handles Undefined instruction or illegal state.
- void SVC_Handler (void)

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

  This function handles Debug monitor.
- void PendSV_Handler (void)

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

  This function handles System tick timer.
- void TIM1_CC_IRQHandler (void)

  This function handles TIM1 capture compare interrupt.

Variables

- TIM_HandleTypeDef htim1

7.29.1 Detailed Description

Interrupt Service Routines.

Attention

7.29.2 Function Documentation

7.29.2.1 BusFault_Handler()

```
void BusFault_Handler (
             void  )
```

This function handles Prefetch fault, memory access fault.

7.29.2.2 DebugMon_Handler()

```
void DebugMon_Handler (
             void  )
```

This function handles Debug monitor.

7.29.2.3 HardFault_Handler()

```
void HardFault_Handler (
             void  )
```

This function handles Hard fault interrupt.

7.29.2.4 MemManage_Handler()

```
void MemManage_Handler (
             void  )
```

This function handles Memory management fault.

7.29.2.5 NMI_Handler()

```
void NMI_Handler (
             void  )
```

This function handles Non maskable interrupt.

7.29.2.6 PendSV_Handler()

```
void PendSV_Handler (
             void  )
```

This function handles Pendable request for system service.

7.29.2.7 SVC_Handler()

```
void SVC_Handler (
            void  )
```

This function handles System service call via SWI instruction.

7.29.2.8 SysTick_Handler()

```
void SysTick_Handler (
            void  )
```

This function handles System tick timer.

7.29.2.9 TIM1_CC_IRQHandler()

```
void TIM1_CC_IRQHandler (
            void  )
```

This function handles TIM1 capture compare interrupt.

7.29.2.10 UsageFault_Handler()

```
void UsageFault_Handler (
            void  )
```

This function handles Undefined instruction or illegal state.

7.29.3 Variable Documentation

7.29.3.1 htim1

```
TIM_HandleTypeDef htim1  [extern]
```

7.30 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>
```

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

7.30.1 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

7.30.2 Function Documentation

7.30.2.1 __attribute__()

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

7.30.2.2 __io_getchar()

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

7.30.2.3 __io_putchar()

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

7.30.2.4 _close()

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

7.30.2.5 _execve()

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

7.30.2.6 _exit()

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

7.30.2.7 _fork()

```
int _fork (
            void  )
```

7.30.2.8 _fstat()

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

7.30.2.9 _getpid()

```
int _getpid (
            void  )
```

7.30.2.10 _isatty()

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

7.30.2.11 _kill()

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

7.30.2.12 _link()

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

7.30.2.13 _lseek()

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

7.30.2.14 _open()

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

7.30.2.15 _stat()

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

7.30.2.16 _times()

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

7.30.2.17 _unlink()

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

7.30.2.18 _wait()

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

7.30.2.19 initialise_monitor_handles()

```
void initialise_monitor_handles ( )
```

7.30.3 Variable Documentation

7.30.3.1 environ

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

7.31 Core/Src/sysmem.c File Reference

STM32CubeIDE System Memory calls file.

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

Functions

- void * _sbrk (ptrdiff_t incr)

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

Variables

- static uint8_t * __sbrk_heap_end = NULL

7.31.1 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

7.31.2 Function Documentation

7.31.2.1 _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

7.31.3 Variable Documentation

7.31.3.1 __sbrk_heap_end

```
uint8_t* __sbrk_heap_end = NULL  [static]
```

Pointer to the current high watermark of the heap usage

7.32 Core/Src/system_stm32l4xx.c File Reference

CMSIS Cortex-M4 Device Peripheral Access Layer System Source File.

```
#include "stm32l4xx.h"
```

Macros

- #define HSE_VALUE 8000000U
- #define MSI_VALUE 4000000U
- #define HSI_VALUE 16000000U

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 = 4000000U
- const uint8_t AHBPrescTable [16] = {0U, 0U, 0U, 0U, 0U, 0U, 0U, 0U, 1U, 2U, 3U, 4U, 6U, 7U, 8U, 9U}
- const uint8_t APBPrescTable [8] = {0U, 0U, 0U, 0U, 1U, 2U, 3U, 4U}
- const uint32_t MSIRangeTable [12]

7.32.1 Detailed Description

CMSIS Cortex-M4 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_stm32l4xx.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 MSI (4 MHz) is used as system clock source. Then SystemInit() function is called, in "startup_stm32l4xx.s" file, to configure the system clock before to branch to main program.

7.32.2 This file configures the system clock as follows:

7.32.2.1 System Clock source | MSI

7.32.2.2 SYSCLK(Hz) | 4000000

7.32.2.3 HCLK(Hz) | 4000000

7.32.2.4 AHB Prescaler | 1

7.32.2.5 APB1 Prescaler | 1

7.32.2.6 APB2 Prescaler | 1

7.32.2.7 PLL_M | 1

7.32.2.8 PLL_N | 8

7.32.2.9 PLL_P | 7

7.32.2.10 PLL_Q | 2

7.32.2.11 PLL_R | 2

7.32.2.12 PLLSAI1_P | NA

7.32.2.13 PLLSAI1_Q | NA

7.32.2.14 PLLSAI1_R | NA

7.32.2.15 PLLSAI2_P | NA

7.32.2.16 PLLSAI2_Q | NA

7.32.2.17 PLLSAI2_R | NA

Require 48MHz for USB OTG FS, | Disabled

7.32.2.18 SDIO and RNG clock |

==========

Attention

Index