STM32 Blue Pill Drivers

Generated on Fri Jun 30 2023 13:09:10 for STM32 Blue Pill Drivers by Doxygen 1.9.5

Fri Jun 30 2023 13:09:10

Chapter 1

STM32 Blue Pill Drivers

Drivers that could be used to interface and interact with STM32F103C8T6 Microcontroller

View the PDF documentation `here`

Chapter 2

Testing applications

These testing applications to test most of the drivers' functionalities and make sure they do what they intend to do.

2.1 ## Applications list

Below is the applications list and will follow this template of describing each application.

```
Name: <Application name>
Description: <Application description>
Activision Macro: <Application macro name>
```

Testing application's directory has the following structure:

```
- APPS_main.h
- APPS_main.c
- <Application name>/
    - TestApp_<Application name>_main.h
    - TestApp_<Application name>_main.c
```

2.2 ### List

```
Name: TestingGPIO
Description: This application tests the GPIO driver.
Activision Macro: TestingGPIO
```

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Module Documentation

6.1 Bit Manipulation Math Macros

This module contains the bit math manipulation macro-functions.

Collaboration diagram for Bit Manipulation Math Macros:

Macros

- #define SET_BIT(REG, BITNUM) (REG) |= (1 << (BITNUM))

  Set a certain bit's value.
- #define CLEAR_BIT(REG, BITNUM) (REG) &= ~(1 << (BITNUM))

  Clear a certain bit's value to.
- #define TOGGLE_BIT(REG, BITNUM) (REG) ^= (1 << (BITNUM))

  Toggle a bit to `0` if it's `1`, `1` otherwise.
- #define GET_BIT(REG, BITNUM) (((REG) >> (BITNUM)) & 1)

  Return the value of the bit whether it's `1` or `0`

6.1.1 Detailed Description

This module contains the bit math manipulation macro-functions.

6.1.2 Macro Definition Documentation

6.1.2.1 SET_BIT

```
#define SET_BIT(
                REG,
                BITNUM ) (REG) |= (1 << (BITNUM))
```

```
#include <LIB/LSTD_BITMATH.h>
```

Set a certain bit's value.

Parameters

in	*REG*	The register to set its bit
in	*BITNUM*	The bit number to set

6.1.2.2 CLEAR_BIT

```
#define CLEAR_BIT(
              REG,
              BITNUM ) (REG) &= ~(1 << (BITNUM))
```

`#include <LIB/LSTD_BITMATH.h>`

Clear a certain bit's value to.

Parameters

in	*REG*	The register to clear its bit
in	*BITNUM*	The bit number to clear

6.1.2.3 TOGGLE_BIT

```
#define TOGGLE_BIT(
              REG,
              BITNUM ) (REG) ^= (1 << (BITNUM))
```

`#include <LIB/LSTD_BITMATH.h>`

Toggle a bit to `0` if it's `1`, `1` otherwise.

Parameters

in	*REG*	The register to toggle its bit
in	*BITNUM*	The bit number to toggle

6.1.2.4 GET_BIT

```
#define GET_BIT(
              REG,
              BITNUM ) (((REG) >> (BITNUM)) & 1)
```

`#include <LIB/LSTD_BITMATH.h>`

Return the value of the bit whether it's `1` or `0`

Parameters

in	*REG*	The register to get its bit
in	*BITNUM*	The bit number to get

6.2 Compiler standard macros

This file contains the compiler standard macros.

Collaboration diagram for Compiler standard macros:

Macros

- #define CONST const

  Declare a standard constant variable with the specified type.
- #define STATIC static

  Declare a standard static variable/function.
- #define VOLATILE volatile

  Declare a standard volatile variable.
- #define P2VAR(ptrtype) ptrtype ∗

  Declare a pointer-to-variable with the specified type.
- #define P2CONST(ptrtype) CONST ptrtype ∗

  Declare a constant pointer-to-variable with the specified type.
- #define CONSTP2VAR(ptrtype) ptrtype ∗CONST

  Declare a pointer-to-variable constant with the specified type.
- #define CONSTP2CONST(ptrtype) CONST ptrtype ∗CONST

  Declare a constant pointer-to-variable constant with the specified type.
- #define P2FUNC(rettype, fctname) rettype(∗fctname)

  Declare a pointer-to-function with the specified return type.

6.2.1 Detailed Description

This file contains the compiler standard macros.

6.2.2 Macro Definition Documentation

6.2.2.1 CONST

```
#define CONST const
```

```
#include <LIB/LSTD_COMPILER.h>
```

Declare a standard constant variable with the specified type.

6.2.2.2 STATIC

```
#define STATIC static
```

```
#include <LIB/LSTD_COMPILER.h>
```

Declare a standard static variable/function.

6.2.2.3 VOLATILE

```
#define VOLATILE volatile
```

```
#include <LIB/LSTD_COMPILER.h>
```

Declare a standard volatile variable.

6.2.2.4 P2VAR

```
#define P2VAR(
              ptrtype ) ptrtype *
```

```
#include <LIB/LSTD_COMPILER.h>
```

Declare a pointer-to-variable with the specified type.

Parameters

in	*ptrtype*	The type of the pointer

6.2.2.5 P2CONST

```
#define P2CONST(
              ptrtype ) CONST ptrtype *
```

```
#include <LIB/LSTD_COMPILER.h>
```

Declare a constant pointer-to-variable with the specified type.

Parameters

in	*ptrtype*	The type of the pointer

6.2.2.6 CONSTP2VAR

```
#define CONSTP2VAR(
              ptrtype ) ptrtype *CONST
```

`#include <LIB/LSTD_COMPILER.h>`

Declare a pointer-to-variable constant with the specified type.

Parameters

in	*ptrtype*	The type of the pointer

6.2.2.7 CONSTP2CONST

```
#define CONSTP2CONST(
              ptrtype ) CONST ptrtype *CONST
```

`#include <LIB/LSTD_COMPILER.h>`

Declare a constant pointer-to-variable constant with the specified type.

Parameters

in	*ptrtype*	The type of the pointer

6.2.2.8 P2FUNC

```
#define P2FUNC(
              rettype,
              fctname ) rettype(*fctname)
```

`#include <LIB/LSTD_COMPILER.h>`

Declare a pointer-to-function with the specified return type.

Parameters

in	*rettype*	The return type of the function
in	*fctname*	The name of the function

6.3 Hardware registers macro-functions

These macro-functions help in mapping and accessing the hardware registers.

Collaboration diagram for Hardware registers macro-functions:

Macros

- #define REGISTER_ADDRESS(ADDRESS, OFFSET) ((ADDRESS) + (OFFSET))

  Placeholder for declaring a register address.
- #define REGISTER(REG_TYPE, ADDRESS) (∗(VOLATILE P2VAR(REG_TYPE))(ADDRESS))

  Map to a certain register by its address in the memory.
- #define REGISTER_U8(ADDRESS) REGISTER(t_u8, ADDRESS)

  Map to a certain register by its 8-bit address in the memory (used for 8-bit registers)
- #define REGISTER_U16(ADDRESS) REGISTER(t_u16, ADDRESS)

  Map to a certain register by its 16-bit address in the memory (used for 16-bit registers)
- #define REGISTER_U32(ADDRESS) REGISTER(t_u32, ADDRESS)

  Map to a certain register by its 32-bit address in the memory (used for 32-bit registers)

6.3.1 Detailed Description

These macro-functions help in mapping and accessing the hardware registers.

6.3.2 Macro Definition Documentation

6.3.2.1 REGISTER_ADDRESS

```
#define REGISTER_ADDRESS(
              ADDRESS,
              OFFSET ) ((ADDRESS) + (OFFSET))
```

```
#include <LIB/LSTD_HW_REGS.h>
```

Placeholder for declaring a register address.

Parameters

in	*ADDRESS*	The address of the register
in	*OFFSET*	The offset of the register

Returns

The final address of the register

6.3.2.2 REGISTER

```
#define REGISTER(
              REG_TYPE,
              ADDRESS ) (*(VOLATILE P2VAR(REG_TYPE))(ADDRESS))
```

`#include <LIB/LSTD_HW_REGS.h>`

Map to a certain register by its address in the memory.

Parameters

in	*ADDRESS*	The address of the register
in	*REG_TYPE*	The type of the register

Note

REG_TYPE can be a standard type (e.g. t_u8, t_u16, t_u32, ...) or a user-defined type

Returns

The value of the register

See also

Standard data types

6.3.2.3 REGISTER_U8

```
#define REGISTER_U8(
              ADDRESS ) REGISTER(t_u8, ADDRESS)
```

`#include <LIB/LSTD_HW_REGS.h>`

Map to a certain register by its 8-bit address in the memory (used for 8-bit registers)

Parameters

in	*ADDRESS*	The address of the register

Returns

The value of the register

6.3.2.4 REGISTER_U16

```
#define REGISTER_U16(
              ADDRESS ) REGISTER(t_u16, ADDRESS)
```

`#include <LIB/LSTD_HW_REGS.h>`

Map to a certain register by its 16-bit address in the memory (used for 16-bit registers)

Parameters

in	*ADDRESS*	The address of the register

Returns

The value of the register

6.3.2.5 REGISTER_U32

```
#define REGISTER_U32(
              ADDRESS ) REGISTER(t_u32, ADDRESS)
```

`#include <LIB/LSTD_HW_REGS.h>`

Map to a certain register by its 32-bit address in the memory (used for 32-bit registers)

Parameters

in	*ADDRESS*	The address of the register

Returns

The value of the register

6.4 Standard Library

This module contains the standard library-related macros, types, and functions.

Collaboration diagram for Standard Library:

Modules

- Bit Manipulation Math Macros

  This module contains the bit math manipulation macro-functions.
- Compiler standard macros

This file contains the compiler standard macros.

- Hardware registers macro-functions

  These macro-functions help in mapping and accessing the hardware registers.
- Standard data types

  This module contains the standard data types.
- Standard values

  This module contains the standard values.
- Shared library

  This library contains the shared macros and functions for all the standard types.

6.4.1 Detailed Description

This module contains the standard library-related macros, types, and functions.

6.5 Standard data types

This module contains the standard data types.

Collaboration diagram for Standard data types:

Typedefs

- typedef unsigned char t_bool

  Type definition for boolean.
- typedef unsigned char t_u8

  Type definition for 8-bit unsigned INT.
- typedef signed char t_s8

  Type definition for 8-bit signed INT.
- typedef unsigned char t_c8

  Type definition for 8-bit char.
- typedef unsigned short int t_u16

  Type definition for 16-bit unsigned int.
- typedef signed short int t_s16

  Type definition for 16-bit signed INT.
- typedef unsigned int t_u32

  Type definition for 32-bit unsigned int.
- typedef signed int t_s32

  Type definition for 32-bit signed INT.
- typedef unsigned long int t_u64

  Type definition for 64-bit unsigned int.
- typedef signed long int t_s64

  Type definition for 64-bit signed int.
- typedef float t_fl32

  Type definition for 32-bit float.
- typedef double t_fl64

  Type definition for 64-bit float.

6.5.1 Detailed Description

This module contains the standard data types.

6.5.2 Typedef Documentation

6.5.2.1 t_bool

`t_bool`

`#include <LIB/LSTD_TYPES.h>`

Type definition for boolean.

6.5.2.2 t_u8

`t_u8`

`#include <LIB/LSTD_TYPES.h>`

Type definition for 8-bit unsigned INT.

6.5.2.3 t_s8

`t_s8`

`#include <LIB/LSTD_TYPES.h>`

Type definition for 8-bit signed INT.

6.5.2.4 t_c8

`t_c8`

`#include <LIB/LSTD_TYPES.h>`

Type definition for 8-bit char.

6.5.2.5 t_u16

t_u16

`#include <LIB/LSTD_TYPES.h>`

Type definition for 16-bit unsigned int.

6.5.2.6 t_s16

t_s16

`#include <LIB/LSTD_TYPES.h>`

Type definition for 16-bit signed INT.

6.5.2.7 t_u32

t_u32

`#include <LIB/LSTD_TYPES.h>`

Type definition for 32-bit unsigned int.

6.5.2.8 t_s32

t_s32

`#include <LIB/LSTD_TYPES.h>`

Type definition for 32-bit signed INT.

6.5.2.9 t_u64

t_u64

`#include <LIB/LSTD_TYPES.h>`

Type definition for 64-bit unsigned int.

6.5.2.10 t_s64

`t_s64`

`#include <LIB/LSTD_TYPES.h>`

Type definition for 64-bit signed int.

6.5.2.11 t_fl32

`t_fl32`

`#include <LIB/LSTD_TYPES.h>`

Type definition for 32-bit float.

6.5.2.12 t_fl64

`t_fl64`

`#include <LIB/LSTD_TYPES.h>`

Type definition for 64-bit float.

6.6 Standard values

This module contains the standard values.

Collaboration diagram for Standard values:

Macros

- #define TRUE ((t_bool)1)

  Type definition for TRUE.
- #define FALSE ((t_bool)0)

  Type definition for FALSE.
- #define NULL ((P2VAR(void))0)

  Type definition for NULL.

6.6.1 Detailed Description

This module contains the standard values.

6.6.2 Macro Definition Documentation

6.6.2.1 TRUE

```
#define TRUE ((t_bool)1)
```

```
#include <LIB/LSTD_VALUES.h>
```

Type definition for TRUE.

6.6.2.2 FALSE

```
#define FALSE ((t_bool)0)
```

```
#include <LIB/LSTD_VALUES.h>
```

Type definition for FALSE.

6.6.2.3 NULL

```
#define NULL ((P2VAR(void))0)
```

```
#include <LIB/LSTD_VALUES.h>
```

Type definition for NULL.

6.7 GPIO Module

GPIO Module.

Collaboration diagram for GPIO Module:

Modules

- GPIO Registers

  GPIO Registers.
- GPIO Addresses

  GPIO Addresses.
- GPIO Pins Constants

  GPIO Pins Constants.

Macros

- #define DEFAULT_PIN_INPUT_TYPE (GPIO_Input_Type_Pull_Down)

  The default input type for the GPIO pins.
- #define DEFAULT_PIN_OUTPUT_TYPE (GPIO_Output_Type_Push_Pull)

  The default output type for the GPIO pins.

Enumerations

- enum t_GPIO_Ports {
  GPIO_Ports_A = 0 , GPIO_Ports_B , GPIO_Ports_C , GPIO_Ports_D ,
  GPIO_Ports_E , GPIO_Ports_F , GPIO_Ports_G }

  GPIO Ports.
- enum t_GPIO_Pins {
  GPIO_Pins_0 = 0 , GPIO_Pins_1 , GPIO_Pins_2 , GPIO_Pins_3 ,
  GPIO_Pins_4 , GPIO_Pins_5 , GPIO_Pins_6 , GPIO_Pins_7 ,
  GPIO_Pins_8 , GPIO_Pins_9 , GPIO_Pins_10 , GPIO_Pins_11 ,
  GPIO_Pins_12 , GPIO_Pins_13 , GPIO_Pins_14 , GPIO_Pins_15 }

  GPIO Pins.
- enum t_GPIO_Direction { GPIO_Direction_Input = 0 , GPIO_Direction_Output_10MHz , GPIO_Direction_Output_2MHz , GPIO_Direction_Output_50MHz }

  GPIO Direction.
- enum t_GPIO_Output_Type { GPIO_Output_Type_Push_Pull = 0 , GPIO_Output_Type_Open_Drain , GPIO_Output_Type_Alternate_Push_Pull , GPIO_Output_Type_Alternate_Open_Drain }

  GPIO Output Type.
- enum t_GPIO_Input_Type { GPIO_Input_Type_Analog = 0 , GPIO_Input_Type_Floating , GPIO_Input_Type_Pull_Down , GPIO_Input_Type_Pull_Up }

  GPIO Input Type.
- enum t_GPIO_Value { GPIO_Value_Low = 0 , GPIO_Value_High }

  GPIO Pin Value.

Functions

- void GPIO_vSetPinDirection (t_GPIO_Ports tPort, t_GPIO_Pins tPin, t_GPIO_Direction tDirection)

  This function is used to set the direction of a GPIO pin.
- void GPIO_vSetPinInputType (t_GPIO_Ports tPort, t_GPIO_Pins tPin, t_GPIO_Input_Type tInputType)

  This function is used to set the input type of a GPIO pin.
- void GPIO_vSetPinOutputType (t_GPIO_Ports tPort, t_GPIO_Pins tPin, t_GPIO_Output_Type tOutputType)

  This function is used to set the output type of a GPIO pin.
- void GPIO_vSetPinValue (t_GPIO_Ports tPort, t_GPIO_Pins tPin, t_GPIO_Value tValue)

  This function is used to set the value of a GPIO pin.
- t_GPIO_Value GPIO_tGetPinValue (t_GPIO_Ports tPort, t_GPIO_Pins tPin)

  This function is used to get the value of a GPIO pin.

6.7.1 Detailed Description

GPIO Module.

This module contains all the APIs related to the GPIO module

6.7.2 Macro Definition Documentation

6.7.2.1 DEFAULT_PIN_INPUT_TYPE

```
#define DEFAULT_PIN_INPUT_TYPE (GPIO_Input_Type_Pull_Down)
```

`#include <MCAL/GPIO/GPIO_config.h>`

The default input type for the GPIO pins.

This macro defines the default input type for the GPIO pins

6.7.2.2 DEFAULT_PIN_OUTPUT_TYPE

```
#define DEFAULT_PIN_OUTPUT_TYPE (GPIO_Output_Type_Push_Pull)
```

`#include <MCAL/GPIO/GPIO_config.h>`

The default output type for the GPIO pins.

This macro defines the default output type for the GPIO pins

6.7.3 Enumeration Type Documentation

6.7.3.1 t_GPIO_Ports

```
enum t_GPIO_Ports
```

`#include <MCAL/GPIO/GPIO_interface.h>`

GPIO Ports.

This type is used to select a GPIO port

Enumerator

GPIO_Ports_A	GPIO Port A.
GPIO_Ports_B	GPIO Port B.
GPIO_Ports_C	GPIO Port C.
GPIO_Ports_D	GPIO Port D.
GPIO_Ports_E	GPIO Port E.
GPIO_Ports_F	GPIO Port F.
GPIO_Ports_G	GPIO Port G.

```
{
    GPIO_Ports_A = 0,
    GPIO_Ports_B,
    GPIO_Ports_C,
    GPIO_Ports_D,
    GPIO_Ports_E,
    GPIO_Ports_F,
    GPIO_Ports_G
} t_GPIO_Ports;
```

6.7.3.2 t_GPIO_Pins

```
enum t_GPIO_Pins
```

```
#include <MCAL/GPIO/GPIO_interface.h>
```

GPIO Pins.

This type is used to select a GPIO pin

Enumerator

GPIO_Pins_0	GPIO Pin 0.
GPIO_Pins_1	GPIO Pin 1.
GPIO_Pins_2	GPIO Pin 2.
GPIO_Pins_3	GPIO Pin 3.
GPIO_Pins_4	GPIO Pin 4.
GPIO_Pins_5	GPIO Pin 5.
GPIO_Pins_6	GPIO Pin 6.
GPIO_Pins_7	GPIO Pin 7.
GPIO_Pins_8	GPIO Pin 8.
GPIO_Pins_9	GPIO Pin 9.
GPIO_Pins_10	GPIO Pin 10.
GPIO_Pins_11	GPIO Pin 11.
GPIO_Pins_12	GPIO Pin 12.
GPIO_Pins_13	GPIO Pin 13.
GPIO_Pins_14	GPIO Pin 14.
GPIO_Pins_15	GPIO Pin 15.

```
{
    GPIO_Pins_0 = 0,
    GPIO_Pins_1,
    GPIO_Pins_2,
    GPIO_Pins_3,
    GPIO_Pins_4,
    GPIO_Pins_5,
    GPIO_Pins_6,
    GPIO_Pins_7,
    GPIO_Pins_8,
    GPIO_Pins_9,
    GPIO_Pins_10,
    GPIO_Pins_11,
    GPIO_Pins_12,
    GPIO_Pins_13,
    GPIO_Pins_14,
    GPIO_Pins_15
} t_GPIO_Pins;
```

6.7.3.3 t_GPIO_Direction

```
enum t_GPIO_Direction
```

```
#include <MCAL/GPIO/GPIO_interface.h>
```

GPIO Direction.

This type is used to select a GPIO pin direction

Enumerator

GPIO_Direction_Input	GPIO Pin Input.
GPIO_Direction_Output_10MHz	GPIO Pin Output with max speed of 10 MHz.
GPIO_Direction_Output_2MHz	GPIO Pin Output with max speed of 2 MHz.
GPIO_Direction_Output_50MHz	GPIO Pin Output with max speed of 50 MHz.

```
{
    GPIO_Direction_Input = 0,
    GPIO_Direction_Output_10MHz,
    GPIO_Direction_Output_2MHz,
    GPIO_Direction_Output_50MHz
} t_GPIO_Direction;
```

6.7.3.4 t_GPIO_Output_Type

```
enum t_GPIO_Output_Type
```

```
#include <MCAL/GPIO/GPIO_interface.h>
```

GPIO Output Type.

This type is used to select a GPIO pin output type

Enumerator

GPIO_Output_Type_Push_Pull	GPIO Pin Output Type: push-pull.
GPIO_Output_Type_Open_Drain	GPIO Pin Output Type: open-drain.
GPIO_Output_Type_Alternate_Push_Pull	GPIO Pin Output Type: alternate push-pull.
GPIO_Output_Type_Alternate_Open_Drain	GPIO Pin Output Type: alternate open-drain.

```
{
    GPIO_Output_Type_Push_Pull = 0,
    GPIO_Output_Type_Open_Drain,
    GPIO_Output_Type_Alternate_Push_Pull,
    GPIO_Output_Type_Alternate_Open_Drain
} t_GPIO_Output_Type;
```

6.7.3.5 t_GPIO_Input_Type

```
enum t_GPIO_Input_Type
```

```
#include <MCAL/GPIO/GPIO_interface.h>
```

GPIO Input Type.

This type is used to select a GPIO pin input type

Enumerator

GPIO_Input_Type_Analog	GPIO Pin Input Type: Analog.
GPIO_Input_Type_Floating	GPIO Pin Input Type: Floating.
GPIO_Input_Type_Pull_Down	GPIO Pin Input Type: Pull-Down.
GPIO_Input_Type_Pull_Up	GPIO Pin Input Type: Pull-Up.

```
{
    GPIO_Input_Type_Analog = 0,
    GPIO_Input_Type_Floating,
    GPIO_Input_Type_Pull_Down,
    GPIO_Input_Type_Pull_Up
} t_GPIO_Input_Type;
```

6.7.3.6 t_GPIO_Value

```
enum t_GPIO_Value
```

```
#include <MCAL/GPIO/GPIO_interface.h>
```

GPIO Pin Value.

This type is used to select a GPIO pin value

Enumerator

GPIO_Value_Low	GPIO Pin Value: LOW (0)
GPIO_Value_High	GPIO Pin Value: HIGH (1)

```
{
    GPIO_Value_Low = 0,
    GPIO_Value_High
} t_GPIO_Value;
```

6.7.4 Function Documentation

6.7.4.1 GPIO_vSetPinDirection()

```
void GPIO_vSetPinDirection (
            t_GPIO_Ports tPort,
            t_GPIO_Pins tPin,
            t_GPIO_Direction tDirection )
```

```
#include <MCAL/GPIO/GPIO_interface.h>
```

This function is used to set the direction of a GPIO pin.

This function is used to set the direction of a GPIO pin

Parameters

in	*tPort*	The GPIO port
in	*tPin*	The GPIO pin
in	*tDirection*	The GPIO pin direction

See also

t_GPIO_Ports t_GPIO_Pins t_GPIO_Direction

```
{
    /* Get the pin span (the number of bits to shift to reach the target pin mode and configuration bits) */
    t_u8 u8PinSpan = GPIO_vGetPinSpan(tPin);
    /* Store the base address of the GPIO port */
    P2VAR(t_GPIOx_RegisterMap)
    pu32PortBaseAddress = (P2VAR(t_GPIOx_RegisterMap))NULL;
    /* Store the pin mode and configuration bits of the pin */
    P2VAR(t_u32)
    pu32TargetPinModeConfig = (P2VAR(t_u32))NULL;

    /* Get the base address of the GPIO port */
    GPIO_vGetPortAddress(tPort, &pu32PortBaseAddress);
    /* Get the pin mode and configuration bits of the pin */
    pu32TargetPinModeConfig = IS_PIN_IN_LOW_REGISTER(tPin) ? &pu32PortBaseAddress->CRL : &pu32PortBaseAddress->CRH;
    /* Set the mode and configuration bits of the pin */
    *pu32TargetPinModeConfig = (*pu32TargetPinModeConfig & PIN_RESET_MASK(u8PinSpan)) | (tDirection « u8PinSpan);

    /* Set the pull-up/pull-down resistor */
    if (tDirection == GPIO_Direction_Input)
    {
        GPIO_vSetPinInputType(tPort, tPin, DEFAULT_PIN_INPUT_TYPE);
    }
    else
    {
        GPIO_vSetPinOutputType(tPort, tPin, DEFAULT_PIN_OUTPUT_TYPE);
    }
}
```

References DEFAULT_PIN_INPUT_TYPE, DEFAULT_PIN_OUTPUT_TYPE, GPIO_Direction_Input, GPIO_vGetPinSpan(), GPIO_vGetPortAddress(), GPIO_vSetPinInputType(), GPIO_vSetPinOutputType(), IS_PIN_IN_LOW_REGISTER, NULL, P2VAR, and PIN_RESET_MASK.

Referenced by vTestApp_TestingGPIO_main().

Here is the call graph for this function: Here is the caller graph for this function:

6.7.4.2 GPIO_vSetPinInputType()

```
void GPIO_vSetPinInputType (
            t_GPIO_Ports tPort,
            t_GPIO_Pins tPin,
            t_GPIO_Input_Type tInputType )
```

`#include <MCAL/GPIO/GPIO_interface.h>`

This function is used to set the input type of a GPIO pin.

This function is used to set the input type of a GPIO pin

Parameters

in	*tPort*	The GPIO port
in	*tPin*	The GPIO pin
in	*tInputType*	The GPIO pin input type

See also

t_GPIO_Ports t_GPIO_Pins t_GPIO_Input_Type

```
{
    /* Get the pin span (the number of bits to shift to reach the target pin mode and configuration
   bits) */
    t_u8 u8PinSpan = GPIO_vGetPinSpan(tPin);
    /* Check if the input type is pull-up or pull-down */
    t_GPIO_Input_Type u8PinInputTypeModeConfigurations = (tInputType == GPIO_Input_Type_Pull_Up) ?
   GPIO_Input_Type_Pull_Down : tInputType;
    /* Store the base address of the GPIO port */
    P2VAR(t_GPIOx_RegisterMap)
    pu32PortBaseAddress = (P2VAR(t_GPIOx_RegisterMap))NULL;
    /* Store the pin mode and configuration bits of the pin */
    P2VAR(t_u32)
    pu32TargetPinModeConfig = (P2VAR(t_u32))NULL;

    /* Get the base address of the GPIO port */
    GPIO_vGetPortAddress(tPort, &pu32PortBaseAddress);
    /* Get the pin mode and configuration bits of the pin */
    pu32TargetPinModeConfig = IS_PIN_IN_LOW_REGISTER(tPin) ? &pu32PortBaseAddress->CRL :
   &pu32PortBaseAddress->CRH;
    /* Set the mode and configuration bits of the pin */
    *pu32TargetPinModeConfig = (*pu32TargetPinModeConfig & PIN_RESET_CONFIGURATIONS_MASK(u8PinSpan)) |
   (u8PinInputTypeModeConfigurations « (u8PinSpan + PIN_CONFIGURATION_BITS_SHIFT_VALUE));
    /* Set the pull-up/pull-down resistor */
    GPIO_vSetPinInputTypePullUpDown(pu32PortBaseAddress, tPin, tInputType);
}
```

References GPIO_Input_Type_Pull_Down, GPIO_Input_Type_Pull_Up, GPIO_vGetPinSpan(), GPIO_vGetPortAddress(), GPIO_vSetPinInputTypePullUpDown(), IS_PIN_IN_LOW_REGISTER, NULL, P2VAR, PIN_CONFIGURATION_BITS_SHIFT_VALUE and PIN_RESET_CONFIGURATIONS_MASK.

Referenced by GPIO_vSetPinDirection(), and vTestApp_TestingGPIO_main().

Here is the call graph for this function: Here is the caller graph for this function:

6.7.4.3 GPIO_vSetPinOutputType()

```
void GPIO_vSetPinOutputType (
            t_GPIO_Ports tPort,
            t_GPIO_Pins tPin,
            t_GPIO_Output_Type tOutputType )
```

`#include <MCAL/GPIO/GPIO_interface.h>`

This function is used to set the output type of a GPIO pin.

This function is used to set the output type of a GPIO pin

Parameters

in	*tPort*	The GPIO port
in	*tPin*	The GPIO pin
in	*tOutputType*	The GPIO pin output type

See also

t_GPIO_Ports t_GPIO_Pins t_GPIO_Output_Type

```
{
    /* Get the pin span (the number of bits to shift to reach the target pin mode and configuration
   bits) */
    t_u8 u8PinSpan = GPIO_vGetPinSpan(tPin);
    /* Store the base address of the GPIO port */
    P2VAR(t_GPIOx_RegisterMap)
    pu32PortBaseAddress = (P2VAR(t_GPIOx_RegisterMap))NULL;
    /* Store the pin mode and configuration bits of the pin */
    P2VAR(t_u32)
    pu32TargetPinModeConfig = (P2VAR(t_u32))NULL;

    /* Get the base address of the GPIO port */
    GPIO_vGetPortAddress(tPort, &pu32PortBaseAddress);
    /* Get the pin mode and configuration bits of the pin */
    pu32TargetPinModeConfig = IS_PIN_IN_LOW_REGISTER(tPin) ? &pu32PortBaseAddress->CRL :
   &pu32PortBaseAddress->CRH;
    /* Set the mode and configuration bits of the pin */
    *pu32TargetPinModeConfig = (*pu32TargetPinModeConfig & PIN_RESET_CONFIGURATIONS_MASK(u8PinSpan)) |
   (t_u32)(tOutputType « (u8PinSpan + PIN_CONFIGURATION_BITS_SHIFT_VALUE));
}
```

References GPIO_vGetPinSpan(), GPIO_vGetPortAddress(), IS_PIN_IN_LOW_REGISTER, NULL, P2VAR, PIN_CONFIGURATION_BITS_SHIFT_VALUE, and PIN_RESET_CONFIGURATIONS_MASK.

Referenced by GPIO_vSetPinDirection().

Here is the call graph for this function: Here is the caller graph for this function:

6.7.4.4 GPIO_vSetPinValue()

```
void GPIO_vSetPinValue (
            t_GPIO_Ports tPort,
            t_GPIO_Pins tPin,
            t_GPIO_Value tValue )
```

```
#include <MCAL/GPIO/GPIO_interface.h>
```

This function is used to set the value of a GPIO pin.

This function is used to set the value of a GPIO pin

Parameters

in	*tPort*	The GPIO port
in	*tPin*	The GPIO pin
in	*tValue*	The GPIO pin value

See also

t_GPIO_Ports t_GPIO_Pins t_GPIO_Value

```
{
    /* Get the pin location (the number of bits to shift to reach the target pin) */
    t_u32 u32PinLocation = tPin;
    /* Store the base address of the GPIO port */
    P2VAR(t_GPIOx_RegisterMap)
    pu32PortBaseAddress = (P2VAR(t_GPIOx_RegisterMap))NULL;

    /* Get the base address of the GPIO port */
    GPIO_vGetPortAddress(tPort, &pu32PortBaseAddress);

    if (tValue == GPIO_Value_Low)
    {
        u32PinLocation += PIN_RESET_SHIFT_VALUE;
    }
    else
    {
        /* Do nothing */
    }

    /* Set the value of the pin */
    pu32PortBaseAddress->BSRR = (t_u32)(TRUE « u32PinLocation);
}
```

References GPIO_Value_Low, GPIO_vGetPortAddress(), NULL, P2VAR, PIN_RESET_SHIFT_VALUE, and TRUE.

Referenced by vTestApp_TestingGPIO_main().

Here is the call graph for this function: Here is the caller graph for this function:

6.7.4.5 GPIO_tGetPinValue()

```
t_GPIO_Value GPIO_tGetPinValue (
            t_GPIO_Ports tPort,
            t_GPIO_Pins tPin )
```

`#include <MCAL/GPIO/GPIO_interface.h>`

This function is used to get the value of a GPIO pin.

This function is used to get the value of a GPIO pin

Parameters

in	*tPort*	The GPIO port
in	*tPin*	The GPIO pin

Returns

The GPIO pin value

See also

t_GPIO_Ports t_GPIO_Pins t_GPIO_Value

```
{
    /* Store the base address of the GPIO port */
    P2VAR(t_GPIOx_RegisterMap)
    pu32PortBaseAddress = (P2VAR(t_GPIOx_RegisterMap))NULL;

    /* Get the base address of the GPIO port */
    GPIO_vGetPortAddress(tPort, &pu32PortBaseAddress);

    /* Get the value of the pin */
    return (t_GPIO_Value)((pu32PortBaseAddress->IDR » tPin) & TRUE);
}
```

References GPIO_vGetPortAddress(), NULL, P2VAR, and TRUE.

Referenced by vTestApp_TestingGPIO_main().

Here is the call graph for this function: Here is the caller graph for this function:

6.8 GPIO Registers

GPIO Registers.

Collaboration diagram for GPIO Registers:

Data Structures

- struct t_GPIOx_RegisterMap
  GPIO Register Map.

6.8.1 Detailed Description

GPIO Registers.

6.9 GPIO Addresses

GPIO Addresses.

Collaboration diagram for GPIO Addresses:

Macros

- #define BASE_ADDRESS_PORT_A REGISTER_ADDRESS(0x40010800, 0)
  Base Address of Port A.
- #define GPIO_A REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_A)
  GPIO Port A.
- #define BASE_ADDRESS_PORT_B REGISTER_ADDRESS(0x40010C00, 0)
  Base Address of Port B.
- #define GPIO_B REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_B)
  GPIO Port B.
- #define BASE_ADDRESS_PORT_C REGISTER_ADDRESS(0x40011000, 0)
  Base Address of Port C.
- #define GPIO_C REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_C)
  GPIO Port C.
- #define BASE_ADDRESS_PORT_D REGISTER_ADDRESS(0x40011400, 0)
  Base Address of Port D.
- #define GPIO_D REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_D)
  GPIO Port D.
- #define BASE_ADDRESS_PORT_E REGISTER_ADDRESS(0x40011800, 0)
  Base Address of Port E.
- #define GPIO_E REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_E)
  GPIO Port E.
- #define BASE_ADDRESS_PORT_F REGISTER_ADDRESS(0x40011C00, 0)
  Base Address of Port F.
- #define GPIO_F REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_F)
  GPIO Port F.
- #define BASE_ADDRESS_PORT_G REGISTER_ADDRESS(0x40012000, 0)
  Base Address of Port G.
- #define GPIO_G REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_G)
  GPIO Port G.

6.9.1 Detailed Description

GPIO Addresses.

This module contains the addresses of the GPIO registers

6.9.2 Macro Definition Documentation

6.9.2.1 BASE_ADDRESS_PORT_A

```
#define BASE_ADDRESS_PORT_A REGISTER_ADDRESS(0x40010800, 0)
```

```
#include <MCAL/GPIO/GPIO_private.h>
```

Base Address of Port A.

6.9.2.2 GPIO_A

```
#define GPIO_A REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_A)
```

```
#include <MCAL/GPIO/GPIO_private.h>
```

GPIO Port A.

6.9.2.3 BASE_ADDRESS_PORT_B

```
#define BASE_ADDRESS_PORT_B REGISTER_ADDRESS(0x40010C00, 0)
```

```
#include <MCAL/GPIO/GPIO_private.h>
```

Base Address of Port B.

6.9.2.4 GPIO_B

```
#define GPIO_B REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_B)
```

```
#include <MCAL/GPIO/GPIO_private.h>
```

GPIO Port B.

6.9.2.5 BASE_ADDRESS_PORT_C

```
#define BASE_ADDRESS_PORT_C REGISTER_ADDRESS(0x40011000, 0)
```

```
#include <MCAL/GPIO/GPIO_private.h>
```

Base Address of Port C.

6.9.2.6 GPIO_C

```
#define GPIO_C REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_C)
```

```
#include <MCAL/GPIO/GPIO_private.h>
```

GPIO Port C.

6.9.2.7 BASE_ADDRESS_PORT_D

```
#define BASE_ADDRESS_PORT_D REGISTER_ADDRESS(0x40011400, 0)
```

```
#include <MCAL/GPIO/GPIO_private.h>
```

Base Address of Port D.

6.9.2.8 GPIO_D

```
#define GPIO_D REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_D)
```

```
#include <MCAL/GPIO/GPIO_private.h>
```

GPIO Port D.

6.9.2.9 BASE_ADDRESS_PORT_E

```
#define BASE_ADDRESS_PORT_E REGISTER_ADDRESS(0x40011800, 0)
```

```
#include <MCAL/GPIO/GPIO_private.h>
```

Base Address of Port E.

6.9.2.10 GPIO_E

```
#define GPIO_E REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_E)
```

`#include <MCAL/GPIO/GPIO_private.h>`

GPIO Port E.

6.9.2.11 BASE_ADDRESS_PORT_F

```
#define BASE_ADDRESS_PORT_F REGISTER_ADDRESS(0x40011C00, 0)
```

`#include <MCAL/GPIO/GPIO_private.h>`

Base Address of Port F.

6.9.2.12 GPIO_F

```
#define GPIO_F REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_F)
```

`#include <MCAL/GPIO/GPIO_private.h>`

GPIO Port F.

6.9.2.13 BASE_ADDRESS_PORT_G

```
#define BASE_ADDRESS_PORT_G REGISTER_ADDRESS(0x40012000, 0)
```

`#include <MCAL/GPIO/GPIO_private.h>`

Base Address of Port G.

6.9.2.14 GPIO_G

```
#define GPIO_G REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_G)
```

`#include <MCAL/GPIO/GPIO_private.h>`

GPIO Port G.

6.10 GPIO Pins Constants

GPIO Pins Constants.

Collaboration diagram for GPIO Pins Constants:

Macros

- #define PIN_SHIFT_VALUE (4)

  Pin Shift Value.
- #define PIN_RESET_MASK_VALUE ((t_u32)0x0000000FU)

  Pin Reset Mask Value.
- #define PIN_RESET_CONFIGURATION_MASK_VALUE ((t_u32)0x0000000CU)

  Pin Reset Configuration Mask Value.
- #define PIN_RESET_ODR_MASK_VALUE ((t_u32)0x00000001U)

  Pin Reset ODR Mask Value.
- #define PIN_CONFIGURATION_BITS_SHIFT_VALUE (2)

  Pin Mode Bits Shift Value.
- #define PIN_RESET_SHIFT_VALUE (16)

  Pin Reset Shift Value.
- #define PIN_RESET_MASK(GPIO_PIN_SPAN) ∼(PIN_RESET_MASK_VALUE << GPIO_PIN_SPAN)

  Pin Reset Mask.
- #define PIN_RESET_CONFIGURATIONS_MASK(GPIO_PIN_SPAN) ∼(PIN_RESET_CONFIGURATION_MASK_VALUE << GPIO_PIN_SPAN)

  Pin Reset Configurations Mask.
- #define PIN_RESET_ODR_MASK(GPIO_PIN_SPAN) ∼(PIN_RESET_ODR_MASK_VALUE << GPIO_↩
PIN_SPAN)

  Pin Reset ODR Mask.

6.10.1 Detailed Description

GPIO Pins Constants.

This module contains a set of constants used in configuring the GPIO pins in the configuration register (CRL or CRH)

6.10.2 Macro Definition Documentation

6.10.2.1 PIN_SHIFT_VALUE

```
#define PIN_SHIFT_VALUE (4)
```

```
#include <MCAL/GPIO/GPIO_private.h>
```

Pin Shift Value.

This value is used to shift to the start of a specific pin in the configuration register (CRL or CRH)

6.10.2.2 PIN_RESET_MASK_VALUE

```
#define PIN_RESET_MASK_VALUE ((t_u32)0x0000000FU)
```

`#include <MCAL/GPIO/GPIO_private.h>`

Pin Reset Mask Value.

This value is used to reset the mode and configuration bits of a certain pin

6.10.2.3 PIN_RESET_CONFIGURATION_MASK_VALUE

```
#define PIN_RESET_CONFIGURATION_MASK_VALUE ((t_u32)0x0000000CU)
```

`#include <MCAL/GPIO/GPIO_private.h>`

Pin Reset Configuration Mask Value.

This value is used to reset the configuration bits of a certain pin

6.10.2.4 PIN_RESET_ODR_MASK_VALUE

```
#define PIN_RESET_ODR_MASK_VALUE ((t_u32)0x00000001U)
```

`#include <MCAL/GPIO/GPIO_private.h>`

Pin Reset ODR Mask Value.

This value is used to reset the ODR bit of a certain pin (in the BSRR register)

6.10.2.5 PIN_CONFIGURATION_BITS_SHIFT_VALUE

```
#define PIN_CONFIGURATION_BITS_SHIFT_VALUE (2)
```

`#include <MCAL/GPIO/GPIO_private.h>`

Pin Mode Bits Shift Value.

This value is used to shift to the configuration bits of a certain pin

6.10.2.6 PIN_RESET_SHIFT_VALUE

```
#define PIN_RESET_SHIFT_VALUE (16)
```

`#include <MCAL/GPIO/GPIO_private.h>`

Pin Reset Shift Value.

This value is used to shift to the start of the position of the first pin bit in the BSRR register

6.10.2.7 PIN_RESET_MASK

```
#define PIN_RESET_MASK(
              GPIO_PIN_SPAN ) ~(PIN_RESET_MASK_VALUE << GPIO_PIN_SPAN)
```

`#include <MCAL/GPIO/GPIO_private.h>`

Pin Reset Mask.

This mask is used to reset the mode and configuration bits of a certain pin

Parameters

in	*GPIO_PIN_SPAN*	The span of the pin in the configuration register (CRL or CRH)

6.10.2.8 PIN_RESET_CONFIGURATIONS_MASK

```
#define PIN_RESET_CONFIGURATIONS_MASK(
              GPIO_PIN_SPAN ) ~(PIN_RESET_CONFIGURATION_MASK_VALUE << GPIO_PIN_SPAN)
```

`#include <MCAL/GPIO/GPIO_private.h>`

Pin Reset Configurations Mask.

This mask is used to reset the configuration bits of a certain pin

Parameters

in	*GPIO_PIN_SPAN*	The span of the pin in the configuration register (CRL or CRH)

6.10.2.9 PIN_RESET_ODR_MASK

```
#define PIN_RESET_ODR_MASK(
              GPIO_PIN_SPAN ) ~(PIN_RESET_ODR_MASK_VALUE << GPIO_PIN_SPAN)
```

`#include <MCAL/GPIO/GPIO_private.h>`

Pin Reset ODR Mask.

This mask is used to reset the ODR bit of a certain pin (in the BSRR register)

Parameters

in	*GPIO_PIN_SPAN*	The span of the pin in the configuration register (CRL or CRH)

6.11 MCAL Peripherals

MCAL Peripheral Drivers.

Collaboration diagram for MCAL Peripherals:

Modules

- GPIO Module

    GPIO Module.
- RCC Module

    RCC Module.

6.11.1 Detailed Description

MCAL Peripheral Drivers.

6.12 RCC Configuration

This group contains the configuration parameters of the RCC module.

Collaboration diagram for RCC Configuration:

Macros

- #define RCC_PLL_SRC (RCC_PLLSource_HSE)

    This macro defines the PLL source.
- #define RCC_PLL_MUL (RCC_PLLMulFactors_9)

    This macro defines the PLL multiplication factor.
- #define RCC_PLL_HSE_DIVIDE_BY_2 (TRUE)

    This macro defines the PLL entry HSE divider.
- #define RCC_SYSTEM_CLOCK_SOURCE (RCC_SystemClock_PLL)

    This macro defines the system clock source.
- #define RCC_MCO_SOURCE (RCC_MCOSources_NoClock)

    This macro defines the MCO source.
- #define RCC_ADC_PRESCALER (RCC_ADCPrescaler_DividedBy4)

    This macro defines the ADC clock prescaler.
- #define RCC_AHB_PRESCALER (RCC_AHBPrescaler_NotDivided)

    This macro defines the AHB clock prescaler.
- #define RCC_APB1_PRESCALER (RCC_APBPrescaler_NotDivided)

    This macro defines the APB1 clock prescaler.
- #define RCC_APB2_PRESCALER (RCC_APBPrescaler_NotDivided)

    This macro defines the APB2 clock prescaler.
- #define RCC_USB_PRESCALER (RCC_USBPrescaler_1)

    This macro defines the USB clock prescaler.
- #define RCC_ENABLE_CSS (FALSE)

    This macro defines the clock security system state.

6.12.1 Detailed Description

This group contains the configuration parameters of the RCC module.

6.12.2 Macro Definition Documentation

6.12.2.1 RCC_PLL_SRC

```
#define RCC_PLL_SRC (RCC_PLLSource_HSE)
```

```
#include <MCAL/RCC/RCC_config.h>
```

This macro defines the PLL source.

This macro defines the PLL source

See also

RCC_PLLSource

6.12.2.2 RCC_PLL_MUL

```
#define RCC_PLL_MUL (RCC_PLLMulFactors_9)
```

```
#include <MCAL/RCC/RCC_config.h>
```

This macro defines the PLL multiplication factor.

This macro defines the PLL multiplication factor

See also

RCC_PLLMulFactors

6.12.2.3 RCC_PLL_HSE_DIVIDE_BY_2

```
#define RCC_PLL_HSE_DIVIDE_BY_2 (TRUE)
```

```
#include <MCAL/RCC/RCC_config.h>
```

This macro defines the PLL entry HSE divider.

This macro defines the PLL entry HSE divider Options:

- FALSE: HSE clock is not divided
- TRUE: HSE clock divided by 2

6.12.2.4 RCC_SYSTEM_CLOCK_SOURCE

```
#define RCC_SYSTEM_CLOCK_SOURCE (RCC_SystemClock_PLL)
```

```
#include <MCAL/RCC/RCC_config.h>
```

This macro defines the system clock source.

This macro defines the system clock source

See also

RCC_SystemClock

6.12.2.5 RCC_MCO_SOURCE

```
#define RCC_MCO_SOURCE (RCC_MCOSources_NoClock)
```

```
#include <MCAL/RCC/RCC_config.h>
```

This macro defines the MCO source.

This macro defines the MCO source

See also

RCC_MCOSources

6.12.2.6 RCC_ADC_PRESCALER

```
#define RCC_ADC_PRESCALER (RCC_ADCPrescaler_DividedBy4)
```

```
#include <MCAL/RCC/RCC_config.h>
```

This macro defines the ADC clock prescaler.

This macro defines the ADC clock prescaler

See also

RCC_ADCPrescaler

6.12.2.7 RCC_AHB_PRESCALER

```
#define RCC_AHB_PRESCALER (RCC_AHBPrescaler_NotDivided)
```

`#include <MCAL/RCC/RCC_config.h>`

This macro defines the AHB clock prescaler.

This macro defines the AHB clock prescaler

See also

RCC_AHBPrescaler

6.12.2.8 RCC_APB1_PRESCALER

```
#define RCC_APB1_PRESCALER (RCC_APBPrescaler_NotDivided)
```

`#include <MCAL/RCC/RCC_config.h>`

This macro defines the APB1 clock prescaler.

This macro defines the APB1 clock prescaler

See also

RCC_APBPrescaler

6.12.2.9 RCC_APB2_PRESCALER

```
#define RCC_APB2_PRESCALER (RCC_APBPrescaler_NotDivided)
```

`#include <MCAL/RCC/RCC_config.h>`

This macro defines the APB2 clock prescaler.

This macro defines the APB2 clock prescaler

See also

RCC_APBPrescaler

6.12.2.10 RCC_USB_PRESCALER

```
#define RCC_USB_PRESCALER (RCC_USBPrescaler_1)
```

```
#include <MCAL/RCC/RCC_config.h>
```

This macro defines the USB clock prescaler.

This macro defines the USB clock prescaler

See also

RCC_USBPrescaler

6.12.2.11 RCC_ENABLE_CSS

```
#define RCC_ENABLE_CSS (FALSE)
```

```
#include <MCAL/RCC/RCC_config.h>
```

This macro defines the clock security system state.

This macro defines the clock security system state Options:

- FALSE: CSS disabled
- TRUE: CSS enabled

6.13 RCC Interface Options

This group contains the options of the RCC module interface.

Collaboration diagram for RCC Interface Options:

Enumerations

- enum RCC_PLLMulFactors {
RCC_PLLMulFactors_2 = 0 , RCC_PLLMulFactors_3 , RCC_PLLMulFactors_4 , RCC_PLLMulFactors_5 ,
RCC_PLLMulFactors_6 , RCC_PLLMulFactors_7 , RCC_PLLMulFactors_8 , RCC_PLLMulFactors_9 ,
RCC_PLLMulFactors_10 , RCC_PLLMulFactors_11 , RCC_PLLMulFactors_12 , RCC_PLLMulFactors_13 ,
RCC_PLLMulFactors_14 , RCC_PLLMulFactors_15 , RCC_PLLMulFactors_16 }

  This enum contains the multiplication factors of the PLL.
- enum RCC_AHBPrescaler {
RCC_AHBPrescaler_NotDivided = 0 , RCC_AHBPrescaler_DividedBy2 = 0b1000 , RCC_AHBPrescaler_DividedBy4
, RCC_AHBPrescaler_DividedBy8 ,
RCC_AHBPrescaler_DividedBy16 , RCC_AHBPrescaler_DividedBy64 , RCC_AHBPrescaler_DividedBy128
, RCC_AHBPrescaler_DividedBy256 ,
RCC_AHBPrescaler_DividedBy512 }

  This enum contains the prescaler factors of the AHB bus.
- enum RCC_APBPrescaler {
RCC_APBPrescaler_NotDivided = 0 , RCC_APBPrescaler_DividedBy2 = 0b100 , RCC_APBPrescaler_DividedBy4
, RCC_APBPrescaler_DividedBy8 ,
RCC_APBPrescaler_DividedBy16 }

  This enum contains the prescaler factors of the APB (1 & 2) bus.
- enum RCC_ADCPrescaler { RCC_ADCPrescaler_DividedBy2 = 0 , RCC_ADCPrescaler_DividedBy4 ,
RCC_ADCPrescaler_DividedBy6 , RCC_ADCPrescaler_DividedBy8 }

  This enum contains the prescaler factors of the ADC bus.
- enum RCC_SystemClock { RCC_SystemClock_HSI = 0 , RCC_SystemClock_HSE , RCC_SystemClock_PLL
}

  This enum contains the system clock sources.
- enum RCC_PLLSource { RCC_PLLSource_HSI_DividedBy2 = 0 , RCC_PLLSource_HSE }

  This enum contains the PLL clock sources.
- enum RCC_MCOSources {
RCC_MCOSources_NoClock = 0 , RCC_MCOSources_SystemClock = 0b100 , RCC_MCOSources_HSI ,
RCC_MCOSources_HSE ,
RCC_MCOSources_PLL_DividedBy2 }

  This enum contains the MCO sources.
- enum RCC_USBPrescaler { RCC_USBPrescaler_1_5 = 0 , RCC_USBPrescaler_1 }

  This enum contains the USB prescaler factors.
- enum t_RCC_APB2Peripherals {
RCC_APB2Peripherals_AFIO = 0 , RCC_APB2Peripherals_PORTA = 2 , RCC_APB2Peripherals_PORTB ,
RCC_APB2Peripherals_PORTC ,
RCC_APB2Peripherals_PORTD , RCC_APB2Peripherals_PORTE , RCC_APB2Peripherals_PORTF ,
RCC_APB2Peripherals_PORTG ,
RCC_APB2Peripherals_ADC1 , RCC_APB2Peripherals_ADC2 , RCC_APB2Peripherals_TIM1 , RCC_APB2Peripherals_SPI1
,
RCC_APB2Peripherals_USART1 = 14 , RCC_APB2Peripherals_ADC3 }

  This enum contains the APB2 peripherals that are connected to the APB2 bus.
- enum t_RCC_APB1Peripherals {
RCC_APB1Peripherals_TIM2 = 0 , RCC_APB1Peripherals_TIM3 , RCC_APB1Peripherals_TIM4 ,
RCC_APB1Peripherals_TIM5 ,
RCC_APB1Peripherals_TIM6 , RCC_APB1Peripherals_TIM7 , RCC_APB1Peripherals_WWDG = 11 ,
RCC_APB1Peripherals_SPI2 = 14 ,
RCC_APB1Peripherals_SPI3 , RCC_APB1Peripherals_USART2 = 17 , RCC_APB1Peripherals_USART3 ,
RCC_APB1Peripherals_UART4 ,
RCC_APB1Peripherals_UART5 , RCC_APB1Peripherals_I2C1 , RCC_APB1Peripherals_I2C2 , RCC_APB1Peripherals_USB
,
RCC_APB1Peripherals_CAN = 25 , RCC_APB1Peripherals_BKP = 27 , RCC_APB1Peripherals_PWR ,
RCC_APB1Peripherals_DAC }

This enum contains the APB1 peripherals that are connected to the APB1 bus.

- enum t_RCC_AHBPeripherals { RCC_AHBPeripherals_DMA1 = 0 , RCC_AHBPeripherals_DMA2 , RCC_AHBPeripherals_CRC = 5 }

  This enum contains the AHB peripherals that are connected to the AHB bus.

6.13.1 Detailed Description

This group contains the options of the RCC module interface.

6.13.2 Enumeration Type Documentation

6.13.2.1 RCC_PLLMulFactors

`enum RCC_PLLMulFactors`

`#include <MCAL/RCC/RCC_interface.h>`

This enum contains the multiplication factors of the PLL.

This enum contains the multiplication factors of the PLL

Enumerator

RCC_PLLMulFactors_2	Multiply the PLL1 input clock by 2.
RCC_PLLMulFactors_3	Multiply the PLL1 input clock by 3.
RCC_PLLMulFactors_4	Multiply the PLL1 input clock by 4.
RCC_PLLMulFactors_5	Multiply the PLL1 input clock by 5.
RCC_PLLMulFactors_6	Multiply the PLL1 input clock by 6.
RCC_PLLMulFactors_7	Multiply the PLL1 input clock by 7.
RCC_PLLMulFactors_8	Multiply the PLL1 input clock by 8.
RCC_PLLMulFactors_9	Multiply the PLL1 input clock by 9.
RCC_PLLMulFactors_10	Multiply the PLL1 input clock by 10.
RCC_PLLMulFactors_11	Multiply the PLL1 input clock by 11.
RCC_PLLMulFactors_12	Multiply the PLL1 input clock by 12.
RCC_PLLMulFactors_13	Multiply the PLL1 input clock by 13.
RCC_PLLMulFactors_14	Multiply the PLL1 input clock by 14.
RCC_PLLMulFactors_15	Multiply the PLL1 input clock by 15.
RCC_PLLMulFactors_16	Multiply the PLL1 input clock by 16.

```
{
    RCC_PLLMulFactors_2 = 0,
    RCC_PLLMulFactors_3,
    RCC_PLLMulFactors_4,
    RCC_PLLMulFactors_5,
    RCC_PLLMulFactors_6,
    RCC_PLLMulFactors_7,
    RCC_PLLMulFactors_8,
    RCC_PLLMulFactors_9,
    RCC_PLLMulFactors_10,
    RCC_PLLMulFactors_11,
    RCC_PLLMulFactors_12,
    RCC_PLLMulFactors_13,
    RCC_PLLMulFactors_14,
    RCC_PLLMulFactors_15,
    RCC_PLLMulFactors_16
};
```

6.13.2.2 RCC_AHBPrescaler

`enum RCC_AHBPrescaler`

`#include <MCAL/RCC/RCC_interface.h>`

This enum contains the prescaler factors of the AHB bus.

This enum contains the prescaler factors of the AHB bus

Enumerator

RCC_AHBPrescaler_NotDivided	AHB bus not divided.
RCC_AHBPrescaler_DividedBy2	AHB bus divided by 2.
RCC_AHBPrescaler_DividedBy4	AHB bus divided by 4.
RCC_AHBPrescaler_DividedBy8	AHB bus divided by 8.
RCC_AHBPrescaler_DividedBy16	AHB bus divided by 16.
RCC_AHBPrescaler_DividedBy64	AHB bus divided by 64.
RCC_AHBPrescaler_DividedBy128	AHB bus divided by 128.
RCC_AHBPrescaler_DividedBy256	AHB bus divided by 256.
RCC_AHBPrescaler_DividedBy512	AHB bus divided by 512.

```
{
    RCC_AHBPrescaler_NotDivided = 0,
    RCC_AHBPrescaler_DividedBy2 = 0b1000,
    RCC_AHBPrescaler_DividedBy4,
    RCC_AHBPrescaler_DividedBy8,
    RCC_AHBPrescaler_DividedBy16,
    RCC_AHBPrescaler_DividedBy64,
    RCC_AHBPrescaler_DividedBy128,
    RCC_AHBPrescaler_DividedBy256,
    RCC_AHBPrescaler_DividedBy512
};
```

6.13.2.3 RCC_APBPrescaler

`enum RCC_APBPrescaler`

`#include <MCAL/RCC/RCC_interface.h>`

This enum contains the prescaler factors of the APB (1 & 2) bus.

This enum contains the prescaler factors of the APB (1 & 2) bus

Enumerator

RCC_APBPrescaler_NotDivided	APB1 bus not divided.

Enumerator

RCC_APBPrescaler_DividedBy2	APB1 bus divided by 2.
RCC_APBPrescaler_DividedBy4	APB1 bus divided by 4.
RCC_APBPrescaler_DividedBy8	APB1 bus divided by 8.
RCC_APBPrescaler_DividedBy16	APB1 bus divided by 16.

```
{
    RCC_APBPrescaler_NotDivided = 0,
    RCC_APBPrescaler_DividedBy2 = 0b100,
    RCC_APBPrescaler_DividedBy4,
    RCC_APBPrescaler_DividedBy8,
    RCC_APBPrescaler_DividedBy16
};
```

6.13.2.4 RCC_ADCPrescaler

`enum RCC_ADCPrescaler`

`#include <MCAL/RCC/RCC_interface.h>`

This enum contains the prescaler factors of the ADC bus.

This enum contains the prescaler factors of the ADC bus

Enumerator

RCC_ADCPrescaler_DividedBy2	ADC bus divided by 2.
RCC_ADCPrescaler_DividedBy4	ADC bus divided by 4.
RCC_ADCPrescaler_DividedBy6	ADC bus divided by 6.
RCC_ADCPrescaler_DividedBy8	ADC bus divided by 8.

```
{
    RCC_ADCPrescaler_DividedBy2 = 0,
    RCC_ADCPrescaler_DividedBy4,
    RCC_ADCPrescaler_DividedBy6,
    RCC_ADCPrescaler_DividedBy8
};
```

6.13.2.5 RCC_SystemClock

`enum RCC_SystemClock`

`#include <MCAL/RCC/RCC_interface.h>`

This enum contains the system clock sources.

This enum contains the system clock sources

Enumerator

RCC_SystemClock_HSI	HSI clock selected as system clock.
RCC_SystemClock_HSE	HSE clock selected as system clock.
RCC_SystemClock_PLL	PLL clock selected as system clock.

```
{
    RCC_SystemClock_HSI = 0,
    RCC_SystemClock_HSE,
    RCC_SystemClock_PLL
};
```

6.13.2.6 RCC_PLLSource

enum RCC_PLLSource

#include <MCAL/RCC/RCC_interface.h>

This enum contains the PLL clock sources.

This enum contains the PLL clock sources

Enumerator

RCC_PLLSource_HSI_DividedBy2	HSI clock divided by 2 selected as PLL clock.
RCC_PLLSource_HSE	HSE clock selected as PLL clock.

```
{
    RCC_PLLSource_HSI_DividedBy2 = 0,
    RCC_PLLSource_HSE
};
```

6.13.2.7 RCC_MCOSources

enum RCC_MCOSources

#include <MCAL/RCC/RCC_interface.h>

This enum contains the MCO sources.

This enum contains the MCO sources

Enumerator

RCC_MCOSources_NoClock	No clock selected as MCO.
RCC_MCOSources_SystemClock	System clock selected as MCO.
RCC_MCOSources_HSI	HSI clock selected as MCO.
RCC_MCOSources_HSE	HSE clock selected as MCO.
RCC_MCOSources_PLL_DividedBy2	PLL clock divided by 2 selected as MCO.

```
{
    RCC_MCOSources_NoClock = 0,
    RCC_MCOSources_SystemClock = 0b100,
    RCC_MCOSources_HSI,
    RCC_MCOSources_HSE,
    RCC_MCOSources_PLL_DividedBy2
};
```

6.13.2.8 RCC_USBPrescaler

```
enum RCC_USBPrescaler
```

```
#include <MCAL/RCC/RCC_interface.h>
```

This enum contains the USB prescaler factors.

This enum contains the USB prescaler factors

Enumerator

RCC_USBPrescaler_1↩_5	USB clock divided by 1.5.
RCC_USBPrescaler_1	USB clock divided by 1.

```
{
    RCC_USBPrescaler_1_5 = 0,
    RCC_USBPrescaler_1
};
```

6.13.2.9 t_RCC_APB2Peripherals

```
enum t_RCC_APB2Peripherals
```

```
#include <MCAL/RCC/RCC_interface.h>
```

This enum contains the APB2 peripherals that are connected to the APB2 bus.

This enum contains the APB2 peripherals that are connected to the APB2 bus

Enumerator

RCC_APB2Peripherals_AFIO	Alternate function I/O clock.
RCC_APB2Peripherals_PORTA	I/O port A clock.
RCC_APB2Peripherals_PORTB	I/O port B clock.
RCC_APB2Peripherals_PORTC	I/O port C clock.
RCC_APB2Peripherals_PORTD	I/O port D clock.
RCC_APB2Peripherals_PORTE	I/O port E clock.
RCC_APB2Peripherals_PORTF	I/O port F clock.
RCC_APB2Peripherals_PORTG	I/O port G clock.
RCC_APB2Peripherals_ADC1	ADC 1 interface clock.
RCC_APB2Peripherals_ADC2	ADC 2 interface clock.
RCC_APB2Peripherals_TIM1	Timer 1 (TIM1) clock.
RCC_APB2Peripherals_SPI1	SPI 1 clock.
RCC_APB2Peripherals_USART1	USART1 clock.
RCC_APB2Peripherals_ADC3	ADC 3 interface clock.

```
{
    RCC_APB2Peripherals_AFIO = 0,
    RCC_APB2Peripherals_PORTA = 2,
    RCC_APB2Peripherals_PORTB,
    RCC_APB2Peripherals_PORTC,
    RCC_APB2Peripherals_PORTD,
    RCC_APB2Peripherals_PORTE,
    RCC_APB2Peripherals_PORTF,
    RCC_APB2Peripherals_PORTG,
    RCC_APB2Peripherals_ADC1,
    RCC_APB2Peripherals_ADC2,
    RCC_APB2Peripherals_TIM1,
    RCC_APB2Peripherals_SPI1,
    RCC_APB2Peripherals_USART1 = 14,
    RCC_APB2Peripherals_ADC3
} t_RCC_APB2Peripherals;
```

6.13.2.10 t_RCC_APB1Peripherals

enum t_RCC_APB1Peripherals

#include <MCAL/RCC/RCC_interface.h>

This enum contains the APB1 peripherals that are connected to the APB1 bus.

This enum contains the APB1 peripherals that are connected to the APB1 bus

Enumerator

RCC_APB1Peripherals_TIM2	Timer 2 (TIM2) clock.
RCC_APB1Peripherals_TIM3	Timer 3 (TIM3) clock.
RCC_APB1Peripherals_TIM4	Timer 4 (TIM4) clock.
RCC_APB1Peripherals_TIM5	Timer 5 (TIM5) clock.
RCC_APB1Peripherals_TIM6	Timer 6 (TIM6) clock.
RCC_APB1Peripherals_TIM7	Timer 7 (TIM7) clock.
RCC_APB1Peripherals_WWDG	Window watchdog (WWDG) clock.
RCC_APB1Peripherals_SPI2	SPI 2 clock.
RCC_APB1Peripherals_SPI3	SPI 3 clock.
RCC_APB1Peripherals_USART2	USART 2 clock.
RCC_APB1Peripherals_USART3	USART 3 clock.
RCC_APB1Peripherals_UART4	UART 4 clock.
RCC_APB1Peripherals_UART5	UART 5 clock.
RCC_APB1Peripherals_I2C1	I2C 1 clock.
RCC_APB1Peripherals_I2C2	I2C 2 clock.
RCC_APB1Peripherals_USB	USB clock.
RCC_APB1Peripherals_CAN	CAN clock.
RCC_APB1Peripherals_BKP	Backup interface clock.
RCC_APB1Peripherals_PWR	Power interface clock.
RCC_APB1Peripherals_DAC	DAC interface clock.

```
{
    RCC_APB1Peripherals_TIM2 = 0,
    RCC_APB1Peripherals_TIM3,
    RCC_APB1Peripherals_TIM4,
    RCC_APB1Peripherals_TIM5,
    RCC_APB1Peripherals_TIM6,
    RCC_APB1Peripherals_TIM7,
    RCC_APB1Peripherals_WWDG = 11,
    RCC_APB1Peripherals_SPI2 = 14,
    RCC_APB1Peripherals_SPI3,
    RCC_APB1Peripherals_USART2 = 17,
    RCC_APB1Peripherals_USART3,
    RCC_APB1Peripherals_UART4,
    RCC_APB1Peripherals_UART5,
    RCC_APB1Peripherals_I2C1,
    RCC_APB1Peripherals_I2C2,
    RCC_APB1Peripherals_USB,
    RCC_APB1Peripherals_CAN = 25,
    RCC_APB1Peripherals_BKP = 27,
    RCC_APB1Peripherals_PWR,
    RCC_APB1Peripherals_DAC
} t_RCC_APB1Peripherals;
```

6.13.2.11 t_RCC_AHBPeripherals

enum t_RCC_AHBPeripherals

#include <MCAL/RCC/RCC_interface.h>

This enum contains the AHB peripherals that are connected to the AHB bus.

This enum contains the AHB peripherals that are connected to the AHB bus

Enumerator

RCC_AHBPeripherals_DMA1	DMA 1 clock.
RCC_AHBPeripherals_DMA2	DMA 2 clock.
RCC_AHBPeripherals_CRC	CRC clock.

```
{
    RCC_AHBPeripherals_DMA1 = 0,
    RCC_AHBPeripherals_DMA2,
    RCC_AHBPeripherals_CRC = 5
} t_RCC_AHBPeripherals;
```

6.14 RCC exported functions

RCC exported functions.

Collaboration diagram for RCC exported functions:

Functions

- void RCC_vInit (void)

  This function initializes the RCC peripheral.
- void RCC_vEnablePeripheralABP2 (t_RCC_APB2Peripherals enuPeripheral)

  This function enables the clock of a peripheral connected to the APB2 bus.
- void RCC_vDisablePeripheralABP2 (t_RCC_APB2Peripherals enuPeripheral)

  This function disables the clock of a peripheral connected to the APB2 bus.
- void RCC_vEnablePeripheralABP1 (t_RCC_APB1Peripherals enuPeripheral)

  This function enables the clock of a peripheral connected to the APB1 bus.
- void RCC_vDisablePeripheralABP1 (t_RCC_APB1Peripherals enuPeripheral)

  This function disables the clock of a peripheral connected to the APB1 bus.
- void RCC_vEnablePeripheralAHB (t_RCC_AHBPeripherals enuPeripheral)

  This function enables the clock of a peripheral connected to the AHB bus.
- void RCC_vDisablePeripheralAHB (t_RCC_AHBPeripherals enuPeripheral)

  This function disables the clock of a peripheral connected to the AHB bus.

6.14.1 Detailed Description

RCC exported functions.

6.14.2 Function Documentation

6.14.2.1 RCC_vInit()

```
void RCC_vInit (
             void )
```

`#include <MCAL/RCC/RCC_interface.h>`

This function initializes the RCC peripheral.

This function initializes the RCC peripheral

```
{
    if (RCC_SYSTEM_CLOCK_SOURCE == RCC_SystemClock_HSE)
    {
        RCC_vInitHSEClock();
    }
    else if (RCC_SYSTEM_CLOCK_SOURCE == RCC_SystemClock_HSI)
    {
        RCC_vInitHSIClock();
    }
    else if (RCC_SYSTEM_CLOCK_SOURCE == RCC_SystemClock_PLL)
    {
        RCC_vInitPLL();
    }
    else
    {
        /* Do nothing */
    }

    RCC_vInitSystemClock();
    RCC_vSetAHBPrescaler();
    RCC_vSetAPB1Prescaler();
    RCC_vSetAPB2Prescaler();
    RCC_vSetADCPreScaler();
    RCC_vSetUSBPrescaler();
    RCC_vInitMCO();
}
```

References RCC_SYSTEM_CLOCK_SOURCE, RCC_SystemClock_HSE, RCC_SystemClock_HSI, RCC_SystemClock_PLL, RCC_vInitHSEClock(), RCC_vInitHSIClock(), RCC_vInitMCO(), RCC_vInitPLL(), RCC_vInitSystemClock(), RCC_vSetADCPreScaler(), RCC_vSetAHBPrescaler(), RCC_vSetAPB1Prescaler(), RCC_vSetAPB2Prescaler(), and RCC_vSetUSBPrescaler().

Referenced by vAPPS_main().

Here is the call graph for this function: Here is the caller graph for this function:

6.14.2.2 RCC_vEnablePeripheralABP2()

```
void RCC_vEnablePeripheralABP2 (
             t_RCC_APB2Peripherals enuPeripheral )
```

`#include <MCAL/RCC/RCC_interface.h>`

This function enables the clock of a peripheral connected to the APB2 bus.

This function enables the clock of a peripheral connected to the APB2 bus

Parameters

in	*enuPeripheral*	Peripheral to be enabled

```
{
    RCC_vSetAPB2PeripheralClockStatus(enuPeripheral, TRUE);
}
```

References RCC_vSetAPB2PeripheralClockStatus(), and TRUE.

Referenced by vTestApp_TestingGPIO_main().

Here is the call graph for this function: Here is the caller graph for this function:

6.14.2.3 RCC_vDisablePeripheralABP2()

```
void RCC_vDisablePeripheralABP2 (
            t_RCC_APB2Peripherals enuPeripheral )
```

```
#include <MCAL/RCC/RCC_interface.h>
```

This function disables the clock of a peripheral connected to the APB2 bus.

This function disables the clock of a peripheral connected to the APB2 bus

Parameters

in	*enuPeripheral*	Peripheral to be disabled

```
{
    RCC_vSetAPB2PeripheralClockStatus(enuPeripheral, FALSE);
}
```

References FALSE, and RCC_vSetAPB2PeripheralClockStatus().

Here is the call graph for this function:

6.14.2.4 RCC_vEnablePeripheralABP1()

```
void RCC_vEnablePeripheralABP1 (
            t_RCC_APB1Peripherals enuPeripheral )
```

```
#include <MCAL/RCC/RCC_interface.h>
```

This function enables the clock of a peripheral connected to the APB1 bus.

This function enables the clock of a peripheral connected to the APB1 bus

Parameters

in	*enuPeripheral*	Peripheral to be enabled

```
{
    RCC_vSetAPB1PeripheralClockStatus(enuPeripheral, TRUE);
}
```

References RCC_vSetAPB1PeripheralClockStatus(), and TRUE.

Here is the call graph for this function:

6.14.2.5 RCC_vDisablePeripheralABP1()

```
void RCC_vDisablePeripheralABP1 (
            t_RCC_APB1Peripherals enuPeripheral )
```

```
#include <MCAL/RCC/RCC_interface.h>
```

This function disables the clock of a peripheral connected to the APB1 bus.

This function disables the clock of a peripheral connected to the APB1 bus

Parameters

in	*enuPeripheral*	Peripheral to be disabled

```
{
    RCC_vSetAPB1PeripheralClockStatus(enuPeripheral, FALSE);
}
```

References FALSE, and RCC_vSetAPB1PeripheralClockStatus().

Here is the call graph for this function:

6.14.2.6 RCC_vEnablePeripheralAHB()

```
void RCC_vEnablePeripheralAHB (
            t_RCC_AHBPeripherals enuPeripheral )
```

```
#include <MCAL/RCC/RCC_interface.h>
```

This function enables the clock of a peripheral connected to the AHB bus.

This function enables the clock of a peripheral connected to the AHB bus

Parameters

in	*enuPeripheral*	Peripheral to be enabled

```
{
    RCC_vSetAHBPeripheralClockStatus(enuPeripheral, TRUE);
}
```

References RCC_vSetAHBPeripheralClockStatus(), and TRUE.

Here is the call graph for this function:

6.14.2.7 RCC_vDisablePeripheralAHB()

```
void RCC_vDisablePeripheralAHB (
            t_RCC_AHBPeripherals enuPeripheral )
```

```
#include <MCAL/RCC/RCC_interface.h>
```

This function disables the clock of a peripheral connected to the AHB bus.

This function disables the clock of a peripheral connected to the AHB bus

Parameters

in	*enuPeripheral*	Peripheral to be disabled

```
{
    RCC_vSetAHBPeripheralClockStatus(enuPeripheral, FALSE);
}
```

References FALSE, and RCC_vSetAHBPeripheralClockStatus().

Here is the call graph for this function:

6.15 RCC Module

RCC Module.

Collaboration diagram for RCC Module:

Modules

- RCC Configuration

  This group contains the configuration parameters of the RCC module.
- RCC Interface Options

  This group contains the options of the RCC module interface.
- RCC exported functions

  RCC exported functions.
- RCC Registers

  RCC Registers.
- RCC Addresses

  RCC Addresses.

6.15.1 Detailed Description

RCC Module.

RCC Module contains the functions for controlling the clock system of the microcontroller.

6.16 RCC Registers

RCC Registers.

Collaboration diagram for RCC Registers:

Data Structures

- struct t_RCC_CR

  RCC clock control register.
- struct t_RCC_CFGR

  RCC clock configuration register.
- struct t_RCC_CIR

  RCC clock interrupt register.
- struct t_RCC_APB2RSTR

  RCC APB2 peripheral reset register.
- struct t_RCC_APB1RSTR

  RCC APB1 peripheral reset register.
- struct t_RCC_AHBENR

  RCC AHB peripheral clock register.
- struct t_RCC_APB2ENR

  RCC APB2 peripheral clock enable register.
- struct t_RCC_APB1ENR

  RCC APB1 peripheral clock enable register.
- struct t_RCC_BDCR

  RCC Backup domain control register.
- struct t_RCC_CSR

  RCC Control/status register.
- struct t_RCC_RegisterMap

  RCC Register Map.

6.16.1 Detailed Description

RCC Registers.

6.17 RCC Addresses

RCC Addresses.

Collaboration diagram for RCC Addresses:

Macros

- #define RCC_BASE_ADDRESS REGISTER_ADDRESS(0x40021000, 0)

  RCC Base Address in the memory.
- #define RCC REGISTER(t_RCC_RegisterMap, RCC_BASE_ADDRESS)

  RCC Register Map.
- #define RCC_SET_REGISTER_BIT_STATUS(REG, BIT, BSTATUS) SET_REGISTER_BIT_STATUS(RCC, REG, BIT, BSTATUS)

  Set RCC register bit status (TRUE or FALSE)

6.17.1 Detailed Description

RCC Addresses.

6.17.2 Macro Definition Documentation

6.17.2.1 RCC_BASE_ADDRESS

```
#define RCC_BASE_ADDRESS REGISTER_ADDRESS(0x40021000, 0)
```

`#include <MCAL/RCC/RCC_private.h>`

RCC Base Address in the memory.

6.17.2.2 RCC

```
#define RCC REGISTER(t_RCC_RegisterMap, RCC_BASE_ADDRESS)
```

`#include <MCAL/RCC/RCC_private.h>`

RCC Register Map.

6.17.2.3 RCC_SET_REGISTER_BIT_STATUS

```
#define RCC_SET_REGISTER_BIT_STATUS(
              REG,
              BIT,
              BSTATUS ) SET_REGISTER_BIT_STATUS(RCC, REG, BIT, BSTATUS)
```

`#include <MCAL/RCC/RCC_private.h>`

Set RCC register bit status (TRUE or FALSE)

This macro sets the RCC register bit status (TRUE or FALSE)

Parameters

in	*REG*	Register name
in	*BIT*	Bit name
in	*BSTATUS*	Bit status value (TRUE or FALSE)

6.18 Testing Applications

Testing Applications.

Functions

- void vAPPS_main (void)

  This function is responsible for running the applications.
- void vTestApp_TestingGPIO_main (void)

  This function is the main function for the TestApp_TestingGPIO application.

6.18.1 Detailed Description

Testing Applications.

This module contains all the testing applications

6.18.2 Function Documentation

6.18.2.1 vAPPS_main()

```
void vAPPS_main (
             void )
```

```
#include <APPS/APPS_main.h>
```

This function is responsible for running the applications.

This function is responsible for running the applications

```
{
    RCC_vInit();
    TESTING_APPLICATION_MAIN_FUNC(TestingGPIO);

    for (;;)
    {
        /* Do nothing */
    }
}
```

References RCC_vInit(), TESTING_APPLICATION_MAIN_FUNC, and TestingGPIO.

Referenced by main().

Here is the call graph for this function: Here is the caller graph for this function:

6.18.2.2 vTestApp_TestingGPIO_main()

```
void vTestApp_TestingGPIO_main (
            void  )
```

`#include <APPS/TestingGPIO/TestApp_TestingGPIO_main.h>`

This function is the main function for the TestApp_TestingGPIO application.

This function is the main function for the TestApp_TestingGPIO application

```
{
    // t_u32 u32Counter = 0;
    RCC_vEnablePeripheralABP2(RCC_APB2Peripherals_PORTA);
    RCC_vEnablePeripheralABP2(RCC_APB2Peripherals_PORTB);
    GPIO_vSetPinDirection(GPIO_Ports_A, GPIO_Pins_1, GPIO_Direction_Output_50MHz);
    GPIO_vSetPinDirection(GPIO_Ports_B, GPIO_Pins_7, GPIO_Direction_Input);
    // GPIO_vSetPinInputType(GPIO_Ports_B, GPIO_Pins_7, GPIO_Input_Type_Pull_Down);
    GPIO_vSetPinInputType(GPIO_Ports_B, GPIO_Pins_7, GPIO_Input_Type_Pull_Up);

    for (;;)
    {
        GPIO_vSetPinValue(GPIO_Ports_A, GPIO_Pins_1, GPIO_tGetPinValue(GPIO_Ports_B, GPIO_Pins_7));
        // if (GPIO_tGetPinValue(GPIO_Ports_B, GPIO_Pins_7) == GPIO_Value_High)
        // if (GPIO_tGetPinValue(GPIO_Ports_B, GPIO_Pins_7) == GPIO_Value_Low)
        // {
            // GPIO_vSetPinValue(GPIO_Ports_A, GPIO_Pins_1, GPIO_Value_High);
        // }
        // else
        // {
            // GPIO_vSetPinValue(GPIO_Ports_A, GPIO_Pins_1, GPIO_Value_Low);
        // }

        // GPIO_vSetPinValue(GPIO_Ports_A, GPIO_Pins_1, GPIO_Value_High);
        // for (u32Counter = 0; u32Counter < 10000000; u32Counter++)
        //  ;
        // GPIO_vSetPinValue(GPIO_Ports_A, GPIO_Pins_1, GPIO_Value_Low);
        // for (u32Counter = 0; u32Counter < 10000000; u32Counter++)
        //  ;
    }
}
```

References GPIO_Direction_Input, GPIO_Direction_Output_50MHz, GPIO_Input_Type_Pull_Up, GPIO_Pins_1, GPIO_Pins_7, GPIO_Ports_A, GPIO_Ports_B, GPIO_tGetPinValue(), GPIO_vSetPinDirection(), GPIO_vSetPinInputType(), GPIO_vSetPinValue(), RCC_APB2Peripherals_PORTA, RCC_APB2Peripherals_PORTB, and RCC_vEnablePeripheralABP2().

Here is the call graph for this function:

6.19 Shared library

This library contains the shared macros and functions for all the standard types.

Collaboration diagram for Shared library:

Macros

- #define SET_REGISTER_BIT_STATUS(PERPH, REG, BIT, BSTATUS) (PERPH.REG.BIT = (BSTATUS == TRUE) ? 1U : 0U)

  Set register bit status (1 or 0)

6.19.1 Detailed Description

This library contains the shared macros and functions for all the standard types.

6.19.2 Macro Definition Documentation

6.19.2.1 SET_REGISTER_BIT_STATUS

```
#define SET_REGISTER_BIT_STATUS(
              PERPH,
              REG,
              BIT,
              BSTATUS ) (PERPH.REG.BIT = (BSTATUS == TRUE) ?  1U : 0U)
```

```
#include <LIB/LSTD_SHARED.h>
```

Set register bit status (1 or 0)

This macro is used to set a specific bit in a specific register

Parameters

in	*PERPH*	Peripheral address
in	*REG*	Register name
in	*BIT*	Bit name
in	*BSTATUS*	Bit status (TRUE or FALSE)

Chapter 7

Data Structure Documentation

7.1 t_GPIOx_RegisterMap Struct Reference

GPIO Register Map.

```
#include "MCAL/GPIO/GPIO_private.h"
```

Collaboration diagram for t_GPIOx_RegisterMap:

Data Fields

- t_u32 CRL

  Port Configuration Register Low.
- t_u32 CRH

  Port Configuration Register High.
- t_u32 IDR

  Port Input Data Register.
- t_u32 ODR

  Port Output Data Register.
- t_u32 BSRR

  Port Bit Set/Reset Register.
- t_u32 BRR

  Port Bit Reset Register.

7.1.1 Detailed Description

GPIO Register Map.

This type is used to access the GPIO registers

7.1.2 Field Documentation

7.1.2.1 CRL

```
t_u32 CRL
```

Port Configuration Register Low.

7.1.2.2 CRH

```
t_u32 CRH
```

Port Configuration Register High.

7.1.2.3 IDR

```
t_u32 IDR
```

Port Input Data Register.

Warning

This register is read-only

7.1.2.4 ODR

```
t_u32 ODR
```

Port Output Data Register.

Referenced by GPIO_vSetPinInputTypePullUpDown().

7.1.2.5 BSRR

```
t_u32 BSRR
```

Port Bit Set/Reset Register.

7.1.2.6 BRR

```
t_u32 BRR
```

Port Bit Reset Register.

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

- MCAL/GPIO/GPIO_private.h

7.2 t_RCC_AHBENR Struct Reference

RCC AHB peripheral clock register.

```
#include "MCAL/RCC/RCC_private.h"
```

Collaboration diagram for t_RCC_AHBENR:

Data Fields

- t_u32 DMA1EN: 1

  DMA1 clock enable.
- t_u32 DMA2EN: 1

  DMA2 clock enable.
- t_u32 __pad0__: 4

  Reserved bit(s)
- t_u32 CRCEN: 1

  CRC clock enable.
- t_u32 __pad1__: 25

  Reserved bit(s)

7.2.1 Detailed Description

RCC AHB peripheral clock register.

7.2.2 Field Documentation

7.2.2.1 DMA1EN

```
t_u32 DMA1EN
```

DMA1 clock enable.

This field enables the DMA1 clock

7.2.2.2 DMA2EN

`t_u32 DMA2EN`

DMA2 clock enable.

This field enables the DMA2 clock

7.2.2.3 __pad0__

`t_u32 __pad0__`

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.2.2.4 CRCEN

`t_u32 CRCEN`

CRC clock enable.

This field enables the CRC clock

7.2.2.5 __pad1__

`t_u32 __pad1__`

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

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

- MCAL/RCC/RCC_private.h

7.3 t_RCC_APB1ENR Struct Reference

RCC APB1 peripheral clock enable register.

```
#include "MCAL/RCC/RCC_private.h"
```

Collaboration diagram for t_RCC_APB1ENR:

Data Fields

- t_u32 TIM2EN: 1

  Timer 2 clock enable.
- t_u32 TIM3EN: 1

  Timer 3 clock enable.
- t_u32 TIM4EN: 1

  Timer 4 clock enable.
- t_u32 TIM5EN: 1

  Timer 5 clock enable.
- t_u32 TIM6EN: 1

  Timer 6 clock enable.
- t_u32 TIM7EN: 1

  Timer 7 clock enable.
- t_u32 __pad0__: 5

  Reserved bit(s)
- t_u32 WWDGEN: 1

  Window watchdog clock enable.
- t_u32 __pad1__: 2

  Reserved bit(s)
- t_u32 SPI2EN: 1

  SPI 2 clock enable.
- t_u32 SPI3EN: 1

  SPI 3 clock enable.
- t_u32 __pad2__: 1

  Reserved bit(s)
- t_u32 USART2EN: 1

  USART 2 clock enable.
- t_u32 USART3EN: 1

  USART 3 clock enable.
- t_u32 UART4EN: 1

  UART 4 clock enable.
- t_u32 UART5EN: 1

  UART 5 clock enable.
- t_u32 I2C1EN: 1

  I2C 1 clock enable.
- t_u32 I2C2EN: 1

  I2C 2 clock enable.
- t_u32 USBEN: 1

  USB clock enable.
- t_u32 __pad3__: 1

  Reserved bit(s)
- t_u32 CANEN: 1

  CAN clock enable.
- t_u32 __pad4__: 1

  Reserved bit(s)
- t_u32 BKPEN: 1

  Backup interface clock enable.
- t_u32 PWREN: 1

  Power interface clock enable.
- t_u32 DACEN: 1

  DAC interface clock enable.
- t_u32 __pad5__: 2

  Reserved bit(s)

7.3.1 Detailed Description

RCC APB1 peripheral clock enable register.

7.3.2 Field Documentation

7.3.2.1 TIM2EN

`t_u32 TIM2EN`

Timer 2 clock enable.

This field enables the Timer 2 clock

7.3.2.2 TIM3EN

`t_u32 TIM3EN`

Timer 3 clock enable.

This field enables the Timer 3 clock

7.3.2.3 TIM4EN

`t_u32 TIM4EN`

Timer 4 clock enable.

This field enables the Timer 4 clock

7.3.2.4 TIM5EN

`t_u32 TIM5EN`

Timer 5 clock enable.

This field enables the Timer 5 clock

7.3.2.5 TIM6EN

`t_u32 TIM6EN`

Timer 6 clock enable.

This field enables the Timer 6 clock

7.3.2.6 TIM7EN

```
t_u32 TIM7EN
```

Timer 7 clock enable.

This field enables the Timer 7 clock

7.3.2.7 __pad0__

```
t_u32 __pad0__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.3.2.8 WWDGEN

```
t_u32 WWDGEN
```

Window watchdog clock enable.

This field enables the Window watchdog clock

7.3.2.9 __pad1__

```
t_u32 __pad1__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.3.2.10 SPI2EN

```
t_u32 SPI2EN
```

SPI 2 clock enable.

This field enables the SPI 2 clock

7.3.2.11 SPI3EN

`t_u32 SPI3EN`

SPI 3 clock enable.

This field enables the SPI 3 clock

7.3.2.12 __pad2__

`t_u32 __pad2__`

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.3.2.13 USART2EN

`t_u32 USART2EN`

USART 2 clock enable.

This field enables the USART 2 clock

7.3.2.14 USART3EN

`t_u32 USART3EN`

USART 3 clock enable.

This field enables the USART 3 clock

7.3.2.15 UART4EN

`t_u32 UART4EN`

UART 4 clock enable.

This field enables the UART 4 clock

7.3.2.16 UART5EN

`t_u32 UART5EN`

UART 5 clock enable.

This field enables the UART 5 clock

7.3.2.17 I2C1EN

```
t_u32 I2C1EN
```

I2C 1 clock enable.

This field enables the I2C 1 clock

7.3.2.18 I2C2EN

```
t_u32 I2C2EN
```

I2C 2 clock enable.

This field enables the I2C 2 clock

7.3.2.19 USBEN

```
t_u32 USBEN
```

USB clock enable.

This field enables the USB clock

7.3.2.20 __pad3__

```
t_u32 __pad3__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.3.2.21 CANEN

```
t_u32 CANEN
```

CAN clock enable.

This field enables the CAN clock

7.3.2.22 __pad4__

```
t_u32 __pad4__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.3.2.23 BKPEN

```
t_u32 BKPEN
```

Backup interface clock enable.

This field enables the Backup interface clock

7.3.2.24 PWREN

```
t_u32 PWREN
```

Power interface clock enable.

This field enables the Power interface clock

7.3.2.25 DACEN

```
t_u32 DACEN
```

DAC interface clock enable.

This field enables the DAC interface clock

7.3.2.26 __pad5__

```
t_u32 __pad5__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

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

- MCAL/RCC/RCC_private.h

7.4 t_RCC_APB1RSTR Struct Reference

RCC APB1 peripheral reset register.

```
#include "MCAL/RCC/RCC_private.h"
```

Collaboration diagram for t_RCC_APB1RSTR:

Data Fields

- t_u32 TIM2RST: 1

  TIM2 timer reset.
- t_u32 TIM3RST: 1

  TIM3 timer reset.
- t_u32 TIM4RST: 1

  TIM4 timer reset.
- t_u32 TIM5RST: 1

  TIM5 timer reset.
- t_u32 TIM6RST: 1

  TIM6 timer reset.
- t_u32 TIM7RST: 1

  TIM7 timer reset.
- t_u32 __pad0__: 5

  Reserved bit(s)
- t_u32 WWDGRST: 1

  Window watchdog reset.
- t_u32 __pad1__: 2

  Reserved bit(s)
- t_u32 SPI2RST: 1

  SPI 2 reset.
- t_u32 SPI3RST: 1

  SPI 3 reset.
- t_u32 __pad2__: 1

  Reserved bit(s)
- t_u32 USART2RST: 1

  USART 2 reset.
- t_u32 USART3RST: 1

  USART 3 reset.
- t_u32 UART4RST: 1

  UART 4 reset.
- t_u32 UART5RST: 1

  UART 5 reset.
- t_u32 I2C1RST: 1

  I2C 1 reset.
- t_u32 I2C2RST: 1

  I2C 2 reset.
- t_u32 USBRST: 1

  USB interface reset.
- t_u32 __pad3__: 1

  Reserved bit(s)

- t_u32 CANRST: 1

  CAN reset.
- t_u32 __pad4__: 1

  Reserved bit(s)
- t_u32 BKPRST: 1

  Backup interface reset.
- t_u32 PWRRST: 1

  Power interface reset.
- t_u32 DACRST: 1

  DAC interface reset.
- t_u32 __pad5__: 2

  Reserved bit(s)

7.4.1 Detailed Description

RCC APB1 peripheral reset register.

7.4.2 Field Documentation

7.4.2.1 TIM2RST

```
t_u32 TIM2RST
```

TIM2 timer reset.

This field resets the TIM2 configuration registers

7.4.2.2 TIM3RST

```
t_u32 TIM3RST
```

TIM3 timer reset.

This field resets the TIM3 configuration registers

7.4.2.3 TIM4RST

```
t_u32 TIM4RST
```

TIM4 timer reset.

This field resets the TIM4 configuration registers

7.4.2.4 TIM5RST

```
t_u32 TIM5RST
```

TIM5 timer reset.

This field resets the TIM5 configuration registers

7.4.2.5 TIM6RST

```
t_u32 TIM6RST
```

TIM6 timer reset.

This field resets the TIM6 configuration registers

7.4.2.6 TIM7RST

```
t_u32 TIM7RST
```

TIM7 timer reset.

This field resets the TIM7 configuration registers

7.4.2.7 __pad0__

```
t_u32 __pad0__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.4.2.8 WWDGRST

```
t_u32 WWDGRST
```

Window watchdog reset.

This field resets the window watchdog configuration registers

7.4.2.9 __pad1__

```
t_u32 __pad1__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.4.2.10 SPI2RST

```
t_u32 SPI2RST
```

SPI 2 reset.

This field resets the SPI 2 configuration registers

7.4.2.11 SPI3RST

```
t_u32 SPI3RST
```

SPI 3 reset.

This field resets the SPI 3 configuration registers

7.4.2.12 __pad2__

```
t_u32 __pad2__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.4.2.13 USART2RST

```
t_u32 USART2RST
```

USART 2 reset.

This field resets the USART 2 configuration registers

7.4.2.14 USART3RST

`t_u32 USART3RST`

USART 3 reset.

This field resets the USART 3 configuration registers

7.4.2.15 UART4RST

`t_u32 UART4RST`

UART 4 reset.

This field resets the UART 4 configuration registers

7.4.2.16 UART5RST

`t_u32 UART5RST`

UART 5 reset.

This field resets the UART 5 configuration registers

7.4.2.17 I2C1RST

`t_u32 I2C1RST`

I2C 1 reset.

This field resets the I2C 1 configuration registers

7.4.2.18 I2C2RST

`t_u32 I2C2RST`

I2C 2 reset.

This field resets the I2C 2 configuration registers

7.4.2.19 USBRST

`t_u32 USBRST`

USB interface reset.

This field resets the USB interface configuration registers

7.4.2.20 __pad3__

```
t_u32 __pad3__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.4.2.21 CANRST

```
t_u32 CANRST
```

CAN reset.

This field resets the CAN configuration registers

7.4.2.22 __pad4__

```
t_u32 __pad4__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.4.2.23 BKPRST

```
t_u32 BKPRST
```

Backup interface reset.

This field resets the backup interface configuration registers

7.4.2.24 PWRRST

```
t_u32 PWRRST
```

Power interface reset.

This field resets the power interface configuration registers

7.4.2.25 DACRST

```
t_u32 DACRST
```

DAC interface reset.

This field resets the DAC interface configuration registers

7.4.2.26 __pad5__

```
t_u32 __pad5__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

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

- MCAL/RCC/RCC_private.h

7.5 t_RCC_APB2ENR Struct Reference

RCC APB2 peripheral clock enable register.

```
#include "MCAL/RCC/RCC_private.h"
```

Collaboration diagram for t_RCC_APB2ENR:

Data Fields

- t_u32 AFIOEN: 1

  Alternate function I/O clock enable.
- t_u32 __pad0__: 1

  Reserved bit(s)
- t_u32 IOPAEN: 1

  I/O port A clock enable.
- t_u32 IOPBEN: 1

  I/O port B clock enable.
- t_u32 IOPCEN: 1

  I/O port C clock enable.
- t_u32 IOPDEN: 1

  I/O port D clock enable.
- t_u32 IOPEEN: 1

  I/O port E clock enable.
- t_u32 IOPFEN: 1

  I/O port F clock enable.

- t_u32 IOPGEN: 1

  I/O port G clock enable.
- t_u32 ADC1EN: 1

  ADC 1 interface clock enable.
- t_u32 ADC2EN: 1

  ADC 2 interface clock enable.
- t_u32 TIM1EN: 1

  TIM1 Timer clock enable.
- t_u32 SPI1EN: 1

  SPI 1 clock enable.
- t_u32 __pad1__: 1

  Reserved bit(s)
- t_u32 USART1EN: 1

  USART1 clock enable.
- t_u32 ADC3EN: 1

  ADC 3 interface clock enable.
- t_u32 __pad2__: 16

  Reserved bit(s)

7.5.1 Detailed Description

RCC APB2 peripheral clock enable register.

7.5.2 Field Documentation

7.5.2.1 AFIOEN

```
t_u32 AFIOEN
```

Alternate function I/O clock enable.

This field enables the alternate function I/O clock

7.5.2.2 __pad0__

```
t_u32 __pad0__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.5.2.3 IOPAEN

```
t_u32 IOPAEN
```

I/O port A clock enable.

This field enables the I/O port A clock

7.5.2.4 IOPBEN

```
t_u32 IOPBEN
```

I/O port B clock enable.

This field enables the I/O port B clock

7.5.2.5 IOPCEN

```
t_u32 IOPCEN
```

I/O port C clock enable.

This field enables the I/O port C clock

7.5.2.6 IOPDEN

```
t_u32 IOPDEN
```

I/O port D clock enable.

This field enables the I/O port D clock

7.5.2.7 IOPEEN

```
t_u32 IOPEEN
```

I/O port E clock enable.

This field enables the I/O port E clock

7.5.2.8 IOPFEN

```
t_u32 IOPFEN
```

I/O port F clock enable.

This field enables the I/O port F clock

7.5.2.9 IOPGEN

`t_u32 IOPGEN`

I/O port G clock enable.

This field enables the I/O port G clock

7.5.2.10 ADC1EN

`t_u32 ADC1EN`

ADC 1 interface clock enable.

This field enables the ADC 1 interface clock

7.5.2.11 ADC2EN

`t_u32 ADC2EN`

ADC 2 interface clock enable.

This field enables the ADC 2 interface clock

7.5.2.12 TIM1EN

`t_u32 TIM1EN`

TIM1 Timer clock enable.

This field enables the TIM1 Timer clock

7.5.2.13 SPI1EN

`t_u32 SPI1EN`

SPI 1 clock enable.

This field enables the SPI 1 clock

7.5.2.14 __pad1__

`t_u32 __pad1__`

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.5.2.15 USART1EN

```
t_u32 USART1EN
```

USART1 clock enable.

This field enables the USART1 clock

7.5.2.16 ADC3EN

```
t_u32 ADC3EN
```

ADC 3 interface clock enable.

This field enables the ADC 3 interface clock

7.5.2.17 __pad2__

```
t_u32 __pad2__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

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

- MCAL/RCC/RCC_private.h

7.6 t_RCC_APB2RSTR Struct Reference

RCC APB2 peripheral reset register.

```
#include "MCAL/RCC/RCC_private.h"
```

Collaboration diagram for t_RCC_APB2RSTR:

Data Fields

- t_u32 AFIORST: 1

  Alternate function I/O reset.
- t_u32 __pad0__: 1

  Reserved bit(s)
- t_u32 IOPARST: 1

  IO port A reset.
- t_u32 IOPBRST: 1

  IO port B reset.
- t_u32 IOPCRST: 1

  IO port C reset.
- t_u32 IOPDRST: 1

  IO port D reset.
- t_u32 IOPERST: 1

  IO port E reset.
- t_u32 IOPFRST: 1

  IO port F reset.
- t_u32 IOPGRST: 1

  IO port G reset.
- t_u32 ADC1RST: 1

  ADC 1 interface reset.
- t_u32 ADC2RST: 1

  ADC 2 interface reset.
- t_u32 TIM1RST: 1

  TIM1 timer reset.
- t_u32 SPI1RST: 1

  SPI 1 reset.
- t_u32 __pad1__: 1

  Reserved bit(s)
- t_u32 USART1RST: 1

  USART1 reset.
- t_u32 ADC3RST: 1

  ADC 3 interface reset.
- t_u32 __pad2__: 16

  Reserved bit(s)

7.6.1 Detailed Description

RCC APB2 peripheral reset register.

7.6.2 Field Documentation

7.6.2.1 AFIORST

`t_u32 AFIORST`

Alternate function I/O reset.

This field resets the IO port configuration registers

7.6.2.2 __pad0__

`t_u32 __pad0__`

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.6.2.3 IOPARST

`t_u32 IOPARST`

IO port A reset.

This field resets the IO port A configuration registers

7.6.2.4 IOPBRST

`t_u32 IOPBRST`

IO port B reset.

This field resets the IO port B configuration registers

7.6.2.5 IOPCRST

`t_u32 IOPCRST`

IO port C reset.

This field resets the IO port C configuration registers

7.6.2.6 IOPDRST

`t_u32 IOPDRST`

IO port D reset.

This field resets the IO port D configuration registers

7.6.2.7 IOPERST

```
t_u32 IOPERST
```

IO port E reset.

This field resets the IO port E configuration registers

7.6.2.8 IOPFRST

```
t_u32 IOPFRST
```

IO port F reset.

This field resets the IO port F configuration registers

7.6.2.9 IOPGRST

```
t_u32 IOPGRST
```

IO port G reset.

This field resets the IO port G configuration registers

7.6.2.10 ADC1RST

```
t_u32 ADC1RST
```

ADC 1 interface reset.

This field resets the ADC 1 configuration registers

7.6.2.11 ADC2RST

```
t_u32 ADC2RST
```

ADC 2 interface reset.

This field resets the ADC 2 configuration registers

7.6.2.12 TIM1RST

```
t_u32 TIM1RST
```

TIM1 timer reset.

This field resets the TIM1 configuration registers

7.6.2.13 SPI1RST

```
t_u32 SPI1RST
```

SPI 1 reset.

This field resets the SPI 1 configuration registers

7.6.2.14 __pad1__

```
t_u32 __pad1__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.6.2.15 USART1RST

```
t_u32 USART1RST
```

USART1 reset.

This field resets the USART1 configuration registers

7.6.2.16 ADC3RST

```
t_u32 ADC3RST
```

ADC 3 interface reset.

This field resets the ADC 3 configuration registers

7.6.2.17 __pad2__

```
t_u32 __pad2__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

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

- MCAL/RCC/RCC_private.h

7.7 t_RCC_BDCR Struct Reference

RCC Backup domain control register.

```
#include "MCAL/RCC/RCC_private.h"
```

Collaboration diagram for t_RCC_BDCR:

Data Fields

- t_u32 LSEON: 1
  External Low Speed oscillator enable.
- t_u32 LSERDY: 1
  External Low Speed oscillator ready.
- t_u32 LSEBYP: 1
  External Low Speed oscillator bypass.
- t_u32 __pad0__: 5
  Reserved bit(s)
- t_u32 RTCSEL: 2
  RTC clock source selection.
- t_u32 __pad1__: 5
  Reserved bit(s)
- t_u32 RTCEN: 1
  RTC clock enable.
- t_u32 BDRST: 1
  Backup domain software reset.
- t_u32 __pad2__: 15
  Reserved bit(s)

7.7.1 Detailed Description

RCC Backup domain control register.

7.7.2 Field Documentation

7.7.2.1 LSEON

```
t_u32 LSEON
```

External Low Speed oscillator enable.

This field enables the external Low Speed oscillator (LSE)

7.7.2.2 LSERDY

```
t_u32 LSERDY
```

External Low Speed oscillator ready.

This field is set and cleared by hardware to indicate when the external Low Speed oscillator (LSE) is stable.

Warning

This field is read-only

7.7.2.3 LSEBYP

```
t_u32 LSEBYP
```

External Low Speed oscillator bypass.

This field is used to bypass the oscillator with an external clock.

7.7.2.4 __pad0__

```
t_u32 __pad0__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.7.2.5 RTCSEL

```
t_u32 RTCSEL
```

RTC clock source selection.

This field selects the clock source to be used for the RTC.

7.7.2.6 __pad1__

```
t_u32 __pad1__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.7.2.7 RTCEN

```
t_u32 RTCEN
```

RTC clock enable.

This field enables the RTC clock

7.7.2.8 BDRST

```
t_u32 BDRST
```

Backup domain software reset.

This field forces the Backup domain to reset.

7.7.2.9 __pad2__

```
t_u32 __pad2__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

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

- MCAL/RCC/RCC_private.h

7.8 t_RCC_CFGR Struct Reference

RCC clock configuration register.

```
#include "MCAL/RCC/RCC_private.h"
```

Collaboration diagram for t_RCC_CFGR:

Data Fields

- t_u32 SW: 2

  System clock switch.
- t_u32 SWS: 2

  System clock switch status.
- t_u32 HPRE: 4

  AHB prescaler.
- t_u32 PPRE1: 3

  APB Low speed prescaler (APB1)
- t_u32 PPRE2: 3

  APB high-speed prescaler (APB2)
- t_u32 ADCPRE: 2

  ADC prescaler.
- t_u32 PLLSRC: 1

  PLL entry/input clock source.
- t_u32 PLLXTPRE: 1

  LSB of PLL division factor PREDIV1.
- t_u32 PLLMUL: 4

  PLL multiplication factor.
- t_u32 USBPRE: 1

  USB prescaler.
- t_u32 __pad0__: 1

  Reserved bit(s)
- t_u32 MCO: 3

  Microcontroller clock output.
- t_u32 __pad1__: 5

  Reserved bit(s)

7.8.1 Detailed Description

RCC clock configuration register.

7.8.2 Field Documentation

7.8.2.1 SW

```
t_u32 SW
```

System clock switch.

Select the system clock source

7.8.2.2 SWS

```
t_u32 SWS
```

System clock switch status.

These bits are set and cleared by hardware to inform about which clock source is used as the system clock

Warning

This field is read-only

7.8.2.3 HPRE

```
t_u32 HPRE
```

AHB prescaler.

Configure the AHB clock (HCLK) prescaler

7.8.2.4 PPRE1

```
t_u32 PPRE1
```

APB Low speed prescaler (APB1)

Configure the APB1 clock (PCLK1) prescaler

Warning

Max frequency for APB1 is 36 MHz

7.8.2.5 PPRE2

```
t_u32 PPRE2
```

APB high-speed prescaler (APB2)

Configure the APB2 clock (PCLK2) prescaler

Warning

Max frequency for APB2 is 72 MHz

7.8.2.6 ADCPRE

```
t_u32 ADCPRE
```

ADC prescaler.

Configure the ADC clock (ADCCLK) prescaler

Warning

Max frequency for ADC is 14 MHz

7.8.2.7 PLLSRC

```
t_u32 PLLSRC
```

PLL entry/input clock source.

Select the PLL input clock source

7.8.2.8 PLLXTPRE

```
t_u32 PLLXTPRE
```

LSB of PLL division factor PREDIV1.

Select the LSB of the division factor for PLL input clock

7.8.2.9 PLLMUL

```
t_u32 PLLMUL
```

PLL multiplication factor.

Configure the PLL multiplication factor

7.8.2.10 USBPRE

```
t_u32 USBPRE
```

USB prescaler.

Configure the USB clock (USBCLK) prescaler

7.8.2.11 __pad0__

```
t_u32 __pad0__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.8.2.12 MCO

```
t_u32 MCO
```

Microcontroller clock output.

Configure the MCO clock

7.8.2.13 __pad1__

```
t_u32 __pad1__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

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

- MCAL/RCC/RCC_private.h

7.9 t_RCC_CIR Struct Reference

RCC clock interrupt register.

```
#include "MCAL/RCC/RCC_private.h"
```

Collaboration diagram for t_RCC_CIR:

Data Fields

- t_u32 LSIRDYF: 1

  LSI ready interrupt flag.
- t_u32 LSERDYF: 1

  LSE ready interrupt flag.
- t_u32 HSIRDYF: 1

  HSI ready interrupt flag.
- t_u32 HSERDYF: 1

  HSE ready interrupt flag.
- t_u32 PLLRDYF: 1

  PLL ready interrupt flag.
- t_u32 __pad0__: 2

  Reserved bit(s)
- t_u32 CSSF: 1

  Clock security system interrupt flag.
- t_u32 LSIRDYIE: 1

  LSI ready interrupt enable.
- t_u32 LSERDYIE: 1

  LSE ready interrupt enable.
- t_u32 HSIRDYIE: 1

  HSI ready interrupt enable.
- t_u32 HSERDYIE: 1

  HSE ready interrupt enable.
- t_u32 PLLRDYIE: 1

  PLL ready interrupt enable.
- t_u32 __pad1__: 3

  Reserved bit(s)
- t_u32 LSIRDYC: 1

  LSI ready interrupt clear.
- t_u32 LSERDYC: 1

  LSE ready interrupt clear.
- t_u32 HSIRDYC: 1

  HSI ready interrupt clear.
- t_u32 HSERDYC: 1

  HSE ready interrupt clear.
- t_u32 PLLRDYC: 1

  PLL ready interrupt clear.
- t_u32 __pad2__: 2

  Reserved bit(s)
- t_u32 CSSC: 1

  Clock security system interrupt clear.
- t_u32 __pad3__: 8

  Reserved bit(s)

7.9.1 Detailed Description

RCC clock interrupt register.

7.9.2 Field Documentation

7.9.2.1 LSIRDYF

```
t_u32 LSIRDYF
```

LSI ready interrupt flag.

This field indicates whether the LSI oscillator is stable or not

Warning

This field is read-only

7.9.2.2 LSERDYF

```
t_u32 LSERDYF
```

LSE ready interrupt flag.

This field indicates whether the LSE oscillator is stable or not

Warning

This field is read-only

7.9.2.3 HSIRDYF

```
t_u32 HSIRDYF
```

HSI ready interrupt flag.

This field indicates whether the HSI oscillator is stable or not

Warning

This field is read-only

7.9.2.4 HSERDYF

`t_u32 HSERDYF`

HSE ready interrupt flag.

This field indicates whether the HSE oscillator is stable or not

Warning

This field is read-only

7.9.2.5 PLLRDYF

`t_u32 PLLRDYF`

PLL ready interrupt flag.

This field indicates whether the PLL oscillator is stable or not

Warning

This field is read-only

7.9.2.6 __pad0__

`t_u32 __pad0__`

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.9.2.7 CSSF

`t_u32 CSSF`

Clock security system interrupt flag.

This field indicates whether a failure is detected on the external 3.3 V oscillator

Warning

This field is read-only

7.9.2.8 LSIRDYIE

```
t_u32 LSIRDYIE
```

LSI ready interrupt enable.

This field enables the LSI ready interrupt

7.9.2.9 LSERDYIE

```
t_u32 LSERDYIE
```

LSE ready interrupt enable.

This field enables the LSE ready interrupt

7.9.2.10 HSIRDYIE

```
t_u32 HSIRDYIE
```

HSI ready interrupt enable.

This field enables the HSI ready interrupt

7.9.2.11 HSERDYIE

```
t_u32 HSERDYIE
```

HSE ready interrupt enable.

This field enables the HSE ready interrupt

7.9.2.12 PLLRDYIE

```
t_u32 PLLRDYIE
```

PLL ready interrupt enable.

This field enables the PLL ready interrupt

7.9.2.13 __pad1__

```
t_u32 __pad1__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.9.2.14 LSIRDYC

```
t_u32 LSIRDYC
```

LSI ready interrupt clear.

This field clears the LSI ready interrupt

7.9.2.15 LSERDYC

```
t_u32 LSERDYC
```

LSE ready interrupt clear.

This field clears the LSE ready interrupt

7.9.2.16 HSIRDYC

```
t_u32 HSIRDYC
```

HSI ready interrupt clear.

This field clears the HSI ready interrupt

7.9.2.17 HSERDYC

```
t_u32 HSERDYC
```

HSE ready interrupt clear.

This field clears the HSE ready interrupt

7.9.2.18 PLLRDYC

```
t_u32 PLLRDYC
```

PLL ready interrupt clear.

This field clears the PLL ready interrupt

7.9.2.19 __pad2__

```
t_u32 __pad2__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.9.2.20 CSSC

```
t_u32 CSSC
```

Clock security system interrupt clear.

This field clears the clock security system interrupt

7.9.2.21 __pad3__

```
t_u32 __pad3__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

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

- MCAL/RCC/RCC_private.h

7.10 t_RCC_CR Struct Reference

RCC clock control register.

```
#include "MCAL/RCC/RCC_private.h"
```

Collaboration diagram for t_RCC_CR:

Data Fields

- t_u32 HSION: 1

  Internal high-speed clock enable.
- t_u32 HSIRDY: 1

  Internal high-speed clock ready flag.
- t_u32 __pad0__: 1

  Reserved bit(s)
- t_u32 HSITRIM: 5

  Internal high-speed clock trimming.
- t_u32 HSICAL: 8

  Internal high-speed clock calibration.
- t_u32 HSEON: 1

  HSE clock enable.
- t_u32 HSERDY: 1

  External high-speed clock ready flag.
- t_u32 HSEBYP: 1

  External high-speed clock bypass.
- t_u32 CSSON: 1

  Clock security system enable.
- t_u32 __pad1__: 4

  Reserved bit(s)
- t_u32 PLLON: 1

  PLL enable.
- t_u32 PLLRDY: 1

  PLL clock ready flag.
- t_u32 __pad2__: 6

  Reserved bit(s)

7.10.1 Detailed Description

RCC clock control register.

7.10.2 Field Documentation

7.10.2.1 HSION

```
t_u32 HSION
```

Internal high-speed clock enable.

Enable or disable the internal high-speed clock (HSI)

7.10.2.2 HSIRDY

```
t_u32 HSIRDY
```

Internal high-speed clock ready flag.

Whether the internal high-speed clock is ready or not

Warning

This field is read-only

7.10.2.3 __pad0__

```
t_u32 __pad0__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.10.2.4 HSITRIM

```
t_u32 HSITRIM
```

Internal high-speed clock trimming.

Adjusts the internal high-speed clock (HSI) frequency in conjunction with the HSICAL field

Note

Default value: 16

7.10.2.5 HSICAL

```
t_u32 HSICAL
```

Internal high-speed clock calibration.

This field holds the factory calibration value

Warning

This field is read-only

7.10.2.6 HSEON

```
t_u32 HSEON
```

HSE clock enable.

Enable or disable the external high-speed clock (HSE)

7.10.2.7 HSERDY

```
t_u32 HSERDY
```

External high-speed clock ready flag.

Whether the external high-speed clock is ready or not

Warning

This field is read-only

7.10.2.8 HSEBYP

```
t_u32 HSEBYP
```

External high-speed clock bypass.

Bypass the oscillator with an external clock

7.10.2.9 CSSON

```
t_u32 CSSON
```

Clock security system enable.

Enable or disable the clock security system

7.10.2.10 __pad1__

```
t_u32 __pad1__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.10.2.11 PLLON

```
t_u32 PLLON
```

PLL enable.

Enable or disable the PLL

7.10.2.12 PLLRDY

```
t_u32 PLLRDY
```

PLL clock ready flag.

This bit is set by hardware to indicate that the PLL clock is locked

Warning

This field is read-only

7.10.2.13 __pad2__

```
t_u32 __pad2__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

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

- MCAL/RCC/RCC_private.h

7.11 t_RCC_CSR Struct Reference

RCC Control/status register.

```
#include "MCAL/RCC/RCC_private.h"
```

Collaboration diagram for t_RCC_CSR:

Data Fields

- t_u32 LSION: 1

  Internal low speed oscillator enable.
- t_u32 LSIRDY: 1

  Internal low speed oscillator ready.
- t_u32 __pad0__: 22

  Reserved bit(s)
- t_u32 RMVF: 1

  Remove reset flag.
- t_u32 __pad1__: 1

  Reserved bit(s)
- t_u32 PINRSTF: 1

  Pin reset flag.
- t_u32 PORRSTF: 1

  POR/PDR reset flag.
- t_u32 SFTRSTF: 1

  Software reset flag.
- t_u32 IWDGRSTF: 1

  Independent watchdog reset flag.
- t_u32 WWDGRSTF: 1

  Window watchdog reset flag.
- t_u32 LPWRRSTF: 1

  Low-power reset flag.

7.11.1 Detailed Description

RCC Control/status register.

7.11.2 Field Documentation

7.11.2.1 LSION

```
t_u32 LSION
```

Internal low speed oscillator enable.

This field enables the internal low speed oscillator (LSI)

7.11.2.2 LSIRDY

```
t_u32 LSIRDY
```

Internal low speed oscillator ready.

This field is set and cleared by hardware to indicate when the internal low speed oscillator (LSI) is stable.

Warning

This field is read-only

7.11.2.3 __pad0__

```
t_u32 __pad0__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.11.2.4 RMVF

```
t_u32 RMVF
```

Remove reset flag.

This field resets the reset flags.

7.11.2.5 __pad1__

```
t_u32 __pad1__
```

Reserved bit(s)

Attention

This field is reserved and must be kept at reset value.

7.11.2.6 PINRSTF

```
t_u32 PINRSTF
```

Pin reset flag.

This field indicates that a reset has been caused by an NRST pin reset

Warning

This field is read-only

7.11.2.7 PORRSTF

```
t_u32 PORRSTF
```

POR/PDR reset flag.

This field indicates that a reset has been caused by a POR/PDR reset

Warning

This field is read-only

7.11.2.8 SFTRSTF

```
t_u32 SFTRSTF
```

Software reset flag.

This field indicates that a reset has been caused by a software reset

Warning

This field is read-only

7.11.2.9 IWDGRSTF

```
t_u32 IWDGRSTF
```

Independent watchdog reset flag.

This field indicates that a reset has been caused by an independent watchdog reset

Warning

This field is read-only

7.11.2.10 WWDGRSTF

```
t_u32 WWDGRSTF
```

Window watchdog reset flag.

This field indicates that a reset has been caused by a window watchdog reset

Warning

This field is read-only

7.11.2.11 LPWRRSTF

```
t_u32 LPWRRSTF
```

Low-power reset flag.

This field indicates that a reset has been caused by a low-power reset

Warning

This field is read-only

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

- MCAL/RCC/RCC_private.h

7.12 t_RCC_RegisterMap Struct Reference

RCC Register Map.

```
#include "MCAL/RCC/RCC_private.h"
```

Collaboration diagram for t_RCC_RegisterMap:

Data Fields

- t_RCC_CR CR

    Clock control register.
- t_RCC_CFGR CFGR

    Clock configuration register.
- t_RCC_CIR CIR

    Clock interrupt register.
- t_RCC_APB2RSTR APB2RSTR

    RCC APB2 peripheral reset register.
- t_RCC_APB1RSTR APB1RSTR

    RCC APB1 peripheral reset register.
- t_RCC_AHBENR AHBENR

    RCC AHB peripheral clock register.
- t_RCC_APB2ENR APB2ENR

    RCC APB2 peripheral clock enable register.
- t_RCC_APB1ENR APB1ENR

    RCC APB1 peripheral clock enable register.
- t_RCC_BDCR BDCR

    RCC Backup domain control register.
- t_RCC_CSR CSR

    RCC Control/status register.

7.12.1 Detailed Description

RCC Register Map.

7.12.2 Field Documentation

7.12.2.1 CR

```
t_RCC_CR CR
```

Clock control register.

7.12.2.2 CFGR

```
t_RCC_CFGR CFGR
```

Clock configuration register.

7.12.2.3 CIR

`t_RCC_CIR CIR`

Clock interrupt register.

7.12.2.4 APB2RSTR

`t_RCC_APB2RSTR APB2RSTR`

RCC APB2 peripheral reset register.

7.12.2.5 APB1RSTR

`t_RCC_APB1RSTR APB1RSTR`

RCC APB1 peripheral reset register.

7.12.2.6 AHBENR

`t_RCC_AHBENR AHBENR`

RCC AHB peripheral clock register.

7.12.2.7 APB2ENR

`t_RCC_APB2ENR APB2ENR`

RCC APB2 peripheral clock enable register.

7.12.2.8 APB1ENR

`t_RCC_APB1ENR APB1ENR`

RCC APB1 peripheral clock enable register.

7.12.2.9 BDCR

`t_RCC_BDCR BDCR`

RCC Backup domain control register.

7.12.2.10 CSR

`t_RCC_CSR CSR`

RCC Control/status register.

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

- MCAL/RCC/RCC_private.h

Chapter 8

File Documentation

8.1 APPS/APPS_main.c File Reference

This file contains the implementation of the main function that is responsible for running the applications.

```
#include "APPS_main.h"
#include "../COTS/MCAL/RCC/RCC_interface.h"
#include "TestingGPIO/TestApp_TestingGPIO_main.h"
```

Include dependency graph for APPS_main.c:

Macros

- #define TESTING_APPLICATION_MAIN_FUNC(APP_NAME) vTestApp_##APP_NAME##_main()

    This macro is used to define the main function of the application.
- #define TestingGPIO

Functions

- void vAPPS_main (void)

    This function is responsible for running the applications.

8.1.1 Detailed Description

This file contains the implementation of the main function that is responsible for running the applications.

Author

Mohamed Alaa

Version

1.0.0

Date

2023-06-16

8.1.2 Macro Definition Documentation

8.1.2.1 TESTING_APPLICATION_MAIN_FUNC

```
#define TESTING_APPLICATION_MAIN_FUNC(
             APP_NAME ) vTestApp_##APP_NAME##_main()
```

This macro is used to define the main function of the application.

This macro is used to define the main function of the application

Parameters

in	*APP_NAME*	The name of the application

8.1.2.2 TestingGPIO

```
#define TestingGPIO
```

8.2 APPS/APPS_main.h File Reference

This file contains the prototypes of the main function of the testing applications.

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

Functions

- void vAPPS_main (void)

  This function is responsible for running the applications.

8.2.1 Detailed Description

This file contains the prototypes of the main function of the testing applications.

Author

Mohamed Alaa

This file contains the prototypes of the main function that is responsible for running the applications

Version

1.0.0

Date

2023-06-16

8.3 /github/workspace/README.md File Reference

8.4 APPS/README.md File Reference

8.5 APPS/TestingGPIO/TestApp_TestingGPIO_main.c File Reference

This file contains the main implementation for the TestApp_TestingGPIO application.

```
#include "TestApp_TestingGPIO_main.h"
#include "../../COTS/LIB/LSTD_TYPES.h"
#include "../../COTS/MCAL/RCC/RCC_interface.h"
#include "../../COTS/MCAL/GPIO/GPIO_interface.h"
```
Include dependency graph for TestApp_TestingGPIO_main.c:

Functions

- void vTestApp_TestingGPIO_main (void)

  This function is the main function for the TestApp_TestingGPIO application.

8.5.1 Detailed Description

This file contains the main implementation for the TestApp_TestingGPIO application.

Author

Mohamed Alaa

Version

1.0.0

This file contains the main implementation for the TestApp_TestingGPIO application.

Date

2023-06-25

8.6 APPS/TestingGPIO/TestApp_TestingGPIO_main.h File Reference

This file contains the main header file for the TestApp_TestingGPIO application.

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

Functions

- void vTestApp_TestingGPIO_main (void)

  This function is the main function for the TestApp_TestingGPIO application.

8.6.1 Detailed Description

This file contains the main header file for the TestApp_TestingGPIO application.

Author

Mohamed Alaa

Version

1.0.0

This file contains the main header file for the TestApp_TestingGPIO application.

Date

2023-06-25

8.7 LIB/LSTD_BITMATH.h File Reference

This file contains the bit math manipulation macro-functions.

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

Macros

- #define SET_BIT(REG, BITNUM) (REG) |= (1 << (BITNUM))

  Set a certain bit's value.
- #define CLEAR_BIT(REG, BITNUM) (REG) &= ∼(1 << (BITNUM))

  Clear a certain bit's value to.
- #define TOGGLE_BIT(REG, BITNUM) (REG) ^= (1 << (BITNUM))

  Toggle a bit to `0` if it's `1`, `1` otherwise.
- #define GET_BIT(REG, BITNUM) (((REG) >> (BITNUM)) & 1)

  Return the value of the bit whether it's `1` or `0`

8.7.1 Detailed Description

This file contains the bit math manipulation macro-functions.

Author

Mohamed alaa

Version

1.0.0

Date

2023-06-18

8.8 LIB/LSTD_COMPILER.h File Reference

This file contains the compiler standard macros.

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

Macros

- #define CONST const

  Declare a standard constant variable with the specified type.
- #define STATIC static

  Declare a standard static variable/function.
- #define VOLATILE volatile

  Declare a standard volatile variable.
- #define P2VAR(ptrtype) ptrtype ∗

  Declare a pointer-to-variable with the specified type.
- #define P2CONST(ptrtype) CONST ptrtype ∗

  Declare a constant pointer-to-variable with the specified type.
- #define CONSTP2VAR(ptrtype) ptrtype ∗CONST

  Declare a pointer-to-variable constant with the specified type.
- #define CONSTP2CONST(ptrtype) CONST ptrtype ∗CONST

  Declare a constant pointer-to-variable constant with the specified type.
- #define P2FUNC(rettype, fctname) rettype(∗fctname)

  Declare a pointer-to-function with the specified return type.

8.8.1 Detailed Description

This file contains the compiler standard macros.

Author

Mohamed Alaa

Version

1.0.0

Date

2023-06-18

8.9 LIB/LSTD_HW_REGS.h File Reference

This file contains the hardware registers macro-functions for memory addresses mapping and accessing.

```
#include "LSTD_TYPES.h"
#include "LSTD_COMPILER.h"
```
Include dependency graph for LSTD_HW_REGS.h: This graph shows which files directly or indirectly include this file:

Macros

- #define REGISTER_ADDRESS(ADDRESS, OFFSET) ((ADDRESS) + (OFFSET))

  Placeholder for declaring a register address.
- #define REGISTER(REG_TYPE, ADDRESS) (∗(VOLATILE P2VAR(REG_TYPE))(ADDRESS))

  Map to a certain register by its address in the memory.
- #define REGISTER_U8(ADDRESS) REGISTER(t_u8, ADDRESS)

  Map to a certain register by its 8-bit address in the memory (used for 8-bit registers)
- #define REGISTER_U16(ADDRESS) REGISTER(t_u16, ADDRESS)

  Map to a certain register by its 16-bit address in the memory (used for 16-bit registers)
- #define REGISTER_U32(ADDRESS) REGISTER(t_u32, ADDRESS)

  Map to a certain register by its 32-bit address in the memory (used for 32-bit registers)

8.9.1 Detailed Description

This file contains the hardware registers macro-functions for memory addresses mapping and accessing.

Author

Mohamed Alaa

Version

1.0.0

Date

2023-06-18

8.10 LIB/LSTD_SHARED.h File Reference

Shared library for all the standard types.

```
#include "LSTD_TYPES.h"
```
Include dependency graph for LSTD_SHARED.h: This graph shows which files directly or indirectly include this file:

Macros

- #define SET_REGISTER_BIT_STATUS(PERPH, REG, BIT, BSTATUS) (PERPH.REG.BIT = (BSTATUS == TRUE) ? 1U : 0U)

  Set register bit status (1 or 0)

8.10.1 Detailed Description

Shared library for all the standard types.

Author

Mohamed Alaa

Version

1.0.0

Date

2023-06-24

8.11 LIB/LSTD_TYPES.h File Reference

This file contains the standard data types.

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

Typedefs

- typedef unsigned char t_bool

  Type definition for boolean.
- typedef unsigned char t_u8

  Type definition for 8-bit unsigned INT.
- typedef signed char t_s8

  Type definition for 8-bit signed INT.
- typedef unsigned char t_c8

  Type definition for 8-bit char.
- typedef unsigned short int t_u16

  Type definition for 16-bit unsigned int.
- typedef signed short int t_s16

  Type definition for 16-bit signed INT.
- typedef unsigned int t_u32

  Type definition for 32-bit unsigned int.
- typedef signed int t_s32

  Type definition for 32-bit signed INT.
- typedef unsigned long int t_u64

  Type definition for 64-bit unsigned int.
- typedef signed long int t_s64

  Type definition for 64-bit signed int.
- typedef float t_fl32

  Type definition for 32-bit float.
- typedef double t_fl64

  Type definition for 64-bit float.

8.11.1 Detailed Description

This file contains the standard data types.

Author

Mohamed Alaa

Version

1.0.0

Date

2023-06-18

8.12 LIB/LSTD_VALUES.h File Reference

This file contains the standard values.

```
#include "LSTD_TYPES.h"
#include "LSTD_COMPILER.h"
```

Include dependency graph for LSTD_VALUES.h: This graph shows which files directly or indirectly include this file:

Macros

- #define TRUE ((t_bool)1)

  Type definition for TRUE.
- #define FALSE ((t_bool)0)

  Type definition for FALSE.
- #define NULL ((P2VAR(void))0)

  Type definition for NULL.

8.12.1 Detailed Description

This file contains the standard values.

Author

Mohamed Alaa

Version

1.0.0

Date

2023-06-18

8.13 MCAL/GPIO/GPIO_config.h File Reference

This file contains the configuration parameters for the GPIO module.

```
#include "GPIO_interface.h"
```

Include dependency graph for GPIO_config.h: This graph shows which files directly or indirectly include this file:

Macros

- #define DEFAULT_PIN_INPUT_TYPE (GPIO_Input_Type_Pull_Down)

  The default input type for the GPIO pins.
- #define DEFAULT_PIN_OUTPUT_TYPE (GPIO_Output_Type_Push_Pull)

  The default output type for the GPIO pins.

8.13.1 Detailed Description

This file contains the configuration parameters for the GPIO module.

Author

Mohamed Alaa

Version

1.0.0

This file contains the configuration parameters for the GPIO module.

Date

2023-06-25

8.14 MCAL/GPIO/GPIO_interface.h File Reference

This file is the interface file for the GPIO module.

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

Enumerations

- enum t_GPIO_Ports {
  GPIO_Ports_A = 0 , GPIO_Ports_B , GPIO_Ports_C , GPIO_Ports_D ,
  GPIO_Ports_E , GPIO_Ports_F , GPIO_Ports_G }

  GPIO Ports.
- enum t_GPIO_Pins {
  GPIO_Pins_0 = 0 , GPIO_Pins_1 , GPIO_Pins_2 , GPIO_Pins_3 ,
  GPIO_Pins_4 , GPIO_Pins_5 , GPIO_Pins_6 , GPIO_Pins_7 ,
  GPIO_Pins_8 , GPIO_Pins_9 , GPIO_Pins_10 , GPIO_Pins_11 ,
  GPIO_Pins_12 , GPIO_Pins_13 , GPIO_Pins_14 , GPIO_Pins_15 }

  GPIO Pins.
- enum t_GPIO_Direction { GPIO_Direction_Input = 0 , GPIO_Direction_Output_10MHz , GPIO_Direction_Output_2MHz , GPIO_Direction_Output_50MHz }

  GPIO Direction.
- enum t_GPIO_Output_Type { GPIO_Output_Type_Push_Pull = 0 , GPIO_Output_Type_Open_Drain , GPIO_Output_Type_Alternate_Push_Pull , GPIO_Output_Type_Alternate_Open_Drain }

  GPIO Output Type.
- enum t_GPIO_Input_Type { GPIO_Input_Type_Analog = 0 , GPIO_Input_Type_Floating , GPIO_Input_Type_Pull_Down , GPIO_Input_Type_Pull_Up }

  GPIO Input Type.
- enum t_GPIO_Value { GPIO_Value_Low = 0 , GPIO_Value_High }

  GPIO Pin Value.

Functions

- void GPIO_vSetPinDirection (t_GPIO_Ports tPort, t_GPIO_Pins tPin, t_GPIO_Direction tDirection)

  This function is used to set the direction of a GPIO pin.
- void GPIO_vSetPinInputType (t_GPIO_Ports tPort, t_GPIO_Pins tPin, t_GPIO_Input_Type tInputType)

  This function is used to set the input type of a GPIO pin.
- void GPIO_vSetPinOutputType (t_GPIO_Ports tPort, t_GPIO_Pins tPin, t_GPIO_Output_Type tOutputType)

  This function is used to set the output type of a GPIO pin.
- void GPIO_vSetPinValue (t_GPIO_Ports tPort, t_GPIO_Pins tPin, t_GPIO_Value tValue)

  This function is used to set the value of a GPIO pin.
- t_GPIO_Value GPIO_tGetPinValue (t_GPIO_Ports tPort, t_GPIO_Pins tPin)

  This function is used to get the value of a GPIO pin.

8.14.1 Detailed Description

This file is the interface file for the GPIO module.

Author

Mohamed Alaa

Version

1.0.0

This file contains the prototypes of the functions of the GPIO module.

Date

2023-06-25

8.15 MCAL/GPIO/GPIO_private.h File Reference

This file contains the private macros and registers for the GPIO module.

```
#include "../../LIB/LSTD_TYPES.h"
#include "../../LIB/LSTD_HW_REGS.h"
```
Include dependency graph for GPIO_private.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct t_GPIOx_RegisterMap

  GPIO Register Map.

Macros

- #define BASE_ADDRESS_PORT_A REGISTER_ADDRESS(0x40010800, 0)

  Base Address of Port A.
- #define GPIO_A REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_A)

  GPIO Port A.
- #define BASE_ADDRESS_PORT_B REGISTER_ADDRESS(0x40010C00, 0)

  Base Address of Port B.
- #define GPIO_B REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_B)

  GPIO Port B.
- #define BASE_ADDRESS_PORT_C REGISTER_ADDRESS(0x40011000, 0)

  Base Address of Port C.
- #define GPIO_C REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_C)

  GPIO Port C.
- #define BASE_ADDRESS_PORT_D REGISTER_ADDRESS(0x40011400, 0)

  Base Address of Port D.
- #define GPIO_D REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_D)

  GPIO Port D.
- #define BASE_ADDRESS_PORT_E REGISTER_ADDRESS(0x40011800, 0)

  Base Address of Port E.
- #define GPIO_E REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_E)

  GPIO Port E.
- #define BASE_ADDRESS_PORT_F REGISTER_ADDRESS(0x40011C00, 0)

  Base Address of Port F.
- #define GPIO_F REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_F)

  GPIO Port F.
- #define BASE_ADDRESS_PORT_G REGISTER_ADDRESS(0x40012000, 0)

  Base Address of Port G.
- #define GPIO_G REGISTER(t_GPIOx_RegisterMap, BASE_ADDRESS_PORT_G)

  GPIO Port G.
- #define PIN_SHIFT_VALUE (4)

  Pin Shift Value.
- #define PIN_RESET_MASK_VALUE ((t_u32)0x0000000FU)

  Pin Reset Mask Value.
- #define PIN_RESET_CONFIGURATION_MASK_VALUE ((t_u32)0x0000000CU)

  Pin Reset Configuration Mask Value.

- #define PIN_RESET_ODR_MASK_VALUE ((t_u32)0x00000001U)

  Pin Reset ODR Mask Value.
- #define PIN_CONFIGURATION_BITS_SHIFT_VALUE (2)

  Pin Mode Bits Shift Value.
- #define PIN_RESET_SHIFT_VALUE (16)

  Pin Reset Shift Value.
- #define PIN_RESET_MASK(GPIO_PIN_SPAN) ∼(PIN_RESET_MASK_VALUE << GPIO_PIN_SPAN)

  Pin Reset Mask.
- #define PIN_RESET_CONFIGURATIONS_MASK(GPIO_PIN_SPAN) ∼(PIN_RESET_CONFIGURATION_MASK_VALUE << GPIO_PIN_SPAN)

  Pin Reset Configurations Mask.
- #define PIN_RESET_ODR_MASK(GPIO_PIN_SPAN) ∼(PIN_RESET_ODR_MASK_VALUE << GPIO_↩ PIN_SPAN)

  Pin Reset ODR Mask.

8.15.1 Detailed Description

This file contains the private macros and registers for the GPIO module.

Author

Mohamed Alaa

Version

1.0.0

This file contains the private macros and registers for the GPIO module.

Date

2023-06-25

8.16 MCAL/GPIO/GPIO_program.c File Reference

This file contains the implementation for the GPIO module.

```
#include "../../LIB/LSTD_TYPES.h"
#include "../../LIB/LSTD_COMPILER.h"
#include "../../LIB/LSTD_BITMATH.h"
#include "../../LIB/LSTD_VALUES.h"
#include "GPIO_private.h"
#include "GPIO_interface.h"
#include "GPIO_config.h"
```

Include dependency graph for GPIO_program.c:

Macros

- #define IS_PIN_IN_LOW_REGISTER(GPIO_PIN) (GPIO_PIN <= GPIO_Pins_7)

  Check if the pin is in the low register.

Functions

- static void GPIO_vGetPortAddress (t_GPIO_Ports tPort, t_GPIOx_RegisterMap ∗∗ppu32PortBaseAddress)

  This function gets the base address of a GPIO port.
- static t_u8 GPIO_vGetPinSpan (t_GPIO_Pins tPin)

  This function gets the pin span.
- static void GPIO_vSetPinInputTypePullUpDown (t_GPIOx_RegisterMap ∗pu32PortBaseAddress, t_GPIO_Pins tPin, t_GPIO_Input_Type tInputType)

  This function is used to set the pin input type.
- void GPIO_vSetPinDirection (t_GPIO_Ports tPort, t_GPIO_Pins tPin, t_GPIO_Direction tDirection)

  This function is used to set the direction of a GPIO pin.
- void GPIO_vSetPinInputType (t_GPIO_Ports tPort, t_GPIO_Pins tPin, t_GPIO_Input_Type tInputType)

  This function is used to set the input type of a GPIO pin.
- void GPIO_vSetPinOutputType (t_GPIO_Ports tPort, t_GPIO_Pins tPin, t_GPIO_Output_Type tOutputType)

  This function is used to set the output type of a GPIO pin.
- void GPIO_vSetPinValue (t_GPIO_Ports tPort, t_GPIO_Pins tPin, t_GPIO_Value tValue)

  This function is used to set the value of a GPIO pin.
- t_GPIO_Value GPIO_tGetPinValue (t_GPIO_Ports tPort, t_GPIO_Pins tPin)

  This function is used to get the value of a GPIO pin.

8.16.1 Detailed Description

This file contains the implementation for the GPIO module.

Author

Mohamed Alaa

Version

1.0.0

This file contains the implementation for the GPIO module.

Date

2023-06-25

8.16.2 Macro Definition Documentation

8.16.2.1 IS_PIN_IN_LOW_REGISTER

```
#define IS_PIN_IN_LOW_REGISTER(
               GPIO_PIN ) (GPIO_PIN <= GPIO_Pins_7)
```

Check if the pin is in the low register.

This macro checks if the pin is in the low register (CRL)

Parameters

in	*GPIO_PIN*	The pin to check if it is in the low register (CRL) or not

8.16.3 Function Documentation

8.16.3.1 GPIO_vGetPortAddress()

```
static void GPIO_vGetPortAddress (
            t_GPIO_Ports tPort,
            t_GPIOx_RegisterMap ** ppu32PortBaseAddress )  [static]
```

This function gets the base address of a GPIO port.

This function gets the base address of a GPIO port

Parameters

in	*tPort*	The GPIO port to get its base address
out	*ppu32PortBaseAddress*	The base address of the GPIO port

```
{
    P2VAR(VOLATILE t_GPIOx_RegisterMap)
    pu32PortBaseAddress = (P2VAR(t_GPIOx_RegisterMap))NULL;

    switch (tPort)
    {
    case GPIO_Ports_A:
        pu32PortBaseAddress = &GPIO_A;
        break;
    case GPIO_Ports_B:
        pu32PortBaseAddress = &GPIO_B;
        break;
    case GPIO_Ports_C:
        pu32PortBaseAddress = &GPIO_C;
        break;
    case GPIO_Ports_D:
        pu32PortBaseAddress = &GPIO_D;
        break;
    case GPIO_Ports_E:
        pu32PortBaseAddress = &GPIO_E;
        break;
    case GPIO_Ports_F:
        pu32PortBaseAddress = &GPIO_F;
        break;
    case GPIO_Ports_G:
        pu32PortBaseAddress = &GPIO_G;
        break;
    default:
        break;
    }

    *ppu32PortBaseAddress = (P2VAR(t_GPIOx_RegisterMap))pu32PortBaseAddress;
}
```

References GPIO_A, GPIO_B, GPIO_C, GPIO_D, GPIO_E, GPIO_F, GPIO_G, GPIO_Ports_A, GPIO_Ports_B, GPIO_Ports_C, GPIO_Ports_D, GPIO_Ports_E, GPIO_Ports_F, GPIO_Ports_G, NULL, P2VAR, and VOLATILE.

Referenced by GPIO_tGetPinValue(), GPIO_vSetPinDirection(), GPIO_vSetPinInputType(), GPIO_vSetPinOutputType(), and GPIO_vSetPinValue().

Here is the caller graph for this function:

8.16.3.2 GPIO_vGetPinSpan()

```
static t_u8 GPIO_vGetPinSpan (
            t_GPIO_Pins tPin )  [static]
```

This function gets the pin span.

This function gets the pin span, which is the number of bits to shift to reach the target pin mode and configuration bits

Parameters

in	*tPin*	The target pin

Returns

t_u8 The pin span

```
{
    /* Get the pin span (the number of bits to shift to reach the target pin mode and configuration bits)
    */
    return (t_u8)(((tPin <= GPIO_Pins_7) ? tPin : (tPin - GPIO_Pins_8)) * PIN_SHIFT_VALUE);
}
```

References GPIO_Pins_7, GPIO_Pins_8, and PIN_SHIFT_VALUE.

Referenced by GPIO_vSetPinDirection(), GPIO_vSetPinInputType(), and GPIO_vSetPinOutputType().

Here is the caller graph for this function:

8.16.3.3 GPIO_vSetPinInputTypePullUpDown()

```
static void GPIO_vSetPinInputTypePullUpDown (
            t_GPIOx_RegisterMap * pu32PortBaseAddress,
            t_GPIO_Pins tPin,
            t_GPIO_Input_Type tInputType )  [static]
```

This function is used to set the pin input type.

This function is used to set the pin input type in case of it is pull-up or pull-down in the ODR register

Parameters

in	*pu32PortBaseAddress*	The base address of the GPIO port
in	*tPin*	The target pin
in	*tInputType*	The input type of the target pin (pull-up or pull-down)

See also

t_GPIO_Input_Type

```
{
    if (tInputType >= GPIO_Input_Type_Pull_Down)
    {
        /* Get the pin input type (pull-up or pull-down) */
        t_u8 u8PinInputType = GET_BIT(tInputType, 0);

        /* Set the pull-up/pull-down resistor */
        pu32PortBaseAddress->ODR = (pu32PortBaseAddress->ODR & PIN_RESET_ODR_MASK(tPin)) |
    (t_u32)(u8PinInputType « tPin);
    }
    else
    {
        /* Do nothing */
     }
}
```

References GET_BIT, GPIO_Input_Type_Pull_Down, t_GPIOx_RegisterMap::ODR, and PIN_RESET_ODR_MASK.

Referenced by GPIO_vSetPinInputType().

Here is the caller graph for this function:

8.17 MCAL/index.h File Reference

8.18 MCAL/RCC/RCC_config.h File Reference

This file contains the configuration parameters for the RCC module.

```
#include "../../LIB/LSTD_VALUES.h"
#include "RCC_interface.h"
```

Include dependency graph for RCC_config.h: This graph shows which files directly or indirectly include this file:

Macros

- #define RCC_PLL_SRC (RCC_PLLSource_HSE)

  This macro defines the PLL source.
- #define RCC_PLL_MUL (RCC_PLLMulFactors_9)

  This macro defines the PLL multiplication factor.
- #define RCC_PLL_HSE_DIVIDE_BY_2 (TRUE)

  This macro defines the PLL entry HSE divider.
- #define RCC_SYSTEM_CLOCK_SOURCE (RCC_SystemClock_PLL)

  This macro defines the system clock source.
- #define RCC_MCO_SOURCE (RCC_MCOSources_NoClock)

  This macro defines the MCO source.
- #define RCC_ADC_PRESCALER (RCC_ADCPrescaler_DividedBy4)

  This macro defines the ADC clock prescaler.
- #define RCC_AHB_PRESCALER (RCC_AHBPrescaler_NotDivided)

  This macro defines the AHB clock prescaler.
- #define RCC_APB1_PRESCALER (RCC_APBPrescaler_NotDivided)

  This macro defines the APB1 clock prescaler.
- #define RCC_APB2_PRESCALER (RCC_APBPrescaler_NotDivided)

  This macro defines the APB2 clock prescaler.
- #define RCC_USB_PRESCALER (RCC_USBPrescaler_1)

  This macro defines the USB clock prescaler.
- #define RCC_ENABLE_CSS (FALSE)

  This macro defines the clock security system state.

8.18.1 Detailed Description

This file contains the configuration parameters for the RCC module.

Author

Mohamed Alaa

Version

1.0.0

This file contains the configuration parameters for the RCC module.

Date

2023-06-18

8.19 MCAL/RCC/RCC_interface.h File Reference

This file is the interface file for the RCC module.

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

Enumerations

- enum RCC_PLLMulFactors {
  RCC_PLLMulFactors_2 = 0 , RCC_PLLMulFactors_3 , RCC_PLLMulFactors_4 , RCC_PLLMulFactors_5 ,
  RCC_PLLMulFactors_6 , RCC_PLLMulFactors_7 , RCC_PLLMulFactors_8 , RCC_PLLMulFactors_9 ,
  RCC_PLLMulFactors_10 , RCC_PLLMulFactors_11 , RCC_PLLMulFactors_12 , RCC_PLLMulFactors_13 ,
  RCC_PLLMulFactors_14 , RCC_PLLMulFactors_15 , RCC_PLLMulFactors_16 }

  This enum contains the multiplication factors of the PLL.
- enum RCC_AHBPrescaler {
  RCC_AHBPrescaler_NotDivided = 0 , RCC_AHBPrescaler_DividedBy2 = 0b1000 , RCC_AHBPrescaler_DividedBy4
  , RCC_AHBPrescaler_DividedBy8 ,
  RCC_AHBPrescaler_DividedBy16 , RCC_AHBPrescaler_DividedBy64 , RCC_AHBPrescaler_DividedBy128
  , RCC_AHBPrescaler_DividedBy256 ,
  RCC_AHBPrescaler_DividedBy512 }

  This enum contains the prescaler factors of the AHB bus.
- enum RCC_APBPrescaler {
  RCC_APBPrescaler_NotDivided = 0 , RCC_APBPrescaler_DividedBy2 = 0b100 , RCC_APBPrescaler_DividedBy4
  , RCC_APBPrescaler_DividedBy8 ,
  RCC_APBPrescaler_DividedBy16 }

  This enum contains the prescaler factors of the APB (1 & 2) bus.
- enum RCC_ADCPrescaler { RCC_ADCPrescaler_DividedBy2 = 0 , RCC_ADCPrescaler_DividedBy4 ,
  RCC_ADCPrescaler_DividedBy6 , RCC_ADCPrescaler_DividedBy8 }

  This enum contains the prescaler factors of the ADC bus.
- enum RCC_SystemClock { RCC_SystemClock_HSI = 0 , RCC_SystemClock_HSE , RCC_SystemClock_PLL
  }

  This enum contains the system clock sources.
- enum RCC_PLLSource { RCC_PLLSource_HSI_DividedBy2 = 0 , RCC_PLLSource_HSE }

This enum contains the PLL clock sources.

- enum RCC_MCOSources {
  RCC_MCOSources_NoClock = 0 , RCC_MCOSources_SystemClock = 0b100 , RCC_MCOSources_HSI ,
  RCC_MCOSources_HSE ,
  RCC_MCOSources_PLL_DividedBy2 }

  This enum contains the MCO sources.
- enum RCC_USBPrescaler { RCC_USBPrescaler_1_5 = 0 , RCC_USBPrescaler_1 }

  This enum contains the USB prescaler factors.
- enum t_RCC_APB2Peripherals {
  RCC_APB2Peripherals_AFIO = 0 , RCC_APB2Peripherals_PORTA = 2 , RCC_APB2Peripherals_PORTB ,
  RCC_APB2Peripherals_PORTC ,
  RCC_APB2Peripherals_PORTD , RCC_APB2Peripherals_PORTE , RCC_APB2Peripherals_PORTF ,
  RCC_APB2Peripherals_PORTG ,
  RCC_APB2Peripherals_ADC1 , RCC_APB2Peripherals_ADC2 , RCC_APB2Peripherals_TIM1 , RCC_APB2Peripherals_SPI1
  ,
  RCC_APB2Peripherals_USART1 = 14 , RCC_APB2Peripherals_ADC3 }

  This enum contains the APB2 peripherals that are connected to the APB2 bus.
- enum t_RCC_APB1Peripherals {
  RCC_APB1Peripherals_TIM2 = 0 , RCC_APB1Peripherals_TIM3 , RCC_APB1Peripherals_TIM4 ,
  RCC_APB1Peripherals_TIM5 ,
  RCC_APB1Peripherals_TIM6 , RCC_APB1Peripherals_TIM7 , RCC_APB1Peripherals_WWDG = 11 ,
  RCC_APB1Peripherals_SPI2 = 14 ,
  RCC_APB1Peripherals_SPI3 , RCC_APB1Peripherals_USART2 = 17 , RCC_APB1Peripherals_USART3 ,
  RCC_APB1Peripherals_UART4 ,
  RCC_APB1Peripherals_UART5 , RCC_APB1Peripherals_I2C1 , RCC_APB1Peripherals_I2C2 , RCC_APB1Peripherals_USB
  ,
  RCC_APB1Peripherals_CAN = 25 , RCC_APB1Peripherals_BKP = 27 , RCC_APB1Peripherals_PWR ,
  RCC_APB1Peripherals_DAC }

  This enum contains the APB1 peripherals that are connected to the APB1 bus.
- enum t_RCC_AHBPeripherals { RCC_AHBPeripherals_DMA1 = 0 , RCC_AHBPeripherals_DMA2 ,
  RCC_AHBPeripherals_CRC = 5 }

  This enum contains the AHB peripherals that are connected to the AHB bus.

Functions

- void RCC_vInit (void)

  This function initializes the RCC peripheral.
- void RCC_vEnablePeripheralABP2 (t_RCC_APB2Peripherals enuPeripheral)

  This function enables the clock of a peripheral connected to the APB2 bus.
- void RCC_vDisablePeripheralABP2 (t_RCC_APB2Peripherals enuPeripheral)

  This function disables the clock of a peripheral connected to the APB2 bus.
- void RCC_vEnablePeripheralABP1 (t_RCC_APB1Peripherals enuPeripheral)

  This function enables the clock of a peripheral connected to the APB1 bus.
- void RCC_vDisablePeripheralABP1 (t_RCC_APB1Peripherals enuPeripheral)

  This function disables the clock of a peripheral connected to the APB1 bus.
- void RCC_vEnablePeripheralAHB (t_RCC_AHBPeripherals enuPeripheral)

  This function enables the clock of a peripheral connected to the AHB bus.
- void RCC_vDisablePeripheralAHB (t_RCC_AHBPeripherals enuPeripheral)

  This function disables the clock of a peripheral connected to the AHB bus.

8.19.1 Detailed Description

This file is the interface file for the RCC module.

Author

Mohamed Alaa

Version

1.0.0

This file contains the prototypes of the functions of the RCC module.

Date

2023-06-18

8.20 MCAL/RCC/RCC_private.h File Reference

This file contains the private macros and registers for the RCC module.

```
#include "../../LIB/LSTD_HW_REGS.h"
#include "../../LIB/LSTD_TYPES.h"
#include "../../LIB/LSTD_VALUES.h"
#include "../../LIB/LSTD_SHARED.h"
```
Include dependency graph for RCC_private.h:

8.21 MCAL/RCC/RCC_program.c File Reference

This file contains the implementation for the RCC module.

```
#include "../../LIB/LSTD_TYPES.h"
#include "../../LIB/LSTD_BITMATH.h"
#include "../../LIB/LSTD_COMPILER.h"
#include "../../LIB/LSTD_VALUES.h"
#include "RCC_private.h"
#include "RCC_interface.h"
#include "RCC_config.h"
```
Include dependency graph for RCC_program.c:

Functions

- static void RCC_vInitHSEClock (void)

  This function initializes the HSE clock.
- static void RCC_vInitHSIClock (void)

  This function initializes the HSI clock.
- static void RCC_vInitPLL (void)

  This function initializes the PLL clock.
- static void RCC_vInitSystemClock (void)

  This function initializes the system clock.
- static void RCC_vInitMCO (void)

  This function initializes the MCO.
- static void RCC_vSetAPB2PeripheralClockStatus (t_RCC_APB2Peripherals enuPeripheral, t_bool bStatus)

  This function initializes a certain peripheral clock on APB2 bus.
- static void RCC_vSetAPB1PeripheralClockStatus (t_RCC_APB1Peripherals enuPeripheral, t_bool bStatus)

  This function initializes a certain peripheral clock on APB1 bus.
- static void RCC_vSetAHBPeripheralClockStatus (t_RCC_AHBPeripherals enuPeripheral, t_bool bStatus)

  This function initializes a certain peripheral clock on AHB bus.
- static void RCC_vSetADCPreScaler (void)

  This function sets the ADC prescaler.
- static void RCC_vSetUSBPrescaler (void)

  This function sets the USB prescaler.
- static void RCC_vSetAHBPrescaler (void)

  This function sets the AHB prescaler.
- static void RCC_vSetAPB1Prescaler (void)

  This function sets the APB1 prescaler.
- static void RCC_vSetAPB2Prescaler (void)

  This function sets the APB2 prescaler.
- void RCC_vInit (void)

  This function initializes the RCC peripheral.
- void RCC_vEnablePeripheralABP2 (t_RCC_APB2Peripherals enuPeripheral)

  This function enables the clock of a peripheral connected to the APB2 bus.
- void RCC_vDisablePeripheralABP2 (t_RCC_APB2Peripherals enuPeripheral)

  This function disables the clock of a peripheral connected to the APB2 bus.
- void RCC_vEnablePeripheralABP1 (t_RCC_APB1Peripherals enuPeripheral)

  This function enables the clock of a peripheral connected to the APB1 bus.
- void RCC_vDisablePeripheralABP1 (t_RCC_APB1Peripherals enuPeripheral)

  This function disables the clock of a peripheral connected to the APB1 bus.
- void RCC_vEnablePeripheralAHB (t_RCC_AHBPeripherals enuPeripheral)

  This function enables the clock of a peripheral connected to the AHB bus.
- void RCC_vDisablePeripheralAHB (t_RCC_AHBPeripherals enuPeripheral)

  This function disables the clock of a peripheral connected to the AHB bus.

8.21.1 Detailed Description

This file contains the implementation for the RCC module.

Author

Mohamed Alaa

Version

1.0.0

This file contains the implementation for the RCC module.

Date

2023-06-18

8.21.2 Function Documentation

8.21.2.1 RCC_vInitHSEClock()

```
static void RCC_vInitHSEClock (
              void  )  [static]
```

This function initializes the HSE clock.

This function initializes the HSE clock and waits until it is ready

```
{
    /* Enable HSE bypass */
    RCC.CR.HSEBYP = TRUE;
    /* Enable HSE */
    RCC.CR.HSEON = TRUE;
    /* Wait until HSE is ready */
    while (RCC.CR.HSERDY == FALSE)
        ;

    if (RCC_ENABLE_CSS == TRUE)
    {
        /* Enable CSS */
        RCC.CR.CSSON = TRUE;
    }
    else
    {
        /* Disable CSS */
        RCC.CR.CSSON = FALSE;
    }
}
```

References FALSE, RCC, RCC_ENABLE_CSS, and TRUE.

Referenced by RCC_vInit(), and RCC_vInitPLL().

Here is the caller graph for this function:

8.21.2.2 RCC_vInitHSIClock()

```
static void RCC_vInitHSIClock (
              void  )  [static]
```

This function initializes the HSI clock.

This function initializes the HSI clock and waits until it is ready

```
{
    /* Enable HSI */
    RCC.CR.HSION = TRUE;
    /* Wait until HSI is ready */
    while (RCC.CR.HSIRDY == FALSE)
        ;
}
```

References FALSE, RCC, and TRUE.

Referenced by RCC_vInit(), and RCC_vInitPLL().

Here is the caller graph for this function:

8.21.2.3 RCC_vInitPLL()

```
static void RCC_vInitPLL (
              void  )  [static]
```

This function initializes the PLL clock.

This function initializes the PLL clock and waits until it is ready

```
{
    if (RCC_PLL_SRC == RCC_PLLSource_HSE)
    {
        RCC_vInitHSEClock();
    }
    else if (RCC_PLL_SRC == RCC_PLLSource_HSI_DividedBy2)
    {
        RCC_vInitHSIClock();
    }
    else
    {
        /* Do nothing */
    }

    /* Set PLL source to HSE */
    RCC.CFGR.PLLSRC = RCC_PLL_SRC;
    /* Set PLL multiplication factor */
    RCC.CFGR.PLLMUL = RCC_PLL_MUL;
    /* Set PLL entry HSE divider */
    RCC.CFGR.PLLXTPRE = RCC_PLL_HSE_DIVIDE_BY_2;
    /* Enable PLL */
    RCC.CR.PLLON = TRUE;
    /* Wait until PLL is ready */
    while (RCC.CR.PLLRDY == FALSE)
        ;
}
```

References FALSE, RCC, RCC_PLL_HSE_DIVIDE_BY_2, RCC_PLL_MUL, RCC_PLL_SRC, RCC_PLLSource_HSE, RCC_PLLSource_HSI_DividedBy2, RCC_vInitHSEClock(), RCC_vInitHSIClock(), and TRUE.

Referenced by RCC_vInit().

Here is the call graph for this function: Here is the caller graph for this function:

8.21.2.4 RCC_vInitSystemClock()

```
static void RCC_vInitSystemClock (
              void  )  [static]
```

This function initializes the system clock.

This function initializes the system clock

```
{
    /* Set system clock source */
    RCC.CFGR.SW = RCC_SYSTEM_CLOCK_SOURCE;
    /* Wait until system clock source is set */
    while (RCC.CFGR.SWS != RCC_SYSTEM_CLOCK_SOURCE)
        ;
}
```

References RCC, and RCC_SYSTEM_CLOCK_SOURCE.

Referenced by RCC_vInit().

Here is the caller graph for this function:

8.21.2.5 RCC_vInitMCO()

```
static void RCC_vInitMCO (
            void )  [static]
```

This function initializes the MCO.

This function initializes the MCO (Microcontroller Clock Output) and waits until it is ready

```
{
    /* Set MCO source */
    RCC.CFGR.MCO = RCC_MCO_SOURCE;
}
```

References RCC, and RCC_MCO_SOURCE.

Referenced by RCC_vInit().

Here is the caller graph for this function:

8.21.2.6 RCC_vSetAPB2PeripheralClockStatus()

```
static void RCC_vSetAPB2PeripheralClockStatus (
            t_RCC_APB2Peripherals enuPeripheral,
            t_bool bStatus )  [static]
```

This function initializes a certain peripheral clock on APB2 bus.

This function initializes a certain peripheral clock on APB2 bus

Parameters

in	*enuPeripheral*	Peripheral to enable/disable its clock
in	*bStatus*	Enable/Disable peripheral clock

See also

RCC_APB2Peripherals

```
{
    switch (enuPeripheral)
    {
    case RCC_APB2Peripherals_AFIO:
        RCC_SET_REGISTER_BIT_STATUS(APB2ENR, AFIOEN, bStatus);
        break;
    case RCC_APB2Peripherals_PORTA:
        RCC_SET_REGISTER_BIT_STATUS(APB2ENR, IOPAEN, bStatus);
        break;
    case RCC_APB2Peripherals_PORTB:
        RCC_SET_REGISTER_BIT_STATUS(APB2ENR, IOPBEN, bStatus);
        break;
    case RCC_APB2Peripherals_PORTC:
        RCC_SET_REGISTER_BIT_STATUS(APB2ENR, IOPCEN, bStatus);
        break;
    case RCC_APB2Peripherals_PORTD:
        RCC_SET_REGISTER_BIT_STATUS(APB2ENR, IOPDEN, bStatus);
        break;
    case RCC_APB2Peripherals_PORTE:
        RCC_SET_REGISTER_BIT_STATUS(APB2ENR, IOPEEN, bStatus);
        break;
    case RCC_APB2Peripherals_PORTF:
        RCC_SET_REGISTER_BIT_STATUS(APB2ENR, IOPFEN, bStatus);
        break;
    case RCC_APB2Peripherals_PORTG:
        RCC_SET_REGISTER_BIT_STATUS(APB2ENR, IOPGEN, bStatus);
        break;
    case RCC_APB2Peripherals_ADC1:
        RCC_SET_REGISTER_BIT_STATUS(APB2ENR, ADC1EN, bStatus);
        break;
    case RCC_APB2Peripherals_ADC2:
        RCC_SET_REGISTER_BIT_STATUS(APB2ENR, ADC2EN, bStatus);
        break;
    case RCC_APB2Peripherals_TIM1:
        RCC_SET_REGISTER_BIT_STATUS(APB2ENR, TIM1EN, bStatus);
        break;
    case RCC_APB2Peripherals_SPI1:
        RCC_SET_REGISTER_BIT_STATUS(APB2ENR, SPI1EN, bStatus);
        break;
    case RCC_APB2Peripherals_USART1:
        RCC_SET_REGISTER_BIT_STATUS(APB2ENR, USART1EN, bStatus);
        break;
    case RCC_APB2Peripherals_ADC3:
        RCC_SET_REGISTER_BIT_STATUS(APB2ENR, ADC3EN, bStatus);
        break;
    default:
        /* Do nothing */
        break;
    }
}
```

References RCC_APB2Peripherals_ADC1, RCC_APB2Peripherals_ADC2, RCC_APB2Peripherals_ADC3, RCC_APB2Peripherals_AFIO, RCC_APB2Peripherals_PORTA, RCC_APB2Peripherals_PORTB, RCC_APB2Peripherals_PORTC, RCC_APB2Peripherals_PORTD, RCC_APB2Peripherals_PORTE, RCC_APB2Peripherals_PORTF, RCC_APB2Peripherals_PORTG RCC_APB2Peripherals_SPI1, RCC_APB2Peripherals_TIM1, RCC_APB2Peripherals_USART1, and RCC_SET_REGISTER_BIT_ST

Referenced by RCC_vDisablePeripheralABP2(), and RCC_vEnablePeripheralABP2().

Here is the caller graph for this function:

8.21.2.7 RCC_vSetAPB1PeripheralClockStatus()

```
static void RCC_vSetAPB1PeripheralClockStatus (
              t_RCC_APB1Peripherals enuPeripheral,
              t_bool bStatus )  [static]
```

This function initializes a certain peripheral clock on APB1 bus.

This function initializes a certain peripheral clock on APB1 bus

Parameters

in	*enuPeripheral*	Peripheral to enable/disable its clock
in	*bStatus*	Enable/Disable peripheral clock

See also

t_RCC_APB1Peripherals

```
{
    switch (enuPeripheral)
    {
    case RCC_APB1Peripherals_TIM2:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, TIM2EN, bStatus);
        break;
    case RCC_APB1Peripherals_TIM3:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, TIM3EN, bStatus);
        break;
    case RCC_APB1Peripherals_TIM4:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, TIM4EN, bStatus);
        break;
    case RCC_APB1Peripherals_TIM5:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, TIM5EN, bStatus);
        break;
    case RCC_APB1Peripherals_TIM6:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, TIM6EN, bStatus);
        break;
    case RCC_APB1Peripherals_TIM7:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, TIM7EN, bStatus);
        break;
    case RCC_APB1Peripherals_WWDG:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, WWDGEN, bStatus);
        break;
    case RCC_APB1Peripherals_SPI2:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, SPI2EN, bStatus);
        break;
    case RCC_APB1Peripherals_SPI3:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, SPI3EN, bStatus);
        break;
    case RCC_APB1Peripherals_USART2:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, USART2EN, bStatus);
        break;
    case RCC_APB1Peripherals_USART3:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, USART3EN, bStatus);
        break;
    case RCC_APB1Peripherals_UART4:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, UART4EN, bStatus);
        break;
    case RCC_APB1Peripherals_UART5:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, UART5EN, bStatus);
        break;
    case RCC_APB1Peripherals_I2C1:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, I2C1EN, bStatus);
        break;
    case RCC_APB1Peripherals_I2C2:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, I2C2EN, bStatus);
        break;
    case RCC_APB1Peripherals_USB:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, USBEN, bStatus);
        break;
    case RCC_APB1Peripherals_CAN:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, CANEN, bStatus);
        break;
    case RCC_APB1Peripherals_BKP:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, BKPEN, bStatus);
        break;
    case RCC_APB1Peripherals_PWR:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, PWREN, bStatus);
        break;
    case RCC_APB1Peripherals_DAC:
        RCC_SET_REGISTER_BIT_STATUS(APB1ENR, DACEN, bStatus);
        break;
    default:
        /* Do nothing */
        break;
    }
}
```

References RCC_APB1Peripherals_BKP, RCC_APB1Peripherals_CAN, RCC_APB1Peripherals_DAC, RCC_APB1Peripherals_I2C1 RCC_APB1Peripherals_I2C2, RCC_APB1Peripherals_PWR, RCC_APB1Peripherals_SPI2, RCC_APB1Peripherals_SPI3, RCC_APB1Peripherals_TIM2, RCC_APB1Peripherals_TIM3, RCC_APB1Peripherals_TIM4, RCC_APB1Peripherals_TIM5, RCC_APB1Peripherals_TIM6, RCC_APB1Peripherals_TIM7, RCC_APB1Peripherals_UART4, RCC_APB1Peripherals_UART5, RCC_APB1Peripherals_USART2, RCC_APB1Peripherals_USART3, RCC_APB1Peripherals_USB, RCC_APB1Peripherals_WWDG, and RCC_SET_REGISTER_BIT_STATUS.

Referenced by RCC_vDisablePeripheralABP1(), and RCC_vEnablePeripheralABP1().

Here is the caller graph for this function:

8.21.2.8 RCC_vSetAHBPeripheralClockStatus()

```
static void RCC_vSetAHBPeripheralClockStatus (
              t_RCC_AHBPeripherals enuPeripheral,
              t_bool bStatus )  [static]
```

This function initializes a certain peripheral clock on AHB bus.

This function initializes a certain peripheral clock on AHB bus

Parameters

in	*enuPeripheral*	Peripheral to enable/disable its clock
in	*bStatus*	Enable/Disable peripheral clock

See also

t_RCC_AHBPeripherals

```
{
    switch (enuPeripheral)
    {
    case RCC_AHBPeripherals_DMA1:
        RCC_SET_REGISTER_BIT_STATUS(AHBENR, DMA1EN, bStatus);
        break;
    case RCC_AHBPeripherals_DMA2:
        RCC_SET_REGISTER_BIT_STATUS(AHBENR, DMA2EN, bStatus);
        break;
    case RCC_AHBPeripherals_CRC:
        RCC_SET_REGISTER_BIT_STATUS(AHBENR, CRCEN, bStatus);
        break;
    default:
        /* Do nothing */
        break;
    }
}
```

References RCC_AHBPeripherals_CRC, RCC_AHBPeripherals_DMA1, RCC_AHBPeripherals_DMA2, and RCC_SET_REGISTER_BIT_STATUS.

Referenced by RCC_vDisablePeripheralAHB(), and RCC_vEnablePeripheralAHB().

Here is the caller graph for this function:

8.21.2.9 RCC_vSetADCPreScaler()

```
static void RCC_vSetADCPreScaler (
              void ) [static]
```

This function sets the ADC prescaler.

This function sets the ADC prescaler to be used by ADC

```
{
    /* Set ADC prescaler */
    RCC.CFGR.ADCPRE = RCC_ADC_PRESCALER;
}
```

References RCC, and RCC_ADC_PRESCALER.

Referenced by RCC_vInit().

Here is the caller graph for this function:

8.21.2.10 RCC_vSetUSBPrescaler()

```
static void RCC_vSetUSBPrescaler (
              void ) [static]
```

This function sets the USB prescaler.

This function sets the USB prescaler to be used by USB

```
{
    /* Set USB prescaler */
    RCC.CFGR.USBPRE = RCC_USB_PRESCALER;
}
```

References RCC, and RCC_USB_PRESCALER.

Referenced by RCC_vInit().

Here is the caller graph for this function:

8.21.2.11 RCC_vSetAHBPrescaler()

```
static void RCC_vSetAHBPrescaler (
            void ) [static]
```

This function sets the AHB prescaler.

This function sets the AHB prescaler to be used by AHB bus

```
{
    /* Set AHB prescaler */
    RCC.CFGR.HPRE = RCC_AHB_PRESCALER;
}
```

References RCC, and RCC_AHB_PRESCALER.

Referenced by RCC_vInit().

Here is the caller graph for this function:

8.21.2.12 RCC_vSetAPB1Prescaler()

```
static void RCC_vSetAPB1Prescaler (
            void ) [static]
```

This function sets the APB1 prescaler.

This function sets the APB1 prescaler to be used by APB1 bus

```
{
    /* Set APB1 prescaler */
    RCC.CFGR.PPRE1 = RCC_APB1_PRESCALER;
}
```

References RCC, and RCC_APB1_PRESCALER.

Referenced by RCC_vInit().

Here is the caller graph for this function:

8.21.2.13 RCC_vSetAPB2Prescaler()

```
static void RCC_vSetAPB2Prescaler (
            void ) [static]
```

This function sets the APB2 prescaler.

This function sets the APB2 prescaler to be used by APB2 bus

```
{
    /* Set APB2 prescaler */
    RCC.CFGR.PPRE2 = RCC_APB2_PRESCALER;
}
```

References RCC, and RCC_APB2_PRESCALER.

Referenced by RCC_vInit().

Here is the caller graph for this function:

8.22 main.c File Reference

```
#include "APPS/APPS_main.h"
```

Include dependency graph for main.c:

Functions

- int main (void)

8.22.1 Function Documentation

8.22.1.1 main()

```
int main (
            void )
```

```
{
    vAPPS_main();

    for (;;)
    {
    }
}
```

References vAPPS_main().

Here is the call graph for this function:

Index