STM32 Blue Pill Drivers

Generated on Sun Jun 18 2023 13:57:43 for STM32 Blue Pill Drivers by Doxygen 1.9.5

Sun Jun 18 2023 13:57:43

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

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Module Documentation

5.1 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`

5.1.1 Detailed Description

This module contains the bit math manipulation macro-functions.

5.1.2 Macro Definition Documentation

5.1.2.1 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

5.1.2.2 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

5.1.2.3 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

5.1.2.4 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

5.2 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.
- #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.

5.2.1 Detailed Description

This file contains the compiler standard macros.

5.2.2 Macro Definition Documentation

5.2.2.1 CONST

```
#define CONST const
```

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

Declare a standard constant variable with the specified type.

5.2.2.2 STATIC

```
#define STATIC static
```

`#include <LIB/LSTD_COMPILER.h>`

Declare a standard static variable.

5.2.2.3 VOLATILE

```
#define VOLATILE volatile
```

`#include <LIB/LSTD_COMPILER.h>`

Declare a standard volatile variable.

5.2.2.4 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

5.2.2.5 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

5.2.2.6 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

5.2.2.7 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

5.2.2.8 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

5.3 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)

5.3.1 Detailed Description

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

5.3.2 Macro Definition Documentation

5.3.2.1 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

5.3.2.2 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

5.3.2.3 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

5.3.2.4 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

5.3.2.5 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

5.4 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.

5.4.1 Detailed Description

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

5.5 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.

5.5.1 Detailed Description

This module contains the standard data types.

5.5.2 Typedef Documentation

5.5.2.1 t_bool

`t_bool`

`#include <LIB/LSTD_TYPES.h>`

Type definition for boolean.

5.5.2.2 t_u8

`t_u8`

`#include <LIB/LSTD_TYPES.h>`

Type definition for 8-bit unsigned INT.

5.5.2.3 t_s8

`t_s8`

`#include <LIB/LSTD_TYPES.h>`

Type definition for 8-bit signed INT.

5.5.2.4 t_c8

`t_c8`

`#include <LIB/LSTD_TYPES.h>`

Type definition for 8-bit char.

5.5.2.5 t_u16

`t_u16`

`#include <LIB/LSTD_TYPES.h>`

Type definition for 16-bit unsigned int.

5.5.2.6 t_s16

`t_s16`

`#include <LIB/LSTD_TYPES.h>`

Type definition for 16-bit signed INT.

5.5.2.7 t_u32

`t_u32`

`#include <LIB/LSTD_TYPES.h>`

Type definition for 32-bit unsigned int.

5.5.2.8 t_s32

`t_s32`

`#include <LIB/LSTD_TYPES.h>`

Type definition for 32-bit signed INT.

5.5.2.9 t_u64

`t_u64`

`#include <LIB/LSTD_TYPES.h>`

Type definition for 64-bit unsigned int.

5.5.2.10 t_s64

`t_s64`

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

Type definition for 64-bit signed int.

5.5.2.11 t_fl32

`t_fl32`

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

Type definition for 32-bit float.

5.5.2.12 t_fl64

`t_fl64`

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

Type definition for 64-bit float.

5.6 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.

5.6.1 Detailed Description

This module contains the standard values.

5.6.2 Macro Definition Documentation

5.6.2.1 TRUE

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

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

Type definition for TRUE.

5.6.2.2 FALSE

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

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

Type definition for FALSE.

5.6.2.3 NULL

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

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

Type definition for NULL.

Chapter 6

File Documentation

6.1 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 dependency graph for APPS_main.c:

Functions

- void vAPPS_main (void)

  Change this to the macro of the desired application to run.

6.1.1 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

6.1.2 Function Documentation

6.1.2.1 vAPPS_main()

```
void vAPPS_main (
            void )
```

Change this to the macro of the desired application to run.

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

Referenced by main().

Here is the caller graph for this function:

6.2 APPS_main.h File Reference

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

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

Functions

- void vAPPS_main (void)

  Change this to the macro of the desired application to run.

6.2.1 Detailed Description

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

Author

Mohamed Alaa

Version

1.0.0

Date

2023-06-16

6.2.2 Function Documentation

6.2.2.1 vAPPS_main()

```
void vAPPS_main (
             void  )
```

Change this to the macro of the desired application to run.

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

Referenced by main().

Here is the caller graph for this function:

6.3 github/workspace/README.md File Reference

6.4 README.md File Reference

6.5 LIB/LSTD_BITMATH.h File Reference

This file contains the bit math manipulation macro-functions.

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

6.5.1 Detailed Description

This file contains the bit math manipulation macro-functions.

Author

Mohamed alaa

Version

1.0.0

Date

2023-06-18

6.6 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.
- #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.

6.6.1 Detailed Description

This file contains the compiler standard macros.

Author

Mohamed Alaa

Version

1.0.0

Date

2023-06-18

6.7 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:

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)

6.7.1 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

6.8 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.

6.8.1 Detailed Description

This file contains the standard data types.

Author

Mohamed Alaa

Version

1.0.0

Date

2023-06-18

6.9 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:

6.10 main.c File Reference

```
#include "APPS/APPS_main.h"
```
Include dependency graph for main.c:

Functions

- int main (void)

6.10.1 Function Documentation

6.10.1.1 main()

```
int main (
               void  )
```

```
{
    vAPPS_main();

    for (;;)
    {
    }
}
```

References vAPPS_main().

Here is the call graph for this function:

Index