Back to main page

STM8S Firmware Library: *Overview*

The ***STM8S Firmware library*** covers 3 abstraction levels, and includes:

- A complete register address mapping with all bits, bit fields and registers declared in C. This avoids a cumbersome task and more important, it brings the benefits of a bug free reference mapping file, speeding up the early project phase.

- A collection of routines and data structures which covers all peripheral functions (drivers with common API). It can directly be used as a reference framework, since it also includes macros for supporting core-related intrinsic features and common constants and data types definition.

- A set of examples covering all available peripherals with template projects for the most common development toolchains. With the appropriate hardware evaluation board, this allows to get started with a brand new micro within few hours.

Each driver consists of a set of functions covering all peripheral functionalities. The development of each driver is driven by a common API (application programming interface) which standardizes the driver structure, the functions and the parameter names.
The driver source code is developed in 'Strict ANSI-C' (relaxed ANSI-C for projects and example files). It is fully documented and is MISRA-C 2004 compliant (the compliancy matrix is available upon request). Writing the whole library in 'Strict ANSI-C' makes it independent from the software toolchain. Only the start-up files depend on the toolchain.

The Firmware library implements run-time failure detection by checking the input values for all library functions. Such dynamic checking contributes towards enhancing the robustness of the software. Run-time detection is suitable for user application development and debugging. It adds an overhead which can be removed from the final application code to minimize code size and execution speed. For more details refer to Section Run-time checking.

Since the Firmware library is generic and covers all peripheral functionalities, the size and/or execution speed of the application code may not be optimized. For many applications, the library may be used as is. However, for applications having tough constraints in terms of code size and/or execution speed, the library drivers should be used as a reference on how to configure the peripheral and tailor them to specific application requirements.

Note: The performance of application code, in terms of size and/or speed, depends also from the C compiler optimization settings. To help make the

application code smaller, faster or balanced between size and speed you should consider to fine-tune the optimizations according to your application needs.
As example : to have the smallest code size the following options should be enabled, depending on your C compiler :
- **Cosmic** (default configuration used in Firmware library template project):
  - +compact: produce a smaller code but slower than the default behavior.
  - +split: create a separate sub-section per function thus allowing the linker to remove unused functions and variables from the final application image.
- **Raisonance** (default configuration used in Firmware library template project):
  - O(3,SIZE) produce a smaller code but slower than the default behavior.

For more information please refer to your C compiler documentation.

Supported STM8S devices

The STM8S 8-bit Flash microcontrollers family is divided into two product lines, and within each product line there are two main sub-families:

- ***STM8S Performance line*** includes ***STM8S208*** and ***STM8S207*** sub-families
- ***STM8S Access line*** includes ***STM8S105*** and ***STM8S103*** sub-families

The STM8S family also includes some application-specific microcontrollers (ASSM) like the ***STM8S903*** which is derived from the STM8S103 sub-family and which has extra features.

The STM8S Firmware library supports all these sub-families (devices), and by using this library it becomes straightforward to move the application firmware from one STM8S device to another. User has only to select which device he will use by controlling preprocessor define declared in ***stm8s.h*** file (*STM8S208* default device) then the library will be configured accordingly:

stm8s.h

```
...
/* Uncomment the line below according to the target STM8S device used in your
   application.
   Tip: To avoid modifying this file each time you need to switch between these
        devices, you can define the device in your toolchain compiler preprocessor. */
#if !defined (STM8S208) && !defined (STM8S207) && !defined (STM8S105) && !defined (STM8S103) && !defined
(STM8S903)
 #define STM8S208
 /* #define STM8S207 */
 /* #define STM8S105 */
 /* #define STM8S103 */
 /* #define STM8S903 */
#endif
...
```

This define will control the declaration of the following part of the Firmware library:

- IRQ channel definition

- Peripheral memory mapping and physical registers address definition
- Peripheral pointer declaration and driver header file inclusion
- Product miscellaneous configuration: external quartz (HSE) value…
- Peripheral with Features w/ different/incompatible implementation across the family

Note: This define doesn't apply to peripheral drivers, these drivers are always supporting features of the family's superset.

Description of the Firmware library files

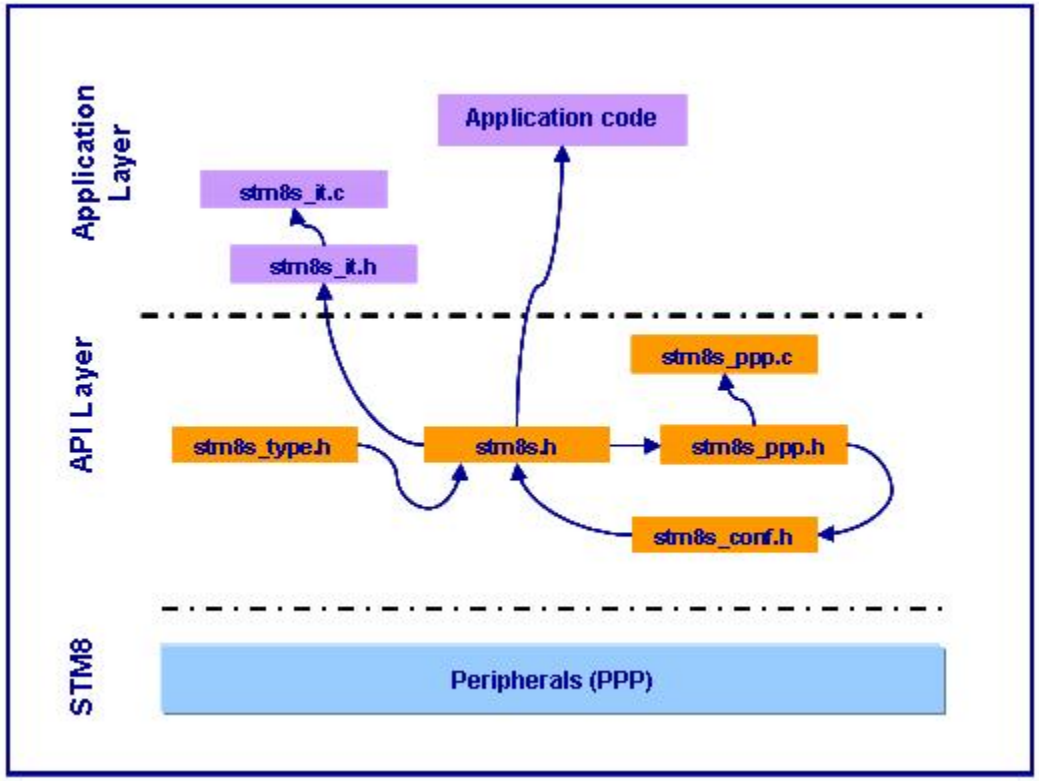
Each peripheral has a source code file ***stm8s_ppp.c*** and a header file ***stm8s_ppp.h***. The ***stm8s_ppp.c*** file contains all the firmware functions required to use the PPP peripheral.
A single memory mapping file, ***stm8s.h***, is supplied for all peripherals. It contains all the register declarations and bits definition. This is the only file that needs to be included in the user application to interface with the library.
The ***stm8s_conf.h*** file is used to specify the set of parameters to interface with the library drivers before running any application.

The table bellow lists and describes the different files used by the Firmware library:

File name	Description
stm8s.h	The file is the unique include file that the application programmer is using in the C source code, usually in main.c. This file contains: ■ data structures and the address mapping for all peripherals ■ peripheral all registers declarations and bits definition ■ macros to access peripherals registers hardware ■ configuration section that allows to select: – the device used in the target application – to use or not the peripherals drivers in application code (i.e. code will be based on direct access to peripherals registers rather than drivers API) , this option is controlled by the #define USE_STDPERIPH_DRIVER
stm8s_type.h	Common declarations file This file includes common types and constants used by all peripheral drivers.
stm8s_conf.h	Peripherals drivers configuration file It must be modified by the user to define the parameters used to interface with the library drivers before running any application. The user can enable or disable peripheral header file inclusion by using the template and can also change few application-specific parameters such as the crystal frequency. This file can also be used to enable or disable the Library run-time failure detection before compiling the firmware library drivers.

stm8s_ppp.h	Header file of PPP peripheral This file includes the PPP peripheral function and variable definitions used within these functions.
stm8s_ppp.c	Driver source code file of PPP peripheral written in C language
stm8s_it.h	Header file including all interrupt handlers prototypes
stm8_interrupt_vector.c	Source file containing the interrupt vector table

The firmware library file inclusions relationship are shown in the following figure:



Run-time checking

The firmware library implements run-time failure detection by checking the input values of all library functions. The run-time checking is achieved by using an ***assert_param*** macro. This macro is used in all library functions which have at least an input parameter. It allows the user to check that the

input value lies within the defined parameter values.

BEEP_Init function example

stm8s_beep.c

```
void BEEP_Init(BEEP_Frequency_TypeDef BEEP_Frequency)
{
 /* Check parameter */
  assert_param(IS_BEEP_FREQ_OK(BEEP_Frequency));
...
}
```

stm8s_beep.h

```
#define IS_BEEP_FREQ_OK(FREQ) \
(((FREQ) == BEEP_FREQ_1KHZ) || \
((FREQ) == BEEP_FREQ_2KHZ) || \
((FREQ) == BEEP_FREQ_4KHZ))
```

If the expression passed to the ***assert_param*** macro is false, the ***assert_failed*** function is called, otherwise nothing happens.

Two implementation of assert_failed function are provided, depending on FULL_ASSERT define (declared in stm8s_conf.h file).

- *#define USE_FULL_ASSERT* line is uncommented, if the expression passed to the ***assert_param*** macro is false, the ***assert_failed*** function will have as input the name of the source file and the source line number of the call that failed.
- *#define USE_FULL_ASSERT* line is commented, if the expression passed to the ***assert_param*** macro is true, the ***assert_failed*** function will have takes no does not returns the name of the source file and the source line number of the call that failed.

The ***assert_param*** macro is implemented in stm8s_conf.h as follows:

```
/**
  * @brief The assert_param macro is used for function's parameters check.
  * @param expr: If expr is false, it calls assert_failed function
  * which reports the name of the source file and the source
  * line number of the call that failed.
  * If expr is true, it returns no value.
  * @retval : None
  */
#ifdef USE_FULL_ASSERT
#define assert_param(expr) ((expr) ? (void)0 : assert_failed((u8 *)__FILE__, __LINE__))
 void assert_failed(u8* file, u32 line);
#else
  #define assert_param(expr) ((void)0)
#endif /* USE_FULL_ASSERT */
```

The ***assert_failed*** function is implemented in the ***main.c*** file or in any other user C files and can be modified by the user in order to take action when an error occurs.

```
/**
  * @brief Reports the name of the source file and the source line number where
  * the assert_param error has occurred.
  * @param file: pointer to the source file name
  * @param line: assert_param error line source number
  * @retval
  * None
  */
#ifdef USE_FULL_ASSERT
void assert_failed(u8* file, u32 line)
{
  /* User can add his own implementation to report the file name and line number,
  ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* Infinite loop */
  while (1)
  {
  }
}
#endif
```

It is recommended to use ***run-time checking*** during application code development and debugging, and to remove it from the final application to improve code size and speed (because of the overhead it introduces).
However if the user wants to keep this functionality in his final application, he can re-use the ***assert_param*** macro defined within the library to test the parameter values before calling the library functions.
