Back to main page

STM8S Firmware Library: *Coding rules and conventions*

The Firmware library uses rules and conventions described in the sections below.

Acronyms

The Table below describes the acronyms used in the firmware library.

Acronym	Peripheral/unit
ADC1 and ADC2	Analog/digital converter
AWU	Auto wakeup
BEEP	Beeper
CAN	Controller area network
CLK	Clock controller
EXTI	External interrupt/event controller
FLASH	Flash memory
GPIO	General purpose I/O
I2C	Inter-integrated circuit
ITC	Interrupt controller
IWDG	Independent watchdog
RST	Reset controller
SPI	Serial peripheral interface

TIM1	16-bit advanced control timer
TIM2, TIM3, TIM5	16-bit general purpose timers
TIM6	8-bit basic timer
UART1, UART2 and UART3	Universal synchronous asynchronous receiver transmitter
WWDG	Window watchdog

Naming conventions

The Firmware library uses the following naming conventions:

- ***PPP*** refers to any peripheral acronym, for example ADC1 and ADC2.
- System and source/header file names are preceded by the prefix '***stm8s_***'.
- Constants used in one file are defined within this file. A constant used in more than one file is defined in a header file. All constants are written in upper case.
- Registers are considered as constants. Their names are in upper case. In most cases, the same acronyms as in the product reference manual document are used.
- Names of peripheral functions are preceded by the corresponding peripheral acronym in upper case followed by an underscore. The first letter in each word is in upper case, for example ***SPI_SendData***. Only one underscore is allowed in a function name to separate the peripheral acronym from the rest of the function name.
- Functions used to initialize the PPP peripheral according to parameters specified in the header file are named ***PPP_Init***, for example ***TIM1_Init***.
- Functions used to reset the PPP peripheral registers to their default values are named ***PPP_DeInit***, for example ***TIM1_DeInit***.
- Functions used to enable or disable the specified PPP peripheral are named ***PPP_Cmd***, for example ***SPI_Cmd***.
- Functions used to enable or disable an interrupt source of the specified PPP peripheral are named ***PPP_ITConfig***, for example ***SPI_ITConfig***.
- Functions used to configure a peripheral function always end with the string 'Config', for example ***TIM1_ForcedOC1Config***.
- Functions used to check whether the specified PPP flag is set or reset are named ***PPP_GetFlagStatus***, for example ***I2C_GetFlagStatus***.
- Functions used to clear a PPP flag are named ***PPP_ClearFlag***, for example ***I2C_ClearFlag***.
- Functions used to check whether the specified PPP interrupt has occurred or not are named ***PPP_GetITStatus***, for example ***TIM1_GetITStatus***.
- Functions used to clear a PPP interrupt pending bit are named ***PPP_ClearITPendingBit***, for example ***TIM1_ClearITPendingBit***.

Coding rules

This section describes the coding rules used in the Firmware library.

Variable types

Specific variable types are already defined with a fixed type and size. These types are defined in the file ***stm8s_type.h***

```
typedef signed long  s32;
typedef signed short s16;
typedef signed char  s8;

typedef signed long  const sc32;  /* Read Only */
typedef signed short const sc16;  /* Read Only */
typedef signed char  const sc8;   /* Read Only */

typedef volatile signed long  vs32;
typedef volatile signed short vs16;
typedef volatile signed char  vs8;

typedef volatile signed long  const vsc32;  /* Read Only */
typedef volatile signed short const vsc16;  /* Read Only */
typedef volatile signed char  const vsc8;   /* Read Only */

typedef unsigned long  u32;
typedef unsigned short u16;
typedef unsigned char  u8;

typedef unsigned long  const uc32;  /* Read Only */
typedef unsigned short const uc16;  /* Read Only */
typedef unsigned char  const uc8;   /* Read Only */

typedef volatile unsigned long  vu32;
typedef volatile unsigned short vu16;
typedef volatile unsigned char  vu8;

typedef volatile unsigned long  const vuc32;  /* Read Only */
typedef volatile unsigned short const vuc16;  /* Read Only */
typedef volatile unsigned char  const vuc8;   /* Read Only */

typedef enum
{
  FALSE = 0,
```

```
  TRUE = !FALSE
}
bool;

typedef enum {
  RESET = 0,
  SET = !RESET
}
FlagStatus, ITStatus, BitStatus;

typedef enum {
  DISABLE = 0,
  ENABLE = !DISABLE
}
FunctionalState;

typedef enum {
  ERROR = 0,
  SUCCESS = !ERROR
}
ErrorStatus;
```

Peripheral registers

Peripheral registers are accessed through a pointer of structure. Each peripheral has its own structure and pointer. All the structures and declarations are defined in the ***stm8s.h*** file.

Registers structure

The example below illustrates the Serial Peripheral Interface (SPI) registers structure declaration:

```
typedef struct SPI_struct
{
  vu8 CR1;    /*!< SPI control register 1 */
  vu8 CR2;    /*!< SPI control register 2 */
  vu8 ICR;    /*!< SPI interrupt control register */
  vu8 SR;     /*!< SPI status register */
  vu8 DR;     /*!< SPI data I/O register */
  vu8 CRCPR;  /*!< SPI CRC polynomial register */
  vu8 RXCRCR; /*!< SPI Rx CRC register */
  vu8 TXCRCR; /*!< SPI Tx CRC register */
}
```

```
SPI_TypeDef;
```

Register names are the register acronyms written in upper case for each peripheral.
RESERVEDi indicates a reserved field (i being an integer that indexes the reserved field).

Peripheral declaration
The following example shows the declaration of the SPI peripheral:

```
#define SPI_BaseAddress         0x5200
...
#define SPI ((SPI_TypeDef *) SPI_BaseAddress)
```

In order to access to the SPI peripheral registers, it is necessary to define the label ***_SPI*** in the ***stm8s_conf.h*** file.
Example:

```
#define _SPI (1)
```

The peripheral registers are accessed as follows:

```
SPI->CR1 = 0x01;
```

Each peripheral has several dedicated registers which contain different flags. Registers are defined within a dedicated structure for each peripheral. Flags are defined as acronyms written in upper case and preceded by '***PPP_FLAG_***'. Flag definition is adapted to each peripheral case and defined in ***stm8s_ppp.h*** file.

Registers reset values
The reset values of all peripheral registers are defined as constants in the ***stm8s.h*** file. They are defined as acronyms written in upper-case into the form:

```
PPP_<register_name>_RESET_VALUE
```

Example:

```
#define SPI_CR1_RESET_VALUE    ((u8)0x00) /*!< Control Register 1 reset value */
#define SPI_CR2_RESET_VALUE    ((u8)0x00) /*!< Control Register 2 reset value */
#define SPI_ICR_RESET_VALUE    ((u8)0x00) /*!< Interrupt Control Register reset value */
#define SPI_SR_RESET_VALUE     ((u8)0x02) /*!< Status Register reset value */
#define SPI_DR_RESET_VALUE     ((u8)0x00) /*!< Data Register reset value */
#define SPI_CRCPR_RESET_VALUE  ((u8)0x07) /*!< Polynomial Register reset value */
```

```
#define SPI_RXCRCR_RESET_VALUE ((u8)0x00) /*!< RX CRC Register reset value */
#define SPI_TXCRCR_RESET_VALUE ((u8)0x00) /*!< TX CRC Register reset value */
```

Registers bits

All the peripheral registers bits are defined as constants in the ***stm8s.h*** file. They are defined as acronyms written in upper-case into the form:

```
PPP_<register_name>_<bit_name>
```

Example:

```
#define SPI_CR1_LSBFIRST ((u8)0x80) /*!< Frame format mask */
#define SPI_CR1_SPE      ((u8)0x40) /*!< Enable bits mask */
#define SPI_CR1_BR       ((u8)0x38) /*!< Baud rate control mask */
#define SPI_CR1_MSTR     ((u8)0x04) /*!< Master Selection mask */
#define SPI_CR1_CPOL     ((u8)0x02) /*!< Clock Polarity mask */
#define SPI_CR1_CPHA     ((u8)0x01) /*!< Clock Phase mask */
```
