STM8S 系列 8 位微控制器 固件函数库

version 1.1.0

北京微芯力科 & 沈阳微扬电机整理

INDEX

STM8S_FWLIB version V1.1.0

****

file stm8s_adc1.   version V1.1.0

****

*****************************   STM8S FWLIB   ***************************************

```
ADC1_DeInit(void);
ADC1_Init(    ADC1_ConvMode_TypeDef ADC1_ConversionMode,
              ADC1_Channel_TypeDef ADC1_Channel,
              ADC1_PresSel_TypeDef ADC1_PrescalerSelection,
              ADC1_ExtTrig_TypeDef ADC1_ExtTrigger,
              FunctionalState ADC1_ExtTriggerState,
              ADC1_Align_TypeDef ADC1_Align,
              ADC1_SchmittTrigg_TypeDef ADC1_SchmittTriggerChannel,
              FunctionalState ADC1_SchmittTriggerState);
ADC1_Cmd(FunctionalState NewState);
ADC1_ScanModeCmd(FunctionalState NewState);
ADC1_DataBufferCmd(FunctionalState NewState);
ADC1_ITConfig(ADC1_IT_TypeDef ADC1_IT,    FunctionalState NewState);
ADC1_PrescalerConfig(ADC1_PresSel_TypeDef ADC1_Prescaler);
ADC1_SchmittTriggerConfig(      ADC1_SchmittTrigg_TypeDef ADC1_SchmittTriggerChannel,
                                FunctionalState NewState);
ADC1_ConversionConfig(    ADC1_ConvMode_TypeDef ADC1_ConversionMode,
                          ADC1_Channel_TypeDef ADC1_Channel,
                          ADC1_Align_TypeDef ADC1_Align);
ADC1_ExternalTriggerConfig(ADC1_ExtTrig_TypeDef ADC1_ExtTrigger,      FunctionalState NewState);
ADC1_AWDChannelConfig(ADC1_Channel_TypeDef Channel,      FunctionalState NewState);
ADC1_StartConversion(void);
ADC1_GetConversionValue(void);
ADC1_SetHighThreshold(u16 Threshold);
ADC1_SetLowThreshold(u16 Threshold);
ADC1_GetBufferValue(u8 Buffer);
ADC1_GetAWDChannelStatus(ADC1_Channel_TypeDef Channel);
ADC1_GetFlagStatus(ADC1_Flag_TypeDef Flag);
```

ADC1_ClearFlag(ADC1_Flag_TypeDef Flag);

ADC1_GetITStatus(ADC1_IT_TypeDef ITPendingBit);

ADC1_ClearITPendingBit(ADC1_IT_TypeDef ITPendingBit);

****

****

ADC1_DeInit(void);

****

```
ADC1_Init(   ADC1_ConvMode_TypeDef ADC1_ConversionMode,
             ADC1_Channel_TypeDef ADC1_Channel,
             ADC1_PresSel_TypeDef ADC1_PrescalerSelection,
             ADC1_ExtTrig_TypeDef ADC1_ExtTrigger,
             FunctionalState ADC1_ExtTriggerState,
             ADC1_Align_TypeDef ADC1_Align,
             ADC1_SchmittTrigg_TypeDef ADC1_SchmittTriggerChannel,
             FunctionalState ADC1_SchmittTriggerState);
```

INPUT :

```
        //ADC1 conversion mode selection
    ADC1_CONVERSIONMODE_SINGLE
    ADC1_CONVERSIONMODE_CONTINUOUS
        //ADC1 analog channel selection
    ADC1_CHANNEL_0
        . . . . . .
    ADC1_CHANNEL_9
        //ADC1 clock prescaler selection
    ADC1_PRESSEL_FCPU_D2
        D3,  D4,  D6,  D8,  D10,  D12
    ADC1_PRESSEL_FCPU_D18
        //ADC1 External conversion trigger event selection
    ADC1_EXTTRIG_TIM
    ADC1_EXTTRIG_GPIO
        //FunctionalState ADC1_ExtTriggerState
    ADC1_EXTTRIG_TIM      = (u8)0x00,
    ADC1_EXTTRIG_GPIO     = (u8)0x10,
        //ADC1 data alignment
    ADC1_ALIGN_LEFT
    ADC1_ALIGN_RIGHT
        //ADC1 schmitt Trigger
    ADC1_SCHMITTTRIG_CHANNEL0
        . . . . . .
    ADC1_SCHMITTTRIG_CHANNEL9
    ADC1_SCHMITTTRIG_ALL
        //FunctionalState ADC1_SchmittTriggerState
    ADC1_SCHMITTTRIG_CHANNEL0    = (u8)0x00,
    ADC1_SCHMITTTRIG_CHANNEL1    = (u8)0x01,
    ADC1_SCHMITTTRIG_CHANNEL2    = (u8)0x02,
    ADC1_SCHMITTTRIG_CHANNEL3    = (u8)0x03,
    ADC1_SCHMITTTRIG_CHANNEL4    = (u8)0x04,
    ADC1_SCHMITTTRIG_CHANNEL5    = (u8)0x05,
    ADC1_SCHMITTTRIG_CHANNEL6    = (u8)0x06,
    ADC1_SCHMITTTRIG_CHANNEL7    = (u8)0x07,
    ADC1_SCHMITTTRIG_CHANNEL8    = (u8)0x08,
    ADC1_SCHMITTTRIG_CHANNEL9    = (u8)0x09,
    ADC1_SCHMITTTRIG_ALL         = (u8)0xFF
```

****

ADC1_Cmd(FunctionalState NewState);

INPUT : DISABLE ; ENABLE

****

ADC1_ScanModeCmd(FunctionalState NewState);

INPUT : DISABLE ; ENABLE

****

ADC1_DataBufferCmd(FunctionalState NewState);

INPUT : DISABLE ; ENABLE

****

ADC1_ITConfig(ADC1_IT_TypeDef ADC1_IT, FunctionalState NewState);

INPUT :

```
        //ADC1 Interrupt source
    ADC1_IT_AWDIE     = (u16)0x10,      /**< Analog WDG interrupt enable */
    ADC1_IT_EOCIE     = (u16)0x20,      /**< EOC iterrupt enable */
    ADC1_IT_AWD       = (u16)0x140,     /**< Analog WDG status */
    ADC1_IT_AWS0      = (u16)0x110,     /**< Analog channel 0 status */
    ADC1_IT_AWS1      = (u16)0x111,     /**< Analog channel 1 status */
    ADC1_IT_AWS2      = (u16)0x112,     /**< Analog channel 2 status */
    ADC1_IT_AWS3      = (u16)0x113,     /**< Analog channel 3 status */
    ADC1_IT_AWS4      = (u16)0x114,     /**< Analog channel 4 status */
    ADC1_IT_AWS5      = (u16)0x115,     /**< Analog channel 5 status */
    ADC1_IT_AWS6      = (u16)0x116,     /**< Analog channel 6 status */
    ADC1_IT_AWS7      = (u16)0x117,     /**< Analog channel 7 status */
    ADC1_IT_AWS8      = (u16)0x118,     /**< Analog channel 8 status */
    ADC1_IT_AWS9      = (u16)0x119,     /**< Analog channel 9 status */
    ADC1_IT_EOC       = (u16)0x80       /**< EOC pending bit */
```

FunctionalState NewState : DISABLE ; ENABLE

****

ADC1_PrescalerConfig(ADC1_PresSel_TypeDef ADC1_Prescaler);

INPUT : //ADC1 clock prescaler selection

```
    ADC1_PRESSEL_FCPU_D2     = (u8)0x00,   /**< Prescaler selection fADC1 = fcpu/2 */
    ADC1_PRESSEL_FCPU_D3     = (u8)0x10,   /**< Prescaler selection fADC1 = fcpu/3 */
    ADC1_PRESSEL_FCPU_D4     = (u8)0x20,   /**< Prescaler selection fADC1 = fcpu/4 */
    ADC1_PRESSEL_FCPU_D6     = (u8)0x30,   /**< Prescaler selection fADC1 = fcpu/6 */
    ADC1_PRESSEL_FCPU_D8     = (u8)0x40,   /**< Prescaler selection fADC1 = fcpu/8 */
    ADC1_PRESSEL_FCPU_D10    = (u8)0x50,   /**< Prescaler selection fADC1 = fcpu/10 */
    ADC1_PRESSEL_FCPU_D12    = (u8)0x60,   /**< Prescaler selection fADC1 = fcpu/12 */
    ADC1_PRESSEL_FCPU_D18    = (u8)0x70    /**< Prescaler selection fADC1 = fcpu/18 */
```

****

ADC1_SchmittTriggerConfig(ADC1_SchmittTrigg_TypeDef ADC1_SchmittTriggerChannel,
FunctionalState NewState);

INPUT: //ADC1 schmitt Trigger

```
    ADC1_SCHMITTTRIG_CHANNEL0   = (u8)0x00,  /**< Schmitt trigger disable on AIN0 */
    ADC1_SCHMITTTRIG_CHANNEL1   = (u8)0x01,  /**< Schmitt trigger disable on AIN1 */
    ADC1_SCHMITTTRIG_CHANNEL2   = (u8)0x02,  /**< Schmitt trigger disable on AIN2 */
    ADC1_SCHMITTTRIG_CHANNEL3   = (u8)0x03,  /**< Schmitt trigger disable on AIN3 */
    ADC1_SCHMITTTRIG_CHANNEL4   = (u8)0x04,  /**< Schmitt trigger disable on AIN4 */
    ADC1_SCHMITTTRIG_CHANNEL5   = (u8)0x05,  /**< Schmitt trigger disable on AIN5 */
    ADC1_SCHMITTTRIG_CHANNEL6   = (u8)0x06,  /**< Schmitt trigger disable on AIN6 */
    ADC1_SCHMITTTRIG_CHANNEL7   = (u8)0x07,  /**< Schmitt trigger disable on AIN7 */
    ADC1_SCHMITTTRIG_CHANNEL8   = (u8)0x08,  /**< Schmitt trigger disable on AIN8 */
    ADC1_SCHMITTTRIG_CHANNEL9   = (u8)0x09,  /**< Schmitt trigger disable on AIN9 */
    ADC1_SCHMITTTRIG_ALL        = (u8)0xFF   /**< Schmitt trigger disable on All channels */
FunctionalState NewState   :   DISABLE  ;  ENABLE
```

****

ADC1_ConversionConfig(ADC1_ConvMode_TypeDef ADC1_ConversionMode,
ADC1_Channel_TypeDef ADC1_Channel,
ADC1_Align_TypeDef ADC1_Align);

INPUT: SEE：**ADC1_Init** ( )

****

ADC1_ExternalTriggerConfig(ADC1_ExtTrig_TypeDefADC1_ExtTrigger, FunctionalState NewState);

INPUT: SEE：**ADC1_Init** ( )

FunctionalState NewState : DISABLE ; ENABLE

****

ADC1_AWDChannelConfig(ADC1_Channel_TypeDef Channel, FunctionalState NewState);

INPUT: SEE：**ADC1_Init( ) -**> ADC1_Channel_TypeDef ADC1_Channel

FunctionalState NewState : DISABLE ; ENABLE

****

ADC1_StartConversion(void);

****

ADC1_GetConversionValue(void);

Return : (u16) ConversionValue

Examples: ADC1ConversionValue= **ADC1_GetConversionValue**( );

****

ADC1_SetHighThreshold(u16 Threshold); // Sets the high threshold of the analog watchdog

INPUT : u16 DATA

****

ADC1_SetLowThreshold(u16 Threshold); // Sets the high threshold of the analog watchdog

INPUT : u16 DATA

****

ADC1_GetBufferValue(u8 Buffer); //Read ADC1ConversionValue from the DATA buffer

INPUT : (u8) Buffer Value

Return : (u16) ADC1ConversionValue

****

ADC1_GetAWDChannelStatus(ADC1_Channel_TypeDef Channel);

// Checks the specified analog watchdog channel status

INPUT : (u8) ADC1_Channel_TypeDef Channel 0 ~ 9

Return ： (u8) ((FlagStatus)status) 0 or 1

****

ADC1_GetFlagStatus(ADC1_Flag_TypeDef Flag);

//Checks the specified ADC1 flag status : REG ADC3_CR3_DBUF

INPUT : //ADC1 flag.

ADC1_FLAG_OVR = (u8)0x41, /**< Overrun status flag */
ADC1_FLAG_AWD = (u8)0x40, /**< Analog WDG status */
ADC1_FLAG_AWS0 = (u8)0x10, /**< Analog channel 0 status */
ADC1_FLAG_AWS1 = (u8)0x11, /**< Analog channel 1 status */
ADC1_FLAG_AWS2 = (u8)0x12, /**< Analog channel 2 status */
ADC1_FLAG_AWS3 = (u8)0x13, /**< Analog channel 3 status */
ADC1_FLAG_AWS4 = (u8)0x14, /**< Analog channel 4 status */
ADC1_FLAG_AWS5 = (u8)0x15, /**< Analog channel 5 status */
ADC1_FLAG_AWS6 = (u8)0x16, /**< Analog channel 6 status */
ADC1_FLAG_AWS7 = (u8)0x17, /**< Analog channel 7 status */
ADC1_FLAG_AWS8 = (u8)0x18, /**< Analog channel 8 status*/
ADC1_FLAG_AWS9 = (u8)0x19, /**< Analog channel 9 status */
ADC1_FLAG_EOC = (u8)0x80 /**< EOC falg */

Return ： 0 or 1 //FlagStatus Status of the ADC1 flag.

****

ADC1_ClearFlag(ADC1_Flag_TypeDef Flag); // Clear the specified ADC1 Flag.

INPUT : //ADC1 flag. SEE **ADC1_GetFlagStatus**( );

****

ADC1_GetITStatus(ADC1_IT_TypeDef ITPendingBit); // Returns the specified pending bit status

INPUT : // ITPendingBit : the IT pending bit to check.

```
        ADC1_IT_AWDIE        = (u16)0x10,       /**< Analog WDG interrupt enable */
        ADC1_IT_EOCIE        = (u16)0x20,       /**< EOC iterrupt enable */
        ADC1_IT_AWD          = (u16)0x140,      /**< Analog WDG status */
        ADC1_IT_AWS0         = (u16)0x110,      /**< Analog channel 0 status */
        ADC1_IT_AWS1         = (u16)0x111,      /**< Analog channel 1 status */
        ADC1_IT_AWS2         = (u16)0x112,      /**< Analog channel 2 status */
        ADC1_IT_AWS3         = (u16)0x113,      /**< Analog channel 3 status */
        ADC1_IT_AWS4         = (u16)0x114,      /**< Analog channel 4 status */
        ADC1_IT_AWS5         = (u16)0x115,      /**< Analog channel 5 status */
        ADC1_IT_AWS6         = (u16)0x116,      /**< Analog channel 6 status */
        ADC1_IT_AWS7         = (u16)0x117,      /**< Analog channel 7 status */
        ADC1_IT_AWS8         = (u16)0x118,      /**< Analog channel 8 status */
        ADC1_IT_AWS9         = (u16)0x119,      /**< Analog channel 9 status */
        ADC1_IT_EOC          = (u16)0x80        /**< EOC pending bit */
```

Return ： 0 or 1 // status of the specified pending bit.

****

ADC1_ClearITPendingBit(ADC1_IT_TypeDef ITPendingBit);

INPUT : SEE **ADC1_GetITStatus**( );

**************************** STM8S FWLIB **************************************

*****************************   STM8S FWLIB   ***************************************

file stm8s_beep.   version V1.1.0

****

BEEP_DeInit(void);
BEEP_Init(BEEP_Frequency_TypeDef BEEP_Frequency);
BEEP_Cmd(FunctionalState NewState);
BEEP_LSICalibrationConfig(u32 LSIFreqHz);

****

****

BEEP_DeInit(void);   // Deinitializes the BEEP peripheral registers to their default reset

****

BEEP_Init(BEEP_Frequency_TypeDef BEEP_Frequency);

// Initializes the BEEP function according to the specified parameters.

INPUT : // BEEP_Frequency Frequency selection.

```
BEEP_FREQUENCY_1KHZ     = (u8)0x00,   /*!< Beep signal output frequency equals to 1 KHz */
BEEP_FREQUENCY_2KHZ     = (u8)0x40,   /*!< Beep signal output frequency equals to 2 KHz */
BEEP_FREQUENCY_4KHZ     = (u8)0x80    /*!< Beep signal output frequency equals to 4 KHz */
```

****

BEEP_Cmd(FunctionalState NewState);

INPUT :   DISABLE ; ENABLE

****

BEEP_LSICalibrationConfig(u32 LSIFreqHz);

// Update CSR register with the measured LSI frequency.

INPUT :   u32 LSIFreqHz

*****************************   STM8S FWLIB   ***************************************

```
****************************      STM8S FWLIB      ***************************************

   file stm8s_clk.          version V1.1.0

******************************************************************************************

CLK_DeInit    (void);

CLK_HSECmd    (FunctionalState NewState);

CLK_HSICmd    (FunctionalState NewState);

CLK_LSICmd    (FunctionalState NewState);

CLK_CCOCmd    (FunctionalState NewState);

CLK_ClockSwitchCmd    (FunctionalState NewState);

CLK_FastHaltWakeUpCmd    (FunctionalState NewState);

CLK_SlowActiveHaltWakeUpCmd    (FunctionalState NewState);

CLK_PeripheralClockConfig    (CLK_Peripheral_TypeDef CLK_Peripheral,    FunctionalState NewState);

CLK_ClockSwitchConfig            (CLK_SwitchMode_TypeDef  CLK_SwitchMode,    CLK_Source_TypeDef

                                 CLK_NewClock,

                                 FunctionalState ITState,

                                 CLK_CurrentClockState_TypeDef CLK_CurrentClockState);

CLK_HSIPrescalerConfig    (CLK_Prescaler_TypeDef HSIPrescaler);

CLK_CCOConfig    (CLK_Output_TypeDef CLK_CCO);

CLK_ITConfig    (CLK_IT_TypeDef CLK_IT, FunctionalState NewState);

CLK_SYSCLKConfig    (CLK_Prescaler_TypeDef CLK_Prescaler);

CLK_SWIMConfig    (CLK_SWIMDivider_TypeDef CLK_SWIMDivider);

CLK_CANConfig    (CLK_CANDivider_TypeDef CLK_CANDivider);

CLK_ClockSecuritySystemEnable    (void);

CLK_SYSCLKEmergencyClear    (void);

CLK_AdjustHSICalibrationValue    (CLK_HSITrimValue_TypeDef CLK_HSICalibrationValue);

CLK_GetClockFreq    (void);

CLK_GetSYSCLKSource    (void);

CLK_GetFlagStatus    (CLK_Flag_TypeDef CLK_FLAG);

CLK_GetITStatus    (CLK_IT_TypeDef CLK_IT);

CLK_ClearITPendingBit    (CLK_IT_TypeDef CLK_IT);
******************************************************************************************
```

****

CLK_DeInit (void);

****

CLK_HSECmd (FunctionalState NewState);

INPUT : DISABLE ; ENABLE

****

CLK_HSICmd (FunctionalState NewState);

INPUT : DISABLE ; ENABLE

****

CLK_LSICmd (FunctionalState NewState);

INPUT : DISABLE ; ENABLE

****

CLK_CCOCmd (FunctionalState NewState); // Enables or disablle the Configurable Clock Output

---

INPUT : DISABLE ; ENABLE

****

CLK_ClockSwitchCmd (FunctionalState NewState); // Starts or Stops manually clock switch execution

---

INPUT : DISABLE ; ENABLE

****

CLK_FastHaltWakeUpCmd (FunctionalState NewState);

INPUT : DISABLE ; ENABLE

****

CLK_SlowActiveHaltWakeUpCmd (FunctionalState NewState); //Configures the slow active halt wake up

---

INPUT : DISABLE ; ENABLE

****

CLK_PeripheralClockConfig (CLK_Peripheral_TypeDef CLK_Peripheral,
FunctionalState NewState);

INPUT : // CLK Enable peripheral

```
    CLK_PERIPHERAL_I2C       = (u8)0x00,   /*!< Peripheral Clock Enable 1, I2C */
    CLK_PERIPHERAL_SPI       = (u8)0x01,   /*!< Peripheral Clock Enable 1, SPI */
    CLK_PERIPHERAL_UART1     = (u8)0x02,   /*!< Peripheral Clock Enable 1, UART1 */
    CLK_PERIPHERAL_UART2     = (u8)0x03,   /*!< Peripheral Clock Enable 1, UART2 */
    CLK_PERIPHERAL_UART3     = (u8)0x03,   /*!< Peripheral Clock Enable 1, UART3 */
    CLK_PERIPHERAL_TIMER6    = (u8)0x04,   /*!< Peripheral Clock Enable 1, Timer6 */
    CLK_PERIPHERAL_TIMER4    = (u8)0x04,   /*!< Peripheral Clock Enable 1, Timer4 */
    CLK_PERIPHERAL_TIMER5    = (u8)0x05,   /*!< Peripheral Clock Enable 1, Timer5 */
    CLK_PERIPHERAL_TIMER2    = (u8)0x05,   /*!< Peripheral Clock Enable 1, Timer2 */
    CLK_PERIPHERAL_TIMER3    = (u8)0x06,   /*!< Peripheral Clock Enable 1, Timer3 */
    CLK_PERIPHERAL_TIMER1    = (u8)0x07,   /*!< Peripheral Clock Enable 1, Timer1 */
    CLK_PERIPHERAL_AWU       = (u8)0x12,   /*!< Peripheral Clock Enable 2, AWU */
    CLK_PERIPHERAL_ADC          = (u8)0x13,    /*!< Peripheral Clock Enable 2, ADC */
    CLK_PERIPHERAL_CAN          = (u8)0x17     /*!< Peripheral Clock Enable 2, CAN */
```

FunctionalState NewState   :   DISABLE ; ENABLE

****

```
    CLK_ClockSwitchConfig ( CLK_SwitchMode_TypeDef   CLK_SwitchMode,
                            CLK_Source_TypeDef CLK_NewClock,
                            FunctionalState ITState,
                            CLK_CurrentClockState_TypeDef CLK_CurrentClockState   );
```

INPUT :

//Switch Mode Auto, Manual.

```
    CLK_SWITCHMODE_MANUAL       = (u8)0x00,    /*!< Enable the manual clock switching mode */
    CLK_SWITCHMODE_AUTO         = (u8)0x01     /*!< Enable the automatic clock switching mode */
```

//CLK Clock Source.

```
    CLK_SOURCE_HSI              = (u8)0xE1,    /*!< Clock Source HSI. */
    CLK_SOURCE_LSI              = (u8)0xD2,    /*!< Clock Source LSI. */
    CLK_SOURCE_HSE              = (u8)0xB4     /*!< Clock Source HSE. */
```

//FunctionalState ITState

```
    DISABLE  ;  ENABLE
```

//CLK_CurrentClockState_TypeDef

```
    CLK_CURRENTCLOCKSTATE_DISABLE     = (u8)0x00,    /*!< Current clock disable */
    CLK_CURRENTCLOCKSTATE_ENABLE      = (u8)0x01     /*!< Current clock enable */
```

Return   :   SUCCESS   or   ERROR;

****

```
    CLK_HSIPrescalerConfig   (CLK_Prescaler_TypeDef HSIPrescaler);
```

INPUT : //CLK Clock Divisor.

```
    CLK_PRESCALER_HSIDIV1       = (u8)0x00,    /*!< High speed internal clock prescaler: 1 */
    CLK_PRESCALER_HSIDIV2       = (u8)0x08,    /*!< High speed internal clock prescaler: 2 */
    CLK_PRESCALER_HSIDIV4       = (u8)0x10,    /*!< High speed internal clock prescaler: 4 */
    CLK_PRESCALER_HSIDIV8       = (u8)0x18,    /*!< High speed internal clock prescaler: 8 */
    CLK_PRESCALER_CPUDIV1       = (u8)0x80,    /*!< CPU clock division factors 1 */
    CLK_PRESCALER_CPUDIV2       = (u8)0x81,    /*!< CPU clock division factors 2 */
    CLK_PRESCALER_CPUDIV4       = (u8)0x82,    /*!< CPU clock division factors 4 */
    CLK_PRESCALER_CPUDIV8       = (u8)0x83,    /*!< CPU clock division factors 8 */
    CLK_PRESCALER_CPUDIV16      = (u8)0x84,    /*!< CPU clock division factors 16 */
    CLK_PRESCALER_CPUDIV32      = (u8)0x85,    /*!< CPU clock division factors 32 */
    CLK_PRESCALER_CPUDIV64      = (u8)0x86,    /*!< CPU clock division factors 64 */
    CLK_PRESCALER_CPUDIV128     = (u8)0x87     /*!< CPU clock division factors 128 */
```

****

```
    CLK_CCOConfig   (CLK_Output_TypeDef CLK_CCO);
```

INPUT : //CLK Clock Output

```
    CLK_OUTPUT_HSI          = (u8)0x00,    /*!< Clock Output HSI */
    CLK_OUTPUT_LSI          = (u8)0x02,    /*!< Clock Output LSI */
    CLK_OUTPUT_HSE          = (u8)0x04,    /*!< Clock Output HSE */
    CLK_OUTPUT_CPU          = (u8)0x08,    /*!< Clock Output CPU */
    CLK_OUTPUT_CPUDIV2       = (u8)0x0A,   /*!< Clock Output CPU/2 */
    CLK_OUTPUT_CPUDIV4       = (u8)0x0C,   /*!< Clock Output CPU/4 */
    CLK_OUTPUT_CPUDIV8       = (u8)0x0E,   /*!< Clock Output CPU/8 */
    CLK_OUTPUT_CPUDIV16      = (u8)0x10,   /*!< Clock Output CPU/16 */
    CLK_OUTPUT_CPUDIV32      = (u8)0x12,   /*!< Clock Output CPU/32 */
    CLK_OUTPUT_CPUDIV64      = (u8)0x14,   /*!< Clock Output CPU/64 */
    CLK_OUTPUT_HSIRC         = (u8)0x16,   /*!< Clock Output HSI RC */
    CLK_OUTPUT_MASTER        = (u8)0x18,   /*!< Clock Output Master */
    CLK_OUTPUT_OTHERS        = (u8)0x1A    /*!< Clock Output OTHER */
```

****

CLK_ITConfig   (CLK_IT_TypeDef CLK_IT, FunctionalState NewState);

INPUT :   //CLK interrupt configuration and Flags cleared by software.

```
    CLK_IT_CSSD    = (u8)0x0C,   /*!< Clock security system detection Flag */
    CLK_IT_SWIF    = (u8)0x1C    /*!< Clock switch interrupt Flag */
```

****

CLK_SYSCLKConfig   (CLK_Prescaler_TypeDef CLK_Prescaler);

INPUT :   //CLK Clock Divisor.

```
    CLK_PRESCALER_HSIDIV1        = (u8)0x00,   /*!< High speed internal clock prescaler: 1 */
    CLK_PRESCALER_HSIDIV2        = (u8)0x08,   /*!< High speed internal clock prescaler: 2 */
    CLK_PRESCALER_HSIDIV4        = (u8)0x10,   /*!< High speed internal clock prescaler: 4 */
    CLK_PRESCALER_HSIDIV8        = (u8)0x18,   /*!< High speed internal clock prescaler: 8 */
    CLK_PRESCALER_CPUDIV1        = (u8)0x80,   /*!< CPU clock division factors 1 */
    CLK_PRESCALER_CPUDIV2        = (u8)0x81,   /*!< CPU clock division factors 2 */
    CLK_PRESCALER_CPUDIV4        = (u8)0x82,   /*!< CPU clock division factors 4 */
    CLK_PRESCALER_CPUDIV8        = (u8)0x83,   /*!< CPU clock division factors 8 */
    CLK_PRESCALER_CPUDIV16       = (u8)0x84,   /*!< CPU clock division factors 16 */
    CLK_PRESCALER_CPUDIV32       = (u8)0x85,   /*!< CPU clock division factors 32 */
    CLK_PRESCALER_CPUDIV64       = (u8)0x86,   /*!< CPU clock division factors 64 */
    CLK_PRESCALER_CPUDIV128      = (u8)0x87    /*!< CPU clock division factors 128 *
```

****

CLK_SWIMConfig   (CLK_SWIMDivider_TypeDef CLK_SWIMDivider);

INPUT :   //SWIM Clock divider.

```
    CLK_SWIMDIVIDER_2        = (u8)0x00,   /*!< SWIM clock is divided by 2 */
    CLK_SWIMDIVIDER_OTHER    = (u8)0x01    /*!< SWIM clock is not divided by 2 */
```

****

CLK_CANConfig   (CLK_CANDivider_TypeDef CLK_CANDivider);

INPUT :   //External CAN clock dividern.

```
    CLK_CANDIVIDER_1     = (u8)0x00,   /*!< External CAN clock = HSE/1 */
    CLK_CANDIVIDER_2     = (u8)0x01,   /*!< External CAN clock = HSE/2 */
    CLK_CANDIVIDER_3     = (u8)0x02,   /*!< External CAN clock = HSE/3 */
    CLK_CANDIVIDER_4     = (u8)0x03,   /*!< External CAN clock = HSE/4 */
    CLK_CANDIVIDER_5     = (u8)0x04,   /*!< External CAN clock = HSE/5 */
    CLK_CANDIVIDER_6     = (u8)0x05,   /*!< External CAN clock = HSE/6 */
    CLK_CANDIVIDER_7        = (u8)0x06,    /*!< External CAN clock = HSE/7 */
    CLK_CANDIVIDER_8        = (u8)0x07     /*!< External CAN clock = HSE/8 */
```

****

CLK_ClockSecuritySystemEnable (void); // Enables the Clock Security System.

****

CLK_SYSCLKEmergencyClear (void); // Reset the SWBSY flag (SWICR Reister)

****

CLK_AdjustHSICalibrationValue (CLK_HSITrimValue_TypeDef CLK_HSICalibrationValue);

INPUT : //CLK HSI Calibration Value.

```
    CLK_HSITRIMVALUE_0      = (u8)0x00,   /*!< HSI Calibtation Value 0 */
    CLK_HSITRIMVALUE_1      = (u8)0x01,   /*!< HSI Calibtation Value 1 */
    CLK_HSITRIMVALUE_2      = (u8)0x02,   /*!< HSI Calibtation Value 2 */
    CLK_HSITRIMVALUE_3      = (u8)0x03,   /*!< HSI Calibtation Value 3 */
    CLK_HSITRIMVALUE_4      = (u8)0x04,   /*!< HSI Calibtation Value 4 */
    CLK_HSITRIMVALUE_5      = (u8)0x05,   /*!< HSI Calibtation Value 5 */
    CLK_HSITRIMVALUE_6      = (u8)0x06,   /*!< HSI Calibtation Value 6 */
    CLK_HSITRIMVALUE_7      = (u8)0x07    /*!< HSI Calibtation Value 7 */
```

****

CLK_GetClockFreq (void); //eturns the frequencies of different on chip clocks.

Return : ((u32)clockfrequency)

Examples: (u32)clockfrequency = CLK_GetClockFreq ();

****

CLK_GetSYSCLKSource (void);

Return : // Returns the clock source used as system clock.

```
    (u8)0xE1,     /*!< Clock Source HSI. */
    (u8)0xD2,     /*!< Clock Source LSI. */
    (u8)0xB4      /*!< Clock Source HSE. */
```

****

CLK_GetFlagStatus (CLK_Flag_TypeDef CLK_FLAG);

// Checks whether the specified CLK flag is set or not.

INPUT : // CLK_FLAG Flag to check.

```
    CLK_FLAG_LSIRDY         = (u16)0x0110,    /*!< Low speed internal oscillator ready Flag */
    CLK_FLAG_HSIRDY         = (u16)0x0102,    /*!< High speed internal oscillator ready Flag */
    CLK_FLAG_HSERDY         = (u16)0x0202,    /*!< High speed external oscillator ready Flag */
    CLK_FLAG_SWIF           = (u16)0x0308,    /*!< Clock switch interrupt Flag */
    CLK_FLAG_SWBSY          = (u16)0x0301,    /*!< Switch busy Flag */
    CLK_FLAG_CSSD           = (u16)0x0408,    /*!< Clock security system detection Flag */
    CLK_FLAG_AUX            = (u16)0x0402,    /*!< Auxiliary oscillator connected to master clock */
    CLK_FLAG_CCOBSY         = (u16)0x0504,    /*!< Configurable clock output busy */
    CLK_FLAG_CCORDY         = (u16)0x0502     /*!< Configurable clock output ready */
```

Return : RESET or SET // FlagStatus, status of the checked flag

****

CLK_GetITStatus (CLK_IT_TypeDef CLK_IT);

INPUT : // CLK_IT specifies the CLK interrupt.

CLK_IT_CSSD = (u8)0x0C, /*!< Clock security system detection Flag */

CLK_IT_SWIF = (u8)0x1C /*!< Clock switch interrupt Flag */

Return : RESET or SET: //ITStatus, new state of CLK_IT (SET or RESET).

****

CLK_ClearITPendingBit (CLK_IT_TypeDef CLK_IT);

INPUT : //CLK_IT specifies the interrupt pending bits. SEE **CLK_GetITStatus** ( );

**************************** STM8S FWLIB **************************************

****************************   STM8S FWLIB   ***************************************

file stm8s_exti.

****

EXTI_DeInit(void);

EXTI_SetExtIntSensitivity(EXTI_Port_TypeDef Port,   EXTI_Sensitivity_TypeDef SensitivityValue);

EXTI_SetTLISensitivity(EXTI_TLISensitivity_TypeDef SensitivityValue   );

EXTI_GetExtIntSensitivity(EXTI_Port_TypeDef Port   );

EXTI_GetTLISensitivity(void);

****

****

EXTI_DeInit(void);   //Deinitializes the external interrupt control registers to their default reset value.

****

EXTI_SetExtIntSensitivity(EXTI_Port_TypeDef Port,   EXTI_Sensitivity_TypeDef SensitivityValue);

//Set the external interrupt sensitivity of the selected port.

INPUT :   // EXTI PortNum possible values

```
EXTI_PORT_GPIOA    = (u8)0x00,    /*!< GPIO Port A */
EXTI_PORT_GPIOB    = (u8)0x01,    /*!< GPIO Port B */
EXTI_PORT_GPIOC    = (u8)0x02,    /*!< GPIO Port C */
EXTI_PORT_GPIOD    = (u8)0x03,    /*!< GPIO Port D */
EXTI_PORT_GPIOE    = (u8)0x04     /*!< GPIO Port E */
          // EXTI Sensitivity values for PORTA to PORTE
EXTI_SENSITIVITY_FALL_LOW     = (u8)0x00,    /*!< Interrupt on Falling edge and Low level */
EXTI_SENSITIVITY_RISE_ONLY    = (u8)0x01,    /*!< Interrupt on Rising edge only */
EXTI_SENSITIVITY_FALL_ONLY    = (u8)0x02,    /*!< Interrupt on Falling edge only */
EXTI_SENSITIVITY_RISE_FALL    = (u8)0x03     /*!< Interrupt on Rising and Falling edges */
```

****

EXTI_SetTLISensitivity(EXTI_TLISensitivity_TypeDef SensitivityValue);

// Set the TLI interrupt sensitivity.

INPUT :   //EXTI Sensitivity values for TLI

```
EXTI_TLISENSITIVITY_FALL_ONLY = (u8)0x00,      /*!< Top Level Interrupt on Falling edge only */
EXTI_TLISENSITIVITY_RISE_ONLY = (u8)0x04       /*!< Top Level Interrupt on Rising edge only */
```

****

EXTI_GetExtIntSensitivity(EXTI_Port_TypeDef Port);

// Get the external interrupt sensitivity of the selected port.

INPUT :   // Port The port number to access.

```
EXTI_PORT_GPIOA = (u8)0x00,      /*!< GPIO Port A */
EXTI_PORT_GPIOB = (u8)0x01,      /*!< GPIO Port B */
EXTI_PORT_GPIOC = (u8)0x02,      /*!< GPIO Port C */
EXTI_PORT_GPIOD = (u8)0x03,      /*!< GPIO Port D */
```

EXTI_PORT_GPIOE = (u8)0x04 /*!< GPIO Port E */

Return ((EXTI_Sensitivity_TypeDef)value);

(u8)0x00, /*!< Interrupt on Falling edge and Low level */

(u8)0x01, /*!< Interrupt on Rising edge only */

(u8)0x02, /*!< Interrupt on Falling edge only */

(u8)0x03 /*!< Interrupt on Rising and Falling edges */

****

EXTI_GetTLISensitivity(void); // Get the TLI interrupt sensitivity.

Return ((EXTI_TLISensitivity_TypeDef)value);

(u8)0x00, /*!< Top Level Interrupt on Falling edge only */

(u8)0x04 /*!< Top Level Interrupt on Rising edge only */

**************************** STM8S FWLIB ***************************************

****************************   STM8S FWLIB   **************************************

file stm8s_flash.

****

FLASH_Unlock(FLASH_MemType_TypeDef MemType);

FLASH_Lock(FLASH_MemType_TypeDef MemType);

FLASH_DeInit(void);

FLASH_ITConfig(FunctionalState NewState);

FLASH_EraseByte(u32 Address);

FLASH_ProgramByte(u32 Address, u8 Data);

FLASH_ReadByte(u32 Address);

FLASH_ProgramWord(u32 Address, u32 Data);

FLASH_ReadOptionByte(u16 Address);

FLASH_ProgramOptionByte(u16 Address, u8 Data);

FLASH_EraseOptionByte(u16 Address);

FLASH_SetLowPowerMode(FLASH_LPMode_TypeDef LPMode);

FLASH_SetProgrammingTime(FLASH_ProgramTime_TypeDef ProgTime);

FLASH_GetLowPowerMode(void);

FLASH_GetProgrammingTime(void);

FLASH_GetBootSize(void);

FLASH_GetFlagStatus(FLASH_Flag_TypeDef FLASH_FLAG);

/* Function to be executed from RAM -- */

FLASH_EraseBlock(u16 BlockNum, FLASH_MemType_TypeDef MemType);

FLASH_ProgramBlock(u16 BlockNum, FLASH_MemType_TypeDef MemType,

FLASH_ProgramMode_TypeDef ProgMode, u8 *Buffer);

FLASH_WaitForLastOperation(FLASH_MemType_TypeDef MemType);

****

****

FLASH_Unlock(FLASH_MemType_TypeDef MemType); // Unlocks the program or data EEPROM memory

INPUT : //FLASH Memory types

```
      FLASH_MEMTYPE_PROG          = (u8)0x00,         /*!< Program memory */
      FLASH_MEMTYPE_DATA          = (u8)0x01          /*!< Data EEPROM memory */
```

****

FLASH_Lock(FLASH_MemType_TypeDef MemType); //Locks the program or data EEPROM memory

---

INPUT : SEE **FLASH_Unlock**( )

****

FLASH_DeInit(void); // Deinitializes the FLASH peripheral registers to their default reset values

****

FLASH_ITConfig(FunctionalState NewState); // Enables or Disables the Flash interrupt mode

---

INPUT : DISABLE ; ENABLE

****

FLASH_EraseByte(u32 Address); // Erases one byte in the program or data EEPROM memory

---

INPUT : //Address of the byte to erase

u32 Address

****

FLASH_ProgramByte(u32 Address, u8 Data); // Programs one byte in program or data EEPROM memory

---

INPUT : // Adress where the byte is written & Data Value to be writtenu32 Address

u32 Address ; u8 Data

****

u8 FLASH_ReadByte(u32 Address); // Reads any byte from flash memory

---

INPUT : //Address to read

u32 Address

return : u8 Value read

****

FLASH_ProgramWord(u32 Address, u32 Data);

// Programs one word (4 bytes) in program or data EEPROM memory

---

INPUT : // Address Adress where the byte is written & Data Value to be written

u32 Address ; u32 Data

****

u16 FLASH_ReadOptionByte(u16 Address); // Reads one option byte 读选项字节，参考其它资料

INPUT : // option byte address to read.

u16 Address

return ： u16 res_value (Value read + complement value read.)

or : FLASH_OPTIONBYTE_ERROR ((u16)0x5555)

/*!< Error code option byte (if value read is not equal to complement value read) */

****

FLASH_ProgramOptionByte(u16 Address, u8 Data); // Programs an option byte 参考前一函数

INPUT : // option byte address to program & Data Value to write

u16 Address , u8 Data

****

FLASH_EraseOptionByte(u16 Address); // Erases an option byte

INPUT : // Option byte address to erase

u16 Address

****

FLASH_SetLowPowerMode(FLASH_LPMode_TypeDef LPMode);

// Select the Flash behaviour in low power mode

INPUT : // Low power mode selection

FLASH_LPMODE_POWERDOWN = (u8)0x04,

/*!< HALT: Power-Down / ACTIVE-HALT: Power-Down */

FLASH_LPMODE_STANDBY = (u8)0x08,

/*!< HALT: Standby / ACTIVE-HALT: Standby */

FLASH_LPMODE_POWERDOWN_STANDBY = (u8)0x00,

/*!< HALT: Power-Down / ACTIVE-HALT: Standby */

FLASH_LPMODE_STANDBY_POWERDOWN = (u8)0x0C

/*!< HALT: Standby / ACTIVE-HALT: Power-Down */

****

FLASH_SetProgrammingTime(FLASH_ProgramTime_TypeDef ProgTime);

// Sets the fixed programming time

INPUT : // ProgTime Indicates the programming time to be fixed

FLASH_PROGRAMTIME_STANDARD = (u8)0x00, /*!< Standard programming time fixed at 1/2 tprog */

FLASH_PROGRAMTIME_TPROG = (u8)0x01 /*!< Programming time fixed at tprog */

****

FLASH_GetLowPowerMode(void); // Returns the Flash behaviour type in low power mode

Return : //FLASH_LPMode_TypeDef Flash behaviour type in low power mode

FLASH_LPMODE_POWERDOWN = (u8)0x04,

FLASH_LPMODE_STANDBY = (u8)0x08,

FLASH_LPMODE_POWERDOWN_STANDBY = (u8)0x00,

FLASH_LPMODE_STANDBY_POWERDOWN = (u8)0x0C

****

FLASH_GetProgrammingTime(void); // Returns the fixed programming time

Return : // FLASH_ProgramTime_TypeDef Fixed programming time value

FLASH_PROGRAMTIME_STANDARD = (u8)0x00, /*!< Standard programming time fixed at 1/2 tprog */

FLASH_PROGRAMTIME_TPROG = (u8)0x01 /*!< Programming time fixed at tprog */

****

FLASH_GetBootSize(void); // Returns the Boot memory size in bytes

Return : u32 Boot memory size in bytes

****

FLASH_GetFlagStatus(FLASH_Flag_TypeDef FLASH_FLAG);

// Checks whether the specified SPI flag is set or not.

INPUT : // FLASH_FLAG : Specifies the flag to check.

FLASH_FLAG_DUL = (u8)0x08, /*!< Data EEPROM unlocked flag */

FLASH_FLAG_EOP = (u8)0x04, /*!< End of programming (write or erase operation) flag */

FLASH_FLAG_PUL = (u8)0x02, /*!< Flash Program memory unlocked flag */
FLASH_FLAG_WR_PG_DIS = (u8)0x01 /*!< Write attempted to protected page flag */

Return : FlagStatus : Indicates the state of FLASH_FLAG.

SET ; RESET

****

FLASH_EraseBlock(u16 BlockNum, FLASH_MemType_TypeDef MemType);

// Erases a block in the program or data memory.

INPUT : block number to erase & Memory type

u16 BlockNum

FLASH Memory types

FLASH_MEMTYPE_PROG = (u8)0x00, /*!< Program memory */
FLASH_MEMTYPE_DATA = (u8)0x01 /*!< Data EEPROM memory */

****

FLASH_ProgramBlock(u16 BlockNum, FLASH_MemType_TypeDef MemType,
FLASH_ProgramMode_TypeDef ProgMode, u8 *Buffer);

// Programs a memory block

INPUT : //MemType The type of memory to program ; BlockNum The block number ;
ProgMode The programming mode. ; Buffer The buffer address of source data.

u16 BlockNum

MemType

FLASH_MEMTYPE_PROG = (u8)0x00
FLASH_MEMTYPE_DATA) = (u8)0x01

ProgMode

FLASH_PROGRAMMODE_STANDARD = (u8)0x00,
FLASH_PROGRAMMODE_FAST = (u8)0x10

u8 *Buffer // buffer address of source data.

****

FLASH_WaitForLastOperation(FLASH_MemType_TypeDef MemType);

// Wait for a Flash operation to complete.

INPUT : //MemType Memory type

FLASH_MEMTYPE_PROG = (u8)0x00, /*!< Program memory */
FLASH_MEMTYPE_DATA = (u8)0x01 /*!< Data EEPROM memory */

Return : //FLASH_Status_TypeDef State of the last operation

FLASH_STATUS_END_HIGH_VOLTAGE = (u8)0x40, /*!< End of high voltage */
FLASH_STATUS_SUCCESSFUL_OPERATION = (u8)0x04, /*!< End of operation flag */
FLASH_STATUS_TIMEOUT = (u8)0x02, /*!< Time out error */
FLASH_STATUS_WRITE_PROTECTION_ERROR = (u8)0x01 /*Write attempted to protected page */

***************************** STM8S FWLIB **

file stm8s_gpio.

****

```
GPIO_DeInit(GPIO_TypeDef* GPIOx);
GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_Pin_TypeDef GPIO_Pin, GPIO_Mode_TypeDef GPIO_Mode);
GPIO_Write(GPIO_TypeDef* GPIOx, u8 PortVal);
GPIO_WriteHigh(GPIO_TypeDef* GPIOx, GPIO_Pin_TypeDef PortPins);
GPIO_WriteLow(GPIO_TypeDef* GPIOx, GPIO_Pin_TypeDef PortPins);
GPIO_WriteReverse(GPIO_TypeDef* GPIOx, GPIO_Pin_TypeDef PortPins);
GPIO_ReadInputData(GPIO_TypeDef* GPIOx);
GPIO_ReadOutputData(GPIO_TypeDef* GPIOx);
GPIO_ReadInputPin(GPIO_TypeDef* GPIOx, GPIO_Pin_TypeDef GPIO_Pin);
GPIO_ExternalPullUpConfig(    GPIO_TypeDef* GPIOx, GPIO_Pin_TypeDef GPIO_Pin,
                              FunctionalState NewState);
```

****

****

GPIO_DeInit(GPIO_TypeDef* GPIOx);   // Deinitializes the GPIOx peripheral registers to their default reset

---

INPUT：GPIOx : Select the GPIO peripheral number (x = A to I)

****

GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_Pin_TypeDef GPIO_Pin,
GPIO_Mode_TypeDef GPIO_Mode);   //Initializes the GPIOx according to the specified parameters.

INPUT：GPIOx   :   //Select the GPIO peripheral number (x = A to I).

```
        //GPIO_Pin : This parameter contains the pin number, it can be one or many members
    GPIO_PIN_0      = ((u8)0x01),     /*!< Pin 0 selected */
    GPIO_PIN_1      = ((u8)0x02),     /*!< Pin 1 selected */
    GPIO_PIN_2      = ((u8)0x04),     /*!< Pin 2 selected */
    GPIO_PIN_3      = ((u8)0x08),     /*!< Pin 3 selected */
    GPIO_PIN_4      = ((u8)0x10),     /*!< Pin 4 selected */
    GPIO_PIN_5      = ((u8)0x20),     /*!< Pin 5 selected */
    GPIO_PIN_6      = ((u8)0x40),     /*!< Pin 6 selected */
    GPIO_PIN_7      = ((u8)0x80),     /*!< Pin 7 selected */
    GPIO_PIN_LNIB   = ((u8)0x0F),     /*!< Low nibble pins selected */
    GPIO_PIN_HNIB   = ((u8)0xF0),     /*!< High nibble pins selected */
    GPIO_PIN_ALL    = ((u8)0xFF)      /*!< All pins selected */
        //GPIO_Mode : This parameter can be any of the @Ref GPIO_Mode_TypeDef enumeration.
    GPIO_MODE_IN_FL_NO_IT     = (u8)0b00000000,   /*!< Input floating, no external interrupt */
    GPIO_MODE_IN_PU_NO_IT     = (u8)0b01000000,   /*!< Input pull-up, no external interrupt */
```

```
        GPIO_MODE_IN_FL_IT          = (u8)0b00100000,   /*!< Input floating, external interrupt */
        GPIO_MODE_IN_PU_IT          = (u8)0b01100000,   /*!< Input pull-up, external interrupt */
        GPIO_MODE_OUT_OD_LOW_FAST       = (u8)0b10000000,
                                                /*!< Output open-drain, low level, no slope control */
        GPIO_MODE_OUT_PP_LOW_FAST       = (u8)0b11000000,
                                                /*!< Output push-pull, low level, no slope control */
        GPIO_MODE_OUT_OD_LOW_SLOW       = (u8)0b10100000,
                                                /*!< Output open-drain, low level, slow slope */
        GPIO_MODE_OUT_PP_LOW_SLOW       = (u8)0b11100000,
                                                /*!< Output push-pull, low level, slow slope */
        GPIO_MODE_OUT_OD_HIZ_FAST       = (u8)0b10010000,
                                    /*!< Output open-drain, high-impedance level, no slope control */
        GPIO_MODE_OUT_PP_HIGH_FAST      = (u8)0b11010000,
                                    /*!< Output push-pull, high level, no slope control */
        GPIO_MODE_OUT_OD_HIZ_SLOW       = (u8)0b10110000,
                                    /*!< Output open-drain, high-impedance level, slow slope */
        GPIO_MODE_OUT_PP_HIGH_SLOW      = (u8)0b11110000
                                    /*!< Output push-pull, high level, slow slope */
```

****

GPIO_Write(GPIO_TypeDef* GPIOx, u8 PortVal); // Writes data to the specified GPIO data port.

```
INPUT:  GPIOx         //Select the GPIO peripheral number (x = A to I).
        u8 PortVal    //Specifies the value to be written to the port output.
```

****

GPIO_WriteHigh(GPIO_TypeDef* GPIOx, GPIO_Pin_TypeDef PortPins);

// Writes high level to the specified GPIO pins

```
INPUT:  GPIOx         // Select the GPIO peripheral number (x = A to I).
        //GPIO_Pin : This parameter contains the pin number, it can be one or many members
        GPIO_PIN_0      = ((u8)0x01),      /*!< Pin 0 selected */
        GPIO_PIN_1      = ((u8)0x02),      /*!< Pin 1 selected */
        GPIO_PIN_2      = ((u8)0x04),      /*!< Pin 2 selected */
        GPIO_PIN_3      = ((u8)0x08),      /*!< Pin 3 selected */
        GPIO_PIN_4      = ((u8)0x10),      /*!< Pin 4 selected */
        GPIO_PIN_5      = ((u8)0x20),      /*!< Pin 5 selected */
        GPIO_PIN_6      = ((u8)0x40),      /*!< Pin 6 selected */
        GPIO_PIN_7      = ((u8)0x80),      /*!< Pin 7 selected */
        GPIO_PIN_LNIB   = ((u8)0x0F),      /*!< Low nibble pins selected */
        GPIO_PIN_HNIB   = ((u8)0xF0),      /*!< High nibble pins selected */
        GPIO_PIN_ALL    = ((u8)0xFF)       /*!< All pins selected */
```

****

GPIO_WriteLow(GPIO_TypeDef* GPIOx, GPIO_Pin_TypeDef PortPins);

//Writes low level to the specified GPIO pins.

INPUT： SEE **GPIO_WriteHigh( )**

****

GPIO_WriteReverse(GPIO_TypeDef* GPIOx, GPIO_Pin_TypeDef PortPins);

//Writes reverse level to the specified GPIO pins. 反转特定引脚电平

INPUT：GPIOx；PortPins //see(**GPIO_WriteHigh( )**);

****

GPIO_ReadInputData(GPIO_TypeDef* GPIOx); // Reads the specified GPIO output data port.

INPUT：GPIOx

Return : u8 GPIO output data port value.

****

GPIO_ReadOutputData(GPIO_TypeDef* GPIOx);

See **GPIO_ReadInputData**( );

****

GPIO_ReadInputPin(GPIO_TypeDef* GPIOx, GPIO_Pin_TypeDef GPIO_Pin);

// Reads the specified GPIO input data pin.

INPUT ：GPIOx ; GPIO_Pin

Return : (BitStatus) 0 or 1 //BitStatus : GPIO input pin status.

****

GPIO_ExternalPullUpConfig(GPIO_TypeDef* GPIOx, GPIO_Pin_TypeDef GPIO_Pin,
FunctionalState NewState); // Configures the external pull-up on GPIOx pins.内部悬浮，使用外接电阻上拉

INPUT：GPIOx ; GPIO_Pin ; NewState : DISABLE or ENABLE

**************************** STM8S FWLIB **************************************

*****************************   STM8S FWLIB   **

file stm8s_iwdg.

****

IWDG_WriteAccessCmd(IWDG_WriteAccess_TypeDef IWDG_WriteAccess);

IWDG_SetPrescaler(IWDG_Prescaler_TypeDef IWDG_Prescaler);

IWDG_SetReload(u8 Reload);

IWDG_ReloadCounter(void);

IWDG_Enable(void);

****

****

★ **IWDG_WriteAccessCmd**(IWDG_WriteAccess_TypeDef IWDG_WriteAccess); //向看门狗写命令值

INPUT ： IWDG_WriteAccess_Enable = (u8)0x55,
IWDG_WriteAccess_Disable = (u8)0x00

****

IWDG_SetPrescaler(IWDG_Prescaler_TypeDef IWDG_Prescaler); //Sets IWDG Prescaler value.

INPUT ： //IWDG_Prescaler set the value of the prescaler register.

IWDG_Prescaler_4 = (u8)0x00
IWDG_Prescaler_8 = (u8)0x01
IWDG_Prescaler_16 = (u8)0x02
IWDG_Prescaler_32 = (u8)0x03
IWDG_Prescaler_64 = (u8)0x04,
IWDG_Prescaler_128 = (u8)0x05
IWDG_Prescaler_256 = (u8)0x06

****

IWDG_SetReload(u8 Reload); // Sets IWDG Reload value.

INPUT ： // IWDG_Reload Specifies the IWDG Reload value (from 0x00 to 0xFF)

****

IWDG_ReloadCounter(void); // Reload IWDG counter

****

IWDG_Enable(void); // Enable IWDG registers access.(Write ((u8)0xCC) to reg IWDG_KR)

*****************************   STM8S FWLIB   **

****************************   STM8S FWLIB   **************************************

file stm8s_tim1.

****

```
TIM1_DeInit(void);

TIM1_TimeBaseInit(   u16 TIM1_Prescaler,
                     TIM1_CounterMode_TypeDef TIM1_CounterMode,
                     u16 TIM1_Period,
                     u8 TIM1_RepetitionCounter);


TIM1_OC1Init(    TIM1_OCMode_TypeDef TIM1_OCMode,
                 TIM1_OutputState_TypeDef TIM1_OutputState,
                 TIM1_OutputNState_TypeDef TIM1_OutputNState,
                 u16 TIM1_Pulse,
                 TIM1_OCPolarity_TypeDef TIM1_OCPolarity,
                 TIM1_OCNPolarity_TypeDef TIM1_OCNPolarity,
                 TIM1_OCIdleState_TypeDef TIM1_OCIdleState,
                 TIM1_OCNIdleState_TypeDef TIM1_OCNIdleState);


TIM1_OC2Init(    TIM1_OCMode_TypeDef TIM1_OCMode,
                 TIM1_OutputState_TypeDef TIM1_OutputState,
                 TIM1_OutputNState_TypeDef TIM1_OutputNState,
                 u16 TIM1_Pulse,
                 TIM1_OCPolarity_TypeDef TIM1_OCPolarity,
                 TIM1_OCNPolarity_TypeDef TIM1_OCNPolarity,
                 TIM1_OCIdleState_TypeDef TIM1_OCIdleState,
                 TIM1_OCNIdleState_TypeDef TIM1_OCNIdleState);


TIM1_OC3Init(    TIM1_OCMode_TypeDef TIM1_OCMode,
                 TIM1_OutputState_TypeDef TIM1_OutputState,
                 TIM1_OutputNState_TypeDef TIM1_OutputNState,
                 u16 TIM1_Pulse,
                 TIM1_OCPolarity_TypeDef TIM1_OCPolarity,
                 TIM1_OCNPolarity_TypeDef TIM1_OCNPolarity,
                 TIM1_OCIdleState_TypeDef TIM1_OCIdleState,
                 TIM1_OCNIdleState_TypeDef TIM1_OCNIdleState);

TIM1_OC4Init(   TIM1_OCMode_TypeDef TIM1_OCMode,
                 TIM1_OutputState_TypeDef TIM1_OutputState,
                 u16 TIM1_Pulse,
                 TIM1_OCPolarity_TypeDef TIM1_OCPolarity,
                 TIM1_OCIdleState_TypeDef TIM1_OCIdleState);

TIM1_BDTRConfig(    TIM1_OSSIState_TypeDef TIM1_OSSIState,
                     TIM1_LockLevel_TypeDef TIM1_LockLevel,
                     u8 TIM1_DeadTime,
                     TIM1_BreakState_TypeDef TIM1_Break,
                     TIM1_BreakPolarity_TypeDefTIM1_BreakPolarity,
                     TIM1_AutomaticOutput_TypeDef TIM1_AutomaticOutput);

TIM1_ICInit(    TIM1_Channel_TypeDef TIM1_Channel,
                 TIM1_ICPolarity_TypeDef TIM1_ICPolarity,
                 TIM1_ICSelection_TypeDef TIM1_ICSelection,
                 TIM1_ICPSC_TypeDef TIM1_ICPrescaler,
                 u8 TIM1_ICFilter);

TIM1_PWMIConfig(    TIM1_Channel_TypeDef TIM1_Channel,
                     TIM1_ICPolarity_TypeDef TIM1_ICPolarity,
                     TIM1_ICSelection_TypeDef TIM1_ICSelection,
                     TIM1_ICPSC_TypeDef TIM1_ICPrescaler,
                     u8 TIM1_ICFilter);

TIM1_Cmd(   FunctionalState NewState);
```

```
TIM1_CtrlPWMOutputs(      FunctionalState Newstate);
TIM1_ITConfig(    TIM1_IT_TypeDef TIM1_IT,       FunctionalState NewState);
TIM1_InternalClockConfig(void);
TIM1_ETRClockMode1Config(   TIM1_ExtTRGPSC_TypeDefTIM1_ExtTRGPrescaler,
                            TIM1_ExtTRGPolarity_TypeDef TIM1_ExtTRGPolarity,
                            u8 ExtTRGFilter);
TIM1_ETRClockMode2Config(   TIM1_ExtTRGPSC_TypeDefTIM1_ExtTRGPrescaler,
                            TIM1_ExtTRGPolarity_TypeDef TIM1_ExtTRGPolarity,
                            u8 ExtTRGFilter);
TIM1_ETRConfig(        TIM1_ExtTRGPSC_TypeDef TIM1_ExtTRGPrescaler,
                       TIM1_ExtTRGPolarity_TypeDef TIM1_ExtTRGPolarity,
                       u8 ExtTRGFilter);
TIM1_TIxExternalClockConfig(   TIM1_TIxExternalCLK1Source_TypeDef TIM1_TIxExternalCLKSource,
                               TIM1_ICPolarity_TypeDef TIM1_ICPolarity,
                               u8 ICFilter);
TIM1_SelectInputTrigger(     TIM1_TS_TypeDef TIM1_InputTriggerSource);
TIM1_UpdateDisableConfig(       FunctionalState Newstate);
TIM1_UpdateRequestConfig(       TIM1_UpdateSource_TypeDef TIM1_UpdateSource);
TIM1_SelectHallSensor(       FunctionalState Newstate);
TIM1_SelectOnePulseMode(        TIM1_OPMode_TypeDef TIM1_OPMode);
TIM1_SelectOutputTrigger(          TIM1_TRGOSource_TypeDef TIM1_TRGOSource);
TIM1_SelectSlaveMode(       TIM1_SlaveMode_TypeDef TIM1_SlaveMode);
TIM1_SelectMasterSlaveMode      (FunctionalState NewState);
TIM1_EncoderInterfaceConfig(    TIM1_EncoderMode_TypeDef TIM1_EncoderMode,
                                TIM1_ICPolarity_TypeDef TIM1_IC1Polarity,
                                TIM1_ICPolarity_TypeDef TIM1_IC2Polarity);
TIM1_PrescalerConfig(        u16 Prescaler,         TIM1_PSCReloadMode_TypeDef TIM1_PSCReloadMode);
TIM1_CounterModeConfig(  TIM1_CounterMode_TypeDef TIM1_CounterMode);
TIM1_ForcedOC1Config(TIM1_ForcedAction_TypeDef TIM1_ForcedAction);
TIM1_ForcedOC2Config(TIM1_ForcedAction_TypeDef TIM1_ForcedAction);
TIM1_ForcedOC3Config(TIM1_ForcedAction_TypeDef TIM1_ForcedAction);
```

```
TIM1_ForcedOC4Config(TIM1_ForcedAction_TypeDef TIM1_ForcedAction);
TIM1_ARRPreloadConfig(FunctionalState Newstate);
TIM1_SelectCOM(FunctionalState Newstate);
TIM1_CCPreloadControl(FunctionalState Newstate);
TIM1_OC1PreloadConfig(FunctionalState Newstate);
TIM1_OC2PreloadConfig(FunctionalState Newstate);
TIM1_OC3PreloadConfig(FunctionalState Newstate);
TIM1_OC4PreloadConfig(FunctionalState Newstate);
TIM1_OC1FastConfig(FunctionalState Newstate);
TIM1_OC2FastConfig(FunctionalState Newstate);
TIM1_OC3FastConfig(FunctionalState Newstate);
TIM1_OC4FastConfig(FunctionalState Newstate);
TIM1_GenerateEvent(TIM1_EventSource_TypeDef TIM1_EventSource);
TIM1_OC1PolarityConfig(TIM1_OCPolarity_TypeDef TIM1_OCPolarity);
TIM1_OC1NPolarityConfig(TIM1_OCNPolarity_TypeDef TIM1_OCNPolarity);
TIM1_OC2PolarityConfig(TIM1_OCPolarity_TypeDef TIM1_OCPolarity);
TIM1_OC2NPolarityConfig(TIM1_OCNPolarity_TypeDef TIM1_OCNPolarity);
TIM1_OC3PolarityConfig(TIM1_OCPolarity_TypeDef TIM1_OCPolarity);
TIM1_OC3NPolarityConfig(TIM1_OCNPolarity_TypeDef TIM1_OCNPolarity);
TIM1_OC4PolarityConfig(TIM1_OCPolarity_TypeDef TIM1_OCPolarity);
TIM1_CCxCmd(TIM1_Channel_TypeDef TIM1_Channel,      FunctionalState Newstate);
TIM1_CCxNCmd(TIM1_Channel_TypeDef TIM1_Channel,     FunctionalState Newstate);
TIM1_SelectOCxM(    TIM1_Channel_TypeDef TIM1_Channel,
                    TIM1_OCMode_TypeDef TIM1_OCMode);
TIM1_SetCounter(u16 Counter);
TIM1_SetAutoreload(u16 Autoreload);
TIM1_SetCompare1(u16 Compare1);
TIM1_SetCompare2(u16 Compare2);
TIM1_SetCompare3(u16 Compare3);
TIM1_SetCompare4(u16 Compare4);
TIM1_SetIC1Prescaler(TIM1_ICPSC_TypeDef TIM1_IC1Prescaler);
```

```
TIM1_SetIC2Prescaler(TIM1_ICPSC_TypeDef TIM1_IC2Prescaler);

TIM1_SetIC3Prescaler(TIM1_ICPSC_TypeDef TIM1_IC3Prescaler);

TIM1_SetIC4Prescaler(TIM1_ICPSC_TypeDef TIM1_IC4Prescaler);

TIM1_GetCapture1(void);

TIM1_GetCapture2(void);

TIM1_GetCapture3(void);

TIM1_GetCapture4(void);

TIM1_GetCounter(void);

TIM1_GetPrescaler(void);

TIM1_GetFlagStatus(TIM1_FLAG_TypeDef TIM1_FLAG);

TIM1_ClearFlag(TIM1_FLAG_TypeDef TIM1_FLAG);

TIM1_GetITStatus(TIM1_IT_TypeDef TIM1_IT);

TIM1_ClearITPendingBit(TIM1_IT_TypeDef TIM1_IT);
****************************************************************************************

****************************************************************************************
    TIM1_DeInit(void);    // Deinitializes the TIM1 peripheral registers to their default reset values.
****************************************************************************************
    TIM1_TimeBaseInit(  u16 TIM1_Prescaler,     TIM1_CounterMode_TypeDef TIM1_CounterMode,
                        u16 TIM1_Period,        u8 TIM1_RepetitionCounter);
                    // Initializes the TIM1 Time Base Unit according to the specified parameters.
----------------------------------------------------------------------------------------
INPUT :  u16 TIM1_Prescaler                                              时钟预分频
             //TIM1_CounterMode specifies the counter mode               计数模式
         TIM1_COUNTERMODE_UP                    = ((u8)0x00),
         TIM1_COUNTERMODE_DOWN                  = ((u8)0x10),
         TIM1_COUNTERMODE_CENTERALIGNED1        = ((u8)0x20),
         TIM1_COUNTERMODE_CENTERALIGNED2        = ((u8)0x40),
         TIM1_COUNTERMODE_CENTERALIGNED3        = ((u8)0x60)
             // TIM1_Period specifies the Period value.                  周期值
         u16 TIM1_Period
             // TIM1_RepetitionCounter specifies the Repetition counter value
         u8 TIM1_RepetitionCounter                                       重复计数的次数
****************************************************************************************
    TIM1_OC1Init(   TIM1_OCMode_TypeDef TIM1_OCMode,
                TIM1_OutputState_TypeDef TIM1_OutputState,
                TIM1_OutputNState_TypeDef TIM1_OutputNState,
                u16 TIM1_Pulse,
                TIM1_OCPolarity_TypeDef TIM1_OCPolarity,
                TIM1_OCNPolarity_TypeDef TIM1_OCNPolarity,
                TIM1_OCIdleState_TypeDef TIM1_OCIdleState,
                TIM1_OCNIdleState_TypeDef TIM1_OCNIdleState);
                    //初始化输出比较通道 1
```

INPUT :

```
// TIM1_OCMode specifies the Output Compare mode from @ref TIM1_OCMode_TypeDef.
      TIM1_OCMODE_TIMING              = ((u8)0x00),
      TIM1_OCMODE_ACTIVE              = ((u8)0x10),
      TIM1_OCMODE_INACTIVE            = ((u8)0x20),
      TIM1_OCMODE_TOGGLE              = ((u8)0x30),
      TIM1_OCMODE_PWM1                = ((u8)0x60),
      TIM1_OCMODE_PWM2                = ((u8)0x70)
// TIM1_OutputState specifies the Output State from @ref TIM1_OutputState_TypeDef.
      TIM1_OUTPUTSTATE_DISABLE        = ((u8)0x00),
      TIM1_OUTPUTSTATE_ENABLE         = ((u8)0x11)
// TIM1_OutputNState specifies the Complementary Output State from @ref TIM1_OutputNState_TypeDef.
      TIM1_OUTPUTNSTATE_DISABLE       = ((u8)0x00),
      TIM1_OUTPUTNSTATE_ENABLE        = ((u8)0x44)
// TIM1_Pulse specifies the Pulse width value.
      u16 TIM1_Pulse
// TIM1_OCPolarity specifies the Output Compare Polarity from @ref TIM1_OCPolarity_TypeDef.
      TIM1_OCPOLARITY_HIGH            = ((u8)0x00),
      TIM1_OCPOLARITY_LOW             = ((u8)0x22)
// TIM1_OCNPolarity specifies the Complementary Output Compare Polarity from @ref TIM1_OCNPolarity_TypeDef.
      TIM1_OCNPOLARITY_HIGH           = ((u8)0x00),
      TIM1_OCNPOLARITY_LOW            = ((u8)0x88)
// TIM1_OCIdleState specifies the Output Compare Idle State from @ref TIM1_OCIdleState_TypeDef.
      TIM1_OCIDLESTATE_SET            = ((u8)0x55),
      TIM1_OCIDLESTATE_RESET          = ((u8)0x00)
// TIM1_OCNIdleState specifies the Complementary Output Compare Idle State from @ref TIM1_OCNIdleState_TypeDef.
      TIM1_OCNIDLESTATE_SET           = ((u8)0x2A),
      TIM1_OCNIDLESTATE_RESET         = ((u8)0x00)
//IDLE 详见输出空闲状态寄存器 TIM1_OISR
*************************************************************************************
```

```
      TIM1_OC2Init(   TIM1_OCMode_TypeDef TIM1_OCMode,
                      TIM1_OutputState_TypeDef TIM1_OutputState,
                      TIM1_OutputNState_TypeDef TIM1_OutputNState,
                      u16 TIM1_Pulse,
                      TIM1_OCPolarity_TypeDef TIM1_OCPolarity,
                      TIM1_OCNPolarity_TypeDef TIM1_OCNPolarity,
                      TIM1_OCIdleState_TypeDef TIM1_OCIdleState,
                      TIM1_OCNIdleState_TypeDef TIM1_OCNIdleState);
```

SEE   TIM1_OC1Init( )

```
*************************************************************************************
      TIM1_OC3Init(   TIM1_OCMode_TypeDef TIM1_OCMode,
                      TIM1_OutputState_TypeDef TIM1_OutputState,
                      TIM1_OutputNState_TypeDef TIM1_OutputNState,
                      u16 TIM1_Pulse,
                      TIM1_OCPolarity_TypeDef TIM1_OCPolarity,
                      TIM1_OCNPolarity_TypeDef TIM1_OCNPolarity,
                      TIM1_OCIdleState_TypeDef TIM1_OCIdleState,
                      TIM1_OCNIdleState_TypeDef TIM1_OCNIdleState);
```

SEE TIM1_OC1Init( )

****

TIM1_OC4Init(

```
                    TIM1_OCMode_TypeDef TIM1_OCMode,
                    TIM1_OutputState_TypeDef TIM1_OutputState,
                    u16 TIM1_Pulse,
                    TIM1_OCPolarity_TypeDef TIM1_OCPolarity,
                    TIM1_OCIdleState_TypeDef TIM1_OCIdleState);
```

SEE TIM1_OC1Init( )

****

TIM1_BDTRConfig(

```
                    TIM1_OSSIState_TypeDef TIM1_OSSIState,
                    TIM1_LockLevel_TypeDef TIM1_LockLevel,
                    u8 TIM1_DeadTime,
                    TIM1_BreakState_TypeDef TIM1_Break,
                    TIM1_BreakPolarity_TypeDefTIM1_BreakPolarity,
                    TIM1_AutomaticOutput_TypeDef TIM1_AutomaticOutput);
```

// Configures the Break feature, dead time, Lock level, the OSSI, 参考 REG TIM1_BKR ; TIM1_DTR

INPUT : //TIM1_OSSIState specifies the OSSI State

```
    TIM1_OSSISTATE_ENABLE               = ((u8)0x04),
    TIM1_OSSISTATE_DISABLE              = ((u8)0x00)
        // TIM1_Lock Level specifies the lock level
    TIM1_LOCKLEVEL_OFF                  = ((u8)0x00),
    TIM1_LOCKLEVEL_1                    = ((u8)0x01),
    TIM1_LOCKLEVEL_2                    = ((u8)0x02),
    TIM1_LOCKLEVEL_3                    = ((u8)0x03)
        // TIM1_DeadTime specifies the dead time value.
    u8 TIM1_DeadTime
        // TIM1_Break specifies the Break state
    TIM1_BREAK_ENABLE                   = ((u8)0x10),
    TIM1_BREAK_DISABLE                  = ((u8)0x00)
        // TIM1_BreakPolarity specifies the Break polarity from @ref TIM1_BreakPolarity_TypeDef.
    TIM1_BREAKPOLARITY_LOW              = ((u8)0x00),
    TIM1_BREAKPOLARITY_HIGH             = ((u8)0x20)
        //TIM1_AutomaticOutput specifies the Automatic Output configuration
    TIM1_AUTOMATICOUTPUT_ENABLE         = ((u8)0x40),
    TIM1_AUTOMATICOUTPUT_DISABLE        = ((u8)0x00)
```

****

TIM1_ICInit(

```
                    TIM1_Channel_TypeDef TIM1_Channel,
                    TIM1_ICPolarity_TypeDef TIM1_ICPolarity,
```

TIM1_ICSelection_TypeDef TIM1_ICSelection,
TIM1_ICPSC_TypeDef TIM1_ICPrescaler,
u8 TIM1_ICFilter);

INPUT : //TIM1_Channel specifies the input capture channel from TIM1_Channel_TypeDef.

TIM1_CHANNEL_1 = ((u8)0x00),
TIM1_CHANNEL_2 = ((u8)0x01),
TIM1_CHANNEL_3 = ((u8)0x02),
TIM1_CHANNEL_4 = ((u8)0x03)

// TIM1_ICPolarity specifies the Input capture polarity from TIM1_ICPolarity_TypeDef .

TIM1_ICPOLARITY_RISING = ((u8)0x00),
TIM1_ICPOLARITY_FALLING = ((u8)0x01)

// TIM1_ICSelection specifies the Input capture source selection from TIM1_ICSelection_TypeDef.

TIM1_ICSELECTION_DIRECTTI = ((u8)0x01),
TIM1_ICSELECTION_INDIRECTTI = ((u8)0x02),
TIM1_ICSELECTION_TRGI = ((u8)0x03)

// TIM1_ICPrescaler specifies the Input capture Prescaler from TIM1_ICPSC_TypeDef.

TIM1_ICPSC_DIV1 = ((u8)0x00),
TIM1_ICPSC_DIV2 = ((u8)0x04),
TIM1_ICPSC_DIV4 = ((u8)0x08),
TIM1_ICPSC_DIV8 = ((u8)0x0C)

// TIM1_ICFilter specifies the Input capture filter value.

u8 TIM1_ICFilter

****

TIM1_PWMIConfig(TIM1_Channel_TypeDef TIM1_Channel,
TIM1_ICPolarity_TypeDef TIM1_ICPolarity,
TIM1_ICSelection_TypeDef TIM1_ICSelection,
TIM1_ICPSC_TypeDef TIM1_ICPrescaler,
u8 TIM1_ICFilter);

// Configures the TIM1 peripheral in PWM Input Mode according to the specified parameters.

SEE **TIM1_ICInit**( )

****

TIM1_Cmd(FunctionalState NewState); // Enables or disables the TIM1 peripheral.

INPUT : DISABLE ; ENABLE

****

TIM1_CtrlPWMOutputs(FunctionalState Newstate); // Enables or disables the TIM1 peripheral Main Outputs.

INPUT : DISABLE ; ENABLE

****

TIM1_ITConfig(TIM1_IT_TypeDef TIM1_IT, FunctionalState NewState);

// Enables or disables the specified TIM1 interrupts.

INPUT : //TIM1_IT specifies the TIM1 interrupts sources to be enabled or disabled.

TIM1_IT_UPDATE = ((u8)0x01),
TIM1_IT_CC1 = ((u8)0x02),

```
TIM1_IT_CC2                  = ((u8)0x04),
TIM1_IT_CC3                  = ((u8)0x08),
TIM1_IT_CC4                  = ((u8)0x10),
TIM1_IT_COM                  = ((u8)0x20),
TIM1_IT_TRIGGER              = ((u8)0x40),
TIM1_IT_BREAK                = ((u8)0x80)
```

// NewState new state of the TIM1 peripheral.

ENABLE   or   DISABLE

****

TIM1_InternalClockConfig(void);

****

TIM1_ETRClockMode1Config(TIM1_ExtTRGPSC_TypeDefTIM1_ExtTRGPrescaler,
TIM1_ExtTRGPolarity_TypeDef TIM1_ExtTRGPolarity,
u8 ExtTRGFilter);

// Configures the TIM1 External clock Mode1.   参考 REG TIM1_ETR

INPUT :   // TIM1_ExtTRGPrescaler specifies the external Trigger Prescaler.

```
TIM1_EXTTRGPSC_OFF                    = ((u8)0x00),
TIM1_EXTTRGPSC_DIV2                   = ((u8)0x10),
TIM1_EXTTRGPSC_DIV4                   = ((u8)0x20),
TIM1_EXTTRGPSC_DIV8                   = ((u8)0x30)
```

// TIM1_ExtTRGPolarity specifies the external Trigger Polarity.

```
TIM1_EXTTRGPOLARITY_INVERTED            = ((u8)0x80),
TIM1_EXTTRGPOLARITY_NONINVERTED          = ((u8)0x00)
```

// ExtTRGFilter specifies the External Trigger Filter.

u8 ExtTRGFilter

****

TIM1_ETRClockMode2Config(TIM1_ExtTRGPSC_TypeDefTIM1_ExtTRGPrescaler,
TIM1_ExtTRGPolarity_TypeDef TIM1_ExtTRGPolarity,
u8 ExtTRGFilter);

// Configures the TIM1 External clock Mode2.

SEE   **TIM1_ETRClockMode1Config** ( )

****

TIM1_ETRConfig(TIM1_ExtTRGPSC_TypeDef TIM1_ExtTRGPrescaler,
TIM1_ExtTRGPolarity_TypeDef TIM1_ExtTRGPolarity,
u8 ExtTRGFilter);

//配置 TIM1 外部触发

SEE   **TIM1_ETRClockMode1Config** ( )

****

TIM1_TIxExternalClockConfig(TIM1_TIxExternalCLK1Source_TypeDef
TIM1_TIxExternalCLKSource,
TIM1_ICPolarity_TypeDef TIM1_ICPolarity,
u8 ICFilter);

// Configures the TIM1 Trigger as External Clock.

INPUT : // TIM1_TIxExternalCLKSource specifies Trigger source.

```
    TIM1_TIXEXTERNALCLK1SOURCE_TI1ED        = ((u8)0x40),
    TIM1_TIXEXTERNALCLK1SOURCE_TI1          = ((u8)0x50),
    TIM1_TIXEXTERNALCLK1SOURCE_TI2          = ((u8)0x60)
```

// TIM1_ICPolarity specifies the TIx Polarity.

```
    TIM1_ICPOLARITY_RISING                  = ((u8)0x00),
    TIM1_ICPOLARITY_FALLING                 = ((u8)0x01)
```

// ICFilter specifies the filter value.

```
    u8 ICFilter
```

****

TIM1_SelectInputTrigger(TIM1_TS_TypeDef TIM1_InputTriggerSource); //Selects Trigger source.

INPUT : // TIM1_InputTriggerSource specifies Input Trigger source.

```
    TIM1_TS_TI1F_ED         = ((u8)0x40),
    TIM1_TS_TI1FP1          = ((u8)0x50),
    TIM1_TS_TI2FP2          = ((u8)0x60),
    TIM1_TS_ETRF            = ((u8)0x70)
```

****

TIM1_UpdateDisableConfig(FunctionalState Newstate); // Enables or Disables the TIM1 Update event.

INPUT : DISABLE ; ENABLE

****

TIM1_UpdateRequestConfig(TIM1_UpdateSource_TypeDef TIM1_UpdateSource);

// Selects the TIM1 Update Request Interrupt source.

INPUT : // TIM1_UpdateSource specifies the Update source.

```
    TIM1_UPDATESOURCE_GLOBAL        = ((u8)0x00),
    TIM1_UPDATESOURCE_REGULAR       = ((u8)0x01)
```

****

TIM1_SelectHallSensor(FunctionalState Newstate); // Enables or Disables the TIM1 Hall sensor interface.

INPUT : DISABLE ; ENABLE

****

TIM1_SelectOnePulseMode(TIM1_OPMode_TypeDef TIM1_OPMode);

INPUT : //TIM1_OPMode specifies the OPM Mode to be used.

```
    TIM1_OPMODE_SINGLE              = ((u8)0x01),
    TIM1_OPMODE_REPETITIVE          = ((u8)0x00)
```

****

TIM1_SelectOutputTrigger(TIM1_TRGOSource_TypeDef TIM1_TRGOSource);

// Selects the TIM1 Trigger Output Mode.

```
INPUT : //TIM1_TRGOSOURCE_RESET         = ((u8)0x00),
  TIM1_TRGOSOURCE_ENABLE                = ((u8)0x10),
  TIM1_TRGOSOURCE_UPDATE                = ((u8)0x20),
  TIM1_TRGOSource_OC1                   = ((u8)0x30),
  TIM1_TRGOSOURCE_OC1REF                = ((u8)0x40),
```

TIM1_TRGOSOURCE_OC2REF	= ((u8)0x50),
TIM1_TRGOSOURCE_OC3REF	= ((u8)0x60)

****

TIM1_SelectSlaveMode(TIM1_SlaveMode_TypeDef TIM1_SlaveMode);

NPUT : //TIM1_SlaveMode specifies the TIM1 Slave Mode.

TIM1_SLAVEMODE_RESET	= ((u8)0x04),
TIM1_SLAVEMODE_GATED	= ((u8)0x05),
TIM1_SLAVEMODE_TRIGGER	= ((u8)0x06),
TIM1_SLAVEMODE_EXTERNAL1	= ((u8)0x07)

****

TIM1_SelectMasterSlaveMode (FunctionalState NewState);

// Sets or Resets the TIM1 Master/Slave Mode.

INPUT : DISABLE ; ENABLE

****

TIM1_EncoderInterfaceConfig(TIM1_EncoderMode_TypeDef TIM1_EncoderMode,
TIM1_ICPolarity_TypeDef TIM1_IC1Polarity,
TIM1_ICPolarity_TypeDef TIM1_IC2Polarity);
// Configures the TIM1 Encoder Interface.

INPUT : // TIM1_EncoderMode specifies the TIM1 Encoder Mode

TIM1_ENCODERMODE_TI1	= ((u8)0x01),
TIM1_ENCODERMODE_TI2	= ((u8)0x02),
TIM1_ENCODERMODE_TI12	= ((u8)0x03)

// TIM1_IC1Polarity specifies the IC1 Polarity.

TIM1_ICPOLARITY_RISING	= ((u8)0x00),
TIM1_ICPOLARITY_FALLING	= ((u8)0x01)

// TIM1_IC1Polarity specifies the IC2 Polarity.

TIM1_ICPOLARITY_RISING	= ((u8)0x00),
TIM1_ICPOLARITY_FALLING	= ((u8)0x01)

****

TIM1_PrescalerConfig(u16 Prescaler, TIM1_PSCReloadMode_TypeDef TIM1_PSCReloadMode);

INPUT : // Prescaler specifies the Prescaler Register value

u16 Prescaler,

// TIM1_PSCReloadMode specifies the TIM1 Prescaler Reload mode.

TIM1_PSCRELOADMODE_UPDATE	= ((u8)0x00)	// The Prescaler is loaded at the update event.
TIM1_PSCRELOADMODE_IMMEDIATE	= ((u8)0x01)	//The Prescaler is loaded immediately.

****

TIM1_CounterModeConfig(TIM1_CounterMode_TypeDef TIM1_CounterMode);

INPUT : // TIM1_CounterMode specifies the Counter Mode to be used

TIM1_COUNTERMODE_UP	= ((u8)0x00),
TIM1_COUNTERMODE_DOWN	= ((u8)0x10),
TIM1_COUNTERMODE_CENTERALIGNED1	= ((u8)0x20),
TIM1_COUNTERMODE_CENTERALIGNED2	= ((u8)0x40),

TIM1_COUNTERMODE_CENTERALIGNED3 = ((u8)0x60)

****

TIM1_ForcedOC1Config(TIM1_ForcedAction_TypeDef TIM1_ForcedAction);

// Forces the TIM1 Channel1 output waveform to active or inactive level.

INPUT : //TIM1_ForcedAction specifies the forced Action to be set to the output waveform.

TIM1_FORCEDACTION_ACTIVE = ((u8)0x50) //强制为有效电平，强制 OC1REF 为高

TIM1_FORCEDACTION_INACTIVE = ((u8)0x40) //强制为无效电平，强制 OC1REF 为低

****

TIM1_ForcedOC2Config(TIM1_ForcedAction_TypeDef TIM1_ForcedAction);

SEE **TIM1_ForcedOC1Config** ( )

****

TIM1_ForcedOC3Config(TIM1_ForcedAction_TypeDef TIM1_ForcedAction);

SEE **TIM1_ForcedOC1Config** ( )

****

TIM1_ForcedOC4Config(TIM1_ForcedAction_TypeDef TIM1_ForcedAction);

SEE **TIM1_ForcedOC1Config** ( )

****

TIM1_ARRPreloadConfig(FunctionalState Newstate);

// Enables or disables TIM1 peripheral Preload register on ARR.

INPUT : DISABLE ; ENABLE

****

TIM1_SelectCOM(FunctionalState Newstate);

INPUT : DISABLE ; ENABLE

****

TIM1_CCPreloadControl(FunctionalState Newstate);

// Sets or Resets the TIM1 peripheral Capture Compare Preload Control bit.

INPUT : DISABLE ; ENABLE

****

TIM1_OC1PreloadConfig(FunctionalState Newstate);

// Enables or disables the TIM1 peripheral Preload Register on CCR1.

INPUT : DISABLE ; ENABLE

****

TIM1_OC2PreloadConfig(FunctionalState Newstate);

INPUT : DISABLE ; ENABLE

****

TIM1_OC3PreloadConfig(FunctionalState Newstate);

INPUT : DISABLE ; ENABLE

****

TIM1_OC4PreloadConfig(FunctionalState Newstate);

INPUT : DISABLE ; ENABLE

****

TIM1_OC1FastConfig(FunctionalState Newstate); // Configures the TIM1 Capture Compare 1 Fast feature.

INPUT : DISABLE ; ENABLE

****

TIM1_OC2FastConfig(FunctionalState Newstate);

INPUT : DISABLE ; ENABLE

****

TIM1_OC3FastConfig(FunctionalState Newstate);

INPUT : DISABLE ; ENABLE

****

TIM1_OC4FastConfig(FunctionalState Newstate);

INPUT : DISABLE ; ENABLE

****

TIM1_GenerateEvent(TIM1_EventSource_TypeDef TIM1_EventSource); //配置将由软件引发的 TIM 事件

INPUT : // TIM1_EventSource specifies the event source.

```
    TIM1_EVENTSOURCE_UPDATE     = ((u8)0x01),
    TIM1_EVENTSOURCE_CC1        = ((u8)0x02),
    TIM1_EVENTSOURCE_CC2        = ((u8)0x04),
    TIM1_EVENTSOURCE_CC3        = ((u8)0x08),
    TIM1_EVENTSOURCE_CC4        = ((u8)0x10),
    TIM1_EVENTSOURCE_COM        = ((u8)0x20),
    TIM1_EVENTSOURCE_TRIGGER    = ((u8)0x40),
    TIM1_EVENTSOURCE_BREAK      = ((u8)0x80)
```

****

TIM1_OC1PolarityConfig(TIM1_OCPolarity_TypeDef TIM1_OCPolarity);

// Configures the TIM1 Channel 1 polarity.

INPUT : // TIM1_OCPolarity specifies the OC1 Polarity.

```
    TIM1_OCPOLARITY_HIGH        = ((u8)0x00),
    TIM1_OCPOLARITY_LOW         = ((u8)0x22)
```

****

TIM1_OC1NPolarityConfig(TIM1_OCNPolarity_TypeDef TIM1_OCNPolarity);

// Configures the TIM1 Channel 1N polarity.

INPUT : // TIM1_OCNPolarity specifies the OC1N Polarity.

```
    TIM1_OCNPOLARITY_HIGH       = ((u8)0x00),
    TIM1_OCNPOLARITY_LOW        = ((u8)0x88)
```

****

TIM1_OC2PolarityConfig(TIM1_OCPolarity_TypeDef TIM1_OCPolarity);

SEE **TIM1_OC1PolarityConfig** ( )

****

TIM1_OC2NPolarityConfig(TIM1_OCNPolarity_TypeDef TIM1_OCNPolarity);

SEE **TIM1_OC1NPolarityConfig** ( )

****

TIM1_OC3PolarityConfig(TIM1_OCPolarity_TypeDef TIM1_OCPolarity);

SEE **TIM1_OC1PolarityConfig** ( )

****

TIM1_OC3NPolarityConfig(TIM1_OCNPolarity_TypeDef TIM1_OCNPolarity);

SEE **TIM1_OC1NPolarityConfig** ( )

****

TIM1_OC4PolarityConfig(TIM1_OCPolarity_TypeDef TIM1_OCPolarity);

SEE **TIM1_OC1PolarityConfig** ( )

****

TIM1_CCxCmd(TIM1_Channel_TypeDef TIM1_Channel, FunctionalState Newstate);

// Enables or disables the TIM1 Capture Compare Channel x (x=1,..,4).

INPUT : // TIM1_Channel specifies the TIM1 Channel.

TIM1_CHANNEL_1 = ((u8)0x00),
TIM1_CHANNEL_2 = ((u8)0x01),
TIM1_CHANNEL_3 = ((u8)0x02),
TIM1_CHANNEL_4 = ((u8)0x03)

// NewState specifies the TIM1 Channel CCxE bit new state.

ENABLE or DISABLE

****

TIM1_CCxNCmd(TIM1_Channel_TypeDef TIM1_Channel, FunctionalState Newstate);

// Enables or disables the TIM1 Capture Compare Channel xN (xN=1,..,3).

SEE **TIM1_CCxCmd** ( ) // CHANNEL_1 / 2 / 3.

****

TIM1_SelectOCxM(TIM1_Channel_TypeDef TIM1_Channel,
TIM1_OCMode_TypeDef TIM1_OCMode);

// Selects the TIM1 Ouput Compare Mode. This function disables the selected channel before changing the Ouput Compare Mode. User has to enable this channel using TIM1_CCxCmd and TIM1_CCxNCmd functions.

INPUT : // TIM1_Channel specifies the TIM1 Channel.

TIM1_CHANNEL_1 = ((u8)0x00),
TIM1_CHANNEL_2 = ((u8)0x01),
TIM1_CHANNEL_3 = ((u8)0x02),
TIM1_CHANNEL_4 = ((u8)0x03)

// TIM1_OCMode specifies the TIM1 Output Compare Mode.

```
    TIM1_OCMODE_TIMING                 = ((u8)0x00),
    TIM1_OCMODE_ACTIVE                 = ((u8)0x10),
    TIM1_OCMODE_TOGGLE                 = ((u8)0x30),
    TIM1_OCMODE_PWM1                   = ((u8)0x60),
    TIM1_OCMODE_PWM2                   = ((u8)0x70)
    TIM1_FORCEDACTION_ACTIVE           = ((u8)0x50),
    TIM1_FORCEDACTION_INACTIVE         = ((u8)0x40)
```

****

TIM1_SetCounter(u16 Counter);   // Sets the TIM1 Counter Register value.

INPUT :   // Counter specifies the Counter register new value.
u16 Counter

****

TIM1_SetAutoreload(u16 Autoreload);   // Sets the TIM1 Autoreload Register value.

INPUT :   // Autoreload specifies the Autoreload register new value.
u16 Autoreload

****

TIM1_SetCompare1(u16 Compare1);   // Sets the TIM1 Capture Compare1 Register value.
TIM1_SetCompare2(u16 Compare2);
TIM1_SetCompare3(u16 Compare3);
TIM1_SetCompare4(u16 Compare4);

INPUT :   // Compare1 specifies the Capture Compare1 register new value.
u16 Compare1

****

TIM1_SetIC1Prescaler(TIM1_ICPSC_TypeDef TIM1_IC1Prescaler);
TIM1_SetIC2Prescaler(TIM1_ICPSC_TypeDef TIM1_IC2Prescaler);
TIM1_SetIC3Prescaler(TIM1_ICPSC_TypeDef TIM1_IC3Prescaler);
TIM1_SetIC4Prescaler(TIM1_ICPSC_TypeDef TIM1_IC4Prescaler);
// Sets the TIMx Input Capture 1 prescaler.

INPUT :   // TIM1_IC1Prescaler specifies the Input Capture prescaler new value

```
    TIM1_ICPSC_DIV1          = ((u8)0x00),
    TIM1_ICPSC_DIV2          = ((u8)0x04),
    TIM1_ICPSC_DIV4          = ((u8)0x08),
    TIM1_ICPSC_DIV8          = ((u8)0x0C)
```

****

TIM1_GetCapture1(void);   // Gets the TIM1 Input Capture 1 value.
TIM1_GetCapture2(void);   //
TIM1_GetCapture3(void);   //
TIM1_GetCapture4(void);   //
TIM1_GetCounter(void);   // Gets the TIM1 Counter value.
TIM1_GetPrescaler(void);   // Gets the TIM1 Prescaler value.

Return   (u16) DATA
Examples: (u16) ReadData = **TIM1_GetCounter**( );;

****

TIM1_GetFlagStatus(TIM1_FLAG_TypeDef TIM1_FLAG);

// Checks whether the specified TIM1 flag is set or not.

INPUT : //TIM1_FLAG specifies the flag to check.

```
    TIM1_FLAG_UPDATE         = ((u16)0x0001),
    TIM1_FLAG_CC1            = ((u16)0x0002),
    TIM1_FLAG_CC2            = ((u16)0x0004),
    TIM1_FLAG_CC3            = ((u16)0x0008),
    TIM1_FLAG_CC4            = ((u16)0x0010),
    TIM1_FLAG_COM            = ((u16)0x0020),
    TIM1_FLAG_TRIGGER        = ((u16)0x0040),
    TIM1_FLAG_BREAK          = ((u16)0x0080),
    TIM1_FLAG_CC1OF          = ((u16)0x0200),
    TIM1_FLAG_CC2OF          = ((u16)0x0400),
    TIM1_FLAG_CC3OF          = ((u16)0x0800),
    TIM1_FLAG_CC4OF          = ((u16)0x1000)
```

Return : SET or RESET //FlagStatus The new state of TIM1_FLAG

****

TIM1_ClearFlag(TIM1_FLAG_TypeDef TIM1_FLAG); // Clears the TIM1 pending flags.

INPUT : // TIM1_FLAG specifies the flag to clear. SEE **TIM1_GetFlagStatus**( );

****

TIM1_GetITStatus(TIM1_IT_TypeDef TIM1_IT); // Checks whether the TIM1 interrupt has occurred or not.

INPUT : // TIM1_IT specifies the TIM1 interrupt source to check.

```
    TIM1_IT_UPDATE         = ((u8)0x01),
    TIM1_IT_CC1            = ((u8)0x02),
    TIM1_IT_CC2            = ((u8)0x04),
    TIM1_IT_CC3            = ((u8)0x08),
    TIM1_IT_CC4            = ((u8)0x10),
    TIM1_IT_COM            = ((u8)0x20),
    TIM1_IT_TRIGGER        = ((u8)0x40),
    TIM1_IT_BREAK          = ((u8)0x80)
```

Return : SET or RESET //ITStatus The new state of the TIM1_IT

****

TIM1_ClearITPendingBit(TIM1_IT_TypeDef TIM1_IT); // Clears the TIM1's interrupt pending bits.

INPUT : // TIM1_IT specifies the pending bit to clear. SEE **TIM1_GetITStatus** ( );

*****************************   STM8S FWLIB   **************************************

***************************** STM8S FWLIB ***************************************

file stm8s_tim2.

****

```
TIM2_DeInit(void);

TIM2_TimeBaseInit(TIM2_Prescaler_TypeDef TIM2_Prescaler ,     u16 TIM2_Period);

TIM2_OC1Init(     TIM2_OCMode_TypeDef TIM2_OCMode,
                  TIM2_OutputState_TypeDef TIM2_OutputState,
                  u16 TIM2_Pulse,
                  TIM2_OCPolarity_TypeDef TIM2_OCPolarity);


TIM2_OC2Init(     TIM2_OCMode_TypeDef TIM2_OCMode,
                  TIM2_OutputState_TypeDef TIM2_OutputState,
                  u16 TIM2_Pulse,
                  TIM2_OCPolarity_TypeDef TIM2_OCPolarity);


TIM2_OC3Init(     TIM2_OCMode_TypeDef TIM2_OCMode,
                  TIM2_OutputState_TypeDef TIM2_OutputState,
                  u16 TIM2_Pulse,
                  TIM2_OCPolarity_TypeDef TIM2_OCPolarity);


TIM2_ICInit(      TIM2_Channel_TypeDef TIM2_Channel,
                  TIM2_ICPolarity_TypeDef TIM2_ICPolarity,
                  TIM2_ICSelection_TypeDef TIM2_ICSelection,
                  TIM2_ICPSC_TypeDef TIM2_ICPrescaler,
                  u8 TIM2_ICFilter);


TIM2_PWMIConfig(     TIM2_Channel_TypeDef TIM2_Channel,
                     TIM2_ICPolarity_TypeDef TIM2_ICPolarity,
                     TIM2_ICSelection_TypeDef TIM2_ICSelection,
                     TIM2_ICPSC_TypeDef TIM2_ICPrescaler,
                     u8 TIM2_ICFilter);
```

```
TIM2_Cmd(FunctionalState NewState);
TIM2_ITConfig(TIM2_IT_TypeDef TIM2_IT, FunctionalState NewState);
TIM2_InternalClockConfig(void);
TIM2_UpdateDisableConfig(FunctionalState Newstate);
TIM2_UpdateRequestConfig(TIM2_UpdateSource_TypeDef TIM2_UpdateSource);
TIM2_SelectOnePulseMode(TIM2_OPMode_TypeDef TIM2_OPMode);
TIM2_PrescalerConfig( TIM2_Prescaler_TypeDef Prescaler,
                      TIM2_PSCReloadMode_TypeDef TIM2_PSCReloadMode);
TIM2_ForcedOC1Config(TIM2_ForcedAction_TypeDef TIM2_ForcedAction);
TIM2_ForcedOC2Config(TIM2_ForcedAction_TypeDef TIM2_ForcedAction);
TIM2_ForcedOC3Config(TIM2_ForcedAction_TypeDef TIM2_ForcedAction);
TIM2_ARRPreloadConfig(FunctionalState Newstate);
TIM2_CCPreloadControl(FunctionalState Newstate);
TIM2_OC1PreloadConfig(FunctionalState Newstate);
TIM2_OC2PreloadConfig(FunctionalState Newstate);
TIM2_OC3PreloadConfig(FunctionalState Newstate);
TIM2_GenerateEvent(TIM2_EventSource_TypeDef TIM2_EventSource);
TIM2_OC1PolarityConfig(TIM2_OCPolarity_TypeDef TIM2_OCPolarity);
TIM2_OC2PolarityConfig(TIM2_OCPolarity_TypeDef TIM2_OCPolarity);
TIM2_OC3PolarityConfig(TIM2_OCPolarity_TypeDef TIM2_OCPolarity);
TIM2_CCxCmd(TIM2_Channel_TypeDef TIM2_Channel, FunctionalState Newstate);
TIM2_SelectOCxM(TIM2_Channel_TypeDef TIM2_Channel, TIM2_OCMode_TypeDef TIM2_OCMode);
TIM2_SetCounter(u16 Counter);
TIM2_SetAutoreload(u16 Autoreload);
TIM2_SetCompare1(u16 Compare1);
TIM2_SetCompare2(u16 Compare2);
TIM2_SetCompare3(u16 Compare3);
TIM2_SetIC1Prescaler(TIM2_ICPSC_TypeDef TIM2_IC1Prescaler);
TIM2_SetIC2Prescaler(TIM2_ICPSC_TypeDef TIM2_IC2Prescaler);
TIM2_SetIC3Prescaler(TIM2_ICPSC_TypeDef TIM2_IC3Prescaler);
TIM2_GetCapture1(void);
```

TIM2_GetCapture2(void);

TIM2_GetCapture3(void);

TIM2_GetCounter(void);

TIM2_Prescaler_TypeDef TIM2_GetPrescaler(void);

TIM2_GetFlagStatus(TIM2_FLAG_TypeDef TIM2_FLAG);

TIM2_ClearFlag(TIM2_FLAG_TypeDef TIM2_FLAG);

TIM2_GetITStatus(TIM2_IT_TypeDef TIM2_IT);

TIM2_ClearITPendingBit(TIM2_IT_TypeDef TIM2_IT);

****

****

TIM2_DeInit(void); // Deinitializes the TIM2 peripheral registers to their default reset values.

****

TIM2_TimeBaseInit(TIM2_Prescaler_TypeDef TIM2_Prescaler , u16 TIM2_Period);

// Initializes the TIM2 Time Base Unit according to the specified parameters.

INPUT : // TIM2_Prescaler specifies the Prescaler from TIM2_Prescaler_TypeDef.

```
TIM2_PRESCALER_1      = ((u8)0x00),
TIM2_PRESCALER_2      = ((u8)0x01),
TIM2_PRESCALER_4      = ((u8)0x02),
TIM2_PRESCALER_8      = ((u8)0x03),
TIM2_PRESCALER_16     = ((u8)0x04),
TIM2_PRESCALER_32     = ((u8)0x05),
TIM2_PRESCALER_64     = ((u8)0x06),
TIM2_PRESCALER_128    = ((u8)0x07),
TIM2_PRESCALER_256    = ((u8)0x08),
TIM2_PRESCALER_512    = ((u8)0x09),
TIM2_PRESCALER_1024   = ((u8)0x0A),
TIM2_PRESCALER_2048   = ((u8)0x0B),
TIM2_PRESCALER_4096   = ((u8)0x0C),
TIM2_PRESCALER_8192   = ((u8)0x0D),
TIM2_PRESCALER_16384  = ((u8)0x0E),
TIM2_PRESCALER_32768  = ((u8)0x0F)
```

// TIM2_Period specifies the Period value.

u16 TIM2_Period

****

TIM2_OC1Init(TIM2_OCMode_TypeDef TIM2_OCMode,
TIM2_OutputState_TypeDef TIM2_OutputState,
u16 TIM2_Pulse,
TIM2_OCPolarity_TypeDef TIM2_OCPolarity);

INPUT : // TIM2_OCMode specifies the Output Compare mode from @ref TIM2_OCMode_TypeDef.

```
TIM2_OCMODE_TIMING          = ((u8)0x00),
TIM2_OCMODE_ACTIVE              = ((u8)0x10),
TIM2_OCMODE_INACTIVE            = ((u8)0x20),
TIM2_OCMODE_TOGGLE              = ((u8)0x30),
TIM2_OCMODE_PWM1                = ((u8)0x60),
TIM2_OCMODE_PWM2                = ((u8)0x70)
        //TIM2_OutputState specifies the Output State  from @ref TIM2_OutputState_TypeDef.
TIM2_OUTPUTSTATE_DISABLE        = ((u8)0x00),
TIM2_OUTPUTSTATE_ENABLE         = ((u8)0x11)
        //TIM2_Pulse specifies the Pulse width  value.
u16 TIM2_Pulse,
        //TIM2_OCPolarity specifies the Output Compare Polarity  from @ref TIM2_OCPolarity_TypeDef.
TIM2_OCPOLARITY_HIGH            = ((u8)0x00),
TIM2_OCPOLARITY_LOW             = ((u8)0x22)
```

****

```
    TIM2_OC2Init(   TIM2_OCMode_TypeDef TIM2_OCMode,
                    TIM2_OutputState_TypeDef TIM2_OutputState,
                    u16 TIM2_Pulse,
                    TIM2_OCPolarity_TypeDef TIM2_OCPolarity);
```

SEE **TIM2_OC1Init**( )

****

```
    TIM2_OC3Init(   TIM2_OCMode_TypeDef TIM2_OCMode,
                    TIM2_OutputState_TypeDef TIM2_OutputState,
                    u16 TIM2_Pulse,
                    TIM2_OCPolarity_TypeDef TIM2_OCPolarity);
```

SEE **TIM2_OC1Init**( )

****

```
    TIM2_ICInit(    TIM2_Channel_TypeDef TIM2_Channel,
                    TIM2_ICPolarity_TypeDef TIM2_ICPolarity,
                    TIM2_ICSelection_TypeDef TIM2_ICSelection,
                    TIM2_ICPSC_TypeDef TIM2_ICPrescaler,
                    u8 TIM2_ICFilter);
```

```
INPUT  :    // TIM2_Channel specifies the Input Capture Channel from @ref TIM2_Channel_TypeDef.
    TIM2_CHANNEL_1                  = ((u8)0x00),
    TIM2_CHANNEL_2                  = ((u8)0x01),
    TIM2_CHANNEL_3                  = ((u8)0x02)
            //TIM2_ICPolarity specifies the Input Capture Polarity from @ref TIM2_ICPolarity_TypeDef.
    TIM2_ICPOLARITY_RISING          = ((u8)0x00),
    TIM2_ICPOLARITY_FALLING         = ((u8)0x44)
            //TIM2_ICSelection specifies the Input Capture Selection from @ref TIM2_ICSelection_TypeDef.
    TIM2_ICSELECTION_DIRECTTI       = ((u8)0x01),
    TIM2_ICSELECTION_INDIRECTTI     = ((u8)0x02),
    TIM2_ICSELECTION_TRGI           = ((u8)0x03)
            //TIM2_ICPrescaler specifies the Input Capture Prescaler from @ref TIM2_ICPSC_TypeDef.
    TIM2_ICPSC_DIV1                 = ((u8)0x00),
```

TIM2_ICPSC_DIV2 = ((u8)0x04),
TIM2_ICPSC_DIV4 = ((u8)0x08),
TIM2_ICPSC_DIV8 = ((u8)0x0C)

//TIM2_ICFilter specifies the Input Capture Filter value (value can be an integer from 0x00 to 0x0F).

u8 TIM2_ICFilter

****

TIM2_PWMIConfig(TIM2_Channel_TypeDef TIM2_Channel,
TIM2_ICPolarity_TypeDef TIM2_ICPolarity,
TIM2_ICSelection_TypeDef TIM2_ICSelection,
TIM2_ICPSC_TypeDef TIM2_ICPrescaler,
u8 TIM2_ICFilter);

// Configures the TIM2 peripheral in PWM Input Mode according to the specified parameters.

INPUT : SEE **TIM2_ICInit**( )

****

TIM2_Cmd(FunctionalState NewState);

INPUT : DISABLE ; ENABLE

****

TIM2_ITConfig(TIM2_IT_TypeDef TIM2_IT, FunctionalState NewState);

INPUT : //TIM2_IT specifies the TIM2 interrupts sources

TIM2_IT_UPDATE = ((u8)0x01),
TIM2_IT_CC1 = ((u8)0x02),
TIM2_IT_CC2 = ((u8)0x04),
TIM2_IT_CC3 = ((u8)0x08)

// NewState new state of the TIM2 peripheral.

DISABLE ; ENABLE

****

TIM2_InternalClockConfig(void);

■

****

TIM2_UpdateDisableConfig(FunctionalState Newstate); // Enables or Disables the TIM2 Update event.

INPUT : DISABLE ; ENABLE

****

TIM2_UpdateRequestConfig(TIM2_UpdateSource_TypeDef TIM2_UpdateSource);

// Selects the TIM2 Update Request Interrupt source.

INPUT : // TIM2_UpdateSource specifies the Update source.

TIM2_UPDATESOURCE_GLOBAL = ((u8)0x00),
TIM2_UPDATESOURCE_REGULAR = ((u8)0x01)

****

TIM2_SelectOnePulseMode(TIM2_OPMode_TypeDef TIM2_OPMode);

INPUT : // TIM2_OPMode specifies the OPM Mode to be used.

TIM2_OPMODE_SINGLE = ((u8)0x01),
TIM2_OPMODE_REPETITIVE = ((u8)0x00)

****

TIM2_PrescalerConfig(TIM2_Prescaler_TypeDef Prescaler,
TIM2_PSCReloadMode_TypeDef TIM2_PSCReloadMode);

INPUT : // Prescaler specifies the Prescaler Register value

TIM2_PRESCALER_1 = ((u8)0x00),
TIM2_PRESCALER_2 = ((u8)0x01),
TIM2_PRESCALER_4 = ((u8)0x02),
TIM2_PRESCALER_8 = ((u8)0x03),
TIM2_PRESCALER_16 = ((u8)0x04),
TIM2_PRESCALER_32 = ((u8)0x05),
TIM2_PRESCALER_64 = ((u8)0x06),
TIM2_PRESCALER_128 = ((u8)0x07),
TIM2_PRESCALER_256 = ((u8)0x08),
TIM2_PRESCALER_512 = ((u8)0x09),
TIM2_PRESCALER_1024 = ((u8)0x0A),
TIM2_PRESCALER_2048 = ((u8)0x0B),
TIM2_PRESCALER_4096 = ((u8)0x0C),
TIM2_PRESCALER_8192 = ((u8)0x0D),
TIM2_PRESCALER_16384 = ((u8)0x0E),
TIM2_PRESCALER_32768 = ((u8)0x0F)

// TIM2_PSCReloadMode specifies the TIM2 Prescaler Reload mode.

TIM2_PSCRELOADMODE_UPDATE = ((u8)0x00),
TIM2_PSCRELOADMODE_IMMEDIATE = ((u8)0x01)

****

TIM2_ForcedOC1Config(TIM2_ForcedAction_TypeDef TIM2_ForcedAction);

// Forces the TIM2 Channel1 output waveform to active or inactive level.

INPUT : // TIM2_ForcedAction specifies the forced Action to be set to the output waveform.

TIM2_FORCEDACTION_ACTIVE = ((u8)0x50),
TIM2_FORCEDACTION_INACTIVE = ((u8)0x40)

****

TIM2_ForcedOC2Config(TIM2_ForcedAction_TypeDef TIM2_ForcedAction);

SEE **TIM2_ForcedOC1Config** ( )

****

TIM2_ForcedOC3Config(TIM2_ForcedAction_TypeDef TIM2_ForcedAction);

SEE **TIM2_ForcedOC1Config** ( )

****

TIM2_ARRPreloadConfig(FunctionalState Newstate);

// Enables or disables TIM2 peripheral Preload register on ARR.

INPUT : DISABLE ; ENABLE

****

TIM2_CCPreloadControl(FunctionalState Newstate);

■//

INPUT : DISABLE ; ENABLE

****

TIM2_OC1PreloadConfig(FunctionalState Newstate);

// Enables or disables the TIM2 peripheral Preload Register on CCR1.

INPUT : DISABLE ; ENABLE

****

TIM2_OC2PreloadConfig(FunctionalState Newstate);

INPUT : DISABLE ; ENABLE

****

TIM2_OC3PreloadConfig(FunctionalState Newstate);

INPUT : DISABLE ; ENABLE

****

TIM2_GenerateEvent(TIM2_EventSource_TypeDef TIM2_EventSource);

// Configures the TIM2 event to be generated by software.

INPUT : // TIM2_EventSource specifies the event source.

TIM2_EVENTSOURCE_UPDATE = ((u8)0x01),
TIM2_EVENTSOURCE_CC1 = ((u8)0x02),
TIM2_EVENTSOURCE_CC2 = ((u8)0x04),
TIM2_EVENTSOURCE_CC3 = ((u8)0x08)

****

TIM2_OC1PolarityConfig(TIM2_OCPolarity_TypeDef TIM2_OCPolarity);

INPUT : // TIM2_OCPolarity specifies the OC1 Polarity.

TIM2_OCPOLARITY_HIGH = ((u8)0x00),
TIM2_OCPOLARITY_LOW = ((u8)0x22)

****

TIM2_OC2PolarityConfig(TIM2_OCPolarity_TypeDef TIM2_OCPolarity);

SEE **TIM2_OC1PolarityConfig** ( )

****

TIM2_OC3PolarityConfig(TIM2_OCPolarity_TypeDef TIM2_OCPolarity);

SEE **TIM2_OC1PolarityConfig** ( )

****

TIM2_CCxCmd(TIM2_Channel_TypeDef TIM2_Channel, FunctionalState Newstate);

// Enables or disables the TIM2 Capture Compare Channel x.

INPUT : // TIM2_Channel specifies the TIM2 Channel.

TIM2_CHANNEL_1 = ((u8)0x00),
TIM2_CHANNEL_2 = ((u8)0x01),

TIM2_CHANNEL_3 = ((u8)0x02)

// NewState specifies the TIM2 Channel CCxE bit new state.

ENABLE or DISABLE

****

TIM2_SelectOCxM(TIM2_Channel_TypeDef TIM2_Channel,
TIM2_OCMode_TypeDef TIM2_OCMode);

INPUT : // TIM2_Channel specifies the TIM2 Channel.

TIM2_CHANNEL_1 = ((u8)0x00),
TIM2_CHANNEL_2 = ((u8)0x01),
TIM2_CHANNEL_3 = ((u8)0x02)

// TIM2_OCMode specifies the TIM2 Output Compare Mode.

TIM2_OCMODE_TIMING = ((u8)0x00),
TIM2_OCMODE_ACTIVE = ((u8)0x10),
TIM2_OCMODE_TOGGLE = ((u8)0x30),
TIM2_OCMODE_PWM1 = ((u8)0x60),
TIM2_OCMODE_PWM2 = ((u8)0x70)
TIM2_FORCEDACTION_ACTIVE = ((u8)0x50),
TIM2_FORCEDACTION_INACTIVE = ((u8)0x40)

****

TIM2_SetCounter(u16 Counter); // Sets the TIM2 Counter Register value.

INPUT : // Counter specifies the Counter register new value.

u16 Counter

****

TIM2_SetAutoreload(u16 Autoreload); // Sets the TIM2 Autoreload Register value.

INPUT : // Autoreload specifies the Autoreload register new value.

u16 Autoreload

****

TIM2_SetCompare1(u16 Compare1); // Sets the TIM2 Capture Compare1 Register value.

INPUT : // Compare1 specifies the Capture Compare1 register new value.

u16 Compare1

****

TIM2_SetCompare2(u16 Compare2);

SEE **TIM2_SetCompare1** ( )

****

TIM2_SetCompare3(u16 Compare3);

SEE **TIM2_SetCompare1** ( )

****

TIM2_SetIC1Prescaler(TIM2_ICPSC_TypeDef TIM2_IC1Prescaler);

INPUT : // TIM2_IC1Prescaler specifies the Input Capture prescaler new value

TIM2_ICPSC_DIV1 = ((u8)0x00),

TIM2_ICPSC_DIV2 = ((u8)0x04),
TIM2_ICPSC_DIV4 = ((u8)0x08),
TIM2_ICPSC_DIV8 = ((u8)0x0C)

****

TIM2_SetIC2Prescaler(TIM2_ICPSC_TypeDef TIM2_IC2Prescaler);

SEE **TIM2_SetIC1Prescaler** ( )

****

TIM2_SetIC3Prescaler(TIM2_ICPSC_TypeDef TIM2_IC3Prescaler);

SEE **TIM2_SetIC1Prescaler** ( )

****

TIM2_GetCapture1(void); // Gets the TIM2 Input Capture 1 value.

Return : (u16)DATA //Capture Compare 1 Register value.

****

TIM2_GetCapture2(void);

SEE **TIM2_GetCapture1** ( )

****

TIM2_GetCapture3(void);

SEE **TIM2_GetCapture1** ( )

****

TIM2_GetCounter(void); // Gets the TIM2 Counter value.

Return : (u16)DATA // Counter Register value.

****

TIM2_GetPrescaler(void); // Gets the TIM2 Prescaler value.

Return : // Prescaler Register configuration value @ref TIM2_Prescaler_TypeDef.

((u8)0x00)	TIM2_PRESCALER_1
((u8)0x01)	TIM2_PRESCALER_2
((u8)0x02)	TIM2_PRESCALER_4
((u8)0x03)	TIM2_PRESCALER_8
((u8)0x04)	TIM2_PRESCALER_16
((u8)0x05)	TIM2_PRESCALER_32
((u8)0x06)	TIM2_PRESCALER_64
((u8)0x07)	TIM2_PRESCALER_128
((u8)0x08)	TIM2_PRESCALER_256
((u8)0x09)	TIM2_PRESCALER_512
((u8)0x0A)	TIM2_PRESCALER_1024
((u8)0x0B)	TIM2_PRESCALER_2048
((u8)0x0C)	TIM2_PRESCALER_4096
((u8)0x0D)	TIM2_PRESCALER_8192
((u8)0x0E)	TIM2_PRESCALER_16384
((u8)0x0F)	TIM2_PRESCALER_32768

****

TIM2_GetFlagStatus(TIM2_FLAG_TypeDef TIM2_FLAG);

//Checks whether the specified TIM2 flag is set or not.

INPUT : // TIM2_FLAG specifies the flag to check.

TIM2_FLAG_UPDATE = ((u16)0x0001),
TIM2_FLAG_CC1 = ((u16)0x0002),
TIM2_FLAG_CC2 = ((u16)0x0004),
TIM2_FLAG_CC3 = ((u16)0x0008),
TIM2_FLAG_CC1OF = ((u16)0x0200),
TIM2_FLAG_CC2OF = ((u16)0x0400),
TIM2_FLAG_CC3OF = ((u16)0x0800)

Return : SET or RESET //FlagStatus The new state of TIM2_FLAG (SET or RESET).

****

TIM2_ClearFlag(TIM2_FLAG_TypeDef TIM2_FLAG);

INPUT : // TIM2_FLAG specifies the flag to clear. SEE **TIM2_GetFlagStatus( )**

****

TIM2_GetITStatus(TIM2_IT_TypeDef TIM2_IT);

// Checks whether the TIM2 interrupt has occurred or not.

INPUT : //TIM2_IT specifies the TIM2 interrupt source to check.

TIM2_IT_UPDATE = ((u8)0x01),
TIM2_IT_CC1 = ((u8)0x02),
TIM2_IT_CC2 = ((u8)0x04),
TIM2_IT_CC3 = ((u8)0x08)

Return : SET or RESET //ITStatus The new state of the TIM2_IT(SET or RESET).

****

TIM2_ClearITPendingBit(TIM2_IT_TypeDef TIM2_IT);

//Clears the TIM2's interrupt pending bits.

INPUT : // TIM2_IT specifies the pending bit to clear. SEE **TIM2_GetITStatus( )**

***************************** STM8S FWLIB ***************************************

file stm8s_tim4.

****

TIM4_DeInit(void);

TIM4_TimeBaseInit(TIM4_Prescaler_TypeDef TIM4_Prescaler, u8 TIM4_Period);

TIM4_Cmd(FunctionalState NewState);

TIM4_ITConfig(TIM4_IT_TypeDef TIM4_IT, FunctionalState NewState);

TIM4_UpdateDisableConfig(FunctionalState Newstate);

TIM4_UpdateRequestConfig(TIM4_UpdateSource_TypeDef TIM4_UpdateSource);

TIM4_SelectOnePulseMode(TIM4_OPMode_TypeDef TIM4_OPMode);

TIM4_PrescalerConfig(TIM4_Prescaler_TypeDef Prescaler,

TIM4_PSCReloadMode_TypeDef TIM4_PSCReloadMode);

TIM4_ARRPreloadConfig(FunctionalState Newstate);

TIM4_GenerateEvent(TIM4_EventSource_TypeDef TIM4_EventSource);

TIM4_SetCounter(u8 Counter);

TIM4_SetAutoreload(u8 Autoreload);

TIM4_GetCounter(void);

TIM4_GetPrescaler(void);

TIM4_GetFlagStatus(TIM4_FLAG_TypeDef TIM4_FLAG);

TIM4_ClearFlag(TIM4_FLAG_TypeDef TIM4_FLAG);

TIM4_GetITStatus(TIM4_IT_TypeDef TIM4_IT);

TIM4_ClearITPendingBit(TIM4_IT_TypeDef TIM4_IT);

****

****

TIM4_DeInit(void);

****

TIM4_TimeBaseInit(TIM4_Prescaler_TypeDef TIM4_Prescaler, u8 TIM4_Period);

INPUT :   // TIM4_Prescaler specifies the Prescaler from TIM4_Prescaler_TypeDef.

```
    TIM4_PRESCALER_1          = ((u8)0x00),
    TIM4_PRESCALER_2          = ((u8)0x01),
    TIM4_PRESCALER_4          = ((u8)0x02),
    TIM4_PRESCALER_8          = ((u8)0x03),
    TIM4_PRESCALER_16         = ((u8)0x04),
    TIM4_PRESCALER_32         = ((u8)0x05),
```

TIM4_PRESCALER_64 = ((u8)0x06),

TIM4_PRESCALER_128 = ((u8)0x07)

// TIM4_Period specifies the Period value.

u8 TIM4_Period

****

TIM4_Cmd(FunctionalState NewState);

INPUT : DISABLE ; ENABLE

****

TIM4_ITConfig(TIM4_IT_TypeDef TIM4_IT, FunctionalState NewState);

INPUT : // TIM4_IT specifies the TIM4 interrupts sources

TIM4_IT_UPDATE = ((u8)0x01)

DISABLE ; ENABLE //NewState new state of the TIM4 peripheral.

****

TIM4_UpdateDisableConfig(FunctionalState Newstate); // Enables or Disables the TIM4 Update event.

INPUT : DISABLE ; ENABLE

****

TIM4_UpdateRequestConfig(TIM4_UpdateSource_TypeDef TIM4_UpdateSource);

// Selects the TIM4 Update Request Interrupt source.

INPUT : // TIM4_UpdateSource specifies the Update source.

TIM4_UPDATESOURCE_GLOBAL = ((u8)0x00),

TIM4_UPDATESOURCE_REGULAR = ((u8)0x01)

****

TIM4_SelectOnePulseMode(TIM4_OPMode_TypeDef TIM4_OPMode);

INPUT : // TIM4_OPMode specifies the OPM Mode to be used.

TIM4_OPMODE_SINGLE = ((u8)0x01), //单个

TIM4_OPMODE_REPETITIVE = ((u8)0x00) //重复

****

TIM4_PrescalerConfig(TIM4_Prescaler_TypeDef Prescaler,

TIM4_PSCReloadMode_TypeDef TIM4_PSCReloadMode);

INPUT : // Prescaler specifies the Prescaler Register value

TIM4_PRESCALER_1 = ((u8)0x00),

TIM4_PRESCALER_2 = ((u8)0x01),

TIM4_PRESCALER_4 = ((u8)0x02),

TIM4_PRESCALER_8 = ((u8)0x03),

TIM4_PRESCALER_16 = ((u8)0x04),

TIM4_PRESCALER_32 = ((u8)0x05),

TIM4_PRESCALER_64 = ((u8)0x06),

TIM4_PRESCALER_128 = ((u8)0x07)

// TIM4_PSCReloadMode specifies the TIM4 Prescaler Reload mode.

TIM4_PSCRELOADMODE_UPDATE = ((u8)0x00), //更新时重载

TIM4_PSCRELOADMODE_IMMEDIATE = ((u8)0x01) //立即重载

****

TIM4_ARRPreloadConfig(FunctionalState Newstate);

INPUT : DISABLE ; ENABLE

****

TIM4_GenerateEvent(TIM4_EventSource_TypeDef TIM4_EventSource); //配置将由软件引发的 TIM 事件

INPUT : //TIM4_EventSource specifies the event source.

TIM4_EVENTSOURCE_UPDATE = ((u8)0x01)

****

TIM4_SetCounter(u8 Counter); //Sets the TIM4 Counter Register value.

INPUT : u8 Counter //Counter specifies the Counter register new value.

****

TIM4_SetAutoreload(u8 Autoreload); //Sets the TIM4 Autoreload Register value.

INPUT : u8 Autoreload //Autoreload specifies the Autoreload register new value.

****

TIM4_GetCounter(void); //Gets the TIM4 Counter value.

Return u8 DATA //Counter Register value.

****

TIM4_GetPrescaler(void); //Gets the TIM4 Prescaler value.

Return 0x00 ~ 0x07 // Prescaler Register configuration value. 1 , 2 , 4 , 8 , 16 , 32 , 64 , 128

****

TIM4_GetFlagStatus(TIM4_FLAG_TypeDef TIM4_FLAG); // Checks whether the specified TIM4 flag

---

INPUT : // TIM4_FLAG specifies the flag to check.

TIM4_FLAG_UPDATE = ((u8)0x01)

Return SET or RESET // FlagStatus The new state of TIM4_FLAG (SET or RESET).

****

TIM4_ClearFlag(TIM4_FLAG_TypeDef TIM4_FLAG);

INPUT : // TIM4_FLAG specifies the flag to Clear

TIM4_FLAG_UPDATE = ((u8)0x01)

****

TIM4_GetITStatus(TIM4_IT_TypeDef TIM4_IT); //Checks whether the TIM4 interrupt has occurred or not.

INPUT : TIM4_IT_UPDATE = ((u8)0x01)

Return SET or RESET // ITStatus The new state of the TIM4_IT (SET or RESET).

****

TIM4_ClearITPendingBit(TIM4_IT_TypeDef TIM4_IT);

INPUT : TIM4_IT_UPDATE = ((u8)0x01)

```
****************************    STM8S FWLIB    **************************************
ADC2_DeInit(void);
ADC2_Init(   ADC2_ConvMode_TypeDef ADC2_ConversionMode,
             ADC2_Channel_TypeDef ADC2_Channel,
             ADC2_PresSel_TypeDef ADC2_PrescalerSelection,
             ADC2_ExtTrig_TypeDef ADC2_ExtTrigger,
             DC2_ExtTriggerState,
             ADC2_Align_TypeDef ADC2_Align,
             ADC2_SchmittTrigg_TypeDef ADC2_SchmittTriggerChannel,
             FunctionalState ADC2_SchmittTriggerState);
ADC2_Cmd(FunctionalState NewState);
ADC2_ITConfig(FunctionalState NewState);
ADC2_PrescalerConfig(ADC2_PresSel_TypeDef ADC2_Prescaler);
ADC2_SchmittTriggerConfig(    ADC2_SchmittTrigg_TypeDef ADC2_SchmittTriggerChannel,
                              FunctionalState NewState);
ADC2_ConversionConfig(   ADC2_ConvMode_TypeDef ADC2_ConversionMode,
                         ADC2_Channel_TypeDef ADC2_Channel,
                         ADC2_Align_TypeDef ADC2_Align);
ADC2_ExternalTriggerConfig(ADC2_ExtTrig_TypeDef ADC2_ExtTrigger, FunctionalState NewState);
ADC2_StartConversion(void);
ADC2_GetConversionValue(void);
ADC2_GetFlagStatus(void);
ADC2_ClearFlag(void);
ADC2_GetITStatus(void);
ADC2_ClearITPendingBit(void);
******************************************************************************************
AWU_DeInit(void);
AWU_Init(AWU_Timebase_TypeDef AWU_TimeBase);
AWU_Cmd(FunctionalState NewState);
AWU_LSICalibrationConfig(u32 LSIFreqHz);
AWU_IdleModeEnable(void);
```

```
AWU_ReInitCounter(void);
AWU_GetFlagStatus(void);
************************************************************************************************
CAN_DeInit(void);
CAN_Init(           CAN_MasterCtrl_TypeDef CAN_MasterCtrl,CAN_Mode_TypeDef CAN_Mode,
                    CAN_SynJumpWidth_TypeDef CAN_SynJumpWidth,
                    CAN_BitSeg1_TypeDef CAN_BitSeg1,
                    CAN_BitSeg2_TypeDef CAN_BitSeg2,
                    CAN_ClockSource_TypeDef CAN_ClockSource,
                    u8 CAN_Prescaler);
CAN_FilterInit(     CAN_FilterNumber_TypeDef CAN_FilterNumber,
                    FunctionalState CAN_FilterActivation,
                    CAN_FilterMode_TypeDef CAN_FilterMode,
                    CAN_FilterScale_TypeDef CAN_FilterScale,
                    u8 CAN_FilterID1,
                    u8 CAN_FilterID2,
                    u8 CAN_FilterID3,
                    u8 CAN_FilterID4,
                    u8 CAN_FilterIDMask1,
                    u8 CAN_FilterIDMask2,
                    u8 CAN_FilterIDMask3,
                    u8 CAN_FilterIDMask4);
CAN_ITConfig(CAN_IT_TypeDef CAN_IT, FunctionalState NewState);
CAN_ST7CompatibilityCmd(CAN_ST7Compatibility_TypeDef CAN_ST7Compatibility);
CAN_Transmit(       u32 CAN_Id,  CAN_Id_TypeDef CAN_IDE,
                    CAN_RTR_TypeDef CAN_RTR,  u8 CAN_DLC,     u8 *CAN_Data);
CAN_TTComModeCmd(FunctionalState NewState);
CAN_TransmitStatus(CAN_TransmitMailBox_TypeDef CAN_TransmitMailbox);
CAN_CancelTransmit(CAN_TransmitMailBox_TypeDef CAN_TransmitMailbox);
CAN_FIFORelease(void);
CAN_MessagePending(void);
```

```
CAN_Receive(void);
CAN_GetReceivedId(void);
CAN_GetReceivedIDE(void);
CAN_GetReceivedRTR(void);
CAN_GetReceivedDLC(void);
CAN_GetReceivedData(u8 CAN_DataIndex);
CAN_GetReceivedFMI(void);
CAN_GetMessageTimeStamp(void);
CAN_Sleep(void);
CAN_WakeUp(void);
CAN_SelectClock(CAN_ClockSource_TypeDef CAN_ClockSource);
CAN_OperatingModeRequest(CAN_OperatingMode_TypeDef CAN_OperatingMode);
CAN_GetLastErrorCode(void);
CAN_GetSelectedPage(void);
CAN_SelectPage(CAN_Page_TypeDef CAN_Page);
CAN_GetFlagStatus(CAN_Flag_TypeDef CAN_Flag);
CAN_ClearFlag(CAN_Flag_TypeDef CAN_FLAG);
CAN_GetITStatus(CAN_IT_TypeDef CAN_IT);
CAN_ClearITPendingBit(CAN_IT_TypeDef CAN_IT);
**********************************************************************************************
I2C_DeInit(void);
I2C_Init(       u32 OutputClockFrequencyHz,
                u16 OwnAddress,
                I2C_DutyCycle_TypeDef DutyCycle,
                I2C_Ack_TypeDef Ack,
                I2C_AddMode_TypeDef AddMode,
                u8 InputClockFrequencyMHz );
I2C_Cmd(FunctionalState NewState);
I2C_GeneralCallCmd(FunctionalState NewState);
I2C_GenerateSTART(FunctionalState NewState);
I2C_GenerateSTOP(FunctionalState NewState);
```

```
I2C_SoftwareResetCmd(FunctionalState NewState);

I2C_StretchClockCmd(FunctionalState NewState);

I2C_AcknowledgeConfig(I2C_Ack_TypeDef Ack);

I2C_FastModeDutyCycleConfig(I2C_DutyCycle_TypeDef DutyCycle);

I2C_ITConfig(I2C_IT_TypeDef ITName, FunctionalState NewState);

I2C_CheckEvent(I2C_Event_TypeDef I2C_Event);

I2C_ReceiveData(void);

I2C_Send7bitAddress(u8 Address, I2C_Direction_TypeDef Direction);

I2C_SendData(u8 Data);

I2C_GetFlagStatus(I2C_Flag_TypeDef Flag);

I2C_ClearFlag(I2C_Flag_TypeDef Flag);

I2C_GetITStatus(I2C_ITPendingBit_TypeDef ITPendingBit);

I2C_ClearITPendingBit(I2C_ITPendingBit_TypeDef ITPendingBit);

*********************************************************************************

ITC_GetCPUCC(void);

ITC_DeInit(void);

ITC_GetSoftIntStatus(void);

ITC_SetSoftwarePriority(ITC_Irq_TypeDef IrqNum, ITC_PriorityLevel_TypeDef PriorityValue);

ITC_GetSoftwarePriority(ITC_Irq_TypeDef IrqNum);

*********************************************************************************

RST_GetFlagStatus(RST_Flag_TypeDef RST_Flag);

RST_ClearFlag(RST_Flag_TypeDef RST_Flag);

*********************************************************************************

SPI_DeInit(void);

SPI_Init( SPI_FirstBit_TypeDef FirstBit,

          SPI_BaudRatePrescaler_TypeDef BaudRatePrescaler,

          SPI_Mode_TypeDef Mode,

          SPI_ClockPolarity_TypeDef ClockPolarity,

          SPI_ClockPhase_TypeDef ClockPhase,

          SPI_DataDirection_TypeDef Data_Direction,

          SPI_NSS_TypeDef Slave_Management,
          u8 CRCPolynomial);
SPI_Cmd(FunctionalState NewState);
SPI_ITConfig(SPI_IT_TypeDef SPI_IT, FunctionalState NewState);
SPI_SendData(u8 Data);
SPI_ReceiveData(void);
SPI_NSSInternalSoftwareCmd(FunctionalState NSS_NewState);
SPI_TransmitCRC(void);
SPI_CalculateCRCCmd(FunctionalState NewState);
SPI_GetCRC(SPI_CRC_TypeDef SPI_CRC);
SPI_ResetCRC(void);
SPI_GetCRCPolynomial(void);
SPI_BiDirectionalLineConfig(SPI_Direction_TypeDef SPI_Direction);
SPI_GetFlagStatus(SPI_Flag_TypeDef SPI_FLAG);
SPI_ClearFlag(SPI_Flag_TypeDef SPI_FLAG);
SPI_GetITStatus(SPI_IT_TypeDef SPI_IT);
SPI_ClearITPendingBit(SPI_IT_TypeDef SPI_IT);
******************************************************************************************
UART1_DeInit(void);
UART1_Init(  u32 BaudRate,
             UART1_WordLength_TypeDef WordLength,
             UART1_StopBits_TypeDef StopBits,
             UART1_Parity_TypeDef Parity,
             UART1_SyncMode_TypeDef SyncMode,
             UART1_Mode_TypeDef Mode);
UART1_Cmd(FunctionalState NewState);
UART1_ITConfig(UART1_IT_TypeDef UART1_IT, FunctionalState NewState);
UART1_HalfDuplexCmd(FunctionalState NewState);
UART1_IrDAConfig(UART1_IrDAMode_TypeDef UART1_IrDAMode);
UART1_IrDACmd(FunctionalState NewState);
UART1_LINBreakDetectionConfig(UART1_LINBreakDetectionLength_TypeDef--
                                    UART1_LINBreakDetectionLength);
```

```
UART1_LINCmd(FunctionalState NewState);
UART1_SmartCardCmd(FunctionalState NewState);
UART1_SmartCardNACKCmd(FunctionalState NewState);
UART1_WakeUpConfig(UART1_WakeUp_TypeDef UART1_WakeUp);
UART1_ReceiverWakeUpCmd(FunctionalState NewState);
UART1_ReceiveData8(void);
UART1_ReceiveData9(void);
UART1_SendData8(u8 Data);
UART1_SendData9(u16 Data);
UART1_SendBreak(void);
UART1_SetAddress(u8 UART1_Address);
UART1_SetGuardTime(u8 UART1_GuardTime);
UART1_SetPrescaler(u8 UART1_Prescaler);
UART1_GetFlagStatus(UART1_Flag_TypeDef UART1_FLAG);
UART1_ClearFlag(UART1_Flag_TypeDef UART1_FLAG);
UART1_GetITStatus(UART1_IT_TypeDef UART1_IT);
UART1_ClearITPendingBit(UART1_IT_TypeDef UART1_IT);
*************************************************************************************************
UART2_DeInit(void);
UART2_Init(  u32 BaudRate,
             UART2_WordLength_TypeDef WordLength,
             UART2_StopBits_TypeDef StopBits,
             UART2_Parity_TypeDef Parity,
             UART2_SyncMode_TypeDef SyncMode,
             UART2_Mode_TypeDef Mode);
UART2_Cmd(FunctionalState NewState);
UART2_ITConfig(UART2_IT_TypeDef UART2_IT, FunctionalState NewState);
UART2_HalfDuplexCmd(FunctionalState NewState);
UART2_IrDAConfig(UART2_IrDAMode_TypeDef UART2_IrDAMode);
UART2_IrDACmd(FunctionalState NewState);
UART2_LINBreakDetectionConfig(UART2_LINBreakDetectionLength_TypeDef--
                                    UART2_LINBreakDetectionLength);
UART2_LINConfig(    UART2_LinMode_TypeDef UART2_Mode,
                    UART2_LinAutosync_TypeDef UART2_Autosync,
                    UART2_LinDivUp_TypeDef UART2_DivUp);
UART2_LINCmd(FunctionalState NewState);
UART2_SmartCardCmd(FunctionalState NewState);
UART2_SmartCardNACKCmd(FunctionalState NewState);
UART2_WakeUpConfig(UART2_WakeUp_TypeDef UART2_WakeUp);
UART2_ReceiverWakeUpCmd(FunctionalState NewState);
UART2_ReceiveData8(void);
UART2_ReceiveData9(void);
UART2_SendData8(u8 Data);
UART2_SendData9(u16 Data);
UART2_SendBreak(void);
UART2_SetAddress(u8 UART2_Address);
UART2_SetGuardTime(u8 UART2_GuardTime);
UART2_SetPrescaler(u8 UART2_Prescaler);
UART2_GetFlagStatus(UART2_Flag_TypeDef UART2_FLAG);
UART2_ClearFlag(UART2_Flag_TypeDef UART2_FLAG);
UART2_GetITStatus(UART2_IT_TypeDef UART2_IT);
UART2_ClearITPendingBit(UART2_IT_TypeDef UART2_IT);
************************************************************************************************
UART3_DeInit(void);
UART3_Init(  u32 BaudRate,
             UART3_WordLength_TypeDef WordLength,
             UART3_StopBits_TypeDef StopBits,
             UART3_Parity_TypeDef Parity,
             UART3_Mode_TypeDef Mode);
UART3_Cmd(FunctionalState NewState);
UART3_ITConfig( UART3_IT_TypeDef UART3_IT,
                FunctionalState NewState);
```

```
UART3_LINBreakDetectionConfig(UART3_LINBreakDetectionLength_TypeDef --
                                        UART3_LINBreakDetectionLength);
UART3_LINConfig(     UART3_LinMode_TypeDef UART3_Mode,
                     UART3_LinAutosync_TypeDef UART3_Autosync,
                     UART3_LinDivUp_TypeDef UART3_DivUp);
UART3_LINCmd(FunctionalState NewState);
UART3_ReceiverWakeUpCmd(FunctionalState NewState);
UART3_WakeUpConfig( UART3_WakeUp_TypeDef UART3_WakeUp);
UART3_ReceiveData8(void);
UART3_ReceiveData9(void);
UART3_SendData8(u8 Data);
UART3_SendData9(u16 Data);
UART3_SendBreak(void);
UART3_SetAddress(u8 UART3_Address);
UART3_GetFlagStatus(UART3_Flag_TypeDef UART3_FLAG);
UART3_ClearFlag(UART3_Flag_TypeDef UART3_FLAG);
UART3_GetITStatus(UART3_IT_TypeDef UART3_IT);
UART3_ClearITPendingBit(UART3_IT_TypeDef UART3_IT);
**************************************************************************************************
WWDG_Init(u8 Counter, u8 WindowValue);
WWDG_SetCounter(u8 Counter);
WWDG_GetCounter(void);
WWDG_SWReset(void);
WWDG_SetWindowValue(u8 WindowValue);
**************************************************************************************************
```