UM2001
User manual

Standard peripheral library for STLUX™ and STNRG digital controllers

Introduction

This user manual provides complete information for software developers on the STLUX / STNRG peripheral library. This library provides a set of more than 100 application programming interfaces (API) useful to get familiar developing applications for the STLUX / STNRG digital controller and its peripherals.

The STLUX family of controllers is a part of the STMicroelectronics® digital devices tailored for lighting applications. The STLUX controllers have been successfully integrated in a wide range of architectures and applications, starting from simple buck converters for driving multiple LED strings, boost for power factor corrections, half-bridge resonant converters for high power dimmable LED strings and up to full-bridge controllers for HID lamp ballasts. The STLUX natively supports the DALI via the internal DALI communication module (DCM). The DALI is a serial communication standard used in the lighting industry.

The STNRG devices are a part of the STNRG family of STMicroelectronics digital devices designed for advanced power conversion applications. The STNRG improves the design of the STLUX family to support industrial power conversion applications such as the PFC + LLC, interleaved LC DC-DC, interleaved PFC for smart power supplies as well as the full-bridge for pilot line drivers for electric vehicles.

The heart of the STLUX (and consequently the STNRG where not differently specified) is the SMED (“State Machine, Event Driven”) technology which allows the device to operate several independently configurable PWM clocks with an up to 1.3 ns resolution. An SMED is a powerful autonomous state machine which is programmed to react to both external and internal events and may evolve without any software intervention. The SMED even reaction time can be as low as 10 ns, giving the STLUX the ability of operating in time critical applications.

The SMED devices are configured and programmed via the STLUX internal low power microcontroller (STM8). This manual describes the tools provided in this kit.

Contents

1 Reference documents

- For information on the STLUX / STNRG controller and product, please refer to the STLUX reference manual (RM0380).
- For hardware information on the STLUX controller and product specific SMED configuration, please refer to the STLUX product datasheets for the STLUX385A, STLUX383A, STLUX325A, and STLUX285A.
- For hardware information of the STNRG controller and product specific SMED configuration, please refer to the STNRG product datasheets for the STNRG388A, STNRG328A, and STNRG288A.
- For information about the debug and SWIM (single-wire interface module) refer to the STM8 SWIM communication protocol and debug module user manual (UM0470).
- For information on the STM8 core and assembler instruction please refer to the STM8 CPU programming manual (PM0044).

2 Acronyms

In *Table 1* is a list of acronyms used in this document:

Table 1. List of acronyms

Acronym	Description
ACU	Analog comparator unit
ADC	Analog-to-digital converter
ATM	Auxiliary timer
AWU	Auto-wakeup unit
BL	Bootloader - used to load the user program without the emulator
CCO	Configurable clock output
CKC	Clock control unit
CKM	Clock master
CPU	Central processing unit
CSS	Clock security system
DAC	Digital-to-analog converter
DALI	Digital addressable lighting interface
ECC	Error Correction Code
FSM	Finite state machine
FW	Firmware loaded and running on the CPU
GPIO	General purpose input output
HSE	High-speed external crystal - ceramic resonator
HSI	High-speed internal RC oscillator
I^2C	Inter-integrated circuit interface
IAP	In-application programming
ICP	In-circuit programming
ITC	Interrupt controller
IWDG	Independent watchdog
LSI	Low-speed internal RC oscillator
MCU	Microprocessor central unit
MSC	Miscellaneous
PM	Power management
RFU	Reserved for future use
ROP	Read-out protection
RST	Reset control unit
RTC	Real-time clock

Table 1. List of acronyms (continued)

Acronym	Description
SMED	State machine, event driven
STMR	System timer
SW	Software, is the firmware loaded and running on the CPU (synonymous of FW)
SWI	Clock switch interrupt
SWIM	Single-wire interface module
UART	Universal asynchronous receiver/transmitter
WWDG	Window watchdog

3 STLUX library

3.1 Introduction

The STLUX library is a collection of APIs aiming to simplify the usage of the STLUX SMEDs and peripherals to application developers. Each collection of APIs is dedicated to a specific device or functionality and named so “stlux_xxx”, where “xxx” stands for the name of the peripheral. The library is developed using the C language compatible with the IAR, Raisonance and Cosmic tools and is composed of the “stlux_xxx.c” and “stlux_xxx.h” relative files to be included in your application.

3.2 Tools compatibility

As above mentioned, the STLUX peripheral library is intended as a helpful tool to ease application development providing a hardware abstraction layer to peripherals available on each device. Peripheral libraries also aim to provide an abstraction layer to the specific development tool, so that applications can be easily compiled in different environments without changes. This is achieved thanks to a set of compiler defines made available to this purpose. So while including the needed peripheral library files, be sure to declare in the compiler environment the following defines according to the chosen development environment.

Table 2. Tools compatibility

Development environment	#define
Cosmic	_COSMIC_
IAR Systems	_IAR_
Raisonance	_RAISONANCE_

3.3 Device compatibility

As highlighted also in the STLUX reference manual RM0380, the different devices in the STLUX and STNRG families offer different peripherals and features. In order to make the peripheral library totally compatible and configurable, a set of compiler defines has been made available. So while including the needed peripheral library files, be sure to declare in the compiler environment the following defines according to the chosen device.

Table 3. Device compatibility

Family	Device	#define
STLUX	STLUX385A	_STLUX385A_
	STLUX383A	_STLUX383A_
	STLUX325A	_STLUX325A_
	STLUX285A	_STLUX285A_
STNRG	STNRG388A	_STNR388A_
	STNRG328A	_STNR328A_
	STNRG288A	_STNR288A_

3.4 STLUX clock (stlux_clk)

This APIs library allows to handle the clocks configuration for the device.

3.4.1 CLK_Reset

Resets the clock internal registers to their default initial values.

Syntax

void CLK_Reset(void);

Parameters

None

Return value

None

Remarks

By default the following settings are applied:

- Internal clock control enables high-speed internal RC oscillator
- External clock control disables high-speed external oscillator
- Clock switch is set to 16 MHz HSI
- Clock switch control is disabled
- Clock divider is set to HSI / 8
- All peripheral clocks are enabled
- Clock security system is disabled
- Configurable clock output is disabled
- HSI calibration trimmer is set to zero
- SWIM clock division is set to divide SWIM clock by 2
- SMED clocks are set to 16 MHz HSI
- Clock PLL divider is set to CLKPLL / 6
- Auto-wakeup unit divider is set to 1
- Clock PLL is disabled by default
- Clock master set to 16 MHz HSI
- Configurable clock output divider is set to 1, so CLKCCO = CLK
- ADC clock is set to HIS / 3.

3.4.2 CLK_HSECmd

This function enables or disables the external clock source.

Syntax

void CLK_HSECmd(FunctionalState NewState);

Parameters

NewState can take the values ENABLE or DISABLE.

Return value

None

Remarks

None.

3.4.3 CLK_HSICmd

This function enables or disables the high frequency internal clock source.

Syntax

void CLK_HSICmd(FunctionalState NewState);

Parameters

NewState can take the values ENABLE or DISABLE.

Return value

None

Remarks

None.

3.4.4 CLK_LSICmd

This function enables or disables the low frequency internal clock source.

Syntax

void CLK_LSICmd(FunctionalState NewState);

Parameters

NewState can take the values ENABLE or DISABLE.

Return value

None

Remarks

None.

3.4.5 CLK_CCOCmd

This function enables or disables the configurable clock output.

Syntax

void CLK_CCOCmd(FunctionalState NewState);

Parameters

NewState can take the values ENABLE or DISABLE.

Return value

None

Remarks

None.

3.4.6 CLK_REGAHCmd

This function enables or disables the regulator power off in Active-Halt mode.

Syntax

void CLK_REGAHCmd(FunctionalState NewState);

Parameters

NewState can take the values ENABLE or DISABLE.

Return value

None

Remarks

When enabled, the main voltage regulator is powered off as soon as the MCU enters the Active-Halt mode, so the wakeup time is longer.

3.4.7 CLK_ClockSwitchCmd

This function enables or disables the clock to switch to the clock source pointed by the SWI register.

Syntax

void CLK_ClockSwitchCmd(FunctionalState NewState);

Parameters

NewState can take the values ENABLE or DISABLE.

Return value

None

Remarks

Please note that in case the CLK_SWR oscillator is not yet stable, CKC logic keeps active the previous clock for the stabilization time required by the new source clock.

3.4.8 CLK_FastHaltWakeUpCmd

This function enables or disables the fast Halt/Active-Halt wakeup mode.

Syntax

void CLK_ FastHaltWakeUpCmd(FunctionalState NewState);

Parameters

NewState can take the values ENABLE or DISABLE.

Return value

None

Remarks

When enabled, the HSI oscillator is automatically switched on and selected as a next master clock when resuming from Halt/Active-Halt modes.

3.4.9 CLK_ PeripheralClockConfig

This function enables or disables the specified peripheral clock.

Syntax

```
void CLK_ PeripheralClockConfig(
        CLK_Peripheral_TypeDef CLK_Peripheral,
        FunctionalState NewState
        );
```

Parameters

CLK_Peripheral is the specified peripheral which clock must be set. Valid values are:

- CLK_PERIPHERAL_I2C for the I^2C peripheral
- CLK_PERIPHERAL_GPIO0 for the general purpose IO 0
- CLK_PERIPHERAL_UART for the universal asynchronous receiver/transmitter
- CLK_PERIPHERAL_DALI for the digital addressable lighting interface
- CLK_PERIPHERAL_STMR for the system timer
- CLK_PERIPHERAL_GPIO1 for the general purpose IO 1
- CLK_PERIPHERAL_AWU for the auto-wakeup unit
- CLK_PERIPHERAL_ADC for the analog-to-digital converter
- CLK_PERIPHERAL_SMED0 for the state machine, event driven 0
- CLK_PERIPHERAL_SMED1 for the state machine, event driven 1
- CLK_PERIPHERAL_SMED2 for the state machine, event driven 2
- CLK_PERIPHERAL_SMED3 for the state machine, event driven 3
- CLK_PERIPHERAL_SMED4 for the state machine, event driven 4
- CLK_PERIPHERAL_SMED5 for the state machine, event driven 5
- CLK_PERIPHERAL_MSC for the Miscellaneous interface

NewState can take the values ENABLE or DISABLE.

Return value

None

Remarks

None.

3.4.10 CLK_ ClockSwitchConfig

This function configures the Switch from a clock domain to another.

Syntax

ErrorStatus CLK_ClockSwitchConfig(

CLK_SwitchMode_TypeDef CLK_SwitchMode,

CLK_Source_TypeDef CLK_NewClock,

FunctionalState ITState,

CLK_CurrentClockState_TypeDef CLK_CurrentClockState

);

Parameters

CLK_SwitchMode can be manual (CLK_SWITCHMODE_MANUAL) or automatic (CLK_SWITCHMODE_AUTO).

CLK_NewClock can be the high-speed internal RC oscillator (CLK_SOURCE_HSI), the low-speed internal RC oscillator (CLK_SOURCE_LSI) or the external RC oscillator (CLK_SOURCE_HSE).

ITState can be ENABLE or DISABLE. It enables or disables the clock switch interrupt

CLK_CurrentClockState can be off (CLK_CURRENTCLOCKSTATE_DISABLE) or on (CLK_CURRENTCLOCKSTATE_ENABLE).

Return value

ErrorStatus can be ERROR or SUCCESS.

Remarks

None.

3.4.11 CLK_ HSIPrescalerConfig

This function configures the HSI clock divider.

Syntax

void CLK_HSIPrescalerConfig(CLK_Prescaler_TypeDef HSIPrescaler);

Parameters

HSIPrescaler can take the following values:

- CLK_PRESCALER_HSIDIV1 for the high-speed internal clock prescaler/1
- CLK_PRESCALER_HSIDIV2 for the high-speed internal clock prescaler/2
- CLK_PRESCALER_HSIDIV4 for the high-speed internal clock prescaler/4
- CLK_PRESCALER_HSIDIV8 for the high-speed internal clock prescaler/8
- CLK_PRESCALER_CPUDIV1 for the CPU clock division factor 1
- CLK_PRESCALER_CPUDIV2 for the CPU clock division factor 2
- CLK_PRESCALER_CPUDIV4 for the CPU clock division factor 4
- CLK_PRESCALER_CPUDIV8 for the CPU clock division factor 8
- CLK_PRESCALER_CPUDIV16 for the CPU clock division factor 16
- CLK_PRESCALER_CPUDIV32 for the CPU clock division factor 32
- CLK_PRESCALER_CPUDIV64 for the CPU clock division factor 64
- CLK_PRESCALER_CPUDIV128 for the CPU clock division factor 128

Return value

None

Remarks

None.

3.4.12 CLK_ITConfig

This function enables/disables the specified CLK interrupts.

Syntax

```
void CLK_ITConfig(
        CLK_IT_TypeDef CLK_IT,
        FunctionalState NewState
    );
```

Parameters

CLK_IT can take the values

- CLK_IT_CSSD generates interrupt when a clock security system detection flag occurs
- CLK_IT_SWIF generates interrupt when a clock switch detection flag occurs

NewState can take the values ENABLE or DISABLE to enable or disable the specified clock interrupt.

Return value

None

Remarks

None.

3.4.13 CLK_SYSCLKConfig

This function configures the HSI / CPU clock divider.

Syntax

void CLK_SYSCLKConfig(CLK_Prescaler_TypeDef CLK_Prescaler);

Parameters

HSIPrescaler can take the following values:

- CLK_PRESCALER_HSIDIV1 for the high-speed internal clock prescaler/1
- CLK_PRESCALER_HSIDIV2 for the high-speed internal clock prescaler/2
- CLK_PRESCALER_HSIDIV4 for the high-speed internal clock prescaler/4
- CLK_PRESCALER_HSIDIV8 for the high-speed internal clock prescaler/8
- CLK_PRESCALER_CPUDIV1 for the CPU clock division factor 1
- CLK_PRESCALER_CPUDIV2 for the CPU clock division factor 2
- CLK_PRESCALER_CPUDIV4 for the CPU clock division factor 4
- CLK_PRESCALER_CPUDIV8 for the CPU clock division factor 8
- CLK_PRESCALER_CPUDIV16 for the CPU clock division factor 16
- CLK_PRESCALER_CPUDIV32 for the CPU clock division factor 32
- CLK_PRESCALER_CPUDIV64 for the CPU clock division factor 64
- CLK_PRESCALER_CPUDIV128 for the CPU clock division factor 128

Return value

None

Remarks

None.

3.4.14 CLK_SWIMConfig

This function configures the SWIM clock frequency on the fly.

Syntax

void CLK_SWIMConfig(CLK_SWIMDivider_TypeDef CLK_SWIMDivider);

Parameters

CLK_SWIMDivider can take the values:

- CLK_SWIMDIVIDER_2 generates a SWIM clock / 2
- CLK_SWIMDIVIDER_OTHER generates a SWIM clock / 1

Return value

None

Remarks

None.

3.4.15 CLK_ClockSecuritySystemEnable

This function enables the clock security system.

Syntax

void CLK_ClockSecuritySystemEnable(void);

Parameters

None

Return value

None

Remarks

The clock security system (CSS) monitors any possible HSE crystal clock source failure when fMASTER is provided by a HSE crystal oscillator. Whenever the HSE clock fails due to a broken or disconnected resonator or for any other fail reason, the clock controller activates a stall safe recovery mechanism by automatically switching fMASTER to the auxiliary clock source (HSI/8).

3.4.16 CLK_SYSCLKEmergencyClear

This function clears the clock switch busy flag in case of emergency.

Syntax

void CLK_SYSCLKEmergencyClear(void);

Parameters

None

Return value

None

Remarks

This function resets the clock switch busy SWBSY flag in order to reset clock switch operations (target oscillator is broken, stabilization is longing too much, etc.). If at the same time software attempts to set SWEN and clear SWBSY, SWBSY action takes precedence.

3.4.17 CLK_AdjustHSICalibrationValue

This function adjusts the HSI calibration value.

Syntax

void CLK_AdjustHSICalibrationValue(CLK_HSITrimValue_TypeDef CLK_HSICalibrationValue);

Parameters

CLK_HSICalibrationValue is the calibration trimming value. Allowed values are:

- CLK_HSITRIMVALUE_0 for trimming step 0
- CLK_HSITRIMVALUE_1 for trimming step 1
- CLK_HSITRIMVALUE_2 for trimming step 2
- CLK_HSITRIMVALUE_3 for trimming step 3
- CLK_HSITRIMVALUE_4 for trimming step -4
- CLK_HSITRIMVALUE_5 for trimming step -3
- CLK_HSITRIMVALUE_6 for trimming step -2
- CLK_HSITRIMVALUE_7 for trimming step -1

Return value

None

Remarks

Additional trimming value is added to the internal HSI factory calibration value. Each step corresponds to a mean frequency shift of 200 kHz.

3.4.18 CLK_GetClockFreq

This function returns the current clock frequency value.

Syntax

u32 CLK_GetClockFreq(void);

Parameters

None

Return value

The returned value is the current clock frequency value in Hz.

Remarks

None.

3.4.19 CLK_GetSYSCLKSource

This function returns the current clock source.

Syntax

CLK_Source_TypeDef CLK_GetSYSCLKSource(void);

Parameters

None

Return value

The returned value is the current clock source. It can be the high-speed internal RC oscillator (CLK_SOURCE_HSI), the low-speed internal RC oscillator (CLK_SOURCE_LSI) or the external RC oscillator (CLK_SOURCE_HSE).

Remarks

None.

3.4.20 CLK_GetFlagStatus

Checks whether the specified CLK flag is set or not.

Syntax

FlagStatus CLK_GetFlagStatus(CLK_Flag_TypeDef CLK_FLAG);

Parameters

CLK_FLAG is the clock status flag to be checked. It can take the values:

- CLK_FLAG_LSIRDY is the low-speed internal oscillator ready flag
- CLK_FLAG_HSIRDY is the high-speed internal oscillator ready flag
- CLK_FLAG_HSERDY is the high-speed external oscillator ready flag
- CLK_FLAG_SWIF is the clock switch interrupt flag
- CLK_FLAG_SWBSY is the switch busy flag
- CLK_FLAG_CSSD is the clock security system detection flag
- CLK_FLAG_AUX is the auxiliary oscillator connected to the master clock
- CLK_FLAG_CCOBSY is the configurable clock output busy
- CLK_FLAG_CCORDY is the configurable clock output ready

Return value

FlagStatus is the current flag status value. It can be SET or RESET.

Remarks

None.

3.4.21 CLK_GetITStatus

Checks whether the specified CLK flag is set or not.

Syntax

ITStatus CLK_GetITStatus(CLK_IT_TypeDef CLK_IT);

Parameters

CLK_IT can take the values:

- CLK_IT_CSSD generates interrupt when a clock security system detection flag occurs
- CLK_IT_SWIF generates interrupt when a clock switch detection flag occurs

Return value

ITStatus is the current interrupt status value. It can be SET or RESET.

Remarks

None.

3.4.22 CLK_ClearITPendingBit

Clears the CLK's interrupt pending bits.

Syntax

void CLK_ClearITPendingBit(CLK_IT_TypeDef CLK_IT);

Parameters

CLK_IT can take the values:

- CLK_IT_CSSD generates interrupt when a clock security system detection flag occurs
- CLK_IT_SWIF generates interrupt when a clock switch detection flag occurs

Return value

None

Remarks

None.

3.4.23 CLK_PLLConfig

This function sets the clock source for PLL.

Syntax

```
void CLK_PLLConfig(
        CLK_PLL_Source_TypeDef CLK_PLL_Source,
        CLK_PLL_DIVPRES_TypeDef CLK_PLL_DIVPRES
    );
```

Parameters

CLK_PLL_Source specifies the clock source for the PLL. It can take the values:

- CLK_PLL_SOURCE_HSI for the high-speed internal RC oscillator
- CLK_PLL_SOURCE_HSE for the high-speed external RC oscillator

CLK_PLL_DIVPRES is the division factor for the PLL selected clock. It can take the values:

- CLK_PLL_DIVIDER_4 for the CLKPLL / 4
- CLK_PLL_DIVIDER_5 for the CLKPLL / 5
- CLK_PLL_DIVIDER_6 for the CLKPLL / 6
- CLK_PLL_DIVIDER_7 for the CLKPLL / 7
- CLK_PLL_PRESCALER_1 for the division factor n = 0
- CLK_PLL_PRESCALER_2 for the division factor n = 1
- CLK_PLL_PRESCALER_4 for the division factor n = 2
- CLK_PLL_PRESCALER_8 for the division factor n = 3

Return value

None

Remarks

Please note that CLKPLL_PRES_DIV = CLKPLL_DIV / 2n where *n* is the prescaler division factor.

3.4.24 CLK_PLLCmd

This function enables / disables the PLL.

Syntax

```
void CLK_PLLCmd(FunctionalState NewState);
```

Parameters

NewState specifies whether to ENABLE or DISABLE the PLL.

Return value

None

Remarks

None.

3.4.25 CLK_ CCOConfig

This function sets the clock source for the CCO clock.

Syntax

void CLK_CCOConfig(CLK_Output_TypeDef CLK_CCO,u8 CLK_CCODIVR);

Parameters

CLK_CCO specifies the clock source for the CCO. It can be one of the following values:

- CLK_OUTPUT_HSI for the high-speed internal RC oscillator
- CLK_OUTPUT_LSI for the low-speed internal RC oscillator
- CLK_OUTPUT_HSE for the high-speed external RC oscillator
- CLK_OUTPUT_PLL for the PLL clock
- CLK_OUTPUT_CPU for the CPU clock
- CLK_OUTPUT_CKM for the master clock
- CLK_OUTPUT_SMED0_CK for the SMED0 clock
- CLK_OUTPUT_SMED1_CK for the SMED1 clock
- CLK_OUTPUT_SMED2_CK for the SMED2 clock
- CLK_OUTPUT_SMED3_CK for the SMED3 clock
- CLK_OUTPUT_SMED4_CK for the SMED4 clock
- CLK_OUTPUT_SMED5_CK for the SMED5 clock
- CLK_OUTPUT_ADC_CK for the ADC clock
- CLK_OUTPUT_AWU_CK for the auto-wakeup unit clock
- CLK_OUTPUT_PRESCALED_PLL_CK for the prescaled PLL clock

CLK_CCODIVR specifies the division factor *n* for the CCO clock.

Return value

None

Remarks

Please note that the final CCO clock will be set according to the equation:
CLKCCO = SELCLK / (n + 1), where *SELCLK* is the selected clock.

3.4.26 CLK_ADCConfig

This function sets the clock source for the ADC.

Syntax

void CLK_ADCConfig(CLK_ADC_Source_TypeDef CLK_ADC_Source,u8 CLK_ADC_DIV);

Parameters

CLK_ADC_Source specifies the clock source for the ADC. It can take the following values:

- CLK_ADC_SOURCE_HSI for the high-speed internal RC oscillator
- CLK_ADC_SOURCE_PLL for the PLL clock
- CLK_ADC_SOURCE_LSI for the low-speed internal RC oscillator
- CLK_ADC_SOURCE_HSE for the high-speed external RC oscillator

CLK_ADC_DIV specifies the division factor *n* for the ADC clock.

Return value

None

Remarks

Please note that the final CCO clock will be set according to the equation:
CLKADC = SELCLK / (n + 1), where *SELCLK* is the selected clock.

3.4.27 CLK_AWUConfig

This function sets the clock for the auto-wakeup unit.

Syntax

void CLK_AWUConfig(CLK_AWU_DIVIDER_TypeDef CLK_AWU_DIVIDER);

Parameters

CLK_AWU_DIVIDER specifies the clock division factor for the AWU clock. It can take the following values:

- CLK_AWU_DIVIDER_1 divides the timebase clock (the ADC clock) frequency by 1
- CLK_AWU_DIVIDER_2 divides the timebase clock (the ADC clock) frequency by 2
- CLK_AWU_DIVIDER_4 divides the timebase clock (the ADC clock) frequency by 4
- CLK_AWU_DIVIDER_8 divides the timebase clock (the ADC clock) frequency by 8
- CLK_AWU_DIVIDER_16 divides the timebase clock (the ADC clock) frequency by 16
- CLK_AWU_DIVIDER_32 divides the timebase clock (the ADC clock) frequency by 32
- CLK_AWU_DIVIDER_64 divides the timebase clock (the ADC clock) frequency by 64
- CLK_AWU_DIVIDER_128 divides the timebase clock (the ADC clock) frequency by 128
- CLK_AWU_DIVIDER_256 divides the timebase clock (the ADC clock) frequency by 256

Return value

None

Remarks

None.

3.4.28 CLK_SMEDConfig

This function sets the clock for the selected SMED unit.

Syntax

```
void CLK_SMEDConfig(
        CLK_SMD_TypeDef CLK_SMD,
        CLK_SMED_Source_TypeDef Source,
        CLK_SMED_PRESCALER_TypeDef Prescaler
    );
```

Parameters

CLK_SMD specifies the clock SMEDx to be set. It can take the following values:

- CLK_SMED0 for the SMED0 clock
- CLK_SMED1 for the SMED1 clock
- CLK_SMED2 for the SMED2 clock
- CLK_SMED3 for the SMED3 clock
- CLK_SMED4 for the SMED4 clock
- CLK_SMED5 for the SMED5 clock

Source specifies the selected clock SMEDx source to be applied. It can take the following values:

- CLK_SMED_SOURCE_HSI for the high-speed internal RC oscillator
- CLK_SMED_SOURCE_PLL for the PLL clock
- CLK_SMED_SOURCE_LSI for the low-speed internal RC oscillator
- CLK_SMED_SOURCE_HSE for the high-speed external RC oscillator

Prescaler specifies the clock division factor for the AWU clock. It can take the following values:

- CLK_SMED_PRESCALER_1 divides the SMED clock frequency by 1
- CLK_ SMED _ PRESCALER _2 divides the SMED clock frequency by 2
- CLK_ SMED _ PRESCALER _4 divides the SMED clock frequency by 4
- CLK_ SMED _ PRESCALER _8 divides the SMED clock frequency by 8
- CLK_ SMED _ PRESCALER _16 divides the SMED clock frequency by 16
- CLK_ SMED_ PRESCALER _32 divides the SMED clock frequency by 32
- CLK_ SMED _ PRESCALER _64 divides the SMED clock frequency by 64
- CLK_ SMED _ PRESCALER _128 divides the SMED clock frequency by 128

Return value

None

Remarks

None.

3.5 STLUX SMEDs (stlux_smed)

3.5.1 SMED_Start

This function makes the selected SMEDx start running.

Syntax

void SMED_Start(SMED_TypeDef* SMEDx);

Parameters

SMEDx specifies the selected SMED. The type defines a structure including all the parameters characterizing a SMED like control registers, states, timers, events and interrupts. By default the STLUX peripheral library defines the following values:

- SMED0
- SMED1
- SMED2
- SMED3
- SMED4
- SMED5

Return value

None

Remarks

None.

3.5.2 SMED_Stop

This function makes the selected SMEDx stop running.

Syntax

void SMED_Stop(SMED_TypeDef* SMEDx);

Parameters

SMEDx specifies the selected SMED. The type defines a structure including all the parameters characterizing a SMED like control registers, states, timers, events and interrupts. By default the STLUX peripheral library defines the following values:

- SMED0
- SMED1
- SMED2
- SMED3
- SMED4
- SMED5

Return value

None

Remarks

None.

3.5.3 SMED_SetTime

This function sets the time value to the status TimeRegister for the selected SMEDx.

Syntax

```
u8 SMED_SetTime(
        SMED_TypeDef* SMEDx,
        SMED_TIMES_Typedef TimeRegister,
        u16 value
        );
```

Parameters

SMEDx specifies the selected SMED. The type defines a structure including all the parameters characterizing a SMED like control registers, states, timers, events and interrupts. By default the STLUX peripheral library defines the following values:

- SMED0
- SMED1
- SMED2
- SMED3
- SMED4
- SMED5

TimeRegister specifies the internal status of the SMED for which the time must be set. Allowed values are:

- SMED_T0 for the SMEDx Time 0
- SMED_T1 for the SMEDx Time 1
- SMED_T2 for the SMEDx Time 2
- SMED_T3 for the SMEDx Time 3

Value specifies the time value to be set in terms of clock tics.

Return value

The returned value represents the result of the time setting operation. The function returns zero in case of an unsuccessful operation due to pending time validation, one otherwise.

Remarks

None.

3.5.4 SMED_ValidateTimeValues

This function validates the configuration previously set with SMED_SetTime() for the desired SMEDx timers. This enables the shadow registers update for the selected SMED control time register.

Syntax

```
u8 SMED_ValidateTimeValues(
        SMED_TypeDef* SMEDx,
        SMED_VALIDATE_TypeDef TimeVal
        );
```

Parameters

SMEDx specifies the selected SMED. The type defines a structure including all the parameters characterizing a SMED like control registers, states, timers, events and interrupts. By default the STLUX peripheral library defines the following values:

- SMED0
- SMED1
- SMED2
- SMED3
- SMED4
- SMED5

TimeVal specifies one or more internal timers of the SMED to be validated. Allowed values are:

- SMED_T0_VAL for the timer 0 value
- SMED_T1_VAL for the timer 1 value
- SMED_T2_VAL for the timer 2 value
- SMED_T3_VAL for the timer 3 value
- SMED_DITHER_VAL for the dithering value. For more details on dithering, please refer to the STLUX reference manual RM0380.

Return value

The returned value represents the result of the time setting operation. The function returns zero in case of an unsuccessful operation due to pending time validation, one otherwise.

Remarks

Please note that one or more TimeVal can be passed to the function combined in the logic OR. For example validation of the timers 0, 1 and 2 for the SMED0 can be performed by the instruction: SMED_ValidateTimeValues(SMED0, SMED_T0_VAL| SMED_T1_VAL| SMED_T2_VAL);.

3.5.5 SMED_TrigSWEvent

This macro enables the SW event triggers flags. These SW flags are usually interconnected to the connection box matrix and may be selected as input signals for each SMEDs FSM.

Syntax

SMED_TrigSWEvent(x);

Parameters

X specifies the selected SW event trigger SWx to be enabled. It can take the following values:

- 0 for the SW0 software event on the SMED0
- 1 for the SW0 software event on the SMED1
- 2 for the SW0 software event on the SMED2
- 3 for the SW0 software event on the SMED3
- 4 for the SW0 software event on the SMED4
- 5 for the SW0 software event on the SMED5

Return value

None

Remarks

Please note that SW triggers give the application program the capability to modify the SMED finite state machine evolution like any other HW events interconnected to the input signal lines.

3.5.6 SMED_Init

This function is a weak function which body is meant to be redefined by the user according to its application implementation. This can also be automatically generated by the STLUX SMED configurator tool.

Syntax

void SMED_Init(void);

Parameters

None

Return value

None

Remarks

None.

3.6 STLUX analog comparator unit (stlux_acu)

3.6.1 ACU_Reset

This function sets the analog comparators unit internal registers to their default initialization values.

Syntax

void ACU_Reset(void);

Parameters

None

Return value

None

Remarks

By default all DACs are set to zero and all comparators are disabled.

3.6.2 ACU_Init

This function enables the analog comparators unit.

Syntax

void ACU_Init(void);

Parameters

None

Return value

None

Remarks

When the selected target device is the STLUX385A, STLUX383A or STLUX325A, this function must be invoked before using the ACU peripheral. For all other STLUX and STNRG devices, enabling of the ACU has been automatized and this function is deprecated.

3.6.3 ACU_Read

This function reads the output value for a selected analog comparator.

Syntax

Bool ACU_Read(ACU_Selection_TypeDef ACUx);

Parameters

ACUx specifies the comparator peripheral to be read. It can take the values:

- CMP_0 for the analog comparator 0
- CMP_1 for the analog comparator 1
- CMP_2 for the analog comparator 2
- CMP_3 for the analog comparator 3

Return value

The returned value is the output of the selected analog comparator. It can be 0 when the set condition is false, 1 when it is true.

Remarks

None.

3.6.4 ACU_Enable

This function enables the comparator CPx. When available on the device, it also specifies whether the internal or external reference should be used.

Syntax

```
void ACU_Enable(
        ACU_Selection_TypeDef ACUx,
        ACU_CP_SEL_Typedef CP_SEL
        );
```

Parameters

ACUx specifies the comparator peripheral to be enabled. It can take the values:

- CMP_0 for the analog comparator 0
- CMP_1 for the analog comparator 1
- CMP_2 for the analog comparator 2
- CMP_3 for the analog comparator 3

CP_SEL specifies whether the internal or external reference should be used if available.

On the STLUX385A, STLUX383A and STLUX325A it can take the values:

- ACU_CP3_SEL_EXT for the comparator external reference
- ACU_CP3_SEL_INT for the DAC internal reference

For all the other devices (STNRG and STLUX285A) it can take the values:

- ACU_CP0_SEL_EXT for the comparator external reference
- ACU_CP1_SEL_EXT for the comparator external reference
- ACU_CP2_SEL_EXT for the comparator external reference
- ACU_CP3_SEL_EXT for the comparator external reference
- ACU_CPx_SEL_INT for the DAC internal reference

Return value

None

Remarks

None.

3.6.5 ACU_Disable

This function disables the comparator CPx. When available on the device, it also specifies whether the internal or external reference should be used.

Syntax

void ACU_ Disable (ACU_Selection_TypeDef ACUx);

Parameters

ACUx specifies the comparator peripheral to be disabled. It can take the values:

- CMP_0 for the analog comparator 0
- CMP_1 for the analog comparator 1
- CMP_2 for the analog comparator 2
- CMP_3 for the analog comparator 3

Return value

None

Remarks

None.

3.6.6 ACU_SetCompareLevel

This function assigns a specified voltage level reference to the DAC of a specified analog comparator.

Syntax

```
void ACU_SetCompareLevel(
        ACU_Selection_TypeDef ACUx,
        ACU_DACIN_VALUES_TypeDef DACIN
        );
```

Parameters

ACUx specifies the comparator peripheral to be disabled. It can take the values:

- CMP_0 for the analog comparator 0
- CMP_1 for the analog comparator 1
- CMP_2 for the timebase clock 2
- CMP_3 for the timebase clock 3

DACIN specifies the voltage level to be set in the selected DAC. It can take the following values:

- DACIN_0mV for the reference voltage level set to 0 mV
- DACIN_82mV for the reference voltage level set to 82 mV
- DACIN_164mV for the reference voltage level set to 164 mV
- DACIN_246mV for the reference voltage level set to 246 mV
- DACIN_328mV for the reference voltage level set to 328 mV
- DACIN_410mV for the reference voltage level set to 410 mV
- DACIN_492mV for the reference voltage level set to 492 mV
- DACIN_574mV for the reference voltage level set to 574 mV
- DACIN_656mV for the reference voltage level set to 656 mV
- DACIN_738mV for the reference voltage level set to 738 mV
- DACIN_820mV for the reference voltage level set to 820 mV
- DACIN_902mV for the reference voltage level set to 902 mV
- DACIN_984mV for the reference voltage level set to 984 mV
- DACIN_1066mV for the reference voltage level set to 1066 mV
- DACIN_1148mV for the reference voltage level set to 1148 mV
- DACIN_1230mV for the reference voltage level set to 1230 mV

Return value

None

Remarks

None.

3.6.7 ACU_SetHysteresisLevel

This function assigns a specified hysteresis level to the DAC of a specified analog comparator.

Syntax

```
void ACU_SetHysteresisLevel(
        ACU_Selection_TypeDef ACUx,
        ACU_HYS_VALUES_TypeDef HYSIN
        );
```

Parameters

ACUx specifies the comparator peripheral which DAC hysteresis must be applied. It can take the values:

- CMP_0 for the timebase clock 0
- CMP_1 for the timebase clock 1
- CMP_2 for the timebase clock 2
- CMP_3 for the timebase clock 3

HYSIN specifies the voltage level to be set in the selected DAC. It can take the following values:

- HYSTDN_V_CODE0 for the falling edge hysteresis level 0
- HYSTDN_V_CODE3 for the falling edge hysteresis level 3
- HYSTDN_V_CODE4 for the falling edge hysteresis level 4
- HYSTDN_V_CODE5 for the falling edge hysteresis level 5
- HYSTDN_V_CODE6 for the falling edge hysteresis level 6
- HYSTDN_V_CODE7 for the falling edge hysteresis level 7
- HYSTUP_V_CODE0 for the rising edge hysteresis level 0
- HYSTUP_V_CODE3 for the rising edge hysteresis level 3
- HYSTUP_V_CODE4 for the rising edge hysteresis level 4
- HYSTUP_V_CODE5 for the rising edge hysteresis level 5
- HYSTUP_V_CODE6 for the rising edge hysteresis level 6
- HYSTUP_V_CODE7 for the rising edge hysteresis level 7

Return value

None

Remarks

Please note that once properly set, hysteresis applies to the comparator behavior independently whether it works with internal or external reference. For more information on comparators hysteresis, please refer to the reference manual RM0380.

3.7 STLUX analog-to-digital converter (stlux_adc)

3.7.1 ADC_Reset

This function sets the ADC internal registers to their default initialization values. At reset the ADC is powered down and associated end of conversion mode interrupt and end of sequence mode interrupt are disabled. Also the ADC status register is cleared. The ADC delay count is set to zero.

Syntax

void ADC_Reset(void);

Parameters

None

Return value

None

Remarks

None.

3.7.2 ADC_Init

This function initializes the ADC sequencer.

Syntax

```
void ADC_Init(
        ADC_ConvMode_TypeDef ADC_ConvMode_Init,
        ADC_DataFormat_TypeDef ADC_DataFormat_Init
        );
```

Parameters

ADC_ConvMode_Init specifies the conversion mode to be applied for the ADC. It can take the following values:

- ADC_ConvMode_SEQUENCE for the sequence conversion mode
- ADC_ConvMode_CIRCULAR for the circular conversion mode

ADC_DataFormat_Init determines whether the ADC 10-bit resolution data are left or right aligned. It can take the following values:

- ADC_DataFormat_H8L2 for data left alignment [dataH (9:2), dataL(1:0)]
- ADC_DataFormat_H2L8 for data right alignment [dataH (9:8), dataL(7:0)]

Return value

None

Remarks

None.

3.7.3 ADC_Interrupt

This function configures the ADC interrupts.

Syntax

```
void ADC_Interrupt(
        ADC_IntEndConvMode_TypeDef ADC_IntEndConvMode_Interrupt,
        ADC_IntEndSeqMode_TypeDef ADC_IntEndSeqMode_interrupt,
        ADC_IntSeqFull_TypeDef ADC_IntSeqFull_Interrupt
        );
```

Parameters

ADC_IntEndConvMode_Interrupt specifies whether the end of conversion mode interrupt is enabled or disabled. It can take the following values:

- ADC_IntEndConvMode_DIS to disable the interrupt
- ADC_IntEndConvMode_EN to enable the interrupt

ADC_IntEndSeqMode_interrupt specifies whether the end of sequence mode interrupt is enabled or disabled. It can take the following values:

- ADC_IntEndSeqMode_DIS to disable the interrupt
- ADC_IntEndSeqMode_EN to enable the interrupt

ADC_IntSeqFull_Interrupt specifies whether the sequencer buffer full interrupt is enabled or disabled. It can take the following values:

- ADC_IntSeqFull_ DIS to disable the interrupt
- ADC_IntSeqFull_EN to enable the interrupt

Return value

None

Remarks

None.

3.7.4 ADC_Sequencer

This function initializes the ADC sequencer.

Syntax

```
void ADC_Sequencer(
        ADC_Channel_TypeDef ADC_Channel_Sequencer,
        ADC_Gain_TypeDef ADC_Gain_Sequencer
        );
```

Parameters

ADC_Channel_Sequencer specifies the channel which gain must be set. It can take the following values:

- ADC_CHANNEL_0 for the analog channel 0
- ADC_CHANNEL_1 for the analog channel 1
- ADC_CHANNEL_2 for the analog channel 2
- ADC_CHANNEL_3 for the analog channel 3
- ADC_CHANNEL_4 for the analog channel 4
- ADC_CHANNEL_5 for the analog channel 5
- ADC_CHANNEL_6 for the analog channel 6
- ADC_CHANNEL_7 for the analog channel 7

ADC_Gain_Sequencer specifies the analog gain selection to be applied for the data conversion. It can take the following values:

- ADC_GAIN_16 for setting gain equal to 1.6
- ADC_GAIN_64 for setting gain equal to 6.4

Return value

None

Remarks

Please note that the gain 6.4 is available only for the STLUX385A and STNRG388A. For more details on the ADC gain please refer to the reference manual RM0380.

3.7.5 ADC_Start

This function starts the ADC conversion.

Syntax

```
void ADC_Start (void);
```

Parameters

None

Return value

None

Remarks

Before starting, the ADC always checks if it is not in powerdown and if the sequencer buffer is not empty. If these two conditions are not met, the command does not take effect.

3.7.6 ADC_Stop

This function stops the ADC conversion.

Syntax

void ADC_Stop(void);

Parameters

None

Return value

None

Remarks

The ADC sequencer is reset and returns to idle state. If the enhanced abort functionality is enabled, the request is deferred while a conversion is in progress. Please note that the ADC stop activity in the two clock domains requires 20 ADC clock cycles. For more details refer to the reference manual RM0380.

3.7.7 ADC_PowerUp

This function performs the correct power-on sequence for the ADC (after a power-down to save power).

Syntax

void ADC_PowerUp(void);

Parameters

None

Return value

None

Remarks

Particular caution must be taken when the ADC peripherals are in low power mode. After awaking from the Halt, the proper power-on sequence must be performed, which includes waiting for 30 µs before starting ADC conversions.

3.7.8 ADC_PowerDown

This function performs the correct power-down sequence for the ADC (to save power).

Syntax

void ADC_PowerDown(void);

Parameters

None

Return value

None

Remarks

Particular caution must be taken when the ADC peripherals are in low power mode. The proper power-down sequence must be performed to ensure the ADC is correctly stopped before forcing the system in Halt mode.

3.7.9 ADC_Abort

This function performs the correct abort sequence for the ADC.

Syntax

void ADC_Abort(void);

Parameters

None

Return value

None

Remarks

The ADC may be forced from SW to abort and stop the current conversion cycles before a new reconfiguration or to enter in the power-down state by following the abort procedure.

3.7.10 ADC_Delay

This function introduces a delay between the start of the conversion command and the effective start of the conversion.

Syntax

void ADC_Delay(u8 ADC_Delay_Value);

Parameters

ADC_Delay_Value specifies the number of fMASTER clock cycles to be used as a delay.

Return value

None

Remarks

This delay is applied only to the first conversion following the SOC command; next conversions in a sequence are not delayed. For more details about the ADC conversion delay please refer to the STLUX reference manual RM0380.

3.7.11 ADC_Measure

Given a channel number and a gain factor, the function returns a 16-bit value captured by the ADC.

Syntax

```
u16 ADC_Measure(
        ADC_Channel_TypeDef ADC_Channel_Measure,
        ADC_Gain_TypeDef ADC_Gain_Measure
        );
```

Parameters

ADC_Channel_ Measure specifies the channel to be used for the data conversion. It can take the following values:

- ADC_CHANNEL_0 for the analog channel 0
- ADC_CHANNEL_1 for the analog channel 1
- ADC_CHANNEL_2 for the analog channel 2
- ADC_CHANNEL_3 for the analog channel 3
- ADC_CHANNEL_4 for the analog channel 4
- ADC_CHANNEL_5 for the analog channel 5
- ADC_CHANNEL_6 for the analog channel 6
- ADC_CHANNEL_7 for the analog channel 7

ADC_Gain_ Measure specifies the analog gain selection to be applied for the data conversion. It can take the following values:

- ADC_GAIN_16 for setting gain equal to 1.6
- ADC_GAIN_64 for setting gain equal to 6.4

Return value

The returned value is the 16-bit long data from the conversion on the selected channel. Data alignment is according to the current configuration.

Remarks

Please note that the gain 6.4 is available only for the STLUX385A and STNRG388A. For more details on the ADC gain please refer to the reference manual RM0380.

3.7.12 GetStatus

This function returns the current ADC status.

Syntax

u8 ADC_GetStatus(void);

Parameters

None

Return value

The returned data is the current value of the ADC status register. Single bits are status indicators and in particular:

- Bit 0 indicated the EOC (end of conversion) mode
- Bit 1 indicates the EOS (end of sequence) mode
- Bit 2 indicates the SEQ_FULL (SEQuence buffer FULL) mode.

Remarks

None.

3.7.13 SetStatus

This function returns the current ADC status.

Syntax

```
void ADC_SetStatus(
        ADC_StatusEOS_TypeDef EOS,
        ADC_StatusEOC_TypeDef EOC
        );
```

Parameters

EOS specifies the end of sequence status to be set. It can take the following values:

- ADC_StatusEOS_SET defines the EOS has been set
- ADC_StatusEOS_NUL defines the EOS has been cleared

EOC specifies the end of conversion status to be set. It can take the following values:

- ADC_StatusEOC_SET defines the EOC has been set
- ADC_StatusEOC_NUL defines the EOC has been cleared

Return value

None

Remarks

None.

3.8 STLUX system timer (stlux_stmr)

3.8.1 STMR_Reset

Sets the STMR internal registers to their default initialization values.

Syntax

void STMR_Reset(void);

Parameters

None

Return value

None

Remarks

By default the system timer is disabled and also the associated interrupts and event generation are disabled. The autoreload mode is disabled as well.

3.8.2 STMR_TimeBaseInit

This function initializes the STMR timebase unit according to the specified parameters.

Syntax

```
void STMR_TimeBaseInit(
        STMR_Prescaler_TypeDef STMR_Prescaler,
        u16 STMR_Period
        );
```

Parameters

STMR_Prescaler defines the prescaler division factor to be applied to the master clock. It can take the following values:

- STMR_PRESCALER_1 for the prescaler division factor = 1 (No effect)
- STMR_PRESCALER_2 for the prescaler division factor = 2
- STMR_PRESCALER_4 for the prescaler division factor = 4
- STMR_PRESCALER_8 for the prescaler division factor = 8
- STMR_PRESCALER_16 for the prescaler division factor = 16
- STMR_PRESCALER_32 for the prescaler division factor = 32
- STMR_PRESCALER_64 for the prescaler division factor = 64
- STMR_PRESCALER_128 for the prescaler division factor = 128

STMR_Period defines the time period to be set.

Return value

None

Remarks

None.

3.8.3 STMR_Cmd

This function enables or disables the STMR peripheral.

Syntax

void STMR_Cmd(FunctionalState NewState);

Parameters

NewState specifies whether to ENABLE or DISABLE the system timer STMR.

Return value

None

Remarks

None.

3.8.4 STMR_ITConfig

This function enables or disables the STMR peripheral interrupts.

Syntax

```
void STMR_ITConfig(
        STMR_IT_TypeDef STMR_IT,
        FunctionalState NewState
        );
```

Parameters

STMR_IT specifies the interrupt to be enabled. It can take the value STMR_IT_UPDATE which specifies the interrupt update to be enabled / disabled.

NewState specifies whether to ENABLE or DISABLE the system timer STMR.

Return value

None

Remarks

None.

3.8.5 STMR_UpdateDisableConfig

This function enables or disables the generation of the update event.

Syntax

void STMR_UpdateDisableConfig(FunctionalState Newstate);

Parameters

NewState specifies whether to ENABLE or DISABLE the generation of the update event for the System Timer STMR.

Return value

None

Remarks

None.

3.8.6 STMR_UpdateRequestConfig

This function enables or disables the generation of the interrupt update request by SW.

Syntax

void STMR_UpdateRequestConfig(STMR_UpdateSource_TypeDef STMR_UpdateSource);

Parameters

STMR_UpdateSource specifies whether the interrupt update request must be generated by the regular counter overflow only or also by SW. It can take the following values:

- STMR_UPDATESOURCE_GLOBAL for requests to be sent as soon as registers are updated (counter overflow or set of UG bit)
- STMR_UPDATESOURCE_REGULAR for interrupt requests to be sent only when the counter reaches the overflow

Return value

None

Remarks

None.

3.8.7 STMR_SelectOnePulseMode

This function enables or disables the one pulse mode. This mode makes the counter stop when the next update event is triggered.

Syntax

void STMR_SelectOnePulseMode(STMR_OPMode_TypeDef STMR_OPMode);

Parameters

STMR_OPMode specifies whether the one pulse mode is enabled or disabled. It can take the following values:

- STMR_OPMODE_SINGLE for the single pulse mode enable: the counter stops the count when the next update event is triggered
- STMR_OPMODE_REPETITIVE for the single pulse mode disable: the counter doesn't stop when an update event is triggered

Return value

None

Remarks

None.

3.8.8 STMR_PrescalerConfig

This function configures the STMR system timer prescaler.

Syntax

```
void STMR_PrescalerConfig(
        STMR_Prescaler_TypeDef Prescaler,
        STMR_PSCReloadMode_TypeDef STMR_PSCReloadMode
        );
```

Parameters

Prescaler defines the prescaler division factor to be applied to the master clock. It can take the following values:

- STMR_PRESCALER_1 for the prescaler division factor = 1 (no effect)
- STMR_PRESCALER_2 for the prescaler division factor = 2
- STMR_PRESCALER_4 for the prescaler division factor = 4
- STMR_PRESCALER_8 for the prescaler division factor = 8
- STMR_PRESCALER_16 for the prescaler division factor = 16
- STMR_PRESCALER_32 for the prescaler division factor = 32
- STMR_PRESCALER_64 for the prescaler division factor = 64
- STMR_PRESCALER_128 for the prescaler division factor = 128

STMR_PSCReloadMode specifies the STMR prescaler reload mode. It can take the following values:

- STMR_PSCRELOADMODE_UPDATE no action is taken and the prescaler is updated at the next update event
- STMR_PSCRELOADMODE_IMMEDIATE for the prescaler to be loaded immediately. It also reinitializes the counter.

Return value

Please note that the STMR_PSCReloadMode is set by software and it's automatically cleared by hardware.

Remarks

None.

3.8.9 STMR_ARRPreloadConfig

This function enables or disables the STMR system timer peripheral autoreload preload mode.

Syntax

void STMR_ARRPreloadConfig(FunctionalState NewState);

Parameters

NewState specifies whether to ENABLE or DISABLE the autoreload preload mode for the system timer STMR.

Return value

None

Remarks

None.

3.8.10 STMR_GenerateEvent

This function configures the STMR prescaler reload event to be generated by SW.

Syntax

void STMR_GenerateEvent(STMR_EventSource_TypeDef STMR_EventSource);

Parameters

STMR_EventSource specifies the STMR prescaler reload mode. It can take the following values:

- STMR_EVENTSOURCE_UPDATE for the update event to be immediately set.

Return value

None

Remarks

Please note that the event is set by software and it's automatically cleared by hardware.

3.8.11 STMR_SetCounter

This function initializes the counter to be used by STMR.

Syntax

void STMR_SetCounter(u16 Counter);

Parameters

Counter specifies the value to be set as the STMR counter.

Return value

None

Remarks

None.

3.8.12 STMR_SetAutoreload

This function initializes the autoreload value to be used by STMR.

Syntax

void STMR_SetAutoreload(u16 Autoreload);

Parameters

Autoreload specifies the value to be set as the STMR autoreload value.

Return value

None

Remarks

None.

3.8.13 STMR_GetCounter

This function returns the current STMR counter value.

Syntax

u16 STMR_GetCounter(void);

Parameters

None

Return value

The returned value is the current system timer counter value.

Remarks

None.

3.8.14 STMR_GetPrescaler

This function returns the current STMR prescaler value.

Syntax

STMR_Prescaler_TypeDef STMR_GetPrescaler(void);

Parameters

None

Return value

The returned value is the current system timer prescaler value. It can return the following values:

- STMR_PRESCALER_1 for the prescaler division factor = 1 (no effect)
- STMR_PRESCALER_2 for the prescaler division factor = 2
- STMR_PRESCALER_4 for the prescaler division factor = 4
- STMR_PRESCALER_8 for the prescaler division factor = 8
- STMR_PRESCALER_16 for the prescaler division factor = 16
- STMR_PRESCALER_32 for the prescaler division factor = 32
- STMR_PRESCALER_64 for the prescaler division factor = 64
- STMR_PRESCALER_128 for the prescaler division factor = 128

Remarks

None.

3.8.15 STMR_GetFlagStatus

This function returns whether the specified STMR status flag is set or not.

Syntax

FlagStatus STMR_GetFlagStatus(STMR_FLAG_TypeDef STMR_FLAG);

Parameters

STMR_FLAG specifies the status flag to be verified. It takes the following value:

- STMR_FLAG_UPDATE for the update event flag.

Return value

The returned value is the current system timer flag status. It can be SET or RESET.

Remarks

None.

3.8.16 STMR_ClearFlag

This function resets the specified STMR status flag.

Syntax

void STMR_ClearFlag(STMR_FLAG_TypeDef STMR_FLAG);

Parameters

STMR_FLAG specifies the status flag to be reset. It takes the following value:

- STMR_FLAG_UPDATE for the update event flag.

Return value

None

Remarks

None.

3.8.17 STMR_GetITStatus

This function resets the specified STMR status flag.

Syntax

ITStatus STMR_GetITStatus(STMR_IT_TypeDef STMR_IT);

Parameters

STMR_IT specifies the STMR interrupt status to be returned. It takes the following value:

- STMR_IT_UPDATE for the update interrupt.

Return value

The returned value is the current system timer interrupt status. It can be SET or RESET.

Remarks

None.

3.8.18 STMR_ClearITPendingBit

This function resets the specified STMR interrupt status.

Syntax

void STMR_ClearITPendingBit(STMR_IT_TypeDef STMR_IT);

Parameters

STMR_IT specifies the STMR interrupt status to be reset. It takes the following value:

- STMR_IT_UPDATE for the update interrupt.

Return value

None

Remarks

None.

3.9 STLUX general purpose I/O (stlux_gpio)

3.9.1 GPIO_Reset

This function initializes the GPIOs internal registers to their default values.

Syntax

void GPIO_Reset(GPIO_TypeDef* GPIOx);

Parameters

GPIOx specifies the general purpose IO to be reset. In the STLUX peripheral library by default the following GPIOs are defined:

- GPIO0 for the general purpose IO 0
- GPIO1 for the general purpose IO 1

Return value

None

Remarks

After reset almost all port-P0 I/Os are generally floating input signals although few pins may have different behavior. The port P1 I/Os are configured as output pins. Refer to the STLUX and STNRG product datasheets - pinout descriptions for all details.

3.9.2 GPIO_Init

This function initializes the GPIOs internal registers to a specific configuration.

Syntax

```
void GPIO_Init(
        GPIO_TypeDef* GPIOx,
        GPIO_Pin_TypeDef GPIO_Pin,
        GPIO_Mode_TypeDef GPIO_Mode
        );
```

Parameters

GPIOx specifies the general purpose IO to be initialized. In the STLUX peripheral library by default the following GPIOs are defined:

- GPIO0 for the general purpose IO 0
- GPIO1 for the general purpose IO 1

GPIO_Pin specifies the GPIO pins to be configured. It can take the following values:

- GPIO_PIN_0 to select the pin 0
- GPIO_PIN_1 to select the pin 1
- GPIO_PIN_2 to select the pin 2
- GPIO_PIN_3 to select the pin 3
- GPIO_PIN_4 to select the pin 4
- GPIO_PIN_5 to select the pin 5
- GPIO_PIN_6 to select the pin 6
- GPIO_PIN_7 to select the pin 7
- GPIO_PIN_LNIB to select low nibble pins
- GPIO_PIN_HNIB to select high nibble pins
- GPIO_PIN_ALL to select all pins

GPIO_Mode specifies the mode the selected pins must be configured. It can take the following values:

- GPIO_MODE_IN_FL_NO_IT for floating input, no external interrupt
- GPIO_MODE_IN_PU_NO_IT for pull-up input, no external interrupt
- GPIO_MODE_IN_FL_IT for floating input, external interrupt
- GPIO_MODE_IN_PU_IT for pull-up input, external interrupt
- GPIO_MODE_OUT_OD_LOW_FAST for open-drain output, low level, 10 MHz
- GPIO_MODE_OUT_PP_LOW_FAST for push-pull output, low level, 10 MHz
- GPIO_MODE_OUT_OD_LOW_SLOW for open-drain output, low level, 2 MHz
- GPIO_MODE_OUT_PP_LOW_SLOW for push-pull output, low level, 2 MHz
- GPIO_MODE_OUT_OD_HIZ_FAST for open-drain output, high impedance level, 10 MHz
- GPIO_MODE_OUT_PP_HIGH_FAST for push-pull output, high level, 10 MHz
- GPIO_MODE_OUT_OD_HIZ_SLOW for open-drain output, high impedance level, 2 MHz
- GPIO_MODE_OUT_PP_HIGH_SLOW for push-pull output, high level, 2 MHz

Return value

None

Remarks

Please note the input GPIO_Pin parameters can be put in logic OR, therefore multiple pin assignments can be done at once. I. e.: GPIO_Init(GPIO0, GPIO_PIN_0 | GPIO_PIN_1, GPIO_MODE_IN_FL_NO_IT);.

3.9.3 GPIO_Write

This function writes a specific value to the selected GPIOx output port.

Syntax

```
void GPIO_Write(
        GPIO_TypeDef* GPIOx,
        u8 PortVal
        );
```

Parameters

GPIOx specifies the general purpose IO to be set. In the STLUX peripheral library by default the following GPIOs are defined:

- GPIO0 for the general purpose IO 0
- GPIO1 for the general purpose IO 1

PortVal specifies the value to be written to the output port register.

Return value

None

Remarks

None.

3.9.4 GPIO_WriteHigh

This function sets a high level to specific pins of the selected GPIOx.

Syntax

```
void GPIO_WriteHigh(
        GPIO_TypeDef* GPIOx,
        GPIO_Pin_TypeDef PortPins
        );
```

Parameters

GPIOx specifies the general purpose IO to be set. In the STLUX peripheral library by default the following GPIOs are defined:

- GPIO0 for the general purpose IO 0
- GPIO1 for the general purpose IO 1

PortPins specifies the GPIO pins to be configured. It can take the following values:

- GPIO_PIN_0 to select the pin 0
- GPIO_PIN_1 to select the pin 1
- GPIO_PIN_2 to select the pin 2
- GPIO_PIN_3 to select the pin 3
- GPIO_PIN_4 to select the pin 4
- GPIO_PIN_5 to select the pin 5
- GPIO_PIN_6 to select the pin 6
- GPIO_PIN_7 to select the pin 7
- GPIO_PIN_LNIB to select low nibble pins
- GPIO_PIN_HNIB to select high nibble pins
- GPIO_PIN_ALL to select all pins

Return value

None

Remarks

Please note the input GPIO_Pins parameters can be put in logic OR, therefore multiple pin assignments can be done at once. I. e.: GPIO_ WriteHigh (GPIO0, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2);.

3.9.5 GPIO_WriteLow

This function sets a low level to specific pins of the selected GPIOx.

Syntax

```
void GPIO_WriteLow(
        GPIO_TypeDef* GPIOx,
        GPIO_Pin_TypeDef PortPins
        );
```

Parameters

GPIOx specifies the general purpose IO to be set. In the STLUX peripheral library by default the following GPIOs are defined:

- GPIO0 for the general purpose IO 0
- GPIO1 for the general purpose IO 1

PortPins specifies the GPIO pins to be configured. It can take the following values:

- GPIO_PIN_0 to select the pin 0
- GPIO_PIN_1 to select the pin 1
- GPIO_PIN_2 to select the pin 2
- GPIO_PIN_3 to select the pin 3
- GPIO_PIN_4 to select the pin 4
- GPIO_PIN_5 to select the pin 5
- GPIO_PIN_6 to select the pin 6
- GPIO_PIN_7 to select the pin 7
- GPIO_PIN_LNIB to select low nibble pins
- GPIO_PIN_HNIB to select high nibble pins
- GPIO_PIN_ALL to select all pins

Return value

None

Remarks

Please note the input GPIO_Pins parameters can be put in logic OR, therefore multiple pin assignments can be done at once. I. e.: GPIO_ WriteLow (GPIO0, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2);.

3.9.6 GPIO_WriteReverse

This function sets a reverse level to specific pins of the selected GPIOx.

Syntax

```
void GPIO_WriteReverse (
        GPIO_TypeDef* GPIOx,
        GPIO_Pin_TypeDef PortPins
        );
```

Parameters

GPIOx specifies the general purpose IO to be set. In the STLUX peripheral library by default the following GPIOs are defined:

- GPIO0 for the general purpose IO 0
- GPIO1 for the general purpose IO 1

PortPins specifies the GPIO pins to be configured. It can take the following values:

- GPIO_PIN_0 to select the pin 0
- GPIO_PIN_1 to select the pin 1
- GPIO_PIN_2 to select the pin 2
- GPIO_PIN_3 to select the pin 3
- GPIO_PIN_4 to select the pin 4
- GPIO_PIN_5 to select the pin 5
- GPIO_PIN_6 to select the pin6
- GPIO_PIN_7 to select the pin 7
- GPIO_PIN_LNIB to select low nibble pins
- GPIO_PIN_HNIB to select high nibble pins
- GPIO_PIN_ALL to select all pins

Return value

None

Remarks

Please note the input GPIO_Pins parameters can be put in logic OR, therefore multiple pin assignments can be done at once. I. e. GPIO_ WriteReverse (GPIO0, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2);.

3.9.7 GPIO_ReadInputData

This function reads input data from the selected GPIOx.

Syntax

u8 GPIO_ReadInputData(GPIO_TypeDef* GPIOx);

Parameters

GPIOx specifies the general purpose IO to be read. In the STLUX peripheral library by default the following GPIOs are defined:

- GPIO0 for the general purpose IO 0
- GPIO1 for the general purpose IO 1

Return value

The returned unsigned integer byte is the input data read from the selected GPIOx.

Remarks

None.

3.9.8 GPIO_ReadOutputData

This function reads output data to the selected GPIOx.

Syntax

u8 GPIO_ReadOutputData (GPIO_TypeDef* GPIOx);

Parameters

GPIOx specifies the general purpose IO to be read. In the STLUX peripheral library by default the following GPIOs are defined:

- GPIO0 for the general purpose IO 0
- GPIO1 for the general purpose IO 1

Return value

The returned unsigned integer byte is the output data read to the selected GPIOx.

Remarks

None.

3.9.9 GPIO_ReadInputPin

This function reads the specified input pin value from the selected GPIOx.

Syntax

```
BitStatus GPIO_ReadInputPin(
        GPIO_TypeDef* GPIOx,
        GPIO_Pin_TypeDef GPIO_Pin
        );
```

Parameters

GPIOx specifies the general purpose IO to be read. In the STLUX peripheral library by default the following GPIOs are defined:

- GPIO0 for the general purpose IO 0
- GPIO1 for the general purpose IO 1

GPIO_Pin specifies the GPIO pin or pins to be read. It can take the following values:

- GPIO_PIN_0 to select the pin 0
- GPIO_PIN_1 to select the pin 1
- GPIO_PIN_2 to select the pin 2
- GPIO_PIN_3 to select the pin 3
- GPIO_PIN_4 to select the pin 4
- GPIO_PIN_5 to select the pin 5
- GPIO_PIN_6 to select the pin 6
- GPIO_PIN_7 to select the pin 7
- GPIO_PIN_LNIB to select low nibble pins
- GPIO_PIN_HNIB to select high nibble pins
- GPIO_PIN_ALL to select all pins

Return value

The returned value is the status of the pin read from the selected GPIOx. It can be SET or RESET.

Remarks

Please note the input GPIO_Pin parameter can be a single pin number or more than one pin put in logic OR, therefore multiple pin assignments can be done at once. I. e.: GPIO_ ReadInputPin (GPIO0, (GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2)). In this case the returned value will be according to the input pin mask.

3.9.10 GPIO_ExternalPullUpConfig

This function configures the specified pin from the selected GPIOx in pull-up mode.

Syntax

```
void GPIO_ExternalPullUpConfig(
        GPIO_TypeDef* GPIOx,
        GPIO_Pin_TypeDef GPIO_Pin,
        FunctionalState NewState
    );
```

Parameters

GPIOx specifies the general purpose IO to be configured. In the STLUX peripheral library by default the following GPIOs are defined:

- GPIO0 for the general purpose IO 0
- GPIO1 for the general purpose IO 1

GPIO_Pin specifies the GPIO pin or pins to be configured. It can take the following values:

- GPIO_PIN_0 to select the pin 0
- GPIO_PIN_1 to select the pin 1
- GPIO_PIN_2 to select the pin 2
- GPIO_PIN_3 to select the pin 3
- GPIO_PIN_4 to select the pin 4
- GPIO_PIN_5 to select the pin 5
- GPIO_PIN_6 to select the pin 6
- GPIO_PIN_7 to select the pin 7
- GPIO_PIN_LNIB to select low nibble pins
- GPIO_PIN_HNIB to select high nibble pins
- GPIO_PIN_ALL to select all pins

NewState specifies whether to ENABLE or DISABLE the pull-up mode for the specified pins.

Return value

None

Remarks

Please note - the input GPIO_Pin parameter can be a single pin number or more than one pin put in logic OR, therefore multiple pin assignments can be done at once. I. e.: GPIO_ ReadInputPin (GPIO0, (GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2)).

3.10 STLUX auxiliary timer (stlux_atm)

3.10.1 ATM_Reset

This function sets the auxiliary timer (ATM) internal registers to their default initialization values.

Syntax

void ATM_Reset(void);

Parameters

None

Return value

None

Remarks

The auxiliary timer is a light timer built grouping some functionality already existing in the silicon device and spread on different IPs to optimize the silicon cost . In particular the ATM is based on configurable clock output (CCO), so its reset and configuration impacts the CCO registers value. ATM_Reset disables the CCO and sets the CCO divider to zero.

3.10.2 ATM_Config

This function sets the auxiliary timer (ATM) internal registers to their default initialization values.

Syntax

```
void ATM_Config(
        CLK_Output_TypeDef CLK_CCO,
        u8 CLK_CCODIVR,
        ATM_ITPolarity_TypeDef IT_LEV,
        ATM_ITTrigger_TypeDef IT_SEL,
        ATM_ITType_TypeDef IT_TYPE
        );
```

Parameters

CLK_CCO specifies the clock source for the ATM. It can be one of the following values:

- CLK_OUTPUT_HSI for the high-speed internal RC oscillator
- CLK_OUTPUT_LSI for the low-speed internal RC oscillator
- CLK_OUTPUT_HSE for the high-speed external RC oscillator
- CLK_OUTPUT_PLL for the PLL clock
- CLK_OUTPUT_CPU for the CPU clock
- CLK_OUTPUT_CKM for the master clock
- CLK_OUTPUT_SMED0_CK for the SMED0 clock
- CLK_OUTPUT_SMED1_CK for the SMED1 clock
- CLK_OUTPUT_SMED2_CK for the SMED2 clock
- CLK_OUTPUT_SMED3_CK for the SMED3 clock

- CLK_OUTPUT_SMED4_CK for the SMED4 clock
- CLK_OUTPUT_SMED5_CK for the SMED5 clock
- CLK_OUTPUT_ADC_CK for the ADC clock
- CLK_OUTPUT_AWU_CK for the auto-wakeup unit clock
- CLK_OUTPUT_PRESCALED_PLL_CK for the prescaled PLL clock

CLK_CCODIVR specifies the division factor *n* for the CCO clock to be used as a timebase for the ATM.

IT_LEV specifies the interrupts sensitivity level for the ATM. It can take the following values:

- ATM_IT_LEV_HIGH sets the interrupt sensitivity level to the high level or rising edge
- ATM_IT_LEV_LOW sets the interrupt sensitivity level to the low level or falling edge

IT_SEL specifies the trigger to generate interrupts for the ATM. It can take the following values:

- ATM_IT_SEL_EDGE triggers the interrupt generation on the rising/falling edge
- ATM_IT_SEL_LEVEL triggers the interrupt generation on the high/low level
- ATM_IT_SEL_LEVWAKEUP triggers the interrupt generation on the high/low asynchronous level with the wakeup capability

IT_TYPE specifies the type of the interrupt event generated by the ATM. It can take the following values:

- ATM_IT_TYPE_IRQ sets the interrupt type to maskable interrupt
- ATM_IT_TYPE_NMI sets the interrupt type to non maskable interrupt
- ATM_IT_TYPE_POL sets the interrupt type to polling mode. No interrupt is generated.

Return value

None

Remarks

None.

3.10.3 ATM_OutDigIn0

This function enables or disables the ATM clock to be sent as an output to DIGIN(0).

Syntax

void ATM_OutDigIn0(FunctionalState NewState);

Parameters

NewState specifies whether to ENABLE or DISABLE the ATM clock output to DIGIN(0).

Return value

None

Remarks

None.

3.10.4 ATM_ITConfig

This function enables or disables the ATM clock interrupt generation.

Syntax

void ATM_ITConfig(FunctionalState NewState);

Parameters

NewState specifies whether to ENABLE or DISABLE the ATM clock interrupt.

Return value

None

Remarks

None.

3.10.5 ATM_ITClear

This function clears the ATM clock interrupt status register.

Syntax

void ATM_ITClear (void);

Parameters

None

Return value

None

Remarks

None.

3.11 STLUX basic timer (stlux_btm)

3.11.1 BTM_Reset

This function sets the basic timer (BTM) internal registers to their default initialization values.

Syntax

void BTM_Reset(void);

Parameters

None

Return value

None

Remarks

The basic timer functionality is constituted by two light timers which are available only for the STNRG family devices. The reset function stops the timers and clears pending interrupts. It then disables the timers and sets the default HSI clock as a clock source. It also sends the basic timer clock division factor to zero.

3.11.2 BTM_Config

This function sets the basic timer (BTM) internal registers to their default initialization values.

Syntax

```
void BTM_Config(
        BTM_Selection_TypeDef BTMx,
        CLK_BTM_Source_TypeDef CLK_SRC,
        u8 CLK_DIV,
        u8 CLK_CNT,
        BTM_ITPolarity_TypeDef IT_LEV,
        BTM_ITTrigger_TypeDef IT_SEL,
        BTM_ITType_TypeDef IT_TYPE
    );
```

Parameters

BTMx identifies the selected basic timer to be configured. By default two basic timers are defined:

- BTM_0 for the basic timer 0
- BTM_1 for the basic timer 1

CLK_ SRC specifies the clock source for the BTM. It can be one of the following values:

- CLK_BTM_SOURCE_HSI for the high-speed internal RC oscillator
- CLK_BTM_SOURCE_LSI for the low-speed internal RC oscillator
- CLK_BTM_SOURCE_HSE for the high-speed external RC oscillator
- CLK_BTM_SOURCE_PLL for the PLL clock

CLK_DIV specifies the division factor *n* for the clock to be used as a timebase for the BTM.

CLK_CNT specifies the counter value to be set for the BTM.

IT_LEV specifies the interrupts sensitivity level for the BTM. It can take the following values:

- BTM_IT_LEV_HIGH sets the interrupt sensitivity level to the high level or rising edge
- BTM_IT_LEV_LOW sets the interrupt sensitivity level to the low level or falling edge

IT_SEL specifies the trigger to generate interrupts for the BTM. It can take the following values:

- BTM_IT_SEL_EDGE triggers the interrupt generation on the rising/falling edge
- BTM_IT_SEL_LEVEL triggers the interrupt generation on the high/low level
- BTM_IT_SEL_LEVWAKEUP triggers the interrupt generation on the high/low asynchronous level with the wakeup capability

IT_TYPE specifies the type of the interrupt event generated by the BTM. It can take the following values:

- BTM_IT_TYPE_IRQ sets the interrupt type to maskable interrupt
- BTM_IT_TYPE_NMI sets the interrupt type to non maskable interrupt
- BTM_IT_TYPE_POL sets the interrupt type to polling mode. No interrupt is generated

Return value

None

Remarks

None.

3.11.3 BTM_ITConfig

This function enables or disables the BTM clock interrupt generation.

Syntax

```
void BTM_ITConfig(
        BTM_Selection_TypeDef BTMx,
        FunctionalState NewState
        );
```

Parameters

BTMx identifies the selected basic timer which interrupt must be configured. By default two basic timers are defined:

- BTM_0 for the basic timer 0
- BTM_1 for the basic timer 1

NewState specifies whether to ENABLE or DISABLE the BTM clock interrupt.

Return value

None

Remarks

None.

3.11.4 BTM_ITClear

This function clears the BTM clock interrupt status register.

Syntax

void BTM_ITClear(BTM_Selection_TypeDef BTMx);

Parameters

BTMx identifies the selected basic timer which interrupt must be cleared. By default two basic timers are defined:

- BTM_0 for the basic timer 0
- BTM_1 for the basic timer 1

Return value

None

Remarks

None.

3.11.5 BTM_Cmd

This function enables or disables the selected BTM.

Syntax

```
void BTM_Cmd(
        BTM_Selection_TypeDef BTMx,
        FunctionalState NewState
        );
```

Parameters

BTMx identifies the selected basic timer to be configured. By default two basic timers are defined:

- BTM_0 for the basic timer 0
- BTM_1 for the basic timer 1

NewState specifies whether to ENABLE or DISABLE the selected BTM.

Return value

None

Remarks

None.

3.12 STLUX auto-wakeup unit (stlux_awu)

3.12.1 AWU_Reset

This function sets the auto-wakeup unit (AWU) internal registers to their default initialization values.

Syntax

void AWU_Reset(void);

Parameters

None

Return value

None

Remarks

By default the AWU and AWU interrupt generation are disabled. AWU prescaler value is set to reset value 0x3F. Moreover the AWU timebase is set to zero, which once more means no interrupt generation.

3.12.2 AWU_Init

This function sets the auto-wakeup unit (AWU) internal registers to their default initialization values.

Syntax

```
void AWU_Init(
        u8 AWU_Prescaler,
        AWU_Timebase_TypeDef AWU_TimeBase
        );
```

Parameters

AWU_Prescaler specifies the number of tics to be counted between two AWU interrupts. It must be a value ranging from 0 to 62. Please note that the final AWU divider will be: APRDIV = AWU_Prescaler+2.

AWU_TimeBase specifies the AWU timebase to program the wakeup interrupt frequency. It can take the following values:

- AWU_TIMEBASE_NO_IT means no AWU interrupt is selected
- AWU_TIMEBASE_1 for AWU timebase equal to [2^0 * APRDIV / f_AWU] ms
- AWU_TIMEBASE_2 for AWU timebase equal to [2^1 * APRDIV / f_AWU] ms
- AWU_TIMEBASE_4 for AWU timebase equal to [2^2 * APRDIV / f_AWU] ms
- AWU_TIMEBASE_8 for AWU timebase equal to [2^3 * APRDIV / f_AWU] ms
- AWU_TIMEBASE_16 for AWU timebase equal to [2^4 * APRDIV / f_AWU] ms
- AWU_TIMEBASE_32 for AWU timebase equal to [2^5 * APRDIV / f_AWU] ms
- AWU_TIMEBASE_64 for AWU timebase equal to [2^6 * APRDIV / f_AWU] ms
- AWU_TIMEBASE_128 for AWU timebase equal to [2^7 * APRDIV / f_AWU] ms
- AWU_TIMEBASE_256 for AWU timebase equal to [2^8 * APRDIV / f_AWU] ms
- AWU_TIMEBASE_1024 for AWU timebase equal to [2^9 * APRDIV / f_AWU] ms
- AWU_TIMEBASE_2048 for AWU timebase equal to [2^10 * APRDIV / f_AWU] ms
- AWU_TIMEBASE_4096 for AWU timebase equal to [2^11 * APRDIV / f_AWU] ms
- AWU_TIMEBASE_8192 for AWU timebase equal to [2^12 * APRDIV / f_AWU] ms
- AWU_TIMEBASE_16384 for AWU timebase equal to [2^13 * APRDIV / f_AWU] ms
- AWU_TIMEBASE_32768 for AWU timebase equal to [2^14 * APRDIV / f_AWU] ms

Return value

None

Remarks

None.

3.12.3 AWU_Enable

This function enables or disables the AWU peripheral.

Syntax

void AWU_Enable(FunctionalState NewState);

Parameters

NewState specifies whether to ENABLE or DISABLE the AWU.

Return value

None

Remarks

None.

3.12.4 AWU_IdleModeEnable

This function configures the AWU in idle mode to reduce power consumption.

Syntax

void AWU_IdleModeEnable(void);

Parameters

None

Return value

None

Remarks

The AWU in idle mode is disabled and the AWU timebase is reset to zero meaning no interrupt generation.

3.12.5 AWU_GetStatus

This function returns the current AWU status flag.

Syntax

FlagStatus AWU_GetStatus(void);

Parameters

None

Return value

The returned value specifies whether the AWU is currently working or not. It can be:

- SET meaning the AWU is enabled
- RESET meaning the AWU is disabled

Remarks

None.

3.13 STLUX universal asynchronous receiver/transmitter stlux_uart)

3.13.1 UART_Reset

This function sets the universal asynchronous receiver/transmitter unit (UART) internal registers to their default initialization values.

Syntax

void UART_Reset(void);

Parameters

None

Return value

None

Remarks

By default the UART clock and interrupt generation are disabled.

3.13.2 UART_Init

This function initializes the universal asynchronous receiver/transmitter unit (UART) internal registers to a chosen configuration.

Syntax

void UART_Init(
u32 BaudRate,
UART_WordLength_TypeDef WordLength,
UART_StopBits_TypeDef StopBits,
UART_Parity_TypeDef Parity,
UART_PIN_TypeDef PIN,
UART_Mode_TypeDef Mode
);

Parameters

BaudRate specifies the number of the baud to be used by the UART.

WordLength specifies the number of bits to be used by each transmitted/received data symbol. It can be:

- UART_WORDLENGTH_8D for 8 bits data
- UART_WORDLENGTH_9D for 9 bits data

StopBits specifies the number of bits to be used as a stop sequence symbol. It can be:

- UART_STOPBITS_1 = (u8)0x00, /**< One stop bit is transmitted at the end of frame*/
- UART_STOPBITS_0_5 = (u8)0x10, /**< Half stop bit is transmitted at the end of frame*/
- UART_STOPBITS_2 = (u8)0x20, /**< Two stop bits are transmitted at the end of frame*/
- UART_STOPBITS_1_5 = (u8)0x30 /**< One and half stop bits*/

Parity specifies whether even bits, odd bits or no parity check should be used. It can be:

- UART_PARITY_NO for no parity check
- UART_PARITY_EVEN for even parity check
- UART_PARITY_ODD for odd parity check

PIN specifies the couple of pins to be used as UART Tx/Rx for the current device. It can be:

- UART_PIN_14_15 for pins 14 and 15 to be used
- UART_PIN_20_21 for pins 20 and 21 to be used
- UART_PIN_22_23 for pins 22 and 23 to be used

Mode specifies the user mode to be enabled for the UART. It can be:

- UART_MODE_RX_ENABLE to enable the receive mode
- UART_MODE_TX_ENABLE to enable the transmit mode
- UART_MODE_TX_DISABLE to disable the transmit mode
- UART_MODE_RX_DISABLE for single-wire half duplex mode
- UART_MODE_TXRX_ENABLE to enable both transmit and receive mode

Return value

None

Remarks

Please check the STLUX and STNRG product datasheets for more information about the possible UART configurations.

3.13.3 UART_Cmd

This function initializes the universal asynchronous receiver/transmitter unit (UART) internal registers to a chosen configuration.

Syntax

void UART_Cmd(FunctionalState NewState);

Parameters

NewState specifies whether to ENABLE or DISABLE the UART.

Return value

None

Remarks

None.

3.13.4 UART_ITConfig

This function initializes the UART interrupts.

Syntax

```
void UART_ITConfig(
        UART_IT_TypeDef UART_IT,
        FunctionalState NewState
        );
```

Parameters

UART_IT specifies the interrupt to be enabled or disabled. It can be:

- UART_IT_TXE for the transmit interrupt
- UART_IT_TC for the transmission complete interrupt
- UART_IT_RXNE for the receive interrupt
- UART_IT_IDLE for the IDLE line interrupt
- UART_IT_OR for the overrun error interrupt
- UART_IT_PE for the parity error interrupt
- UART_IT_RXNE_OR for the receive/overrun interrupt

NewState specifies whether to ENABLE or DISABLE the UART.

Return value

None

Remarks

Please note the input UART_IT parameter can be a single interrupt or more than one interrupt put in logic OR, therefore multiple interrupts can be enabled at once. I. e.: void UART_ITConfig((UART_IT_TC| UART_IT_PE), ENABLE). For more details about UART interrupts please refer to the STLUX and STNRG product datasheets.

3.13.5 UART_IsITEnabled

This function initializes the UART interrupts.

Syntax

u8 UART_IsITEnabled(UART_IT_TypeDef UART_IT);

Parameters

UART_IT specifies the interrupt to be enabled or disabled. It can be:

- UART_IT_TXE for the transmit interrupt
- UART_IT_TC for the transmission complete interrupt
- UART_IT_RXNE for the receive interrupt
- UART_IT_IDLE for the IDLE line interrupt
- UART_IT_OR for the overrun error interrupt
- UART_IT_PE for the parity error interrupt
- UART_IT_RXNE_OR for the receive/overrun interrupt

Return value

If the specified interrupt is enabled the function returns 1, otherwise zero is returned.

Remarks

None.

3.13.6 UART_WakeUpConfig

This function sets the UART wakeup method.

Syntax

void UART_WakeUpConfig(UART_WakeUp_TypeDef UART_WakeUp);

Parameters

UART_WakeUp specifies the UART wakeup method; it can take the values:

- UART_WAKEUP_IDLELINE makes the UART wakeup when an idle frame is detected
- UART_WAKEUP_ADDRESSMARK makes the UART wakeup when an address character is received

Return value

None

Remarks

None.

3.13.7 UART_ReceiverWakeUpCmd

This function enables or disables the UART mute mode.

Syntax

void UART_ReceiverWakeUpCmd(FunctionalState NewState);

Parameters

NewState specifies whether to ENABLE or DISABLE the UART mute mode.

Return value

None

Remarks

None.

3.13.8 UART_ReceiveData8

This function returns the most recent received 8-bit data by the UART peripheral.

Syntax

u8 UART_ReceiveData8(void);

Parameters

None

Return value

The returned value is the most recently received 8-bit data.

Remarks

None.

3.13.9 UART_ReceiveData9

This function returns the most recent received 9-bit data by the UART peripheral.

Syntax

u16 UART_ReceiveData9(void);

Parameters

None

Return value

The returned value is the most recently received 9-bit data.

Remarks

None.

3.13.10 UART_SendData8

This function sends 8-bit data through the UART peripheral.

Syntax

void UART_SendData8(u8 Data);

Parameters

Data specifies the 8-bit data to be transmitted.

Return value

None

Remarks

None.

3.13.11 UART_SendData9

This function sends 9-bit data through the UART peripheral.

Syntax

void UART_SendData9(u16 Data);

Parameters

Data specifies the 9-bit data to be transmitted.

Return value

None

Remarks

None.

3.13.12 UART_SendBreak

This function sends break characters through the UART peripheral.

Syntax

void UART_SendBreak(void);

Parameters

None

Return value

None

Remarks

None.

3.13.13 UART_GetFlagStatus

This function checks whether a specified UART status flag is set or not.

Syntax

FlagStatus UART_GetFlagStatus(UART_Flag_TypeDef UART_FLAG);

Parameters

UART_FLAG specifies the status flag to be verified. It can take the following values:

- UART_FLAG_TXE for the transmitter data register empty flag
- UART_FLAG_TC for the transmission complete flag
- UART_FLAG_RXNE for the receiver data register not empty flag
- UART_FLAG_IDLE for the idle line detected flag
- UART_FLAG_OR for the overrun error flag
- UART_FLAG_NF for the noise error flag
- UART_FLAG_FE for the framing error flag
- UART_FLAG_PE for the parity error flag
- UART_FLAG_SBK for the send break characters flag

Return value

The returned value specifies whether the specified status flag is currently active. It can be SET or RESET.

Remarks

None.

3.13.14 UART_ClearFlag

This function clears the specified UART status flag.

Syntax

void UART_ClearFlag(UART_Flag_TypeDef UART_FLAG);

Parameters

UART_FLAG specifies the status flag to be cleared. It can take the following values:

- UART_FLAG_TXE for the transmitter data register empty flag
- UART_FLAG_TC for the transmission complete flag
- UART_FLAG_RXNE for the receiver data register not empty flag
- UART_FLAG_IDLE for the idle line detected flag
- UART_FLAG_OR for the overrun error flag
- UART_FLAG_NF for the noise error flag
- UART_FLAG_FE for the framing error flag
- UART_FLAG_PE for the parity error flag
- UART_FLAG_SBK for the send break characters flag

Return value

None

Remarks

None.

3.13.15 UART_GetITStatus

This function returns the specified UART interrupt status.

Syntax

ITStatus UART_GetITStatus(UART_IT_TypeDef UART_IT);

Parameters

UART_IT specifies the interrupt which status has to be verified. It can be:

- UART_IT_TXE for the transmit interrupt
- UART_IT_TC for the transmission complete interrupt
- UART_IT_RXNE for the receive interrupt
- UART_IT_IDLE for the IDLE line interrupt
- UART_IT_OR for the overrun error interrupt
- UART_IT_PE for the parity error interrupt
- UART_IT_RXNE_OR for the receive/overrun interrupt

Return value

The returned value is the current status for the specified UART interrupt. It can be SET or RESET.

Remarks

None.

3.13.16 UART_ClearITPendingBit

This function clears the specified pending UART interrupt.

Syntax

void UART_ClearITPendingBit(UART_IT_TypeDef UART_IT);

Parameters

UART_IT specifies the pending interrupt that has to be cleared. It can be:

- UART_IT_TXE for the transmit interrupt
- UART_IT_TC for the transmission complete interrupt
- UART_IT_RXNE for the receive interrupt
- UART_IT_IDLE for the IDLE line interrupt
- UART_IT_OR for the overrun error interrupt
- UART_IT_PE for the parity error interrupt
- UART_IT_RXNE_OR for the receive/overrun interrupt

Return value

None

Remarks

None.

3.14 STLUX Flash data memory (stlux_flash)

3.14.1 Unlock_eeprom_data_area

This function disables the EEPROM data memory write protection so that it can be modified.

Syntax

void Unlock_eeprom_data_area(void);

Parameters

None

Return value

None

Remarks

None.

3.14.2 Lock_eeprom_data_area

This function enables the EEPROM data memory write protection so that it can be protected from accidental modifications.

Syntax

void Lock_eeprom_data_area(void);

Parameters

None

Return value

None

Remarks

None.

3.14.3 Dump_data_to_eeprom

This function copies data from a specified RAM location to a specified EEPROM data memory location.

Syntax

void Dump_data_to_eeprom(u16 ramaddr, u16 eeaddr);

Parameters

ramaddr specifies the RAM address for the data to be copied.

eeaddr specifies the EEPROM data memory address where the data must be stored.

Return value

None

Remarks

None.

3.14.4 Write_data_eeprom

This function stores data to a specified EEPROM data memory location.

Syntax

void Write_data_eeprom(u16 addr, u8 val);

Parameters

addr specifies the EEPROM data memory address where the data must be stored.

val specifies the 8-bit value to be stored.

Return value

None

Remarks

None.

3.14.5 Read_8_data_eeprom

This function retrieve data from a specified EEPROM data memory location.

Syntax

u8 Read_8_data_eeprom(u16 addr);

Parameters

addr specifies the EEPROM data memory address where the data is stored.

Return value

The returned value is the 8-bit data read from the specified EEPROM data memory address.

Remarks

None.

3.15 STLUX interrupt controller (stlux_itc)

3.15.1 ITC_GetCPUCC

This function returns the current value for the CPU condition code register (CC).

Syntax

u8 ITC_GetCPUCC(void);

Parameters

None

Return value

The returned value is the 8-bit current value read from the CPU condition code register.

Remarks

None.

3.15.2 ITC_Reset

This function resets the interrupt priority registers to their default values.

Syntax

void ITC_Reset(void);

Parameters

None

Return value

None

Remarks

The interrupt request lines are controlled by 8 internal registers which configure the interrupt request priority level. The default value for the interrupt priority registers is set to non-interruptible. For more information about the interrupt controller please refer to the product reference manual RM0380.

3.15.3 ITC_GetSoftIntStatus

This function returns the current software interrupt priority read from the control code register.

Syntax

u8 ITC_GetSoftIntStatus(void);

Parameters

None

Return value

The returned value is the current software interrupt priority value.

Remarks

None.

3.15.4 ITC_SetSoftwarePriority

This function sets the software interrupt priority level for the specified interrupt source.

Syntax

```
void ITC_SetSoftwarePriority(
        ITC_Irq_TypeDef IrqNum,
        ITC_PriorityLevel_TypeDef PriorityValue
        );
```

Parameters

IrqNum specifies the interrupt source which priority level must be set. It can be:

- ITC_IRQ_NMI for the non maskable interrupt
- ITC_IRQ_AWU for the auto-wakeup unit
- ITC_IRQ_CLK for the clock controller

- ITC_IRQ_PORT0 for the external port 0 corresponding to GPIOs
- ITC_IRQ_PORT1 for the external port 1 corresponding to auxiliary and basic timers
- ITC_IRQ_PORT2 for the external port 2 corresponding to DIGINs
- ITC_IRQ_SMED0 for the SMED 0
- ITC_IRQ_SMED1 for the SMED 1
- ITC_IRQ_INPP3 for the analog comparators unit (ACU)[(a)]
- ITC_IRQ_SMED2 for the SMED 2
- ITC_IRQ_SMED3 for the SMED 3
- ITC_IRQ_UART_TX for the UART transmitter
- ITC_IRQ_UART_RX for the UART receiver
- ITC_IRQ_I2C for the I^2C
- ITC_IRQ_ADC for the analog-to-digital converter
- ITC_IRQ_STMR for the system timer
- ITC_IRQ_FLASH for the Flash memory
- ITC_IRQ_DALI for the DALI interface
- ITC_IRQ_SMED4 for the SMED 4
- ITC_IRQ_SMED5 for the SMED 5

PriorityValue specifies the interrupt priority level to be assigned for the selected source. It can be:

- ITC_PRIORITYLEVEL_0 for the priority level 0
- ITC_PRIORITYLEVEL_1 for the priority level 1
- ITC_PRIORITYLEVEL_2 for the priority level 2
- ITC_PRIORITYLEVEL_3 for the priority level 3

Return value

None

Remarks

(a)

a. Please note that the ACU is available as an interrupt source only for STNRG devices. For more information about the STNRG please refer to the product datasheet.

3.15.5 ITC_GetSoftwarePriority

This function returns the software interrupt priority level for the specified interrupt source.

Syntax

ITC_PriorityLevel_TypeDef ITC_GetSoftwarePriority(ITC_Irq_TypeDef IrqNum);

Parameters

IrqNum specifies the interrupt source which priority level must be set. It can be:

- ITC_IRQ_NMI for the non maskable interrupt
- ITC_IRQ_AWU for the auto-wakeup unit
- ITC_IRQ_CLK for the clock controller
- ITC_IRQ_PORT0 for the external port 0 corresponding to GPIOs
- ITC_IRQ_PORT1 for the external port 1 corresponding to auxiliary and basic timers
- ITC_IRQ_PORT2 for the external port 2 corresponding to DIGINs
- ITC_IRQ_SMED0 for the SMED 0
- ITC_IRQ_SMED1 for the SMED 1
- ITC_IRQ_INPP3 for the analog comparators unit (ACU)[(b)]
- ITC_IRQ_SMED2 for the SMED 2
- ITC_IRQ_SMED3 for the SMED 3
- ITC_IRQ_UART_TX for the UART transmitter
- ITC_IRQ_UART_RX for the UART receiver
- ITC_IRQ_I2C for the I^2C
- ITC_IRQ_ADC for the analog-to-digital converter
- ITC_IRQ_STMR for the system timer
- ITC_IRQ_FLASH for the Flash memory
- ITC_IRQ_DALI for the DALI interface
- ITC_IRQ_SMED4 for the SMED 4
- ITC_IRQ_SMED5 for the SMED 5

Return value

The returned value is the software interrupt priority level assigned for the selected interrupt source. It can be:

- ITC_PRIORITYLEVEL_0 for the priority level 0
- ITC_PRIORITYLEVEL_1 for the priority level 1
- ITC_PRIORITYLEVEL_2 for the priority level 2
- ITC_PRIORITYLEVEL_3 for the priority level 3

Remarks

(b)

b. Please note that the ACU is available as an interrupt source only for STNRG devices. For more information about the STNRG please refer to the product datasheet.

4 Revision history

Table 4. Document revision history

Date	Revision	Changes
14-Jan-2016	1	Initial release.