Services

Overview

APIs

APIs

This section provides a reference to the APIs for services on the SiWx91x™ chipset including functions, data types, and constants.

- Power Manager functions to use power manager functionality in optimizing/handling power consumption.
- Sensor Hub functions to generate sensorhub functionality.
- Sleep Timer functions to access software timers, delays, timekeeping, and date features using a low-frequency real-time clock.
- Input/Output Stream functions to perform input/output by creating streams.
- Non-volatile Memory functions to access the non-volatile memory (NVM3) driver to maintain key-value pairs in flash memory.
- Watchdog Manager The Watchdog Timer Manager provides functions to monitor and manage the health of the system by resetting it if it becomes unresponsive or enters an error state.
- Clock Manager Functions to manage clock configurations, ensuring stable and efficient operation of the Si91x system.
- Firmware Fallback A and B Functions to manage firmware fallback functionality with A/B slot management.

Modules

Power Manager

Sensor Hub

Sleep Timer

Input/Output Stream

Non-volatile Memory

Watchdog Manager

Clock Manager

Firmware Fallback A and B

Power Manager

Power Manager

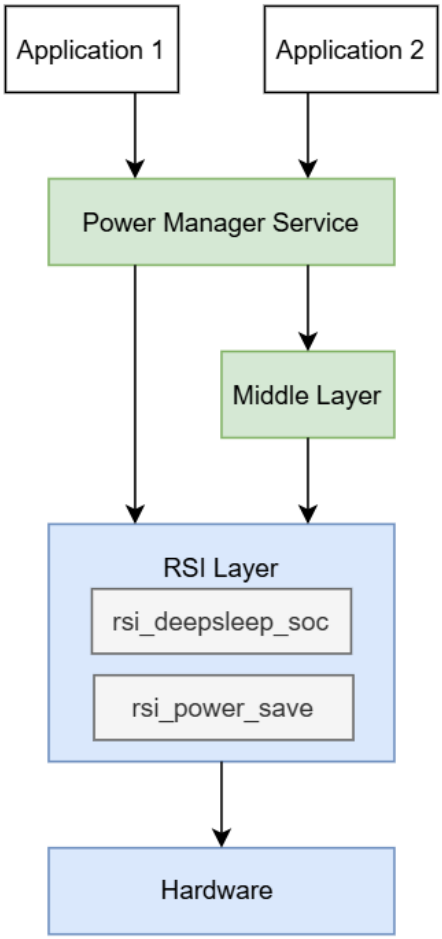
Introduction

The Power Manager is a platform-level software module that manages the system's power states. The power state requirements are set by the different software modules (drivers, stacks, application code, etc...). Power Manager also offers a notification mechanism through which any piece of software module can be notified of power state transitions through callbacks.

Initialization

Power Manager must be initialized before any call to the Power Manager API. If sl_system is used, only sl_system_init() must be called, otherwise sl_si91x_power_manager_init() must be called manually.

Power Manager Architecture



Add and remove requirements

The driver/application can add and remove power state requirements at runtime. Adding requirement function calls changes the power state. Removing requirement function calls will not have any effect for state transition. sl_si91x_power_manager_add_peripheral_requirement()

sl_si91x_power_manager_remove_peripheral_requirement()

Subscribe to events

It is possible to get notified when the system transitions from a power state to another power state. This can allow to do some operations depending on which level the system goes, such as saving/restoring context. sl_si91x_power_manager_subscribe_ps_transition_event()

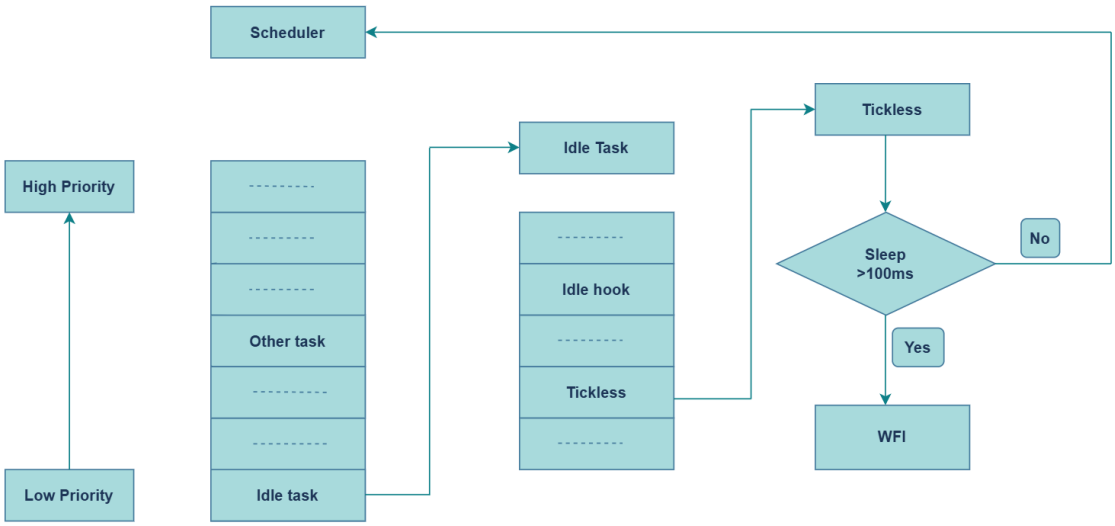
sl_si91x_power_manager_unsubscribe_ps_transition_event()

Memory Footprint Analysis

- Memory footprint analysis helps in understanding the memory usage of the application.
- It includes details about stack, heap, and global/static memory usage.
- Tools like map files or IDE-specific memory analyzers can be used to analyze the memory footprint.
- Optimizing memory usage ensures efficient resource utilization and avoids memory overflows.
- In this example, ensure that the stack size (3072 bytes) and other memory allocations are sufficient for the application's requirements.

Tickless idle Mode

- Application Task: This represents tasks running in your application. These tasks might call vTaskDelay or other blocking functions, leading the scheduler to consider them idle.
- Scheduler (Idle): When no tasks are ready to run, the scheduler enters an idle state.
- Check for Idle: The scheduler checks if FreeRTOS tickless mode is enabled (configUSE_TICKLESS_IDLE=1) and if all tasks are blocked.
- Suspend Ticks & Enter Sleep: If conditions are met, the scheduler calls vPortSuppressTicksAndSleep to:
  - Disable the system tick interrupt.
  - start a timer to track the sleep duration.
  - Enter a low-power mode using MCU power management features (e.g., Wait For Interrupt instruction).
- Wakeup Interrupt: An interrupt (event or timer) wakes up the MCU.
- Scheduler Update: The scheduler calculates the sleep duration and adjusts the system tick counter accordingly.
- Resume Scheduler & Tasks: The scheduler resumes the tick interrupt and exits the low-power mode. Tasks become ready to run again.
- Application Code: Application tasks continue execution based on the updated system time.



Sleep

When the software has no more operations and only needs to wait for an event, the software must call sl_si91x_power_manager_sleep().

Query callback functions

Is OK to sleep

Between the time sl_si91x_power_manager_sleep() is called and the MCU goes to sleep, an ISR may occur and require the system to resume at that time instead of sleeping. In this case, a callback is called in a critical section to validate that the MCU can go to sleep.

The function sl_si91x_power_manager_is_ok_to_sleep() will be generated automatically by Simplicity Studio's wizard. The function will look at multiple software modules from the SDK to make a decision. The application can contribute to the decision by defining the function `app_is_ok_to_sleep()` . If any of the software modules (including the application via app_is_ok_to_sleep()) return false, the process of entering in sleep will be aborted.

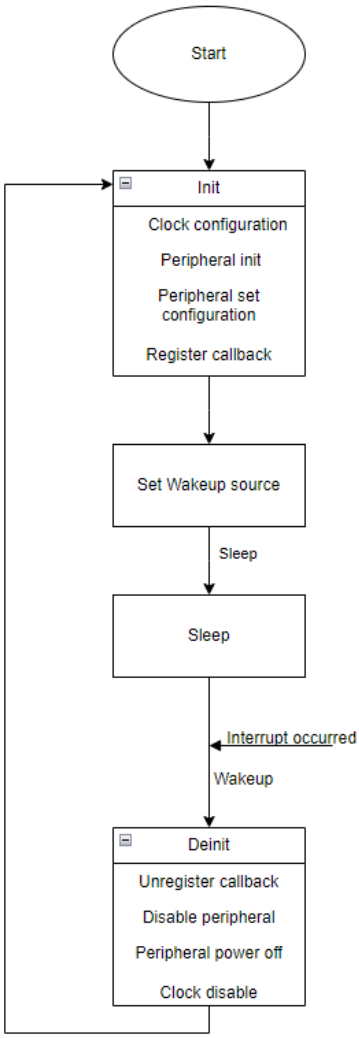
Sleep on ISR exit

When the system enters sleep, the only way to wake it up is via an interrupt or exception. By default, Power Manager will assume that when an interrupt occurs and the corresponding ISR has been executed, the system must not go back to sleep. However, in the case where all the processing related to this interrupt is performed in the ISR, it is possible to go back to sleep by using this hook.

The function sl_si91x_power_manager_sleep_on_isr_exit() will be generated automatically by Simplicity Studio's wizard. The function will look at multiple software modules from the SDK to make a decision. The application can contribute to the decision by defining the function app_sleep_on_isr_exit(). The generated function will make a decision based on the value returned by the different software modules (including the application via app_sleep_on_isr_exit()):

- SL_SI91X_POWER_MANAGER_ISR_IGNORE: if the software module did not cause the system wakeup and/or doesn't want to contribute to the decision.
- SL_SI91X_POWER_MANAGER_ISR_SLEEP: if the software module did cause the system wakeup, but the system should go back to sleep.
- SL_SI91X_POWER_MANAGER_ISR_WAKEUP: if the software module did cause the system wakeup, and the system should not go back to sleep.

If any software module returned SL_SI91X_POWER_MANAGER_ISR_SLEEP and none returned SL_SI91X_POWER_MANAGER_ISR_WAKEUP, the system will go back to sleep. Any other combination will cause the system not to go back to sleep.



Debugging feature

By installing the Power Manager debug component and setting the configuration define SL_SI91X_POWER_MANAGER_DEBUG to 1, it is possible to record the requirements currently set and their owner. It is possible to print a table at any time that lists all the added requirements and their owner. This table can be printed by calling the function: Make sure to add the following define

```
#define CURRENT_MODULE_NAME // Module printable name here
```

to any application code source file that adds and removes requirements.

Configuration

Power Manager allows configuration of RAM retention, peripheral states, wakeup sources, and clock scaling.

- Clock Scaling:
  - Clock can be configured as power-save or performance based on the user requirement. By default after the state change, the clock is configured as powersave. The below API is used to set the clock scaling.

sl_si91x_power_manager_set_clock_scaling(); where mode can be either power-save or performance.

- Peripheral States:
  - High power, ULPSS, and NPSS peripherals can be configured using the below APIs. They can be powered on or off based on user requirements. sl_si91x_power_manager_add_peripheral_requirement()sl_si91x_power_manager_remove_peripheral_requirement();
- RAM retention:
  - Retains the RAM in low power state either by using size or RAM bank as input parameter. sl_si91x_power_manager_configure_ram_retention();
- Wakeup source:
  - The wakeup sources can be configured using the below API. sl_si91x_power_manager_set_wakeup_sources();

Usage

- Initialization:
  - sl_si91x_power_manager_init(): Initializes the Power Manager service and sets the initial power state.
- Peripheral requirements:
  - sl_si91x_power_manager_add_peripheral_requirement(): Turns on/off the peripheral power
  - sl_si91x_power_manager_remove_peripheral_requirement(): Removes the peripheral requirement.
- Power State Transitions:
  - sl_si91x_power_manager_subscribe_ps_transition_event(): Registers a callback for a specific power state transition.
  - sl_si91x_power_manager_sleep(): To Put the device into sleep mode.
  - sl_si91x_power_manager_unsubscribe_ps_transition_event(): Unregisters the callback for the power state transition.
- De-initialization:
  - sl_si91x_power_manager_deinit(): De-initializes the Power Manager service.

Usage Example

```
#define  PS_EVENT_MASK     ( SL_SI91X_POWER_MANAGER_EVENT_TRANSITION_ENTERING_PS4\
                    | SL_SI91X_POWER_MANAGER_EVENT_TRANSITION_LEAVING_PS4 \
                    | SL_SI91X_POWER_MANAGER_EVENT_TRANSITION_ENTERING_PS3\
                    | SL_SI91X_POWER_MANAGER_EVENT_TRANSITION_LEAVING_PS3 \
                    | SL_SI91X_POWER_MANAGER_EVENT_TRANSITION_ENTERING_PS2\
                    | SL_SI91X_POWER_MANAGER_EVENT_TRANSITION_LEAVING_PS2 \
                    | SL_SI91X_POWER_MANAGER_EVENT_TRANSITION_LEAVING_SLEEP)

static void power_manager_app(void)
{
sl_status_t status;
sl_power_manager_ps_transition_event_handle_t handle;
sl_power_manager_ps_transition_event_info_t info = { .event_mask = PS_EVENT_MASK, .on_event = transition_callback };


// Subscribe the state transition callback events, the ored value of flag and function pointer is passed in this API.
status = sl_si91x_power_manager_subscribe_ps_transition_event(&handle, &info);
if (status != SL_STATUS_OK) {
  // If status is not OK, return with the error code.
  return;
 }
// Configuring the RAM retention used for sleep-wakeup.
sl_power_ram_retention_config_t config;
 config.configure_ram_banks = true;
 config.m4ss_ram_banks      = SL_SI91X_POWER_MANAGER_M4SS_RAM_BANK_8 | SL_SI91X_POWER_MANAGER_M4SS_RAM_BANK_9 |
SL_SI91X_POWER_MANAGER_M4SS_RAM_BANK_10;
 config.ulpss_ram_banks     = SL_SI91X_POWER_MANAGER_ULPSS_RAM_BANK_2 | SL_SI91X_POWER_MANAGER_ULPSS_RAM_BANK_3;
 // RAM retention modes are configured and passed into this API.
 status = sl_si91x_power_manager_configure_ram_retention(&config);
 if (status != SL_STATUS_OK) {
 // If status is not OK, return with the error code.
  return;
 }

// Change state to PS2
sl_si91x_power_manager_add_ps_requirement(SL_SI91X_POWER_MANAGER_PS2);

// CODE BLOCK //

// Change state to PS4
// Removed PS2 requirement as it is no longer required.
sl_si91x_power_manager_remove_ps_requirement(SL_SI91X_POWER_MANAGER_PS2);
sl_si91x_power_manager_add_ps_requirement(SL_SI91X_POWER_MANAGER_PS2);

// SLEEP_WAKEUP

// Initialize wakeup source
// Replace the wakeup source peripheral macro defined in sl_si91x_power_manager.h file
// It sets the below peripheral as wakeup source
sl_si91x_power_manager_set_wakeup_sources(WAKEUP_SOURCE, true);
sl_si91x_power_manager_sleep();
}
```

Modules

- sl_power_ram_retention_config_t
- sl_power_peripheral_t
- sl_power_manager_ps_transition_event_info_t
- sl_power_manager_ps_transition_event_handle_t

Enumerations

enum sl_power_state_t {

SL_SI91X_POWER_MANAGER_PS0 = 0
SL_SI91X_POWER_MANAGER_PS1
SL_SI91X_POWER_MANAGER_PS2
SL_SI91X_POWER_MANAGER_PS3
SL_SI91X_POWER_MANAGER_PS4
SL_SI91X_POWER_MANAGER_SLEEP
SL_SI91X_POWER_MANAGER_STANDBY
LAST_ENUM_POWER_STATE

}

Enumeration for the power states.

enum sl_clock_scaling_t {

SL_SI91X_POWER_MANAGER_POWERSAVE
SL_SI91X_POWER_MANAGER_PERFORMANCE
LAST_ENUM_CLOCK_SCALING

}

Enumeration for clock scaling parameters.

enum sl_si91x_power_manager_on_isr_exit_t {

SL_SI91X_POWER_MANAGER_ISR_IGNORE = (1UL << 0UL)
SL_SI91X_POWER_MANAGER_ISR_SLEEP = (1UL << 1UL)
SL_SI91X_POWER_MANAGER_ISR_WAKEUP = (1UL << 2UL)

}

On ISR Exit Hook answer.

Typedefs

typedef uint32_t sl_power_manager_ps_transition_event_t

Mask of all the event(s) to listen to.

typedef void(* sl_power_manager_ps_transition_on_event_t)(sl_power_state_t from, sl_power_state_t to)

Typedef for the user-supplied callback function which is called when a power state transition occurs.

Functions

sl_status_t sl_si91x_power_manager_init(void)

To initialize the Power Manager service.

__STATIC_INLINE void sl_si91x_power_manager_core_entercritical(void)

To disable the interrupts.

__STATIC_INLINE void sl_si91x_power_manager_core_exitcritical(void)

To enable the interrupts.

__STATIC_INLINE sl_status_t sl_si91x_power_manager_add_ps_requirement(sl_power_state_t state)

To add a requirement on power states.

__STATIC_INLINE sl_status_t sl_si91x_power_manager_remove_ps_requirement(sl_power_state_t state)

To remove requirement on power states.

sl_status_t sl_si91x_power_manager_set_clock_scaling(sl_clock_scaling_t mode)

To configure the clock scaling.

sl_clock_scaling_t	sl_si91x_power_manager_get_clock_scaling(void) the clock scaling mode in PS4 and PS3 power state.
sl_status_t	sl_si91x_power_manager_add_peripheral_requirement(sl_power_peripheral_t *peripheral) Adds the peripheral requirement.
sl_status_t	sl_si91x_power_manager_remove_peripheral_requirement(sl_power_peripheral_t *peripheral) To remove the peripheral requirement.
sl_status_t	sl_si91x_power_manager_subscribe_ps_transition_event(sl_power_manager_ps_transition_event_handle_t *event_handle, const sl_power_manager_ps_transition_event_info_t *event_info) To register a callback to be called on given power state transition(s).
sl_status_t	sl_si91x_power_manager_unsubscribe_ps_transition_event(sl_power_manager_ps_transition_event_handle_ *event_handle, const sl_power_manager_ps_transition_event_info_t *event_info) To unregister an event callback handle on power state transition.
sl_status_t	sl_si91x_power_manager_sleep(void) To move into sleep mode and wait for the peripheral to be set as a wakeup source to trigger and wake up the M4 SoC.
void	sl_si91x_power_manager_standby(void) To move into standby state and wait for the interrupt.
sl_status_t	sl_si91x_power_manager_set_wakeup_sources(uint32_t source, boolean_t add) To configure the wakeup sources.
sl_status_t	sl_si91x_power_manager_configure_ram_retention(sl_power_ram_retention_config_t *config) To retain the RAM in low power state either by using size or RAM bank as input parameters.
sl_power_state_t	sl_si91x_power_manager_get_current_state(void) To return the current power state.
uint8_t *	sl_si91x_power_manager_get_requirement_table(void) To get the current requirements on all the power states.
sl_status_t	sl_si91x_power_manager_request_ps1_state(void) Requests the PS1 state requirement during tickless idle mode.
sl_status_t	sl_si91x_power_manager_remove_ps1_state_request(void) Remove the PS1 state request during tickless idle mode.
bool	sl_si91x_power_manager_get_ps1_state_status(void) Get the current ps1 state request status from the power manager during tickless idle mode.
sl_status_t	sl_si91x_power_manager_request_standby_state(void) Requests the standby state during tickless idle mode.
sl_status_t	sl_si91x_power_manager_remove_standby_state_request(void) Remove the standby state request during tickless idle mode.
bool	sl_si91x_power_manager_get_standby_state_status(void) Get the current standby state request status during tickless idle mode.
bool	sl_si91x_power_manager_is_tx_command_in_progress(void) Check if a TX command is currently in progress.
void	sl_si91x_power_manager_deinit(void) To de-initialize the Power Manager service.
bool	sl_si91x_power_manager_ps2_pre_check(void) Performs pre-transition checks before entering PS2 state.

void sl_si91x_power_manager_debug_print_ps_requirements(void)
To print a table that describes the current requirements on each power state and their owner.

Macros

#define SL_SI91X_POWER_MANAGER_EVENT_TRANSITION_ENTERING_PS4 (1 << 0)
sl power manager event transition for entering PS4 state.

#define SL_SI91X_POWER_MANAGER_EVENT_TRANSITION_LEAVING_PS4 (1 << 1)
sl power manager event transition for leaving PS4 state.

#define SL_SI91X_POWER_MANAGER_EVENT_TRANSITION_ENTERING_PS3 (1 << 2)
sl power manager event transition for entering PS3 state.

#define SL_SI91X_POWER_MANAGER_EVENT_TRANSITION_LEAVING_PS3 (1 << 3)
sl power manager event transition for leaving PS3 state.

#define SL_SI91X_POWER_MANAGER_EVENT_TRANSITION_ENTERING_PS2 (1 << 4)
sl power manager event transition for entering PS2 state.

#define SL_SI91X_POWER_MANAGER_EVENT_TRANSITION_LEAVING_PS2 (1 << 5)
sl power manager event transition for leaving PS2 state.

#define SL_SI91X_POWER_MANAGER_EVENT_TRANSITION_LEAVING_PS1 (1 << 6)
sl power manager event transition for leaving PS1 state.

#define SL_SI91X_POWER_MANAGER_EVENT_TRANSITION_LEAVING_SLEEP (1 << 7)
sl power manager event transition for leaving sleep state.

#define SL_SI91X_POWER_MANAGER_EVENT_TRANSITION_LEAVING_STANDBY (1 << 8)
Event transition for leaving standby state.

#define SL_SI91X_POWER_MANAGER_DST_WAKEUP DST_BASED_WAKEUP
Deep Sleep Timer based wakeup source.

#define SL_SI91X_POWER_MANAGER_WIRELESS_WAKEUP WIRELESS_BASED_WAKEUP
Wireless based wakeup source.

#define SL_SI91X_POWER_MANAGER_GPIO_WAKEUP GPIO_BASED_WAKEUP
GPIO based wakeup source.

#define SL_SI91X_POWER_MANAGER_COMPARATOR_WAKEUP COMPR_BASED_WAKEUP
Comparator based wakeup source.

#define SL_SI91X_POWER_MANAGER_SYSRTC_WAKEUP SYSRTC_BASED_WAKEUP
Sysrtc based wakeup source.

#define SL_SI91X_POWER_MANAGER_ULPSS_WAKEUP ULPSS_BASED_WAKEUP
ULP peripheral based wakeup source.

#define SL_SI91X_POWER_MANAGER_SDCSS_WAKEUP SDCSS_BASED_WAKEUP
SDC (Sensor data collector) based wakeup source.

#define SL_SI91X_POWER_MANAGER_ALARM_WAKEUP ALARM_BASED_WAKEUP
Alarm based wakeup source.

#define SL_SI91X_POWER_MANAGER_SEC_WAKEUP SEC_BASED_WAKEUP
Second based wakeup source.

#define SL_SI91X_POWER_MANAGER_MSEC_WAKEUP MSEC_BASED_WAKEUP
Milli second based wakeup source.

#define SL_SI91X_POWER_MANAGER_WDT_WAKEUP WDT_INTR_BASED_WAKEUP
Watchdog interrupt based wakeup source.

#define SL_SI91X_POWER_MANAGER_M4SS_PG_EFUSE M4SS_PWRGATE_ULP_EFUSE_PERI
M4SS EFUSE Power Gate.

#define SL_SI91X_POWER_MANAGER_M4SS_PG_RPDMA M4SS_PWRGATE_ULP_RPDMA
M4SS RPDMA Power Gate.

#define SL_SI91X_POWER_MANAGER_M4SS_PG_SDIO_SPI M4SS_PWRGATE_ULP_SDIO_SPI
M4SS SDIO SPI Power Gate.

#define SL_SI91X_POWER_MANAGER_M4SS_PG_QSPI M4SS_PWRGATE_ULP_QSPI_ICACHE
M4SS QSPI and ICACHE Power Gate.

#define SL_SI91X_POWER_MANAGER_M4SS_PG_IID M4SS_PWRGATE_ULP_IID
M4SS IID Power Gate.

#define SL_SI91X_POWER_MANAGER_M4SS_PG_M4_DEBUG M4SS_PWRGATE_ULP_M4_DEBUG_FPU
M4SS M4 Debug Power Gate.

#define SL_SI91X_POWER_MANAGER_M4SS_PG_M4_CORE M4SS_PWRGATE_ULP_M4_CORE
M4SS M4 Core Power Gate.

#define SL_SI91X_POWER_MANAGER_M4SS_PG_EXTERNAL_ROM M4SS_PWRGATE_ULP_EXT_ROM
M4SS External ROM Power Gate.

#define SL_SI91X_POWER_MANAGER_ULPSS_PG_MISC ULPSS_PWRGATE_ULP_MISC
ULP Miscellaneous Power Gate.

#define SL_SI91X_POWER_MANAGER_ULPSS_PG_CAP ULPSS_PWRGATE_ULP_CAP
DEPRECATED - ULP Capacitive Touch Sensor Power Gate.

#define SL_SI91X_POWER_MANAGER_ULPSS_PG_UART ULPSS_PWRGATE_ULP_UART
ULP UART Power Gate.

#define SL_SI91X_POWER_MANAGER_ULPSS_PG_SSI ULPSS_PWRGATE_ULP_SSI
ULP SSI Power Gate.

#define SL_SI91X_POWER_MANAGER_ULPSS_PG_I2S ULPSS_PWRGATE_ULP_I2S
ULP I2S Power Gate.

#define SL_SI91X_POWER_MANAGER_ULPSS_PG_I2C ULPSS_PWRGATE_ULP_I2C
ULP I2C Power Gate.

#define SL_SI91X_POWER_MANAGER_ULPSS_PG_AUX ULPSS_PWRGATE_ULP_AUX
ULP AUX Power Gate.

#define SL_SI91X_POWER_MANAGER_ULPSS_PG_IR ULPSS_PWRGATE_ULP_IR
ULP IR Power Gate.

#define SL_SI91X_POWER_MANAGER_ULPSS_PG_UDMA ULPSS_PWRGATE_ULP_UDMA
ULP UDMA Power Gate.

#define SL_SI91X_POWER_MANAGER_ULPSS_PG_FIM ULPSS_PWRGATE_ULP_FIM
ULP FIM Power Gate.

#define SL_SI91X_POWER_MANAGER_NPSS_PG_MCUBFFS SLPSS_PWRGATE_ULP_MCUBFFS
NPSS MCU BFFS (Battery FF's) Power Gate.

#define SL_SI91X_POWER_MANAGER_NPSS_PG_MCUFSM SLPSS_PWRGATE_ULP_MCUFSM
NPSS MCU FSM Power Gate.

#define SL_SI91X_POWER_MANAGER_NPSS_PG_MCURTC SLPSS_PWRGATE_ULP_MCURTC
NPSS MCU RTC (Real Time Clock) Power Gate.

#define SL_SI91X_POWER_MANAGER_NPSS_PG_MCUWDT SLPSS_PWRGATE_ULP_MCUWDT
NPSS MCU WDT (Watchdog Timer) Power Gate.

#define SL_SI91X_POWER_MANAGER_NPSS_PG_MCUPS SLPSS_PWRGATE_ULP_MCUPS
NPSS MCU Process Sensor Power Gate.

#define SL_SI91X_POWER_MANAGER_NPSS_PG_MCUTS SLPSS_PWRGATE_ULP_MCUTS
NPSS MCU Temperature Sensor Power Gate.

#define SL_SI91X_POWER_MANAGER_NPSS_PG_MCUSTORE1 SLPSS_PWRGATE_ULP_MCUSTORE1
NPSS MCU Storage 1 Power Gate.

#define SL_SI91X_POWER_MANAGER_NPSS_PG_MCUSTORE2 SLPSS_PWRGATE_ULP_MCUSTORE2
NPSS MCU Storage 2 Power Gate.

#define SL_SI91X_POWER_MANAGER_NPSS_PG_MCUSTORE3 SLPSS_PWRGATE_ULP_MCUSTORE3
NPSS MCU Storage 3 Power Gate.

#define SL_SI91X_POWER_MANAGER_NPSS_PG_TIMEPERIOD SLPSS_PWRGATE_ULP_TIMEPERIOD
NPSS Time Period Power Gate.

#define SL_SI91X_POWER_MANAGER_NPSS_PG_NWPAPB_MCU_CTRL
SLPSS_PWRGATE_ULP_NWPAPB_MCU_CTRL
NPSS MCU APB Control Power Gate.

#define SL_SI91X_POWER_MANAGER_M4SS_RAM_BANK_1 RAM_BANK_0
4 KB (Bank 1 of first 192k chunk)

#define SL_SI91X_POWER_MANAGER_M4SS_RAM_BANK_2 RAM_BANK_1
4 KB (Bank 2 of first 192k chunk)

#define SL_SI91X_POWER_MANAGER_M4SS_RAM_BANK_3 RAM_BANK_2
4 KB (Bank 3 of first 192k chunk)

#define SL_SI91X_POWER_MANAGER_M4SS_RAM_BANK_4 RAM_BANK_3
4 KB (Bank 4 of first 192k chunk)

#define SL_SI91X_POWER_MANAGER_M4SS_RAM_BANK_5 RAM_BANK_4
4 KB (Bank 5 of first 192k chunk)

#define SL_SI91X_POWER_MANAGER_M4SS_RAM_BANK_6 RAM_BANK_5
32 KB (Bank 6-7 of first 192k chunk)

#define SL_SI91X_POWER_MANAGER_M4SS_RAM_BANK_7 RAM_BANK_6
64 KB (Bank 9-11 of first 192k chunk)

#define SL_SI91X_POWER_MANAGER_M4SS_RAM_BANK_8 RAM_BANK_7
64 KB (Bank 12-15 of first 192k chunk)

#define SL_SI91X_POWER_MANAGER_M4SS_RAM_BANK_9 RAM_BANK_8
64 KB (Bank 1-4 of second 192k chunk)

#define SL_SI91X_POWER_MANAGER_M4SS_RAM_BANK_10 RAM_BANK_9
64 KB (Bank 1-4 of third 192k chunk)

#define SL_SI91X_POWER_MANAGER_ULPSS_RAM_BANK_1 ULPSS_2K_BANK_0
2 KB

#define SL_SI91X_POWER_MANAGER_ULPSS_RAM_BANK_2 ULPSS_2K_BANK_1
2 KB

#define SL_SI91X_POWER_MANAGER_ULPSS_RAM_BANK_3 ULPSS_2K_BANK_2
2 KB

#define SL_SI91X_POWER_MANAGER_ULPSS_RAM_BANK_4 ULPSS_2K_BANK_3
2 KB

Enumeration Documentation

sl_power_state_t

> sl_power_state_t

Enumeration for the power states.

Enumerator	
SL_SI91X_POWER_MANAGER_PS0	PS0 Power State.
SL_SI91X_POWER_MANAGER_PS1	PS1 Power State.
SL_SI91X_POWER_MANAGER_PS2	PS2 Power State.
SL_SI91X_POWER_MANAGER_PS3	PS3 Power State.
SL_SI91X_POWER_MANAGER_PS4	PS4 Power State.
SL_SI91X_POWER_MANAGER_SLEEP	Sleep.
SL_SI91X_POWER_MANAGER_STANDBY	Standby.
LAST_ENUM_POWER_STATE	Last enum for validation.

sl_clock_scaling_t

> sl_clock_scaling_t

Enumeration for clock scaling parameters.

Enumerator	
SL_SI91X_POWER_MANAGER_POWERSAVE	Minimum supported frequency in a power state.
SL_SI91X_POWER_MANAGER_PERFORMANCE	Maximum supported frequency in a power state.
LAST_ENUM_CLOCK_SCALING	Last enum for validation.

sl_si91x_power_manager_on_isr_exit_t

> sl_si91x_power_manager_on_isr_exit_t

On ISR Exit Hook answer.

Enumerator

SL_SI91X_POWER_MANAGER_ISR_IGNORE	The module did not trigger an ISR and it does not want to contribute to the decision.
SL_SI91X_POWER_MANAGER_ISR_SLEEP	The module was the one that caused the system wakeup and the system SHOULD go back to sleep.
SL_SI91X_POWER_MANAGER_ISR_WAKEUP	The module was the one that caused the system wakeup and the system MUST NOT go back to sleep.

Typedef Documentation

sl_power_manager_ps_transition_event_t

```
typedef uint32_t sl_power_manager_ps_transition_event_t
```

Mask of all the event(s) to listen to.

sl_power_manager_ps_transition_on_event_t

```
typedef void(* sl_power_manager_ps_transition_on_event_t) (sl_power_state_t from, sl_power_state_t to)
)(sl_power_state_t from, sl_power_state_t to)
```

Typedef for the user-supplied callback function which is called when a power state transition occurs.

Parameters

Type	Direction	Argument Name	Description
	N/A	from	Power state we are leaving.
	N/A	to	Power state we are entering.

This typedef defines a callback function that is called when a power state transition occurs.

Function Documentation

sl_si91x_power_manager_init

```
sl_status_t sl_si91x_power_manager_init (void )
```

To initialize the Power Manager service.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

The application is configured to start in the PS3 Powersave state with a clock frequency of 40MHz.

Note

- The Power Manager initialization is automatically handled within the sl_service_init() function. Users do not need to manually initialize the Power Manager, ensuring a hassle-free setup process.

- Status code indicating the result:
  - SL_STATUS_OK - Success.
  - SL_STATUS_ALREADY_INITIALIZED - Power Manager is already initialized.

For more information on status codes, see [SL STATUS DOCUMENTATION](

sl_si91x_power_manager_core_entercritical

```
__STATIC_INLINE void sl_si91x_power_manager_core_entercritical (void )
```

To disable the interrupts.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Disables all interrupts by setting PRIMASK. Fault exception handlers will still be enabled.

sl_si91x_power_manager_core_exitcritical

```
__STATIC_INLINE void sl_si91x_power_manager_core_exitcritical (void )
```

To enable the interrupts.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Enables interrupts by clearing PRIMASK.

sl_si91x_power_manager_add_ps_requirement

```
__STATIC_INLINE sl_status_t sl_si91x_power_manager_add_ps_requirement (sl_power_state_t state)
```

To add a requirement on power states.

Parameters

Type	Direction	Argument Name	Description
sl_power_state_t	[in]	state	Power state to add requirement: sl_power_state_t • SL_POWER_MANAGER_PS4 • SL_POWER_MANAGER_PS3 • SL_POWER_MANAGER_PS2 • SL_POWER_MANAGER_PS1

The default state for the Power Manager is PS3. The maximum number of requirements that can be added is 255. The Power Manager switches to the state if it is a valid transition. Before transitioning from one state to another, make sure to remove requirements of previous states if any were added. If an invalid state

requirement is added, it returns SL_STATUS_INVALID_PARAMETER. If the Power Manager service is not initialized, it returns SL_STATUS_NOT_INITIALIZED. To initialize, call sl_si91x_power_manager_init. To get the requirements on all power states, call sl_si91x_power_manager_get_requirement_table. To know the current power state, use sl_si91x_power_manager_get_current_state.

- Pre-conditions:
  - sl_si91x_power_manager_init
  - sl_si91x_power_manager_ps2_pre_check (if PS2 state is added along with WiFi component)

Returns

- Status code indicating the result:
  - SL_STATUS_OK - Success.
  - SL_STATUS_NOT_INITIALIZED - Power Manager is not initialized.
  - SL_STATUS_INVALID_PARAMETER - Invalid parameter is passed.

For more information on status codes, see [SL STATUS DOCUMENTATION](

sl_si91x_power_manager_remove_ps_requirement

```
__STATIC_INLINE sl_status_t sl_si91x_power_manager_remove_ps_requirement (sl_power_state_t state)
```

To remove requirement on power states.

Parameters

Type	Direction	Argument Name	Description
sl_power_state_t	[in]	state	Power state to remove requirement: sl_power_state_t • SL_POWER_MANAGER_PS4 • SL_POWER_MANAGER_PS3 • SL_POWER_MANAGER_PS2 • SL_POWER_MANAGER_PS1

Default state for Power Manager is PS3. Requirements added to a power state must be removed in pairs to ensure proper state transitions. If a requirement is added for a specific power state (e.g., PS4) and not removed, the application will be unable to transition to a lower power state. For instance, if a requirement on PS4 is added but not removed, the Power Manager will remain in PS4 and cannot transition to a lower power state.

Note

- : If the current state is PS4 and no other requirements are added, removing the PS4 requirement will cause the device to remain in its last active state, which in this case is PS4.

- Pre-conditions:
  - sl_si91x_power_manager_init

Returns

- sl_status_t Status code indicating the result:
  - SL_STATUS_OK - Success.
  - SL_STATUS_NOT_INITIALIZED - The Power Manager is not initialized.
  - SL_STATUS_INVALID_PARAMETER - Invalid parameter is passed.

For more information on status codes, see [SL STATUS DOCUMENTATION](

sl_si91x_power_manager_set_clock_scaling

> sl_status_t sl_si91x_power_manager_set_clock_scaling (sl_clock_scaling_t mode)

To configure the clock scaling.

Parameters

Type	Direction	Argument Name	Description
sl_clock_scaling_t	[in]	mode	Clock scaling mode (of type sl_clock_scaling_t).

PS4 and PS3 states are supported only. Possible values for clock scaling are:

- POWERSAVE and PERFORMANCE
- PS4 Performance: 180 MHz clock
- PS4 Power-save: 100 MHz clock
- PS3 Performance: 80 MHz clock
- PS3 Power-save: 40 MHz clock
- For PS2 state, 20 MHz clock is default.

If the Power Manager service is not initialized, it returns SL_STATUS_NOT_INITIALIZED. To initialize, call sl_si91x_power_manager_init.

- Pre-conditions:
  - sl_si91x_power_manager_init

Returns

- Status code indicating the result:
  - SL_STATUS_OK - Success.
  - SL_STATUS_NOT_INITIALIZED - Power Manager is not initialized.
  - SL_STATUS_INVALID_PARAMETER - Invalid parameter is passed.
  - SL_STATUS_INVALID_CONFIGURATION - Invalid configuration of mode.

For more information on status codes, see [SL STATUS DOCUMENTATION](

sl_si91x_power_manager_get_clock_scaling

> sl_clock_scaling_t sl_si91x_power_manager_get_clock_scaling (void)

the clock scaling mode in PS4 and PS3 power state.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Possible return values: * SL_SI91X_POWER_MANAGER_POWERSAVE (Minimum supported frequency in a power state) SL_SI91X_POWER_MANAGER_PERFORMANCE (Maximum supported frequency in a power state)

Returns

- The following values are returned:
  - sl_clock_scaling_t enum value indicating current clock scaling mode

sl_si91x_power_manager_add_peripheral_requirement

> sl_status_t sl_si91x_power_manager_add_peripheral_requirement (sl_power_peripheral_t * peripheral)

Adds the peripheral requirement.

Parameters

Type	Direction	Argument Name	Description
sl_power_peripheral_t *	[in]	peripheral	Structure for different peripherals sl_power_peripheral_t.

Power on the peripherals the valid peripherals passed in the structure. Structure member possible values: sl_power_peripheral_t

- m4ss_peripheral -> Accepts masked value of m4ss peripherals.
- ulpss_peripheral -> Accepts masked value of ulpss peripherals.
- npss_peripheral -> Accepts masked value of npss peripherals. The values of enums can be combined by using 'OR' operator and then passed to the variable.

- Pre-conditions:
  - sl_si91x_power_manager_init

Returns

- Status code indicating the result:
  - SL_STATUS_OK - Success.
  - SL_STATUS_INVALID_STATE (0x0002) - Not a valid transition.
  - SL_STATUS_NOT_INITIALIZED - Power Manager is not initialized.
  - SL_STATUS_INVALID_PARAMETER - Invalid parameter is passed.

Note

- The user must take care of the initialization of the peripherals added.

For more information on status codes, see [SL STATUS DOCUMENTATION](

sl_si91x_power_manager_remove_peripheral_requirement

```
sl_status_t sl_si91x_power_manager_remove_peripheral_requirement (sl_power_peripheral_t * peripheral)
```

To remove the peripheral requirement.

Parameters

Type	Direction	Argument Name	Description
sl_power_peripheral_t *	[in]	peripheral	Structure for different peripherals sl_power_peripheral_t.

Powers off the peripherals specified in the structure. Valid peripherals are passed in the structure sl_power_peripheral_t. The structure members can have the following values:

- m4ss_peripheral -> Accepts masked value of M4SS peripherals.
- ulpss_peripheral -> Accepts masked value of ULPSS peripherals.
- npss_peripheral -> Accepts masked value of NPSS peripherals.

The values of enums can be combined using the 'OR' operator and then passed to the variable.

- Pre-conditions:
  - sl_si91x_power_manager_init

Returns

- Status code indicating the result:
  - SL_STATUS_OK - Success.
  - SL_STATUS_NOT_INITIALIZED - Power Manager is not initialized.
  - SL_STATUS_INVALID_PARAMETER - Invalid parameter is passed.

For more information on status codes, see [SL STATUS DOCUMENTATION](

sl_si91x_power_manager_subscribe_ps_transition_event

```
sl_status_t sl_si91x_power_manager_subscribe_ps_transition_event
(sl_power_manager_ps_transition_event_handle_t * event_handle, const
sl_power_manager_ps_transition_event_info_t * event_info)
```

To register a callback to be called on given power state transition(s).

Parameters

Type	Direction	Argument Name	Description
sl_power_manager_ps_transition_event_handle_t *	[in]	event_handle	Event handle (no initialization needed).
const sl_power_manager_ps_transition_event_info_t *	[in]	event_info	Event info structure that contains the event mask and the callback that must be called.

If the Power Manager service is not initialized, it returns SL_STATUS_NOT_INITIALIZED. To initialize, call sl_si91x_power_manager_init.

- Pre-conditions:
  - sl_si91x_power_manager_init

Returns

- Status code indicating the result:
  - SL_STATUS_OK - Success.
  - SL_STATUS_NOT_INITIALIZED - Power Manager is not initialized.
  - SL_STATUS_NULL_POINTER - Null pointer is passed.

For more information on status codes, see [SL STATUS DOCUMENTATION](

Note

- Adding and removing power state transition requirement(s) from a callback on a transition event is not supported.
- The parameters passed must be persistent, meaning that they need to survive until the callback fires.
- An ASSERT is thrown if the handle is not found.

Usage example:

```
#define PS_EVENT_MASK       ( SL_POWER_MANAGER_EVENT_TRANSITION_ENTERING_PS4 \
                            | SL_POWER_MANAGER_EVENT_TRANSITION_LEAVING_PS4  \
                            | SL_POWER_MANAGER_EVENT_TRANSITION_ENTERING_PS3 \
                            | SL_POWER_MANAGER_EVENT_TRANSITION_LEAVING_PS3  \
                            | SL_POWER_MANAGER_EVENT_TRANSITION_ENTERING_PS2 \
                            | SL_POWER_MANAGER_EVENT_TRANSITION_LEAVING_PS2  \
                            | SL_POWER_MANAGER_EVENT_TRANSITION_ENTERING_PS0 \
                            | SL_POWER_MANAGER_EVENT_TRANSITION_LEAVING_SLEEP)

sl_power_manager_ps_transition_event_handle_t handle;
```

```
sl_power_manager_ps_transition_event_info_t info ={.event_mask = PS_EVENT_MASK,.on_event  = transition_callback };

void transition_callback(sl_power_state_t from, sl_power_state_t to){[...]}

void main(void){
 sl_status_t status;

 status =sl_si91x_power_manager_init();// Validate the status

 status =sl_si91x_power_manager_subscribe_ps_transition_event(&handle,&info);// Validate the status}
```

sl_si91x_power_manager_unsubscribe_ps_transition_event

sl_status_t sl_si91x_power_manager_unsubscribe_ps_transition_event (sl_power_manager_ps_transition_event_handle_t * event_handle, const sl_power_manager_ps_transition_event_info_t * event_info)

To unregister an event callback handle on power state transition.

Parameters

Type	Direction	Argument Name	Description
sl_power_manager_ps_transition_event_handle_t *	[in]	event_handle	Event handle which must be unregistered (must have been registered previously).
const sl_power_manager_ps_transition_event_info_t *	[in]	event_info	Event info structure that contains the event mask and the callback that must be called.

If the Power Manager service is not initialized, it returns SL_STATUS_NOT_INITIALIZED. To initialize, call sl_si91x_power_manager_init.

- Pre-conditions:
  - sl_si91x_power_manager_init

Returns

- Status code indicating the result:
  - SL_STATUS_OK - Success.
  - SL_STATUS_NOT_INITIALIZED - Power Manager is not initialized.
  - SL_STATUS_NULL_POINTER - Null pointer is passed. For more information on status codes, see SL STATUS DOCUMENTATION.

For more information on status codes, refer to SL STATUS DOCUMENTATION. Note

- An ASSERT is thrown if the handle is not found.

sl_si91x_power_manager_sleep

sl_status_t sl_si91x_power_manager_sleep (void)

To move into sleep mode and wait for the peripheral to be set as a wakeup source to trigger and wake up the M4 SoC.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

It supports PS4, PS3, and PS2 only; it cannot enter sleep mode from any other active states. If any error occurs, it returns the error code and does not transition to sleep mode.

Note

- This function expects and calls a callback with the following signature: `boolean_t sl_si91x_power_manager_is_ok_to_sleep(void)` . This function can be used to cancel a sleep action and handle the possible race condition where an ISR that would cause a wakeup is triggered right after the decision to call sl_si91x_power_manager_sleep() has been made.

This function also expects and calls a callback with the following signature: `boolean_t sl_si91x_power_manager_isr_wakeup(void)` after wakeup from sleep. The possible return values are:

- SL_SI91X_POWER_MANAGER_ISR_IGNORE
- SL_SI91X_POWER_MANAGER_ISR_SLEEP
- SL_SI91X_POWER_MANAGER_ISR_WAKEUP

Note

- It can end up in an infinite sleep-wakeup loop if continuously SL_SI91X_POWER_MANAGER_ISR_SLEEP return value is passed.
- Pre-conditions:
  - sl_si91x_power_manager_init
  - sl_si91x_power_manager_configure_ram_retention
  - sl_si91x_power_manager_set_wakeup_sources

Returns

- Status code indicating the result:
  - SL_STATUS_OK - Success.
  - SL_STATUS_NOT_INITIALIZED - Power Manager is not initialized.
  - SL_STATUS_INVALID_PARAMETER - Invalid parameter is passed.
  - SL_STATUS_INVALID_STATE (0x0002) - Not a valid transition.

For more information on status codes, see [SL STATUS DOCUMENTATION](

sl_si91x_power_manager_standby

> void sl_si91x_power_manager_standby (void)

To move into standby state and wait for the interrupt.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- Applications using RTOS with tickless mode enabled must not call this API. This API is not supposed to be used directly in the application and is called automatically when the system is in an idle state with tickless mode.

Standby transition is possible from PS4, PS3, and PS2 states only. Transition from sleep, PS1, or PS0 is not supported.

Pre-conditions:
- sl_si91x_power_manager_init

sl_si91x_power_manager_set_wakeup_sources

```
sl_status_t sl_si91x_power_manager_set_wakeup_sources (uint32_t source, boolean_t add)
```

To configure the wakeup sources.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	source	(uint32_t) Wakeup sources.
boolean_t	[in]	add	(boolean_t) True enables and false disables the wakeup source.

Once the wakeup source is set by the Universal Configurator (UC), the Power Manager automatically handles the initialization, User need to install the appropriate wakeup component based on the configured wakeup source.

- Pre-conditions:
  - sl_si91x_power_manager_init
  - sl_si91x_power_manager_configure_ram_retention
-
  - sl_si91x_power_manager_sleep
  - sl_si91x_power_manager_standby

Returns

- Status code indicating the result:
  - SL_STATUS_OK - Success.
  - SL_STATUS_NOT_INITIALIZED - Power Manager is not initialized.
  - SL_STATUS_INVALID_PARAMETER - Invalid parameter is passed.

For more information on status codes, see [SL STATUS DOCUMENTATION](

sl_si91x_power_manager_configure_ram_retention

```
sl_status_t sl_si91x_power_manager_configure_ram_retention (sl_power_ram_retention_config_t * config)
```

To retain the RAM in low power state either by using size or RAM bank as input parameters.

Parameters

Type	Direction	Argument Name	Description
sl_power_ram_retention_config_t *	[in]	config	Structure for the parameters of RAM retention sl_power_ram_retention_config_t.

Structure member possible values: sl_power_ram_retention_config_t

- configure_ram_banks -> Boolean to switch between RAM Bank retentions. Either by size or by RAM bank number.
  - Enable -> Use RAM Bank Number.
  - Disable -> Use Size.
- m4ss_ram_size_kb -> Retains M4SS RAM banks according to the size.
  - Less than 320 KB (Enter 100 for 100 KB).

ulpss_ram_size_kb -> Retains ULPSS RAM banks according to the size.
  - Less than 8 KB (Enter 5 for 5 KB).
- ram_bank_number -> Retains the M4SS and ULPSS RAM Bank using bank number.

- Pre-conditions:
  - sl_si91x_power_manager_init

- sl_si91x_power_manager_sleep

Returns

- Status code indicating the result:
  - SL_STATUS_OK - Success.
  - SL_STATUS_NOT_INITIALIZED - Power Manager is not initialized.
  - SL_STATUS_NULL_POINTER - Null pointer is passed.

For more information on status codes, see [SL STATUS DOCUMENTATION](

sl_si91x_power_manager_get_current_state

```
sl_power_state_t sl_si91x_power_manager_get_current_state (void )
```

To return the current power state.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Possible return values:

- 2: SL_POWER_MANAGER_PS2, ///< PS2 Power State
- 3: SL_POWER_MANAGER_PS3, ///< PS3 Power State
- 4: SL_POWER_MANAGER_PS4, ///< PS4 Power State

Returns

- sl_power_state_t enum value indicating the current power state.

sl_si91x_power_manager_get_requirement_table

```
uint8_t * sl_si91x_power_manager_get_requirement_table (void )
```

To get the current requirements on all the power states.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

It returns 5 values starting from PS0 to PS4.

- Pre-conditions:
  - sl_si91x_power_manager_init

Returns

- Pointer to a uint8_t type array which contains 5 elements.

Usage example:

```
void main()
{
 uint8_t *requirement_table;
 sl_si91x_power_manager_init();
 requirement_table = sl_si91x_power_manager_get_requirement_table();
 DEBUGOUT("PS4: %d, PS3: %d, PS2: %d, PS1: %d, PS0: %d",
        requirement_table[4],
        requirement_table[3],
        requirement_table[2],
        requirement_table[1],
        requirement_table[0]);
}
```

Retrieve the power state requirement table for the power manager.

This function provides access to the power state requirement table, which is used to manage and track the power state requirements of various components or modules in the system. The table contains information about the power states or dependencies required by different modules.

Note

- The returned pointer points to a table managed internally by the power manager. Ensure that the table is not modified directly to avoid unexpected behavior.

Returns

- Pointer to the power state requirement table.

This API can be used to query the current power state requirements of the system and make decisions based on the power states of various components. For example, it can be used in scenarios where power optimization or power state transitions are required.

sl_si91x_power_manager_request_ps1_state

```
sl_status_t sl_si91x_power_manager_request_ps1_state (void )
```

Requests the PS1 state requirement during tickless idle mode.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function requests the PS1 power state from the power manager during tickless idle mode. If the system is in the IDLE state, it transitions into the PS1 state.

Returns

- Status code indicating the result:
  - SL_STATUS_OK (0x0000) - PS1 state requirement successfully added.
  - SL_STATUS_INVALID_STATE (0x0002) - Invalid request to add PS1 state.
  - SL_STATUS_NOT_INITIALIZED (0x0011) - Power manager service is not initialized.
  - SL_STATUS_INVALID_PARAMETER (0x0021) - Invalid parameter.

sl_si91x_power_manager_remove_ps1_state_request

sl_status_t sl_si91x_power_manager_remove_ps1_state_request (void)

Remove the PS1 state request during tickless idle mode.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function removes the PS1 state request from the power manager during tickless idle mode.

Returns

- Status code indicating the result:
  - SL_STATUS_OK (0x0000) - Ps1 state requirement successfully removed.
  - SL_STATUS_INVALID_STATE (0x0002) - Invalid request to remove PS1 state.
  - SL_STATUS_NOT_INITIALIZED (0x0011) - Power manager service is not initialized.
  - SL_STATUS_INVALID_PARAMETER (0x0021) - Invalid parameter.

sl_si91x_power_manager_get_ps1_state_status

bool sl_si91x_power_manager_get_ps1_state_status (void)

Get the current ps1 state request status from the power manager during tickless idle mode.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function indicates whether a PS1 state request has been added to the power manager during tickless idle mode.

Returns

- Status code indicating the result:
  - true - PS1 state requirement is added.
  - false - PS1 state requirement is not added.

sl_si91x_power_manager_request_standby_state

sl_status_t sl_si91x_power_manager_request_standby_state (void)

Requests the standby state during tickless idle mode.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function requests the standby power state from the power manager during tickless idle mode. If the system is in the IDLE state, it transitions into the standby state.

Returns

Status code indicating the result:
- SL_STATUS_OK (0x0000) - Standby state requirement successfully added.
- SL_STATUS_INVALID_STATE (0x0002) - Invalid request to add standby state.

sl_si91x_power_manager_remove_standby_state_request

```
sl_status_t sl_si91x_power_manager_remove_standby_state_request (void )
```

Remove the standby state request during tickless idle mode.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function removes the standby state request from the power manager during tickless idle mode.

Returns

- Status code indicating the result:
  - SL_STATUS_OK (0x0000) : Standby state requirement successfully removed.
  - SL_STATUS_INVALID_STATE (0x0002) - Invalid request to remove standby state.

sl_si91x_power_manager_get_standby_state_status

```
bool sl_si91x_power_manager_get_standby_state_status (void )
```

Get the current standby state request status during tickless idle mode.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function indicates whether a standby state request has been added to the power manager during tickless idle mode.

Returns

- Status code indicating the result:
  - true : Standby state requirement is added.
  - false : Standby state requirement is not added.

sl_si91x_power_manager_is_tx_command_in_progress

```
bool sl_si91x_power_manager_is_tx_command_in_progress (void )
```

Check if a TX command is currently in progress.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function is used to check the TX command status. It returns true if a TX command is currently in progress; otherwise, it returns false otherwise.

- Pre-conditions:
  - The Wi-Fi component must be added to the project for this function to work properly. If it is not included, this function will always return false.

Returns

- bool - Returns true if TX command is in progress; otherwise, returns false otherwise.

Note

- This function is useful for power management decisions and determining when the system is ready for sleep or other power state transitions.

sl_si91x_power_manager_deinit

```
void sl_si91x_power_manager_deinit (void )
```

To de-initialize the Power Manager service.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

It clears all the power requirements and callback event subscriptions. If the Power Manager service is not initialized, it returns SL_STATUS_NOT_INITIALIZED. To initialize, call sl_si91x_power_manager_init.

- Pre-conditions:
  - sl_si91x_power_manager_init

sl_si91x_power_manager_ps2_pre_check

```
bool sl_si91x_power_manager_ps2_pre_check (void )
```

Performs pre-transition checks before entering PS2 state.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function verifies whether the Network Wireless Processor (NWP) has any pending packet transfers to the M4 core. If pending packets are detected, it returns false to prevent PS2 state transition and avoid data loss. Wi-Fi component must be included in the project for this function to operate correctly.

Returns

- Status code indicating the result:
  - true - Safe to enter PS2 state (no pending packets).
  - false - Pending packets detected, PS2 transition should be blocked.

sl_si91x_power_manager_debug_print_ps_requirements

void sl_si91x_power_manager_debug_print_ps_requirements (void)

To print a table that describes the current requirements on each power state and their owner.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sl_power_ram_retention_config_t

Struture to store configuration parameters for RAM retention.

Public Attributes

uint16_t	m4ss_ram_size_kb M4SS RAM size that needs to be restored.
uint16_t	ulpss_ram_size_kb ULPSS RAM size that needs to be restored.
boolean_t	configure_ram_banks Enable will set the RAM banks using size, disable sets RAM banks using bank number.
uint32_t	m4ss_ram_banks M4SS RAM bank number that needs to be restored.
uint32_t	ulpss_ram_banks ULPSS RAM bank number that needs to be restored.

Public Attribute Documentation

m4ss_ram_size_kb

```
uint16_t sl_power_ram_retention_config_t::m4ss_ram_size_kb
```

M4SS RAM size that needs to be restored.

ulpss_ram_size_kb

```
uint16_t sl_power_ram_retention_config_t::ulpss_ram_size_kb
```

ULPSS RAM size that needs to be restored.

configure_ram_banks

```
boolean_t sl_power_ram_retention_config_t::configure_ram_banks
```

Enable will set the RAM banks using size, disable sets RAM banks using bank number.

m4ss_ram_banks

```
uint32_t sl_power_ram_retention_config_t::m4ss_ram_banks
```

M4SS RAM bank number that needs to be restored.

ulpss_ram_banks

```
uint32_t sl_power_ram_retention_config_t::ulpss_ram_banks
```

ULPSS RAM bank number that needs to be restored.

sl_power_peripheral_t

Structure to store masked values of peripherals to enable/disable it.

Public Attributes

uint32_t	m4ss_peripheral Masked value of M4SS Peripherals.
uint32_t	ulpss_peripheral Masked value of ULPSS Peripherals.
uint32_t	npss_peripheral Masked value of NPSS Peripherals.

Public Attribute Documentation

m4ss_peripheral

```
uint32_t sl_power_peripheral_t::m4ss_peripheral
```

Masked value of M4SS Peripherals.

ulpss_peripheral

```
uint32_t sl_power_peripheral_t::ulpss_peripheral
```

Masked value of ULPSS Peripherals.

npss_peripheral

```
uint32_t sl_power_peripheral_t::npss_peripheral
```

Masked value of NPSS Peripherals.

sl_power_manager_ps_transition_event_info_t

Structure representing power state transition event information.

Public Attributes

const sl_power_manager_ps_transition_event_t	event_mask Mask of the transitions on which the callback should be called.
const sl_power_manager_ps_transition_on_event_t	on_event Function that must be called when the event occurs.

Public Attribute Documentation

event_mask

```
const sl_power_manager_ps_transition_event_t
sl_power_manager_ps_transition_event_info_t::event_mask
```

Mask of the transitions on which the callback should be called.

on_event

```
const sl_power_manager_ps_transition_on_event_t
sl_power_manager_ps_transition_event_info_t::on_event
```

Function that must be called when the event occurs.

sl_power_manager_ps_transition_event_handle_t

Structure representing power state transition event handle.

Public Attributes

sl_slist_node_t	node List node.
sl_power_manager_ps_transition_event_info_t *	info Handle event info.

Public Attribute Documentation

node

```
sl_slist_node_t sl_power_manager_ps_transition_event_handle_t::node
```

List node.

info

```
sl_power_manager_ps_transition_event_info_t* sl_power_manager_ps_transition_event_handle_t::info
```

Handle event info.

Sensor Hub

Sensor Hub

Introduction

- Sensor Hub products are primarily intended to address consumer, industrial, and medical applications that require wireless connectivity, extremely low power, and high performance for sensor fusion algorithms. The MCU core, wireless connectivity, and peripherals are identical to the SI917 CCP Radio board.
- Sensor Hub functions as a sensor management system, facilitating hardware abstraction layer, peripheral drivers, and sensor driver layer integration to collect and distribute desired sensor data.
- It enables users to collect sensor data based on time interval or number of samples or when a specific data requirement are met.
- Sensor Hub serves as a framework that can be seamlessly integrated into any application requiring sensor management.
- The Sensor HUB works with sensors as per configurations provided by the application and notifies the necessary events throughout the application.
- The configuration for the sensors must be provided through the sensor hub config file on which the sensor hub should act.
- Sensor Hub relies on FreeRTOS using CMSIS RTOS version 2 wrapper.

Configuration

- Sensor Hub has 2 Sensor Mode:
- - SL_SH_POLLING_MODE: This mode is Timer based.
- - SL_SH_INTERRUPT_MODE: This mode is NPSS Button 0 GPIO Interrupt based.
- There are 3 types of data_deliver.mode in Polling Sensor Mode and they are - SL_SH_THRESHOLD, - SL_SH_TIMEOUT and - SL_SH_NUM_OF_SAMPLES.
- - SL_SH_NO_DATA data_deliver.mode is used when Interrupt Sensor Mode is selected.
- If the user wants a value that sets a limit or boundary, above which the sensor data should be shown then - SL_SH_THRESHOLD mode should be chosen
- - SL_SH_TIMEOUT mode is used when user wants to get the data at some intervals of time for some sampling time
- - SL_SH_NUM_OF_SAMPLES mode is used when user wants to get the particular number of sensor data
- - SL_SH_NO_DATA mode is used when .
- Here data_deliver.timeout is the time for which the sensor should keep collecting the data for.
- sampling_interval is the the amount of time between two sensor data read is sampled or collected.
- sampling_intr_req_pin is the GPIO pin for sampling the sensor data
- Configure the number of sensors info in the /sensors/inc/sensors_config.h file

```
#define - SL_MAX_NUM_SENSORS          5   // Maximum sensors present in the system
```

- Modes: Using the configuration structure, one can configure the following parameters in the sensorhub_config.c file:
  - For POLLING Sensor Mode configure the below parameters:

```
.sensor_mode          = - SL_SH_POLLING_MODE,
.sampling_interval     = 100,
```

    - If sensor_mode is selected as - SL_SH_POLLING_MODE, then data_deliver.mode should be configured as one of the following for a sensor configuration structure:

For TIMEOUT Data Mode configure the below parameters:

```
.data_deliver.mode        = - SL_SH_TIMEOUT,
.data_deliver.timeout     = 1000,
```

- For THRESHOLD Data Mode configure the below parameters:

```
.data_deliver.mode        = - SL_SH_THRESHOLD,
.data_deliver.threshold   = 1000,
```

- For SAMPLING Data Mode configure the below parameters:

```
.data_deliver.mode        = - SL_SH_NUM_OF_SAMPLES,
.data_deliver.numOfSamples  = 5,
```

- For INTERRUPT Sensor Mode configure the below parameters:

```
.sensor_mode              = - SL_SH_INTERRUPT_MODE,
.sampling_intr_req_pin    = BUTTON_0_GPIO_PIN,
.sensor_intr_type         = - SL_SH_FALL_EDGE,
.data_deliver.data_mode   = - SL_SH_NO_DATA_MODE,
```

- To configure the PS2, please update the below macro in the preprocessor settings:

```
- SL_SENSORHUB_POWERSAVE=1
Enabling this macro will move the application from PS4 state to PS2 state. In PS2 state the sensor data will be sampled and collected.
```

- To configure the power states to PS4 sleep or PS2 Sleep, please update the defines in sisdk/util/third_party/freertos/kernel/include/FreeRTOS.h file as below:

```
#ifndef SL_SI91X_TICKLESS_MODE
#define SL_SI91X_TICKLESS_MODE  1         // 1 is to Enable the tickless Idle mode
#endif

#ifndef configPRE_SLEEP_PROCESSING
#define configPRE_SLEEP_PROCESSING(x) sli_si91x_sleep_wakeup(x)            // Here x is idle time,
#endif

// Configure the sleep time by using the below macro.
// If the number of Ideal task ticks exceeds this value, the system is allowed to sleep.
#ifndef configEXPECTED_IDLE_TIME_BEFORE_SLEEP
  #define configEXPECTED_IDLE_TIME_BEFORE_SLEEP    70
#endif
```

Note:

- By using above sleep configuration, sensor hub is going to sleep by using the idle task and idle time.
- If the Ideal time exceeds the expected sleep time value, the system is allowed to sleep.
- The above idle time is fed to the Alarm timer, which we are using as a wake-up source.

ADC Configurations: Configure only below parameters for ADC to change its mode from FIFO to STATIC and vice versa

- For ADC FIFO mode, configure as shown below:

```
.adc_config.adc_cfg.operation_mode       = - SL_ADC_FIFO_MODE,
.adc_config.adc_ch_cfg.sampling_rate[0]  = - SL_SH_ADC_SAMPLING_RATE, // Use 100 for FIFO Mode
```

- For ADC Static mode, configure as shown below:

```
.adc_config.adc_cfg.operation_mode       = - SL_ADC_STATIC_MODE,
.adc_config.adc_ch_cfg.sampling_rate[0]  = - SL_SH_ADC_SAMPLING_RATE, // Use 1000 for Static Mode
```

- To configure the PS1 power state from PS2 State, please update the below macro in the preprocessor settings:

- SL_SH_ADC_PS1=1
  Enabling this macro will move the core from PS2 Active state to PS1 state

- Please update the defines in sisdk/util/third_party/freertos/kernel/include/FreeRTOS.h file as below:

```
#ifndef configUSE_TICKLESS_IDLE
#define configUSE_TICKLESS_IDLE 1         // 1 is to Enable the tickless Idle mode
#endif

#ifndef configPRE_SLEEP_PROCESSING
#define configPRE_SLEEP_PROCESSING(x) sli_si91x_sleep_wakeup(x)            // Here x is idle time,
#endif
```

Note:

- The PS1 state transition only applies to ADC FIFO Mode. Before entering this mode, kindly turn off any other sensors.

Usage

Sensorhub provides two apis which user can use for intergrating into any application

- sl_si91x_sensorhub_app_task(void);
- sl_si91x_sensor_event_handler(uint8_t sensor_id, uint8_t event);

sl_si91x_sensorhub_app_task(void): initialises and starts sensor data collection. it achieves this by calling the following apis

sl_si91x_sensorhub_notify_cb_register(sl_sensor_signalEvent_t cb_event, sl_sensor_id_t *cb_ack): links the event handler provided by the user as a callback function in the event task

sl_si91x_sensorhub_init(): initializes the peripherals I2C, SPI, ADC, SDC

sl_si91x_sensorhub_detect_sensors(sl_sensor_id_t *sensor_id_info, uint8_t num_of_sensors): scans the i2c sensors provided in the sensorhub_config.c and returns the number of these sensors that are currently connected

sl_si91x_sensorhub_create_sensor(sl_sensor_id_t sensor_id): Initializes the sensor by calling the init function of the sensor provided by the user in the respective HAL, Assigs memory to it based on the data delivery mode, Creates a timer for this sensor if required

sl_si91x_sensor_hub_start(): initializes the sensor task, event task and power task

sl_si91x_sensorhub_start_sensor(sl_sensor_id_t sensor_id): starts the timers of the sensors created in create sensor api

sl_si91x_sensor_event_handler(uint8_t sensor_id, uint8_t event) is called by the event task after sensor data is collected ,in the sensorhub appilcation it prints the data based on the sensor id and uploads it to the cloud

Modules

sl_sensorhub_errors_t

sl_data_deliver_type_t

sl_sensor_info_t

sl_sensor_handle_t

sl_sensor_list_t

sl_intr_list_t

sl_intr_list_map_t

sl_em_event_t

sl_sensor_cb_info_t

sl_i2c_config_t

sl_spi_config_t

sl_adc_config

sl_gpio_config_t

Enumerations

enum sl_sensor_mode_t {

SL_SH_POLLING_MODE
SL_SH_INTERRUPT_MODE
SL_SH_NO_MODE

}
Enumeration for Sensor HUB data reading mode.

enum sl_sensorhub_event_t {

SL_SENSOR_CREATION_FAILED
SL_SENSOR_STARTED
SL_SENSOR_STOPPED
SL_SENSOR_DATA_READY
SL_SENSOR_CNFG_INVALID
SL_SENSOR_START_FAILED
SL_SENSOR_STOP_FAILED
SL_SENSOR_DELETED
SL_SENSOR_DELETE_FAILED

}
Enumeration for Sensor Hub Callback Events.

enum sl_gpio_intr_type_t {

SL_SH_RISE_EDGE
SL_SH_FALL_EDGE
SL_SH_LOW_LEVEL
SL_SH_HIGH_LEVEL

}
Enumeration for GPIO Interrupt Configurations.

enum sl_data_deliver_mode_t {

SL_SH_THRESHOLD
SL_SH_TIMEOUT
SL_SH_NUM_OF_SAMPLES
SL_SH_NO_DATA_MODE

}
Enumeration for Sensor HUB data delivery mode.

enum sl_sensor_status_t {

SL_SENSOR_INVALID
SL_SENSOR_VALID
SL_SENSOR_START
SL_SENSOR_STOP

}
Enumeration for Sensors Status.

Typedefs

typedef struct sl_adc_config	sl_adc_cfg_t ADC bus interface configuration structure.

Functions

sl_status_t	sl_si91x_sensorhub_init() To initialize Peripherals of Sensor HUB.
sl_status_t	sl_si91x_sensorhub_detect_sensors(sl_sensor_id_t sensor_id_info[], uint8_t num_of_sensors) To detect the I2C sensors.
sl_status_t	sl_si91x_sensorhub_delete_sensor(sl_sensor_id_t sensor_id) To delete specific sensor from the Sensor HUB list.
sl_status_t	sl_si91x_sensorhub_create_sensor(sl_sensor_id_t sensor_id) To create a sensor instance in the sensor hub list.
sl_status_t	sl_si91x_sensorhub_start_sensor(sl_sensor_id_t sensor_id) To start sensor operation of the given sensor.
sl_status_t	sl_si91x_sensorhub_stop_sensor(sl_sensor_id_t sensor_id) To stop sensor operation of the given sensor.
void	sl_si91x_em_post_event(sl_sensor_id_t sensor_id, sl_sensorhub_event_t event, void *dataPtr, TickType_t ticks_to_wait) SL_DEPRECATED_API_WISECONNECT_4_0 To post the events to Event Manager (EM) to be notified to the application.
void	sl_si91x_sensor_task(void) SL_DEPRECATED_API_WISECONNECT_4_0 Task to handle the sensor operations.
void	sl_si91x_power_state_task(void) SL_DEPRECATED_API_WISECONNECT_4_0 Task to handle the system power operations.
void	sl_si91x_em_task(void) SL_DEPRECATED_API_WISECONNECT_4_0 Task to handle the operations of the Event Manager (EM).
int32_t	sli_si91x_i2c_init(void) SL_DEPRECATED_API_WISECONNECT_4_0 To initialize the I2C interface based on configuration.
int32_t	sli_si91x_spi_init(void) SL_DEPRECATED_API_WISECONNECT_4_0 To initialize SPI Interface based on configuration.
int32_t	sli_si91x_i2c_sensors_scan(uint16_t address) SL_DEPRECATED_API_WISECONNECT_4_0 To scan the I2C sensors.
sl_sensor_impl_type_t *	sli_si91x_get_sensor_implementation(int32_t sensor_id) SL_DEPRECATED_API_WISECONNECT_4_0 To get sensor implementation.
int32_t	sli_si91x_create_sensor_list_index() SL_DEPRECATED_API_WISECONNECT_4_0 To create sensor list index.
uint32_t	sli_si91x_get_sensor_index(sl_sensor_id_t sensor_id) SL_DEPRECATED_API_WISECONNECT_4_0 To get sensor index for the sensor list.
uint32_t	sli_si91x_delete_sensor_list_index(sl_sensor_id_t sensor_id) SL_DEPRECATED_API_WISECONNECT_4_0 To delete the sensor index for the sensor list.
sl_sensor_info_t *	sli_si91x_get_sensor_info(sl_sensor_id_t sensor_id) SL_DEPRECATED_API_WISECONNECT_4_0 To get sensor info from the sensor configuration structure.
sl_status_t	sl_si91x_sensorhub_notify_cb_register(sl_sensor_signalEvent_t cb_event, sl_sensor_id_t

*cb_ack)
To register callback function to the application.

void sl_si91x_sensors_timer_cb(TimerHandle_t xTimer) SL_DEPRECATED_API_WISECONNECT_4_0
Callback function to set the event flag.

sl_status_t sl_si91x_gpio_interrupt_config(uint16_t gpio_pin, sl_si91x_gpio_interrupt_config_flag_t intr_type) SL_DEPRECATED_API_WISECONNECT_4_0
To configuring different types of NPSS GPIO interrupts in the Sensor HUB.

void sl_si91x_gpio_interrupt_start(uint16_t gpio_pin) SL_DEPRECATED_API_WISECONNECT_4_0
To enable and set the priority of NPSS GPIO interrupt.

void sl_si91x_gpio_interrupt_stop(uint16_t gpio_pin) SL_DEPRECATED_API_WISECONNECT_4_0
To mask and disable the NPSS GPIO interrupt.

sl_status_t sl_si91x_sensor_hub_start(void)
To start the sensor hub tasks.

void sli_si91x_set_alarm_intr_time(uint16_t interval) SL_DEPRECATED_API_WISECONNECT_4_0
To set the alarm interrupt time.

void sli_si91x_init_m4alarm_config(void) SL_DEPRECATED_API_WISECONNECT_4_0
To initialize the Alarm block.

void sli_si91x_config_wakeup_source(uint16_t sleep_time) SL_DEPRECATED_API_WISECONNECT_4_0
To configure wake-up source for the system.

void sli_si91x_sleep_wakeup(uint16_t sh_sleep_time) SL_DEPRECATED_API_WISECONNECT_4_0
To configures sleep/wakeup sources for the system.

void sli_si91x_sensorhub_ps4tops2_state(void) SL_DEPRECATED_API_WISECONNECT_4_0
To change the system status from PS4 to PS2.

void sli_si91x_sensorhub_ps2tops4_state(void) SL_DEPRECATED_API_WISECONNECT_4_0
To change the system status from PS2 to PS4.

sl_status_t sli_si91x_adc_init(void) SL_DEPRECATED_API_WISECONNECT_4_0
To initialize ADC Interface based on configuration.

void vPortSetupTimerInterrupt(void)
To initialize and configure systic timer for the RTOS.

void ARM_I2C_SignalEvent(uint32_t event)
I2C event handler.

sl_adc_cfg_t * sl_si91x_fetch_adc_bus_intf_info(void)
To fetch ADC bus interface information.

void sl_si91x_adc_callback(uint8_t channel_no, uint8_t event) SL_DEPRECATED_API_WISECONNECT_4_0
ADC callback to set event flag.

sl_status_t sli_si91x_sdc_init(void) SL_DEPRECATED_API_WISECONNECT_4_0
To initialize sdc Interface based on the configuration.

Macros

#define SL_SH_SENSOR_TASK_STACK_SIZE (1024 * 2)
Sensor task stack size.

#define SL_SH_EM_TASK_STACK_SIZE (512 * 2)
EM task stack size.

#define SL_SH_POWER_SAVE_TASK_STACK_SIZE (512 * 2)
Power task stack size.

#define SL_EM_TASK_RUN_TICKS osWaitForever
Max wait time of message queue in Event task.

#define MAP_TABLE_SIZE 10
Size of the sensors interrupt MAP table.

#define NPSS_GPIO_IRQHandler IRQ021_Handler
NPSS gpio IRQ handler.

#define NPSS_GPIO_NVIC NPSS_TO_MCU_GPIO_INTR_IRQn
NPSS gpio IRQ Number 21.

#define SL_ALARM_PERIODIC_TIME 10
Periodic alarm configuration in milliseconds.

#define RC_TRIGGER_TIME 5
RC clock trigger time.

#define RO_TRIGGER_TIME 0
RO clock trigger time.

#define NO_OF_HOURS_IN_A_DAY 24
Number of hours in a day.

#define NO_OF_MINUTES_IN_AN_HOUR 60
Number of minutes in an hour.

#define NO_OF_SECONDS_IN_A_MINUTE 60
Number of Seconds in a minute.

#define NO_OF_MILLISECONDS_IN_A_SECOND 1000
Number of milliseconds in a second.

#define NO_OF_MONTHS_IN_A_YEAR 12
Number of months in a year.

#define BASE_YEAR 2000
Start year for alarm configuration.

#define NO_OF_DAYS_IN_A_MONTH_1 28
Month with 28 days.

#define NO_OF_DAYS_IN_A_MONTH_2 29
Month with 29 days.

#define NO_OF_DAYS_IN_A_MONTH_3 30
Month with 30 days.

#define NO_OF_DAYS_IN_A_MONTH_4 31
Month with 31 days.

#define RTC_ALARM_IRQHandler IRQ028_Handler
Alarm IRQ handler.

#define NVIC_RTC_ALARM MCU_CAL_ALARM_IRQn
Alarm IRQ number 28.

#define SL_SH_TIMER_CREATION_FAILED (1 << SL_SH_SENSORHUB_ERRORS_MASK)

Creating OS timer for the sensor failed.

#define SL_SH_TIMER_DELETION_FAILED (2 << SL_SH_SENSORHUB_ERRORS_MASK)
Deleting OS timer of the sensor failed.

#define SL_SH_TIMER_START_FAIL (3 << SL_SH_SENSORHUB_ERRORS_MASK)
Starting OS timer of the sensor failed.

#define SL_SH_TIMER_STOP_FAIL (4 << SL_SH_SENSORHUB_ERRORS_MASK)
Stopping OS timer of the sensor failed.

#define SL_SH_MAX_SENSORS_REACHED (5 << SL_SH_SENSORHUB_ERRORS_MASK)
Max number of sensor limit reached.

#define SL_SH_MEMORY_LIMIT_EXCEEDED (6 << SL_SH_SENSORHUB_ERRORS_MASK)
Memory allocation for sensor data storage failed.

#define SL_SH_SENSOR_CREATE_FAIL (7 << SL_SH_SENSORHUB_ERRORS_MASK)
Sensor API is called for a sensor without creating it.

#define SL_SH_COMMAND_SET_POWER_FAIL (8 << SL_SH_SENSORHUB_ERRORS_MASK)
Sensor set power command execution failed.

#define SL_SH_COMMAND_SET_RANGE_FAIL (9 << SL_SH_SENSORHUB_ERRORS_MASK)
Sensor set range command execution failed.

#define SL_SH_COMMAND_SELF_TEST_FAIL (10 << SL_SH_SENSORHUB_ERRORS_MASK)
Sensor self-test command execution failed.

#define SL_SH_INVALID_PARAMETERS (11 << SL_SH_SENSORHUB_ERRORS_MASK)
Generic error code for any invalid parameters.

#define SL_SH_GPIO_OUT_OF_RANGE (12 << SL_SH_SENSORHUB_ERRORS_MASK)
Invalid gpio number.

#define SL_SH_SENSOR_IMPLEMENTATION_NOT_FOUND (13 << SL_SH_SENSORHUB_ERRORS_MASK)
No hal implementation found for given sensor type.

#define SL_SH_INTERRUPT_TYPE_CONFIG_FAIL (14 << SL_SH_SENSORHUB_ERRORS_MASK)
Interrupt type configuration failed.

#define SL_SH_MEMORY_ALLOCATION_FAILED (15 << SL_SH_SENSORHUB_ERRORS_MASK)
Allocating memory for sensor hal failed.

#define SL_SH_SENSOR_INDEX_NOT_FOUND (0xFF)
Sensor not created.

#define SL_SH_CONFIG_NOT_FOUND (16 << SL_SH_SENSORHUB_ERRORS_MASK)
Sensor configuration not found.

#define SL_SH_INVALID_ADDRESS (17 << SL_SH_SENSORHUB_ERRORS_MASK)
No sensor found at given address.

#define SL_SH_WRONG_INTERRUPT_TYPE (18 << SL_SH_SENSORHUB_ERRORS_MASK)
Invalid interrupt type is given for the sensor.

#define SL_SH_INVALID_MODE (19 << SL_SH_SENSORHUB_ERRORS_MASK)
Invalid mode is given for the sensor.

#define SL_SH_INVALID_DELIVERY_MODE (20 << SL_SH_SENSORHUB_ERRORS_MASK)
Invalid delivery mode is given for the sensor.

#define SL_SH_HAL_SENSOR_CREATION_FAILED (21 << SL_SH_SENSORHUB_ERRORS_MASK)

Create sensor in HAL failed.

#define SL_SH_HAL_SENSOR_DELETION_FAILED (22 << SL_SH_SENSORHUB_ERRORS_MASK)
Delete sensor in HAL failed.

#define SL_SH_HAL_SENSOR_SAMPLE_FAIL (23 << SL_SH_SENSORHUB_ERRORS_MASK)
Sample sensor in HAL failed.

#define SL_SH_HAL_SENSOR_CONTROL_FAIL (24 << SL_SH_SENSORHUB_ERRORS_MASK)
Control sensor in HAL failed.

#define SL_SH_POWER_TASK_CREATION_FAILED (25 << SL_SH_SENSORHUB_ERRORS_MASK)
Power task creation failed.

#define SL_SH_SENSOR_TASK_CREATION_FAILED (26 << SL_SH_SENSORHUB_ERRORS_MASK)
Sensor task creation failed.

#define SL_SH_EM_TASK_CREATION_FAILED (27 << SL_SH_SENSORHUB_ERRORS_MASK)
EM task creation failed.

#define SL_ALL_PERIPHERALS_INIT_FAILED (28 << SL_SH_SENSORHUB_ERRORS_MASK)
All(i2c,spi,adc) peripheral's initialization failed.

Enumeration Documentation

sl_sensor_mode_t

> sl_sensor_mode_t

Enumeration for Sensor HUB data reading mode.

Enumerator	
SL_SH_POLLING_MODE	POLLING_MODE.
SL_SH_INTERRUPT_MODE	INTERRUPT_MODE.
SL_SH_NO_MODE	NO_MODE.

sl_sensorhub_event_t

> sl_sensorhub_event_t

Enumeration for Sensor Hub Callback Events.

This enumeration defines various events related to the operation of sensors within a Sensor Hub. The events represent different states and actions of the sensors, such as creation, starting, stopping, and data readiness.

Enumerator	
SL_SENSOR_CREATION_FAILED	Event indicating that sensor creation failed.
SL_SENSOR_STARTED	Event indicating that the sensor has started successfully.
SL_SENSOR_STOPPED	Event indicating that the sensor has stopped.
SL_SENSOR_DATA_READY	Event indicating that sensor data is ready to be read.
SL_SENSOR_CNFG_INVALID	Event indicating that the sensor configuration is invalid.
SL_SENSOR_START_FAILED	Event indicating that the sensor failed to start.
SL_SENSOR_STOP_FAILED	Event indicating that the sensor failed to stop.

SL_SENSOR_DELETED	Event indicating that the sensor has been deleted.
SL_SENSOR_DELETE_FAILED	Event indicating that the sensor deletion failed.

sl_gpio_intr_type_t

```
sl_gpio_intr_type_t
```

Enumeration for GPIO Interrupt Configurations.

Enumerator	
SL_SH_RISE_EDGE	Interrupt at GPIO rise edge.
SL_SH_FALL_EDGE	Interrupt at GPIO fall edge.
SL_SH_LOW_LEVEL	Interrupt at GPIO low level.
SL_SH_HIGH_LEVEL	Interrupt at GPIO high level.

sl_data_deliver_mode_t

```
sl_data_deliver_mode_t
```

Enumeration for Sensor HUB data delivery mode.

Enumerator	
SL_SH_THRESHOLD	Threshold value for sensor data delivery.
SL_SH_TIMEOUT	Timeout value for sensor data delivery.
SL_SH_NUM_OF_SAMPLES	Number of samples for sensor data delivery.
SL_SH_NO_DATA_MODE	Intterupt mode data delivery.

sl_sensor_status_t

```
sl_sensor_status_t
```

Enumeration for Sensors Status.

Enumerator	
SL_SENSOR_INVALID	Sensor is Invalid.
SL_SENSOR_VALID	Sensor is Valid.
SL_SENSOR_START	Sensor has Started.
SL_SENSOR_STOP	Sensor has Stoped.

Typedef Documentation

sl_adc_cfg_t

```
typedef struct sl_adc_config sl_adc_cfg_t
```

ADC bus interface configuration structure.

Function Documentation

sl_si91x_sensorhub_init

```
sl_status_t sl_si91x_sensorhub_init ()
```

To initialize Peripherals of Sensor HUB.

This function will initialize the Peripherals like I2C/SPI/ADC, based on the user configuration.

Returns

- Status code indicating the result:
  - SL_STATUS_OK - Success, peripherals initialization was done properly.
  - SL_STATUS_FAIL - Failed, peripherals initialization failed.
  - SL_ALL_PERIPHERALS_INIT_FAILED , All peripherals initializaion failed.

For more information on status codes, see SL STATUS DOCUMENTATION.

sl_si91x_sensorhub_detect_sensors

```
sl_status_t sl_si91x_sensorhub_detect_sensors (sl_sensor_id_t sensor_id_info, uint8_t num_of_sensors)
```

To detect the I2C sensors.

Parameters

Type	Direction	Argument Name	Description
sl_sensor_id_t	[in]	sensor_id_info	- Sensor IDs.
uint8_t	[in]	num_of_sensors	- Number of sensors given by user.

This function detects I2C sensors according to user configuration and stores the scanned sensor IDs in the provided structure. Returns

- Number of sensors scanned, if successful.
  - SL_STATUS_FAIL - No sensors found. For more information on status codes, see SL STATUS DOCUMENTATION.

sl_si91x_sensorhub_delete_sensor

```
sl_status_t sl_si91x_sensorhub_delete_sensor (sl_sensor_id_t sensor_id)
```

To delete specific sensor from the Sensor HUB list.

Parameters

Type	Direction	Argument Name	Description
sl_sensor_id_t	[in]	sensor_id	- Sensor ID.

This function will delete the specific sensor from the sensor list, modify the sensor status to invalid, and publish the events to the event task.

Returns

- Status code indicating the result:
  - SL_STATUS_OK - Success, delete sensor success.
  - SL_SH_TIMER_DELETION_FAILED - Timer deletion failed.
  - SL_SH_COMMAND_SET_POWER_FAIL - Set power failed.
  - SL_SH_HAL_SENSOR_DELETION_FAILED - Sensor deletion failed at HAL layer.
  - SL_SH_SENSOR_INDEX_NOT_FOUND - Given sensor index not found.

For more information on status codes, see SL STATUS DOCUMENTATION.

sl_si91x_sensorhub_create_sensor

```
sl_status_t sl_si91x_sensorhub_create_sensor (sl_sensor_id_t sensor_id)
```

To create a sensor instance in the sensor hub list.

Parameters

Type	Direction	Argument Name	Description
sl_sensor_id_t	[in]	sensor_id	- Sensor ID.

This function creates a sensor instance as per user configuration. It also allocates max sample memory for the configured sensor.

Returns

- Status code indicating the result:
  - SL_STATUS_OK - Create sensor instance success.
  - SL_SH_TIMER_CREATION_FAILED - Timer creation for the sensor failed.
  - SL_SH_MAX_SENSORS_REACHED - Maximum number of sensors reached.
  - SL_SH_MEMORY_LIMIT_EXCEEDED - Memory limit exceeded.
  - SL_SH_COMMAND_SET_POWER_FAIL - Command to set power failed.
  - SL_SH_COMMAND_SET_RANGE_FAIL - Command to set range failed.
  - SL_SH_COMMAND_SELF_TEST_FAIL - Command for the self test failed.
  - SL_SH_SENSOR_IMPLEMENTATION_NOT_FOUND - Implementation of the sensor not found.
  - SL_SH_CONFIG_NOT_FOUND - Configuration of the sensor not found.
  - SL_SH_INVALID_MODE - Invalid mode.
  - SL_SH_HAL_SENSOR_CREATION_FAILED - Sensor creation failed at HAL. For more information on status codes, see SL STATUS DOCUMENTATION.

sl_si91x_sensorhub_start_sensor

```
sl_status_t sl_si91x_sensorhub_start_sensor (sl_sensor_id_t sensor_id)
```

To start sensor operation of the given sensor.

Parameters

Type	Direction	Argument Name	Description
sl_sensor_id_t	[in]	sensor_id	- Sensor ID.

This function can be called after creating a sensor with given sensor ID It performs the following operations:

- Starts timer for sensor with polling mode.

Enable IRQ handle for sensor with interrupt mode.

Returns

- Status code indicating the result:
  - SL_STATUS_OK - Success, sensor started.
  - SL_SH_TIMER_START_FAIL - Failed to start timer.
  - SL_SH_SENSOR_CREATE_FAIL - Given sensor not created.
  - SL_SH_INVALID_MODE - Invaild mode given.

For more information on status codes, see SL STATUS DOCUMENTATION.

sl_si91x_sensorhub_stop_sensor

```
sl_status_t sl_si91x_sensorhub_stop_sensor (sl_sensor_id_t sensor_id)
```

To stop sensor operation of the given sensor.

Parameters

Type	Direction	Argument Name	Description
sl_sensor_id_t	[in]	sensor_id	- Sensor ID.

This function performs the following operations:

- Stop the sensor operations of the given sensor.
- Based on the sensor mode it will stop the polling/interrupt mode operations and updates the sensor statutes.
- Disable IRQ handles of the interrupt mode sensors.

Returns

- Status code indicating the result:
  - SL_STATUS_OK - Success, sensor stopped.
  - SL_SH_TIMER_STOP_FAIL - Failed to stop timer.
  - SL_SH_SENSOR_CREATE_FAIL - Given sensor not created.

For more information on status codes, see SL STATUS DOCUMENTATION.

sl_si91x_em_post_event

```
void sl_si91x_em_post_event (sl_sensor_id_t sensor_id, sl_sensorhub_event_t event, void * dataPtr,
TickType_t ticks_to_wait)
```

To post the events to Event Manager (EM) to be notified to the application.

Parameters

Type	Direction	Argument Name	Description
sl_sensor_id_t	[in]	sensor_id	- Sensor ID.
sl_sensorhub_event_t	[in]	event	- Sensor hub events.
void *	[in]	dataPtr	- Sensor data pointer.
TickType_t	[in]	ticks_to_wait	- Max time to wait for the post.

It waits for mutex till ticks_to_wait and updates event queue if mutex is acquired

sl_si91x_sensor_task

```
void sl_si91x_sensor_task (void )
```

Task to handle the sensor operations.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Sensor Task performs the following operations:

- Create the OS event and mutex to perform the sensor operations.
- Sample the sensor data based on the event flags.
- Check the events and post the sensor data to the EM task based on the sensor data delivery mode.

sl_si91x_power_state_task

```
void sl_si91x_power_state_task (void )
```

Task to handle the system power operations.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Power state task changes the system from one power save mode to another power save mode like(PS4 to PS2),(PS2toPS4),(Sleep_mode) using Binary semaphore.

sl_si91x_em_task

```
void sl_si91x_em_task (void )
```

Task to handle the operations of the Event Manager (EM).

Parameters

Type	Direction	Argument Name	Description
void	N/A		

EM task performs the following operations:

- Create the osMessageQueue and mutex to perform the event operations.
- Calls the callback event.

sli_si91x_i2c_init

```
int32_t sli_si91x_i2c_init (void )
```

To initialize the I2C interface based on configuration.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function will configure/Initialize I2C Interface based on configuration.

Returns

- I2C Initializtion status

sli_si91x_spi_init

```
int32_t sli_si91x_spi_init (void )
```

To initialize SPI Interface based on configuration.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function will configure/Initialize SPI Interface based on configuration.

Returns

- SPI Initializtion status.

sli_si91x_i2c_sensors_scan

```
int32_t sli_si91x_i2c_sensors_scan (uint16_t address)
```

To scan the I2C sensors.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	address	- Address of sensor

This function will scan the I2C sensors based on the sensor address.

Returns

- Status 0 if successful; otherwise, it will wait for the sensor response.

sli_si91x_get_sensor_implementation

```
sl_sensor_impl_type_t * sli_si91x_get_sensor_implementation (int32_t sensor_id)
```

To get sensor implementation.

Parameters

Type	Direction	Argument Name	Description
Type	Direction	Argument Name	Description
int32_t	[in]	sensor_id	- Sensor ID.

This function will get the sensor implementation of the sensor based on sensor ID.

Returns

- If successful, it returns the Sensor implementation structure. Otherwise, it returns NULL.

sli_si91x_create_sensor_list_index

```
int32_t sli_si91x_create_sensor_list_index ()
```

To create sensor list index.

This function will create the sensor list index based on the sensor status For the maximum sensors available in the sensor hub

Returns

- Returns the sensor index in sensor_list.

sli_si91x_get_sensor_index

```
uint32_t sli_si91x_get_sensor_index (sl_sensor_id_t sensor_id)
```

To get sensor index for the sensor list.

Parameters

Type	Direction	Argument Name	Description
sl_sensor_id_t	[in]	sensor_id	- Sensor ID.

This function will retrieve the sensor index from the sensor list based on the sensor status and sensor ID for the maximum number of sensors available in the sensor hub.

Returns

- If successful, it returns the sensor index. Otherwise, it returns - SL_SH_SENSOR_INDEX_NOT_FOUND.

sli_si91x_delete_sensor_list_index

```
uint32_t sli_si91x_delete_sensor_list_index (sl_sensor_id_t sensor_id)
```

To delete the sensor index for the sensor list.

Parameters

Type	Direction	Argument Name	Description
sl_sensor_id_t	[in]	sensor_id	- Sensor ID.

This function will delete the sensor from the sensor list based on the sensor ID for the maximum number of sensors available in the sensor hub.

Returns

- If successful, it returns the sensor index. Otherwise, it returns error code - SL_SH_SENSOR_INDEX_NOT_FOUND.

sli_si91x_get_sensor_info

```
sl_sensor_info_t * sli_si91x_get_sensor_info (sl_sensor_id_t sensor_id)
```

To get sensor info from the sensor configuration structure.

Parameters

Type	Direction	Argument Name	Description
sl_sensor_id_t	[in]	sensor_id	- Sensor ID.

This function will retrieve sensor data from the sensor configuration structure to update the sensor list configuration structure for the maximum number of sensors available in the sensor hub.

Returns

- If successful, it returns the sensor information. Otherwise, it returns NULL.

sl_si91x_sensorhub_notify_cb_register

```
sl_status_t sl_si91x_sensorhub_notify_cb_register (sl_sensor_signalEvent_t cb_event, sl_sensor_id_t * cb_ack)
```

To register callback function to the application.

Parameters

Type	Direction	Argument Name	Description
sl_sensor_signalEvent_t	[in]	cb_event	- Pointer to the callback event.
sl_sensor_id_t *	[in]	cb_ack	- Pointer to callback acknowledge to the application.

This function will update the callback function event and acknowledgment.

Returns

- Status code indicating the result:
  - SL_STATUS_OK - Success, callback registered.
  - SL_SH_INVALID_PARAMETERS - Invalid parameters.

For more information on status codes, see SL STATUS DOCUMENTATION.

sl_si91x_sensors_timer_cb

```
void sl_si91x_sensors_timer_cb (TimerHandle_t xTimer)
```

Callback function to set the event flag.

Parameters

Type	Direction	Argument Name	Description
TimerHandle_t	[in]	xTimer	- Timer handle.

This function will set the event flag bits based on the sensor sampling intervals. In the polling mode call the timer call-back function.

sl_si91x_gpio_interrupt_config

```
sl_status_t sl_si91x_gpio_interrupt_config (uint16_t gpio_pin, sl_si91x_gpio_interrupt_config_flag_t intr_type)
```

To configuring different types of NPSS GPIO interrupts in the Sensor HUB.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	gpio_pin	- NPSS GPIO pin number.
sl_si91x_gpio_interrupt_config_flag_t	[in]	intr_type	- NPSS GPIO interrupt type.

This function configures the NPSS GPIOs and enables the interrupt mode for the gpios.

Returns

- Status code indicating the result:
  - SL_STATUS_OK - Success, interrupt configured.
  - SL_SH_INTERRUPT_TYPE_CONFIG_FAIL - Invaild interrupt type.

For more information on status codes, see SL STATUS DOCUMENTATION.

sl_si91x_gpio_interrupt_start

```
void sl_si91x_gpio_interrupt_start (uint16_t gpio_pin)
```

To enable and set the priority of NPSS GPIO interrupt.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	gpio_pin	- NPSS GPIO pin number.

This function configures and sets the priority of the NPSS GPIO interrupt. GPIO interrupt priority is (configMAX_SYSCALL_INTERRUPT_PRIORITY - 1)

sl_si91x_gpio_interrupt_stop

```
void sl_si91x_gpio_interrupt_stop (uint16_t gpio_pin)
```

To mask and disable the NPSS GPIO interrupt.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	gpio_pin	- NPSS GPIO pin number.

This function masks and disables the IRQ handler of the NPSS GPIO interrupt.

sl_si91x_sensor_hub_start

```
sl_status_t sl_si91x_sensor_hub_start (void )
```

To start the sensor hub tasks.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This Starts the sensor hub Tasks.

Returns

- Status code indicating the result:
  - SL_STATUS_OK - Success, sensorhub started.
  - SL_SH_POWER_TASK_CREATION_FAILED - Failed to create power task.
  - SL_SH_SENSOR_TASK_CREATION_FAILED - Failed to create sensor task.
  - SL_SH_EM_TASK_CREATION_FAILED - Failed to create EM task.

For more information on status codes, see SL STATUS DOCUMENTATION.

sli_si91x_set_alarm_intr_time

```
void sli_si91x_set_alarm_intr_time (uint16_t interval)
```

To set the alarm interrupt time.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	interval	- interval time

This function will set the alarm interrupt based on the periodic time.

sli_si91x_init_m4alarm_config

```
void sli_si91x_init_m4alarm_config (void )
```

To initialize the Alarm block.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function will initialize the Alarm block.

sli_si91x_config_wakeup_source

```
void sli_si91x_config_wakeup_source (uint16_t sleep_time)
```

To configure wake-up source for the system.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	sleep_time	- Sleep time for the sensor hub.

This function will configure the wake-up source to the system.

sli_si91x_sleep_wakeup

```
void sli_si91x_sleep_wakeup (uint16_t sh_sleep_time)
```

To configures sleep/wakeup sources for the system.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	sh_sleep_time	- Sleep time for the sensor hub, in ADC PS-1 no parameters.

This function will configure sleep/wakeup sources.

sli_si91x_sensorhub_ps4tops2_state

```
void sli_si91x_sensorhub_ps4tops2_state (void )
```

To change the system status from PS4 to PS2.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function will change the system status from PS4 to PS2.

sli_si91x_sensorhub_ps2tops4_state

```
void sli_si91x_sensorhub_ps2tops4_state (void )
```

To change the system status from PS2 to PS4.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function will change the system status from PS2 to PS4.

sli_si91x_adc_init

```
sl_status_t sli_si91x_adc_init (void )
```

To initialize ADC Interface based on configuration.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function will configure/Initialize ADC Interface based on configuration.

Returns

- Status code indicating the result:
  - SL_STATUS_OK - Success, ADC initialized.
  - SL_STATUS_FAIL - Failed to initialize ADC .

For more information on status codes, see SL STATUS DOCUMENTATION.

vPortSetupTimerInterrupt

```
void vPortSetupTimerInterrupt (void )
```

To initialize and configure systic timer for the RTOS.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Set up the systic timer to generate the tick interrupts at the required frequency.

ARM_I2C_SignalEvent

```
void ARM_I2C_SignalEvent (uint32_t event)
```

I2C event handler.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	event	- I2C transmit and receive events.

> sl_adc_cfg_t * sl_si91x_fetch_adc_bus_intf_info (void)

To fetch ADC bus interface information.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This can be used by lower level layers. Returns

- Pointer to ADC configuration structure.

sl_si91x_adc_callback

> void sl_si91x_adc_callback (uint8_t channel_no, uint8_t event)

ADC callback to set event flag.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	channel_no	- Respective channel number.
uint8_t	[in]	event	- Callback event (ADC_STATIC_MODE_CALLBACK, ADC_THRSHOLD_CALLBACK, INTERNAL_DMA, FIFO_MODE_EVENT).

This callback function is called when ADC event occured and it sets event flag corresponding to that event

sli_si91x_sdc_init

> sl_status_t sli_si91x_sdc_init (void)

To initialize sdc Interface based on the configuration.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Returns status 0 if successful. Otherwise, it returns an error code.
  - SL_STATUS_FAIL - Fail.
  - SL_STATUS_OK - Success.

For more information on status codes, see SL STATUS DOCUMENTATION.

sl_sensorhub_errors_t

Used to Monitor the bus errors in the Sensor HUB.

Public Attributes

bool	i2c	I2C bus error status.
bool	spi	SPI bus error status.
bool	adc	ADC bus error status.
bool	sdc	SDC bus error status.
bool	peripheral_global_status	All buses error status.

Public Attribute Documentation

i2c

```
bool sl_sensorhub_errors_t::i2c
```

I2C bus error status.

spi

```
bool sl_sensorhub_errors_t::spi
```

SPI bus error status.

adc

```
bool sl_sensorhub_errors_t::adc
```

ADC bus error status.

sdc

```
bool sl_sensorhub_errors_t::sdc
```

SDC bus error status.

peripheral_global_status

```
bool sl_sensorhub_errors_t::peripheral_global_status
```

All buses error status.

sl_data_deliver_type_t

Sensor data delivery modes in Sensor HUB.

Public Attributes

sl_data_deliver_mode_t	data_mode Enumeration for Sensor HUB data delivery mode.
uint32_t	threshold Threshold value for the sensor.
uint32_t	timeout Timeout values for the sensor.
uint32_t	numofsamples Number of samples for the sensor.
union sl_data_deliver_type_t::@0	@1

Public Attribute Documentation

data_mode

```
sl_data_deliver_mode_t sl_data_deliver_type_t::data_mode
```

Enumeration for Sensor HUB data delivery mode.

threshold

```
uint32_t sl_data_deliver_type_t::threshold
```

Threshold value for the sensor.

timeout

```
uint32_t sl_data_deliver_type_t::timeout
```

Timeout values for the sensor.

numofsamples

```
uint32_t sl_data_deliver_type_t::numofsamples
```

Number of samples for the sensor.

@1

union sl_data_deliver_type_t::@0 sl_data_deliver_type_t::@1

sl_sensor_info_t

Structure for storing the Sensor Configurations from the User.

Public Attributes

char *	sensor_name Name of the sensor.
int16_t	sensor_intr_type GPIO Interrupt Configurations.
uint16_t	sampling_intr_req_pin GPIO pin for sampling the sensor data.
uint32_t	sampling_interval Sensor data sampling interval.
uint8_t	address Address of sensor.
uint16_t	channel Channel for adc.
union sl_sensor_info_t::@2	@3
sl_sensor_id_t	sensor_id Sensor id.
sl_sensor_bus_t	sensor_bus Protocol for the sensor(spi/i2c/adc)
sl_sensor_mode_t	sensor_mode Sensor Mode(Enumeration)
sl_sensor_range_t	sensor_range Range of sensor.
sl_data_deliver_type_t	data_deliver Data delivery mode for the sensor.
sl_sensor_data_group_t *	sensor_data_ptr Pointer to Sensor data storage structure.

Public Attribute Documentation

sensor_name

```
char* sl_sensor_info_t::sensor_name
```

Name of the sensor.

sensor_intr_type

```
int16_t sl_sensor_info_t::sensor_intr_type
```

GPIO Interrupt Configurations.

sampling_intr_req_pin

```
uint16_t sl_sensor_info_t::sampling_intr_req_pin
```

GPIO pin for sampling the sensor data.

sampling_interval

```
uint32_t sl_sensor_info_t::sampling_interval
```

Sensor data sampling interval.

address

```
uint8_t sl_sensor_info_t::address
```

Address of sensor.

channel

```
uint16_t sl_sensor_info_t::channel
```

Channel for adc.

@3

```
union sl_sensor_info_t::@2 sl_sensor_info_t::@3
```

sensor_id

```
sl_sensor_id_t sl_sensor_info_t::sensor_id
```

Sensor id.

sensor_bus

```
sl_sensor_bus_t sl_sensor_info_t::sensor_bus
```

Protocol for the sensor(spi/i2c/adc)

sensor_mode

```
sl_sensor_mode_t sl_sensor_info_t::sensor_mode
```

Sensor Mode(Enumeration)

sensor_range

```
sl_sensor_range_t sl_sensor_info_t::sensor_range
```

Range of sensor.

data_deliver

```
sl_data_deliver_type_t sl_sensor_info_t::data_deliver
```

Data delivery mode for the sensor.

sensor_data_ptr

```
sl_sensor_data_group_t* sl_sensor_info_t::sensor_data_ptr
```

Pointer to Sensor data storage structure.

sl_sensor_handle_t

Structure to monitor and handle the sensors.

Public Attributes

void *	sensor_handle Sensor handle.
void *	ctrl_handle Sensor control handle.
uint8_t	sensor_event_bit Sensor event bits.
uint8_t	event_ack Sensor event acknowledge.
uint16_t	max_samples Maximum number of samples for the sensors.
sl_sensor_info_t *	config_st Sensor configuration structure.
sl_sensor_impl_type_t *	sensor_impl Sensor implementation structure.
sl_sensor_status_t	sensor_status Sensor status.
TimerHandle_t	timer_handle RTOS timer handle.

Public Attribute Documentation

sensor_handle

```
void* sl_sensor_handle_t::sensor_handle
```

Sensor handle.

ctrl_handle

```
void* sl_sensor_handle_t::ctrl_handle
```

Sensor control handle.

sensor_event_bit

```
uint8_t sl_sensor_handle_t::sensor_event_bit
```

Sensor event bits.

event_ack

```
uint8_t sl_sensor_handle_t::event_ack
```

Sensor event acknowledge.

max_samples

```
uint16_t sl_sensor_handle_t::max_samples
```

Maximum number of samples for the sensors.

config_st

```
sl_sensor_info_t* sl_sensor_handle_t::config_st
```

Sensor configuration structure.

sensor_impl

```
sl_sensor_impl_type_t* sl_sensor_handle_t::sensor_impl
```

Sensor implementation structure.

sensor_status

```
sl_sensor_status_t sl_sensor_handle_t::sensor_status
```

Sensor status.

timer_handle

```
TimerHandle_t sl_sensor_handle_t::timer_handle
```

RTOS timer handle.

sl_sensor_list_t

Structure that maintains the Sensors list in Polling mode.

Public Attributes

uint8_t	sensor_index Sensor index.
sl_sensor_handle_t	sl_sensors_st Sensor handle structure.

Public Attribute Documentation

sensor_index

```
uint8_t sl_sensor_list_t::sensor_index
```

Sensor index.

sl_sensors_st

```
sl_sensor_handle_t sl_sensor_list_t::sl_sensors_st[SL_MAX_NUM_SENSORS]
```

Sensor handle structure.

sl_intr_list_t

Strutured used to keep track of the interrupt number and interrupt sensor index.

Public Attributes

uint8_t	sensor_list_index Interrupt mode sensor index.
uint16_t	intr Interrupt GPIO Pin.
uint16_t	adc_intr_channel Channel number for ADC interrupt.
uint16_t	sdc_intr_channel Channel number for SDC interrupt.

Public Attribute Documentation

sensor_list_index

```
uint8_t sl_intr_list_t::sensor_list_index
```

Interrupt mode sensor index.

intr

```
uint16_t sl_intr_list_t::intr
```

Interrupt GPIO Pin.

adc_intr_channel

```
uint16_t sl_intr_list_t::adc_intr_channel
```

Channel number for ADC interrupt.

sdc_intr_channel

```
uint16_t sl_intr_list_t::sdc_intr_channel
```

Channel number for SDC interrupt.

sl_intr_list_map_t

Sensors interrupt mode MAP TABLE.

Public Attributes

uint8_t	map_index Map Table Index.
sl_intr_list_t	map_table Sensor list MAP Table.

Public Attribute Documentation

map_index

```
uint8_t sl_intr_list_map_t::map_index
```

Map Table Index.

map_table

```
sl_intr_list_t sl_intr_list_map_t::map_table[MAP_TABLE_SIZE]
```

Sensor list MAP Table.

sl_em_event_t

Sensor events info structure.

Public Attributes

void *	em_sensor_data String the sensor data address.
sl_sensor_id_t	sensor_id Sensor id information.
sl_sensorhub_event_t	event Sensors HUB Callback Events.

Public Attribute Documentation

em_sensor_data

```
void* sl_em_event_t::em_sensor_data
```

String the sensor data address.

sensor_id

```
sl_sensor_id_t sl_em_event_t::sensor_id
```

Sensor id information.

event

```
sl_sensorhub_event_t sl_em_event_t::event
```

Sensors HUB Callback Events.

sl_sensor_cb_info_t

Event callback configuration structure.

Public Attributes

sl_sensor_signalEvent_t	cb_event Event callback.
sl_sensor_id_t *	cb_event_ack Event callback acknowledge.

Public Attribute Documentation

cb_event

```
sl_sensor_signalEvent_t sl_sensor_cb_info_t::cb_event
```

Event callback.

cb_event_ack

```
sl_sensor_id_t* sl_sensor_cb_info_t::cb_event_ack
```

Event callback acknowledge.

sl_i2c_config_t

I2C bus interface configuration structure.

Public Attributes

uint8_t	i2c_id I2C instances(I2C0/1/2)
uint8_t	i2c_power_state I2C power state.
uint8_t	i2c_control_mode I2C bus control configuration.
uint16_t	i2c_bus_speed I2C bus speed.
uint32_t	i2c_slave_addr I2C slave address.

Public Attribute Documentation

i2c_id

```
uint8_t sl_i2c_config_t::i2c_id
```

I2C instances(I2C0/1/2)

i2c_power_state

```
uint8_t sl_i2c_config_t::i2c_power_state
```

I2C power state.

i2c_control_mode

```
uint8_t sl_i2c_config_t::i2c_control_mode
```

I2C bus control configuration.

i2c_bus_speed

```
uint16_t sl_i2c_config_t::i2c_bus_speed
```

I2C bus speed.

i2c_slave_addr

```
uint32_t sl_i2c_config_t::i2c_slave_addr
```

I2C slave address.

sl_spi_config_t

SPI bus interface configuration structure.

Public Attributes

uint8_t	spi_bit_width SPI bus width.
uint8_t	spi_mode SPI Mode(Master/Slave)
uint8_t	spi_power_state SPI bus power mode.
uint8_t	spi_cs_number Chip select number.
uint8_t	spi_cs_misc_mode SPI miscellaneous for chip select.
uint8_t	spi_sec_sel_sig SPI slave select signal definitions.
uint32_t	spi_baud SPI bus data transmission speed(clock)
uint64_t	spi_control_mode SPI bus phase and polarity.
uint64_t	spi_cs_mode SPI control slave select mode.

Public Attribute Documentation

spi_bit_width

```
uint8_t sl_spi_config_t::spi_bit_width
```

SPI bus width.

spi_mode

```
uint8_t sl_spi_config_t::spi_mode
```

SPI Mode(Master/Slave)

spi_power_state

```
uint8_t sl_spi_config_t::spi_power_state
```

SPI bus power mode.

spi_cs_number

```
uint8_t sl_spi_config_t::spi_cs_number
```

Chip select number.

spi_cs_misc_mode

```
uint8_t sl_spi_config_t::spi_cs_misc_mode
```

SPI miscellaneous for chip select.

spi_sec_sel_sig

```
uint8_t sl_spi_config_t::spi_sec_sel_sig
```

SPI slave select signal definitions.

spi_baud

```
uint32_t sl_spi_config_t::spi_baud
```

SPI bus data transmission speed(clock)

spi_control_mode

```
uint64_t sl_spi_config_t::spi_control_mode
```

SPI bus phase and polarity.

spi_cs_mode

```
uint64_t sl_spi_config_t::spi_cs_mode
```

SPI control slave select mode.

sl_adc_config

ADC bus interface configuration structure.

Public Attributes

uint8_t	adc_init flag to know if adc is initialized, this is necessary to deinit
uint16_t	adc_data_ready flag to indicate data availability for all 16 channels
sl_adc_config_t	adc_cfg adc configuration
sl_adc_channel_config_t	adc_ch_cfg adc channel configuration

Public Attribute Documentation

adc_init

```
uint8_t sl_adc_config::adc_init
```

flag to know if adc is initialized, this is necessary to deinit

adc_data_ready

```
uint16_t sl_adc_config::adc_data_ready
```

flag to indicate data availability for all 16 channels

adc_cfg

```
sl_adc_config_t sl_adc_config::adc_cfg
```

adc configuration

adc_ch_cfg

```
sl_adc_channel_config_t sl_adc_config::adc_ch_cfg
```

adc channel configuration

sl_gpio_config_t

GPIO interface configuration structure.

This structure holds configuration information for the GPIO interface.

Public Attributes

uint8_t	c Description of member c.
uint32_t	d Description of member d.

Public Attribute Documentation

c

```
uint8_t sl_gpio_config_t::c
```

Description of member c.

d

```
uint32_t sl_gpio_config_t::d
```

Description of member d.

Sleep Timer

Sleep Timer

Introduction

The Sleeptimer driver provides software timers and delays functionalities using a low-frequency real-time clock peripheral(SYSRTC) in Si91x.

Refer to the Sleep Timer Documentation for more information. note: Sleeptimer in Si91x doesn't support timekeeping and date functionality. Also, it only uses SYSRTC peripheral.

Input/Output Stream

Input/Output Stream

The input/output (I/O) stream service provides data streams for performing read and write operations on a physical communication interface.

Introduction

The IO Stream module serves as a platform module software, primarily focusing on providing Input/Output functionalities by establishing streams. These streams, acting as abstractions, ensure a uniform approach to read and write data, independent of the specific physical communication interface. Specifically designed for compatibility with UART within the Silicon Labs SDK, the IO Stream module offers a seamless integration.

In practical implementation, the typical setup involves installing the IO Stream: USART component and incorporating the default "vcom" instance to facilitate logging, where the I/O stream service plays a pivotal role. The IO Stream component is equipped with a set of APIs tailored for the efficient reading and writing of serial data. Furthermore, the versatility of this setup is further accentuated by the accessibility of the VCOM serial port over USB, showcasing a comprehensive and effective illustration of serial communication in operation.

Configuration

The default configuration for this IO stream includes the following settings:

- Both Transmit (Tx) and Receive (Rx) functionalities are enabled.
- Operates in Asynchronous mode.
- Utilizes 8-bit data transfer.
- Specifies 1 stop bit
- Implements no parity check.
- Auto Flow control is disabled.
- Baud Rate set at 115200.

Usage

Upon installing the IO Stream: USART component and incorporating the default "vcom" instance for logging purposes, the UART Driver undergoes initialization to facilitate data reading and writing. The IO Stream component offers a suite of APIs expressly designed for efficient handling of serial data. Leveraging these APIs provided by the IO Stream component proves highly convenient for implementing and optimizing serial communication.

IOSTREAM Service in SI91X supports below streams

| Stream | Support | |---------------------------------------| | UART/USART | Supported | |---------------------------------------| | RTT | Not Supported | |---------------------------------------| | SWO | Not Supported | |---------------------------------------| | DEBUG | Not Supported | |---------------------------------------| | VUART | Not Supported | |---------------------------------------| For reconfiguration and initialization of IO Stream, the application should make use of the sl_iostream_init_instances function.

Refer to the I/O Stream Documentation for more Information.

Non-volatile Memory

Non-volatile Memory

The non-volatile memory (NVM3) driver provides a means to store key-value pairs in flash memory.

Introduction

The NVM3 driver provides a way for an application to safely store and retrieve variable-size objects in a page-based non-volatile memory (NVM). Objects are identified with 20-bit object identifiers denoted as keys. The driver is designed to use pages in a sequential order to provide equal usage and wear. The driver is resilient to power loss or reset events, ensuring that objects retrieved from the driver are in a valid state. A valid object will always be the last successfully stored object. NVM3 can detect NVM defects and mark pages as unusable. NVM3 will continue to operate on good pages after defect pages are detected.

Configuration

There are no compile-time configuration options for NVM3. All configuration parameters are contained in nvm3_Init_t.

Refer to the NVM3 Documentation for more information.

Watchdog Manager

Watchdog Manager

Introduction

The Si91x SOC provides a robust mechanism to handle state changes and recovery operations. In standalone mode, it can notify applications running on the M4 core about different state changes, which helps ensure system stability and recovery from faults.

The different boot states managed by the SOC include:

- Cold Boot: A standard boot from a powered-off state.
- Power Save Boot: Resuming from a low-power state, typically retention-based.
- Crash Boot: A reboot triggered by internal events such as hangs, crashes, watchdog triggers, or planned reboots to prevent known issues.

This feature helps detect processor hangs and ensures recovery through system resets, enhancing the overall reliability of the SOC.

Configuration

Configuration involves setting up mechanisms to handle various boot states and utilizing the internal watchdog and reset features to manage unexpected faults or hangs.

- Ensure that all boot states are properly monitored.
- Use watchdog timers to detect hangs or crashes.
- Implement firmware routines to handle planned reboots and crash scenarios effectively.

Usage

The typical sequence to utilize WDT-MANAGER effectively includes:

1. Initialize the watchdog and monitoring mechanisms at startup.
2. Configure the system to handle different boot states:
    - Cold Boot
    - Power Save Boot
    - Crash Boot
3. Implement interrupt handling for detecting system hangs or crashes.
4. Trigger system reset in case of detected faults.
5. Log state changes and recovery actions for diagnostics.

Benefits

- Detects and recovers from processor hangs.
- Provides mechanisms to handle planned and unplanned reboots.
- Ensures system reliability and stability by managing state transitions effectively.

Note

- When the RC clock is used as the source for the LF-FSM, the timeout durations may vary due to the inherent frequency instability of the RC oscillator.

Enumerations

enum sl_watchdog_manager_reset_reason_t {

SL_SI91X_WATCHDOG_MANAGER_RESET_POR
SL_SI91X_WATCHDOG_MANAGER_RESET_WATCHDOG
SL_SI91X_WATCHDOG_MANAGER_RESET_SOFTWARE
SL_SI91X_WATCHDOG_MANAGER_RESET_MAX

}

Reason for the system being reset.

enum sl_watchdog_manager_wdog_timeout_t {

SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_0
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_1
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_2
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_3
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_4
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_5
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_6
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_7
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_8
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_9
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_10
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_11
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_12
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_13
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_14
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_15
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_16
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_17
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_18
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_19
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_20
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_21
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_22
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_23
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_24
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_25
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_26
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_27
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_28
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_29
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_30
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_31
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_MAX

}

Enumeration to represent possible time delay values for Watchdog Timer (WDT) interrupt time and system reset time with 32 kHz clock frequency.

Functions

sl_status_t sl_watchdog_manager_init(void)

To configure the watchdog manager service.

sl_status_t sl_watchdog_manager_start(void)

To start the watchdog manager service.

sl_watchdog_manager_reset_reason_t	sl_wdt_manager_get_system_reset_status(void) To retrieve the system reset reason.
void	sl_watchdog_disable_wdt_in_sleep(void) To disable the hardware watchdog in sleep mode.
void	sl_watchdog_enable_wdt_in_sleep(void) To enable the hardware watchdog in sleep mode.

Macros

#define	SL_SI91X_WAKEUP_INDICATION BIT(0) Wake-up event.
#define	SL_SI91X_TIMEOUT_WAKEUP BIT(1) Timeout wake-up.
#define	SL_SI91X_HOST_BASED_WAKEUP_S BIT(2) Host wake-up.
#define	SL_SI91X_MCU_PROCESSOR_WAKE_STAT BIT(4) MCU wake-up status.
#define	SL_SI91X_MCU_WWD_RESET BIT(5) MCU watchdog reset.
#define	SL_SI91X_MCU_WWD_WINDOW_RESET BIT(6) MCU watchdog window reset.
#define	SL_SI91X_NWP_WWD_RESET BIT(8) Network processor watchdog reset.
#define	SL_SI91X_NWP_WWD_WINDOW_RESET BIT(9) Network processor watchdog window reset.
#define	SL_SI91X_HOST_RESET_REQUEST BIT(10) Host reset request.
#define	SL_SI91X_WDT_RESET undefined Watchdog reset status flags.

Enumeration Documentation

sl_watchdog_manager_reset_reason_t

> sl_watchdog_manager_reset_reason_t

Reason for the system being reset.

This enumeration defines the possible reasons for the system being reset. It includes power-on reset, watchdog reset, and software-invoked reset.

Enumerator	
SL_SI91X_WATCHDOG_MANAGER_RESET_POR	A power-on reset was performed.
SL_SI91X_WATCHDOG_MANAGER_RESET_WATCHDOG	System was reset by the watchdog manager service.
SL_SI91X_WATCHDOG_MANAGER_RESET_SOFTWARE	A reset was invoked from the system firmware.
SL_SI91X_WATCHDOG_MANAGER_RESET_MAX	Maximum value for reset reasons.

sl_watchdog_manager_wdog_timeout_t

sl_watchdog_manager_wdog_timeout_t

Enumeration to represent possible time delay values for Watchdog Timer (WDT) interrupt time and system reset time with 32 kHz clock frequency.

This enumeration defines the possible time delays for the WDT interrupt and system reset times. The delays are based on a 32 kHz clock frequency.

Enumerator	
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_0	Time delay of 0.03125 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_1	Time delay of 0.0625 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_2	Time delay of 0.125 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_3	Time delay of 0.25 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_4	Time delay of 0.5 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_5	Time delay of 1 millisecond.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_6	Time delay of 2 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_7	Time delay of 4 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_8	Time delay of 8 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_9	Time delay of 16 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_10	Time delay of 32 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_11	Time delay of 64 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_12	Time delay of 125 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_13	Time delay of 250 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_14	Time delay of 500 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_15	Time delay of 1000 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_16	Time delay of 2000 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_17	Time delay of 4000 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_18	Time delay of 8000 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_19	Time delay of 16000 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_20	Time delay of 32000 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_21	Time delay of 64000 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_22	Time delay of 128000 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_23	Time delay of 256000 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_24	Time delay of 512000 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_25	Time delay of 10240000 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_26	Time delay of 2048000 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_27	Time delay of 4096000 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_28	Time delay of 8192000 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_29	Time delay of 16384000 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_30	Time delay of 32768000 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_31	Time delay of 65536000 milliseconds.
SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_INDEX_MAX	Maximum value for time delay validation.

Function Documentation

sl_watchdog_manager_init

```
sl_status_t sl_watchdog_manager_init (void )
```

To configure the watchdog manager service.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Configures the hardware watchdog timer.

- The 91x watchdog peripheral's interrupt interval is set to 2 seconds less than the configured value of `SL_WDT_MANAGER_INTERRUPT_TIME`. The hardware watchdog triggers an interrupt when this timeout expires, allowing the software to feed the watchdog within this interval.
  - Assigns a system reset value to 4 seconds. If the WDT is not reset within this interval, the system will reset after 4 seconds. The reset counter begins counting after the interrupt time expires. The system reset value is updated based on the configured interrupt time, referenced by `SL_SI91X_WATCHDOG_MANAGER_TIMEOUT_PERIOD`.

Returns

- sl_status_t Status code indicating the result:
  - SL_STATUS_OK - Success
  - SL_STATUS_FAIL - Generic error

For more information on status codes, refer to SL STATUS DOCUMENTATION.

sl_watchdog_manager_start

```
sl_status_t sl_watchdog_manager_start (void )
```

To start the watchdog manager service.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function starts the WDT and begins counting. The hardware watchdog triggers an interrupt when the WDT timeout interval expires.

- Pre-conditions:
  - sl_watchdog_manager_init must be called before this function.

Returns

- sl_status_t Status code indicating the result:
  - SL_STATUS_OK - Success
  - SL_STATUS_FAIL - Generic error

For more information on status codes, refer to SL STATUS DOCUMENTATION.

sl_wdt_manager_get_system_reset_status

```
sl_watchdog_manager_reset_reason_t sl_wdt_manager_get_system_reset_status (void )
```

To retrieve the system reset reason.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns the reason for the system being reset. This may be invoked during a startup flow to determine if the system has recovered from an error condition such as a crash.

Returns

- sl_watchdog_manager_reset_reason_t Enumeration value indicating the reset reason:
  - SL_SI91X_WATCHDOG_MANAGER_RESET_WATCHDOG - WDT System Reset
  - SL_SI91X_WATCHDOG_MANAGER_RESET_SOFTWARE - Software System Reset
  - SL_SI91X_WATCHDOG_MANAGER_RESET_POR - Power-on Reset

sl_watchdog_disable_wdt_in_sleep

```
void sl_watchdog_disable_wdt_in_sleep (void )
```

To disable the hardware watchdog in sleep mode.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function stops the WDT counter from running when the system enters sleep mode.

- Pre-conditions:
  - sl_watchdog_manager_init must be called before this function.

Note

- This function does not return a status code.

sl_watchdog_enable_wdt_in_sleep

```
void sl_watchdog_enable_wdt_in_sleep (void )
```

To enable the hardware watchdog in sleep mode.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function starts the WDT counter running in sleep mode. It enables the WDT as a wake-up source to the system to serve the ISR and kick the watchdog.

Pre-conditions:

- sl_watchdog_manager_init must be called before this function.

Clock Manager

Clock Manager

Introduction

The Clock Manager module is responsible for configuring the clock tree in the Si91x system. It enhances the clock management features provided by the GSDK's emlib and device_init services, offering additional functionality for selecting clock sources and setting dividers. This module ensures optimized performance and power efficiency across various system components by managing clock configurations according to application requirements.

Configuration

The Clock Manager supports a variety of configurations:

- Clock Source Selection: Allows specific clock sources to be chosen for different branches.
- Divider Configuration: Modifies clock frequencies by setting divider values.

Configuration is typically performed during device initialization using configuration files and source code.

Usage

The following steps outline typical usage of the Clock Manager:

1. Initialize the Clock Manager with desired configuration settings.
2. Set up oscillator sources and configure clock dividers.
3. Use runtime APIs to manage dynamic clock changes.
4. Maintain stable clock configurations to ensure system reliability.

Benefits

- Manages complex clock trees efficiently.
- Enhances system stability and performance.
- Supports a wide range of Si91x devices.
- Optimizes power consumption through efficient clock management.

Typedefs

typedef M4_SOC_CLK_SRC_SEL_T	sl_si91x_m4_soc_clk_src_sel_t Typedef to select the clock source for the M4 core in the Si91x SOC.

Functions

sl_status_t	sl_si91x_clock_manager_init(void) Initializes the M4_SOC and other required clocks.
sl_status_t	sl_si91x_clock_manager_m4_set_core_clk(M4_SOC_CLK_SRC_SEL_T clk_source, uint32_t pll_freq) To configure the M4 core clock source and configure the PLL frequency if selected as source.

sl_status_t	sl_si91x_clock_manager_set_pll_freq(PLL_TYPE_T pll_type, uint32_t pll_freq, uint32_t pll_ref_clk) To set the selected Phase-Locked Loop (PLL) clock to the desired frequency.
sl_si91x_m4_soc_clk_src_sel_t	sl_si91x_clock_manager_m4_get_core_clk_src_freq(uint32_t *m4_core_clk_freq) To read the currently active M4 core clock source and its frequency.
uint32_t	sl_si91x_clock_manager_get_pll_freq(PLL_TYPE_T pll_type) Gets the selected PLL (Phase-Locked Loop) clock to the desired frequency.
sl_status_t	sl_si91x_clock_manager_control_pll(PLL_TYPE_T pll_type, bool enable) Controls the selected PLL (Phase-Locked Loop) clock.
void	sl_si91x_delay_ms(uint32_t milli_seconds) Delays execution for the specified number of milliseconds.
sl_status_t	sl_si91x_clock_manager_ulp_processor_clk_division(uint8_t clk_div) Set M4 ULP Processor clock division factor.

Macros

#define	PLL_REF_CLK_VAL_XTAL (40000000UL) PLL reference clock frequency value of XTAL CLK.
#define	MAX_PLL_FREQUENCY (180000000UL) Max PLL frequency is 180MHz.

Typedef Documentation

sl_si91x_m4_soc_clk_src_sel_t

```
sl_si91x_m4_soc_clk_src_sel_t
```

Typedef to select the clock source for the M4 core in the Si91x SOC.

This typedef maps to the M4_SOC_CLK_SRC_SEL_T type, which defines the possible clock source selections for the M4 core. The clock source selection impacts the operating frequency and performance characteristics of the M4 processor within the Si91x SOC.

Users can configure the clock source to optimize power consumption or performance based on application requirements.

Note

- Ensure that the selected clock source is properly configured and stable before switching to avoid system instability.

Function Documentation

sl_si91x_clock_manager_init

```
sl_status_t sl_si91x_clock_manager_init (void )
```

Initializes the M4_SOC and other required clocks.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- sl_status_t Status code indicating the result:
  - SL_STATUS_OK - Success.
  - Corresponding error code on failure.

For more information on status codes, refer to SL STATUS DOCUMENTATION.

sl_si91x_clock_manager_m4_set_core_clk

> sl_status_t sl_si91x_clock_manager_m4_set_core_clk (M4_SOC_CLK_SRC_SEL_T clk_source, uint32_t pll_freq)

To configure the M4 core clock source and configure the PLL frequency if selected as source.

Parameters

Type	Direction	Argument Name	Description
M4_SOC_CLK_SRC_SEL_T	[in]	clk_source	Enum value representing different core clock sources.
uint32_t	[in]	pll_freq	Desired M4 core frequency in MHz.

Returns

- sl_status_t Status code indicating the result:
  - SL_STATUS_OK - Success.
  - Corresponding error code on failure.

For more information on status codes, see SL STATUS DOCUMENTATION.

sl_si91x_clock_manager_set_pll_freq

> sl_status_t sl_si91x_clock_manager_set_pll_freq (PLL_TYPE_T pll_type, uint32_t pll_freq, uint32_t pll_ref_clk)

To set the selected Phase-Locked Loop (PLL) clock to the desired frequency.

Parameters

Type	Direction	Argument Name	Description
PLL_TYPE_T	[in]	pll_type	Enum specifying the type of PLL to configure.
uint32_t	[in]	pll_freq	Desired frequency for the PLL clock (in MHz).
uint32_t	[in]	pll_ref_clk	Reference clock frequency for the PLL configuration.

Returns

- sl_status_t Status code indicating the result:
  - SL_STATUS_OK - Success.
  - Corresponding error code on failure.

For more information on status codes, see SL STATUS DOCUMENTATION.

sl_si91x_clock_manager_m4_get_core_clk_src_freq

```
sl_si91x_m4_soc_clk_src_sel_t sl_si91x_clock_manager_m4_get_core_clk_src_freq (uint32_t * m4_core_clk_freq)
```

To read the currently active M4 core clock source and its frequency.

Parameters

Type	Direction	Argument Name	Description
uint32_t *	[out]	m4_core_clk_freq	Pointer to a variable where the current core clock frequency will be stored (in MHz).

Returns

- sl_si91x_m4_soc_clk_src_sel_t The currently active core clock source:
  - 0: M4_ULPREFCLK
  - 2: M4_SOCPLLCLK
  - 3: M4_MODEMPLLCLK1
  - 4: M4_INTFPLLCLK
  - 5: M4_SLEEPCLK

sl_si91x_clock_manager_get_pll_freq

```
uint32_t sl_si91x_clock_manager_get_pll_freq (PLL_TYPE_T pll_type)
```

Gets the selected PLL (Phase-Locked Loop) clock to the desired frequency.

Parameters

Type	Direction	Argument Name	Description
PLL_TYPE_T	[in]	pll_type	Enum specifying the type of PLL to configure.

Returns

- uint32_t PLL frequency value in MHz.

sl_si91x_clock_manager_control_pll

```
sl_status_t sl_si91x_clock_manager_control_pll (PLL_TYPE_T pll_type, bool enable)
```

Controls the selected PLL (Phase-Locked Loop) clock.

Parameters

Type	Direction	Argument Name	Description
PLL_TYPE_T	[in]	pll_type	Enum specifying the type of PLL to control.
bool	[in]	enable	Boolean value to enable (true) or disable (false) the PLL.

Returns

- sl_status_t Status code indicating the result:
  - SL_STATUS_OK - Success.

Corresponding error code on failure.

For more information on status codes, see SL STATUS DOCUMENTATION.

sl_si91x_delay_ms

```
void sl_si91x_delay_ms (uint32_t milli_seconds)
```

Delays execution for the specified number of milliseconds.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	milli_seconds	Delay time in milliseconds.

Note

- This function provides a blocking delay.The delay is calibrated based on the SystemCoreClock frequency. If SystemCoreClock < CLOCK_THRESHOLD, the delay is calibrated with a division factor of LOW_FREQ_CLK_DIV_FAC. If SystemCoreClock >= CLOCK_THRESHOLD, the delay is calibrated with a division factor of HIGH_FREQ_CLK_DIV_FAC. This function uses `__NOP()` instructions for the delay loop.

For more information on status codes, see SL STATUS DOCUMENTATION.

sl_si91x_clock_manager_ulp_processor_clk_division

```
sl_si91x_clock_manager_ulp_processor_clk_division (uint8_t clk_div)
```

Set M4 ULP Processor clock division factor.

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	clk_div	Division factor (the clock will be divided by 2*clk_div)

This API sets the M4 ULP Processor clock division factor by configuring the clock control register. The division factor is calculated as 2*clk_div, where clk_div is the input parameter.

Returns

- sl_status_t Status code indicating the result:
  - SL_STATUS_OK - Success.
  - SL_STATUS_INVALID_PARAMETER - Invalid argument.

Note

- This API should be invoked only when the system is in PS2 state, as the source clock in this mode is configured to use a 20 MHz RC.

For more information on status codes, see SL STATUS DOCUMENTATION.

Firmware Fallback A and B

Firmware Fallback A and B

Introduction

The Firmware Fallback feature provides mechanisms for managing A/B firmware upgrades and fallback operations in Si91x devices. It ensures system reliability by maintaining two firmware slots and enabling fallback to a stable firmware version in case of an upgrade failure Additionally, this functionality requires a specific MBR configuration to work.

Features

- A/B Slot Management: Maintains two firmware slots (Slot A and Slot B) for active and backup firmware.
- Integrity Checks: Verifies firmware integrity using CRC and magic word validation.
- Fallback Mechanism: Switches to a backup firmware slot in case of failure.
- OTA Support: Supports Over-The-Air (OTA) firmware upgrades with slot management.

Usage

The following steps outline the typical usage of the Firmware Fallback feature, providing clear guidance on how and when to use each API in the firmware update process:

1. At OTA initiation:
   Use sl_si91x_ab_upgrade_get_rps_configs to parse the incoming firmware image header. This step extracts critical metadata, such as image size and type, and is typically performed when the first chunk of OTA data arrives. This information helps determine the type of update being performed and allocate the necessary resources for the update process.
2. Preparing the inactive slot for new firmware:
   Before writing the new firmware, use sl_si91x_flash_erase to clear the inactive slot. This ensures the slot is ready to receive the new firmware image and prevents flash corruption. The inactive slot can be identified using sl_si91x_ab_get_slot_info, which retrieves the current slot configuration.
3. Writing firmware during OTA download:
   Use sl_si91x_flash_write to store the received firmware chunks in the inactive slot as they arrive from the OTA source. This function should be called for each chunk of data received during the OTA session, ensuring proper addressing and offset management. This step ensures that the new firmware is written correctly to the designated slot.
4. Validating the downloaded firmware:
   After the firmware download is complete, use sl_si91x_verify_image to validate the integrity of the downloaded firmware image. This step ensures that only properly formed and uncorrupted firmware is considered for activation.
5. Updating slot metadata after successful validation:
   Once the new firmware is verified, use sl_si91x_ab_upgrade_set_slot_info to update the slot metadata with information about the new firmware. This prepares the slot for activation and ensures the slot information accurately reflects the valid firmware version. This step is essential to mark the new firmware as ready for execution.
6. Retrieving current slot information:
   Use sl_si91x_ab_get_slot_info to determine which slot (A or B) is currently active for both M4 and NWP processors. This function also retrieves information about firmware locations and sizes in flash memory. It is typically called before performing any operation that depends on knowing the current slot configuration. This step is useful for diagnostics and ensuring the correct slot is being used.
7. Switching slots manually or for recovery:
   Use sl_si91x_toggle_slot_info only in specific scenarios, such as recovering from a failed firmware update by switching back to the previously working slot or manually switching between existing firmware versions. After

calling this function, the system must be reset to load the newly activated firmware. This function should not be part of the regular update process and should only be used when both slots contain valid firmware images.

8. Burning NWP security version:
   Use sl_si91x_burn_nwp_security_version to burn the NWP security version to OTP memory. This step is critical for enforcing firmware security on the NWP (Network Wireless Processor). Burning the security version to OTP (One-Time Programmable) memory ensures that only firmware images with a security version equal to or greater than the burned value can be executed. This mechanism prevents rollback attacks by disallowing the loading of older, potentially vulnerable firmware versions. This operation is irreversible and should be performed with caution, typically after a successful firmware update or when upgrading the device's security requirements.
9. Loading QSPI keys for inline decryption:
   Use sl_si91x_fallback_load_qspi_keys to load the QSPI keys for enabling inline decryption for M4 slot applications during fallback. This function is essential when firmware images are stored in encrypted form in QSPI flash. Loading the QSPI keys allows the hardware to transparently decrypt firmware as it is read and executed, ensuring both security and performance. This step should be performed before attempting to boot or validate encrypted firmware images, and the keys must be securely provisioned and managed.
10. Soft reset for NWP firmware:
    Use sl_si91x_nwp_soft_reset_from_updater to perform a soft reset of the NWP firmware for fallback and keep the NWP firmware in boot mode. This function is typically used after updating or recovering the NWP firmware, ensuring that the NWP restarts cleanly and enters a known state. Keeping the NWP in boot mode allows for further maintenance operations, such as additional firmware updates, diagnostics, or recovery actions, before normal operation resumes. This step is important for robust fallback and recovery workflows.
11. Default NWP firmware selection:
    Use sl_si91x_select_default_nwp_fw to select the default NWP firmware image to load. This API allows you to specify which NWP (Network Wireless Processor) firmware slot (A or B) should be used as the default on the next boot. The function takes a firmware image number as input (0 for slot A, 1 for slot B) and updates the boot configuration accordingly. This is particularly useful for scenarios where you want to manually control which firmware version is active, such as after a successful OTA update, for testing a new firmware image, or for recovery purposes if the current firmware is not functioning as expected. The function returns 0 on success, a negative value if the command fails.
12. Executing the firmware from the active slot:
    Use sl_si91x_jump_to_m4_application to execute the firmware from the currently active slot. This function is primarily used by the M4 Updater to branch to the appropriate application after reading slot information. It transfers control from the updater to the main application, completing the boot process. This step ensures the system boots into the correct firmware version.

The following three APIs are required only when full encryption is enabled:

- sl_si91x_burn_nwp_security_version
- sl_si91x_fallback_load_qspi_keys
- sl_si91x_nwp_soft_reset_from_updater

By following these steps, the Firmware Fallback feature ensures a seamless and reliable firmware update process, with mechanisms for recovery and fallback in case of failures. This design minimizes downtime and ensures the device remains operational even in the event of update issues.

Benefits

- Ensures system reliability with robust fallback mechanisms.
- Simplifies OTA firmware upgrades with A/B slot management.
- Eliminates the need to transfer the image from the download region to the execution region, enabling faster updates.
- Provides APIs for seamless integration into application firmware.
- Optimizes flash memory usage for firmware storage.

Modules

ota_image_info_t

sl_si91x_fw_fallback_request_t

sl_si91x_fw_fallback_config_t

SlotInfo_t

M4_SlotManagement_t

NWP_SlotManagement_t

sl_si91x_fw_ab_slot_management_t

Enumerations

enum sl_si91x_ab_image_type_t {

SL_SI91X_AB_OTA_IMAGE_TYPE_M4 = 1
SL_SI91X_AB_OTA_IMAGE_TYPE_NWP = 2
SL_SI91X_AB_OTA_IMAGE_TYPE_MAX

}
Enum for A/B OTA image types.

enum sl_si91x_ab_slot_t {

SLOT_A = 1
SLOT_B = 2
SLOT_MAX

}
Enum for AB slot identifiers.

enum sl_si91x_chunk_type_t {

SL_SI91X_DATA_PACKET = 0
SL_SI91X_HEADER_PACKET = 1

}
Enum for A/B chunk types(Payload / Header).

enum sl_si91x_ab_fallback_cmd_t {

SL_SI91X_AB_FALLBACK_WRITE_CMD = 1
SL_SI91X_AB_FALLBACK_READ_CMD = 2
SL_SI91X_AB_FALLBACK_ERASE_CMD = 3
SL_SI91X_AB_FALLBACK_INTEGRITY_CHECK_CMD = 4
SL_SI91X_AB_FALLBACK_BURN_SECURITY_VERSION = 5
SL_SI91X_AB_FALLBACK_LOAD_QSPI_KEYS = 6
SL_SI91X_AB_FALLBACK_SOFT_RESET = 7
SL_SI91X_AB_FALLBACK_MAX_CMD = 8

}
SI91x-specific A/B fallback command types.

Functions

sl_status_t sl_si91x_ab_upgrade_get_rps_configs(const uint8_t *image_buffer, ota_image_info_t *ota_st)
Retrieves A/B fallback OTA image configuration from the provided image buffer. This function checks the magic number in the image buffer to validate the firmware header. If valid, it populates the ota_image_info_t structure with the image offset and size from the firmware header.

sl_status_t sl_si91x_flash_write(uint32_t address, const uint8_t *buffer, uint32_t length)
Write data to flash memory. This function writes data to the specified flash memory address. This function will work for A/B firmware only.

sl_status_t sl_si91x_fw_fallback_ota_flash_write(const sl_si91x_fw_fallback_config_t *config, const uint8_t *data_buffer)
Writes firmware data to flash during OTA updates. Call this function when receiving each chunk of OTA firmware data. Must be called after erasing flash and before verifying the image. Works only with A/B firmware configuration.

sl_status_t sl_si91x_flash_read(uint32_t address, uint8_t *buffer, uint32_t length)
Read data from flash memory. This function reads data from the specified flash memory address. This function will work for A/B firmware only.

sl_status_t sl_si91x_ab_upgrade_set_slot_info(uint32_t new_image_offset, uint32_t new_image_size, sl_si91x_ab_image_type_t image_type)
Updates slot information and switches active slots on OTA updates. This function updates the slot information and switches the active slots based on the provided parameters. This function will work for A/B firmware only.

sl_status_t sl_si91x_flash_erase(uint32_t address, uint32_t length)
Erases data from flash memory sector-wise.

sl_status_t sl_si91x_ab_get_slot_info(sl_si91x_fw_ab_slot_management_t *slot_info_t)
Retrieves the current slot information. This function reads the current slot information from flash memory. This function will work for A/B firmware only.

sl_status_t sl_si91x_toggle_slot_info(bool toggle_m4_image, bool toggle_nwp_image)
Toggles the active slot information for M4 or NWP. This function retrieves the current slot information, checks which slot is active, and switches to the other slot by calling the sl_si91x_set_slot_info API.

sl_status_t sl_si91x_get_m4_app_addr(uint32_t *app_addr)
Retrieves the M4 application address from the active firmware slot. This function reads firmware slot information from flash memory, verifies its integrity using a magic word and CRC check, and determines the application address for the currently active M4 slot. This function will work for A/B firmware only.

void sl_si91x_jump_to_m4_application(uint32_t app_addr)
Jumps to the M4 application.

sl_status_t sl_si91x_verify_image(uint32_t flash_address)
Checks the integrity of the firmware for Fallback. This function sends a command to check the integrity of the firmware at the specified flash memory address. It populates the request structure with the necessary parameters and sends the command to the firmware. This function will work for A/B firmware only.

int16_t sl_si91x_select_default_nwp_fw(const uint8_t fw_image_number)
Selects the default NWP firmware image to load. This function sends the appropriate command to select which firmware image (0 for slot A, 1 for slot B) should be loaded on boot.

sl_status_t sl_si91x_burn_nwp_security_version(uint32_t flash_address)
Burns the NWP security version to OTP memory to prevent firmware rollback. Call this function after validating a new NWP firmware version and before using it. This operation is irreversible - once burned, only firmware with equal or higher security versions can be loaded.

sl_status_t sl_si91x_fallback_load_qspi_keys(uint32_t image_offset)
Loads QSPI keys for encrypted firmware execution. Call this function:

void sl_si91x_nwp_soft_reset_from_updater(const uint32_t m4_slot_image_offset)
Performs a soft reset of the NWP firmware during A/B firmware fallback operations. This function is a critical part of the firmware update process that:

Macros

#define SLI_SI91X_RPS_MAGIC_NO 0x900D900D
Magic number located in RPS Header.

#define SLI_SI91X_RPS_HEADER_SIZE sizeof(sl_wifi_firmware_header_t)
Firmware header size in bytes.

#define SL_SI91X_CHUNK_LENGTH 4096
Flash memory chunk size for erase operations.

#define SLI_SI91X_M4_FLASH_BASE_ADDR 0x8000000
Base address of M4 and NWP flash memory.

#define SLI_SI91X_NWP_FLASH_BASE_ADDR 0x4000000
NWP flash base address.

#define SLI_SI91X_SLOT_INFO_FLASH_OFFSET 0x81F1000
Slot information flash offsets.

#define SLI_SI91X_BACKUP_SLOT_INFO_FLASH_OFFSET (SLI_SI91X_SLOT_INFO_FLASH_OFFSET + SL_SI91X_CHUNK_LENGTH)
Backup slot offset.

#define SL_SI91X_M4_RPS_BIT 1
Image type identifiers in RPS.

#define SL_SI91X_NWP_RPS_BIT 0
NWP RPS image type.

#define SL_SI91X_MAX_OTA_IMAGE_CHUNK_SIZE 1024
OTA image chunk size.

#define SL_SI91X_NWP_RESPONSE_TIMEOUT 30000
Timeout for NWP responses.

#define SL_SI91X_DEFAULT_FW_SELECT_CMD_TIMEOUT 10000
Timeout for selecting default firmware.

#define SLI_SI91X_AB_FW_SLOT_MAGIC_WORD 0xA5A5B5B5
Magic word to verify slot information integrity.

#define CRC32_POLYNOMIAL 0x04C11DB7
CRC polynomial for slot information.

#define SL_SI91X_MEM_CHECK_VALUE 0xAB11
Memory value to check for memory match.

#define SL_SI91X_M4_UPDATER_OTA_FLAG 0xEF
M4 updater OTA flag.

#define SL_SI91X_AB_ERR_FLASH_READ ((sl_status_t)0xAB001)
A/B fallback: Flash read operation failed.

#define SL_SI91X_AB_ERR_FLASH_VERIFY ((sl_status_t)0xAB002)
A/B fallback: Flash verification failure.

#define SL_SI91X_AB_ERR_BACKUP_READ ((sl_status_t)0xAB003)
A/B fallback: Backup slot read failure.

#define SL_SI91X_AB_ERR_BACKUP_VERIFY ((sl_status_t)0xAB004)
A/B fallback: CRC verification failed in backup slot.

#define SL_SI91X_AB_ERR_CRC_MISMATCH ((sl_status_t)0xAB005)
A/B fallback: CRC mismatch detected.

#define SL_SI91X_AB_ERR_MAGIC_NUMBER ((sl_status_t)0xAB006)
A/B fallback: Magic number mismatch error.

#define SL_SI91X_AB_ERR_UNKNOWN_IMG ((sl_status_t)0xAB007)
A/B fallback: Unknown image type.

#define SL_SI91X_AB_ERR_INVALID_SLOT_INFO ((sl_status_t)0xAB008)
A/B fallback: Invalid slot info.

#define SL_SI91X_AB_ERR_ERASE_ACTIVE_SLOT ((sl_status_t)0xAB009)
A/B fallback: Invalid slot info.

Enumeration Documentation

sl_si91x_ab_image_type_t

> sl_si91x_ab_image_type_t

Enum for A/B OTA image types.

Enumerator	
SL_SI91X_AB_OTA_IMAGE_TYPE_M4	Image for M4 processor.
SL_SI91X_AB_OTA_IMAGE_TYPE_NWP	Image for NWP processor.
SL_SI91X_AB_OTA_IMAGE_TYPE_MAX	Maximum image type value.

sl_si91x_ab_slot_t

> sl_si91x_ab_slot_t

Enum for AB slot identifiers.

Enumerator	
SLOT_A	Primary slot (typically the active firmware slot)
SLOT_B	Secondary slot (used for updates or fallback)
SLOT_MAX	Maximum number of slots (used for validation)

sl_si91x_chunk_type_t

> sl_si91x_chunk_type_t

Enum for A/B chunk types(Payload / Header).

Enumerator	
SL_SI91X_DATA_PACKET	Chunk type for Data packet.
SL_SI91X_HEADER_PACKET	Chunk type for Header packet.

sl_si91x_ab_fallback_cmd_t

> sl_si91x_ab_fallback_cmd_t

SI91x-specific A/B fallback command types.

Enumerator	
SL_SI91X_AB_FALLBACK_WRITE_CMD	Write firmware data for fallback.
SL_SI91X_AB_FALLBACK_READ_CMD	Read firmware data for verification.
SL_SI91X_AB_FALLBACK_ERASE_CMD	Erase fallback firmware slot.
SL_SI91X_AB_FALLBACK_INTEGRITY_CHECK_CMD	Perform integrity check before fallback.
SL_SI91X_AB_FALLBACK_BURN_SECURITY_VERSION	Burn security version.
SL_SI91X_AB_FALLBACK_LOAD_QSPI_KEYS	Load QSPI keys.
SL_SI91X_AB_FALLBACK_SOFT_RESET	NWP Soft reset for fallback.
SL_SI91X_AB_FALLBACK_MAX_CMD	Maximum fallback command value (for validation)

Function Documentation

sl_si91x_ab_upgrade_get_rps_configs

> sl_status_t sl_si91x_ab_upgrade_get_rps_configs (const uint8_t * image_buffer, ota_image_info_t * ota_st)

Retrieves A/B fallback OTA image configuration from the provided image buffer. This function checks the magic number in the image buffer to validate the firmware header. If valid, it populates the ota_image_info_t structure with the image offset and size from the firmware header.

Parameters

Type	Direction	Argument Name	Description
const uint8_t *	[in]	image_buffer	Pointer to the buffer containing the firmware image.
ota_image_info_t *	[out]	ota_st	Pointer to the ota_image_info_t structure to be populated.

- None

Returns

- sl_status_t SL_STATUS_OK on success, or SL_STATUS_FAIL on failure. For more information on status codes, refer to SL STATUS DOCUMENTATION.

sl_si91x_flash_write

> sl_status_t sl_si91x_flash_write (uint32_t address, const uint8_t * buffer, uint32_t length)

Write data to flash memory. This function writes data to the specified flash memory address. This function will work for A/B firmware only.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	address	Flash memory address to write to.
const uint8_t *	[in]	buffer	Pointer to the data buffer.
uint32_t	[in]	length	Number of bytes to write (min 0 max 4096).

- None

Returns

- sl_status_t SL_STATUS_OK on success, error code otherwise. For more information on status codes, refer to SL STATUS DOCUMENTATION.

sl_si91x_fw_fallback_ota_flash_write

sl_status_t sl_si91x_fw_fallback_ota_flash_write (const sl_si91x_fw_fallback_config_t * config, const uint8_t * data_buffer)

Writes firmware data to flash during OTA updates. Call this function when receiving each chunk of OTA firmware data. Must be called after erasing flash and before verifying the image. Works only with A/B firmware configuration.

Parameters

Type	Direction	Argument Name	Description
const sl_si91x_fw_fallback_config_t *	[in]	config	Pointer to the firmware fallback configuration structure.
const uint8_t *	[in]	data_buffer	Pointer to the data buffer to write.

- 1. Flash region must be erased before writing.
    1. Data chunk size must not exceed 1024 bytes.

Returns

- sl_status_t SL_STATUS_OK on success, error code otherwise. For more information on status codes, refer to SL STATUS DOCUMENTATION.

sl_si91x_flash_read

sl_status_t sl_si91x_flash_read (uint32_t address, uint8_t * buffer, uint32_t length)

Read data from flash memory. This function reads data from the specified flash memory address. This function will work for A/B firmware only.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	address	Flash memory address to read from M4.
uint8_t *	[in]	buffer	Pointer to the buffer to store read data.
uint32_t	[in]	length	Number of bytes to read.

- None

Returns

- sl_status_t SL_STATUS_OK on success, error code otherwise. For more information on status codes, refer to SL STATUS DOCUMENTATION.

sl_si91x_ab_upgrade_set_slot_info

sl_status_t sl_si91x_ab_upgrade_set_slot_info (uint32_t new_image_offset, uint32_t new_image_size, sl_si91x_ab_image_type_t image_type)

Updates slot information and switches active slots on OTA updates. This function updates the slot information and switches the active slots based on the provided parameters. This function will work for A/B firmware only.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	new_image_offset	New image base address (used only during OTA update).
uint32_t	[in]	new_image_size	New image size (used only during OTA update).
sl_si91x_ab_image_type_t	[in]	image_type	Type of the image (M4 or NWP).

- None

Returns

- sl_status_t SL_STATUS_OK if successful, error code otherwise. For more information on status codes, refer to SL STATUS DOCUMENTATION.

sl_si91x_flash_erase

sl_status_t sl_si91x_flash_erase (uint32_t address, uint32_t length)

Erases data from flash memory sector-wise.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	address	Starting flash memory address to erase (must be sector-aligned).
uint32_t	[in]	length	Number of bytes to erase (will be rounded up to sector size).

- None

This function erases data from the specified flash memory address in sector-sized blocks. The erase operation will only work for A/B firmware slots and ensures that no partial sector erases occur.

Note

- The erase operation is performed in full sectors. If the provided address or length does not align with sector boundaries, the function will round to the nearest sector boundary.

Returns

- sl_status_t SL_STATUS_OK on success, error code otherwise. For more information on status codes, refer to SL STATUS DOCUMENTATION.

sl_si91x_ab_get_slot_info

sl_status_t sl_si91x_ab_get_slot_info (sl_si91x_fw_ab_slot_management_t * slot_info_t)

Retrieves the current slot information. This function reads the current slot information from flash memory. This function will work for A/B firmware only.

Parameters

Type	Direction	Argument Name	Description
sl_si91x_fw_ab_slot_management_t *	[out]	slot_info_t	Pointer to the sl_si91x_fw_ab_slot_management_t structure to store the slot information.

- None

Returns

- sl_status_t SL_STATUS_OK if successful, error code otherwise.

sl_si91x_toggle_slot_info

```
sl_status_t sl_si91x_toggle_slot_info (bool toggle_m4_image, bool toggle_nwp_image)
```

Toggles the active slot information for M4 or NWP. This function retrieves the current slot information, checks which slot is active, and switches to the other slot by calling the sl_si91x_set_slot_info API.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	toggle_m4_image	Set to 1 to toggle M4 slot, 0 otherwise.
bool	[in]	toggle_nwp_image	Set to 1 to toggle NWP slot, 0 otherwise.

- None

Returns

- sl_status_t SL_STATUS_OK if successful, error code otherwise. For more information on status codes, refer to SL STATUS DOCUMENTATION.

sl_si91x_get_m4_app_addr

```
sl_status_t sl_si91x_get_m4_app_addr (uint32_t * app_addr)
```

Retrieves the M4 application address from the active firmware slot. This function reads firmware slot information from flash memory, verifies its integrity using a magic word and CRC check, and determines the application address for the currently active M4 slot. This function will work for A/B firmware only.

Parameters

Type	Direction	Argument Name	Description
uint32_t *	[out]	app_addr	Pointer to store the retrieved application address.

- None

If a CRC failure is detected in the primary slot information,
the function attempts to retrieve the slot information from a backup slot. Returns

-

SL_STATUS_OK If the operation is successful.
- SL_SI91X_AB_ERR_FLASH_READ If reading slot information from flash fails.
- SL_SI91X_AB_ERR_MAGIC_NUMBER If the magic word does not match the expected value.
- SL_SI91X_AB_ERR_CRC_MISMATCH If the CRC verification fails.
- SL_STATUS_INVALID_PARAMETER If the active slot is invalid. For more information on status codes, refer to SL STATUS DOCUMENTATION.

sl_si91x_jump_to_m4_application

```
void sl_si91x_jump_to_m4_application (uint32_t app_addr)
```

Jumps to the M4 application.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	app_addr	Address of the application to jump to.

This function disables interrupts, fetches the stack pointer and reset handler from the application's vector table, sets the VTOR to point to the new application vector table, and branches to the reset handler. This function will work for A/B firmware only.

sl_si91x_verify_image

```
sl_status_t sl_si91x_verify_image (uint32_t address)
```

Checks the integrity of the firmware for Fallback. This function sends a command to check the integrity of the firmware at the specified flash memory address. It populates the request structure with the necessary parameters and sends the command to the firmware. This function will work for A/B firmware only.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	address	Flash memory address to check integrity.

- None

Returns

- sl_status_t SL_STATUS_OK on success, error code otherwise. For more information on status codes, refer to SL STATUS DOCUMENTATION.

sl_si91x_select_default_nwp_fw

```
int16_t sl_si91x_select_default_nwp_fw (const uint8_t fw_image_number)
```

Selects the default NWP firmware image to load. This function sends the appropriate command to select which firmware image (0 for slot A, 1 for slot B) should be loaded on boot.

Parameters

Type	Direction	Argument Name	Description
const uint8_t	[in]	fw_image_number	Firmware image number to select (0 for slot A, 1 for slot B)

- None

Returns

- int16_t - Error code
  - 0 : SUCCESS
  - < 0 : Command issue failed
  - -14 : Valid Firmware not present
  - -15 : Invalid Option/Command not supported For more information on status codes, refer to SL STATUS DOCUMENTATION.

sl_si91x_burn_nwp_security_version

```
sl_status_t sl_si91x_burn_nwp_security_version (uint32_t flash_address)
```

Burns the NWP security version to OTP memory to prevent firmware rollback. Call this function after validating a new NWP firmware version and before using it. This operation is irreversible - once burned, only firmware with equal or higher security versions can be loaded.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flash_address	Base address of NWP firmware in flash memory.

- 1. Valid NWP firmware must be present at the specified flash address.
  1. OTP memory must not be programmed with a higher security version.

Returns

- sl_status_t SL_STATUS_OK on success, error code otherwise. For more information on status codes, refer to SL STATUS DOCUMENTATION.

sl_si91x_fallback_load_qspi_keys

```
sl_status_t sl_si91x_fallback_load_qspi_keys (uint32_t image_offset)
```

Loads QSPI keys for encrypted firmware execution. Call this function:

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	image_offset	Base address of encrypted firmware in flash.

- 1. Firmware encryption must be enabled.
  1. Valid firmware must exist at the specified offset.
  - After system reset when using encrypted firmware
  - Before switching to a different encrypted firmware slot
  - Before performing NWP soft reset with encrypted firmware

Returns

sl_si91x_nwp_soft_reset_from_updater

```
void sl_si91x_nwp_soft_reset_from_updater (const uint32_t m4_slot_image_offset)
```

Performs a soft reset of the NWP firmware during A/B firmware fallback operations. This function is a critical part of the firmware update process that:

Parameters

Type	Direction	Argument Name	Description
const uint32_t	[in]	m4_slot_image_offset	Base address of M4 firmware in active slot (A or B). This address points to the firmware image that will be used after the reset.

- 1. Device must be configured with A/B firmware slots.
    1. M4 updater firmware must be running.
    2. Active slot information must be retrieved.
    3. Image verification must be successful.
    4. QSPI keys must be loaded.
  - Ensures safe transition between firmware versions
  - Maintains system stability during updates
  - Prepares NWP for firmware switch
  - Enables fallback to previous version if needed

The function operates in a dual-bank system where two firmware slots (A and B) are maintained. This design allows one slot to remain as a backup while the other is being updated, ensuring system recovery capability if an update fails.

- 1. After this API call, the updater application must branch to the active slot application. No other operations should not be performed after this API call.
    1. No NWP operations allowed between reset and M4 updater

ota_image_info_t

Structure for OTA image information.

Public Attributes

uint8_t	ota_image_type OTA image type.
uint32_t	ota_image_offset OTA image offset.
uint32_t	ota_image_size OTA image size.

Public Attribute Documentation

ota_image_type

```
uint8_t ota_image_info_t::ota_image_type
```

OTA image type.

ota_image_offset

```
uint32_t ota_image_info_t::ota_image_offset
```

OTA image offset.

ota_image_size

```
uint32_t ota_image_info_t::ota_image_size
```

OTA image size.

sl_si91x_fw_fallback_request_t

Firmware update request structure.

Public Attributes

uint16_t	chunk_type Header (1) or Data (0)
uint16_t	sub_command Operation type: WRITE (1), READ (2), ERASE (3), INTEGRITY_CHECK (4)
uint32_t	data_length Length of data (for erase, must be a multiple of 4K)
uint32_t	flash_offset Flash memory address: M4 -> 0x8000000, NWP -> 0x4000000.
uint8_t	m4_updater_ota 0xEF if the updater is M4
uint8_t	reserved1 Reserved for future use.
uint16_t	reserved2 Reserved for future use.
uint32_t	reserved3 Reserved for future use.
uint8_t	data Data buffer (used for WRITE operation, NULL for READ, ERASE, and INTEGRITY_CHECK)

Public Attribute Documentation

chunk_type

```
uint16_t sl_si91x_fw_fallback_request_t::chunk_type
```

Header (1) or Data (0)

sub_command

```
uint16_t sl_si91x_fw_fallback_request_t::sub_command
```

Operation type: WRITE (1), READ (2), ERASE (3), INTEGRITY_CHECK (4)

data_length

```
uint32_t sl_si91x_fw_fallback_request_t::data_length
```

Length of data (for erase, must be a multiple of 4K)

flash_offset

```
uint32_t sl_si91x_fw_fallback_request_t::flash_offset
```

Flash memory address: M4 -> 0x8000000, NWP -> 0x4000000.

m4_updater_ota

```
uint8_t sl_si91x_fw_fallback_request_t::m4_updater_ota
```

0xEF if the updater is M4

reserved1

```
uint8_t sl_si91x_fw_fallback_request_t::reserved1
```

Reserved for future use.

reserved2

```
uint16_t sl_si91x_fw_fallback_request_t::reserved2
```

Reserved for future use.

reserved3

```
uint32_t sl_si91x_fw_fallback_request_t::reserved3
```

Reserved for future use.

data

```
uint8_t sl_si91x_fw_fallback_request_t::data[SL_SI91X_MAX_OTA_IMAGE_CHUNK_SIZE]
```

Data buffer (used for WRITE operation, NULL for READ, ERASE, and INTEGRITY_CHECK)

sl_si91x_fw_fallback_config_t

Configuration structure for firmware fallback requests.

Public Attributes

uint32_t	ota_image_data_length Length of data (for erase, must be a multiple of 4K)
uint32_t	ota_image_flash_offset Flash memory address: M4 -> 0x8000000, NWP -> 0x4000000.
uint16_t	ota_image_chunk_type Header (1) or Data (0)
uint8_t	m4_updater_ota_flag 0xEF if the M4 updater is used for OTA update
uint8_t	ota_image_type Image type: 0=application, 1=updater.
uint8_t	reserved1 Reserved for future use.
uint16_t	reserved2 Reserved for future use.
uint32_t	reserved3 Reserved for future use.

Public Attribute Documentation

ota_image_data_length

```
uint32_t sl_si91x_fw_fallback_config_t::ota_image_data_length
```

Length of data (for erase, must be a multiple of 4K)

ota_image_flash_offset

```
uint32_t sl_si91x_fw_fallback_config_t::ota_image_flash_offset
```

Flash memory address: M4 -> 0x8000000, NWP -> 0x4000000.

ota_image_chunk_type

```
uint16_t sl_si91x_fw_fallback_config_t::ota_image_chunk_type
```

Header (1) or Data (0)

m4_updater_ota_flag

```
uint8_t sl_si91x_fw_fallback_config_t::m4_updater_ota_flag
```

0xEF if the M4 updater is used for OTA update

ota_image_type

```
uint8_t sl_si91x_fw_fallback_config_t::ota_image_type
```

Image type: 0=application, 1=updater.

reserved1

```
uint8_t sl_si91x_fw_fallback_config_t::reserved1
```

Reserved for future use.

reserved2

```
uint16_t sl_si91x_fw_fallback_config_t::reserved2
```

Reserved for future use.

reserved3

```
uint32_t sl_si91x_fw_fallback_config_t::reserved3
```

Reserved for future use.

SlotInfo_t

Structure for slot information.

Public Attributes

uint8_t	slot_id Slot Identifier (e.g., SLOT_A, SLOT_B)
uint32_t	slot_image_offset Base address of the slot in flash memory.
uint32_t	image_size Size of the firmware image in bytes.

Public Attribute Documentation

slot_id

```
uint8_t SlotInfo_t::slot_id
```

Slot Identifier (e.g., SLOT_A, SLOT_B)

slot_image_offset

```
uint32_t SlotInfo_t::slot_image_offset
```

Base address of the slot in flash memory.

image_size

```
uint32_t SlotInfo_t::image_size
```

Size of the firmware image in bytes.

M4_SlotManagement_t

M4 slot management structure.

Public Attributes

SlotInfo_t	m4_slot_A Information for Slot A.
SlotInfo_t	m4_slot_B Information for Slot B.
uint8_t	current_active_M4_slot Currently active slot (A or B)

Public Attribute Documentation

m4_slot_A

```
SlotInfo_t M4_SlotManagement_t::m4_slot_A
```

Information for Slot A.

m4_slot_B

```
SlotInfo_t M4_SlotManagement_t::m4_slot_B
```

Information for Slot B.

current_active_M4_slot

```
uint8_t M4_SlotManagement_t::current_active_M4_slot
```

Currently active slot (A or B)

NWP_SlotManagement_t

NWP slot management structure.

Public Attributes

SlotInfo_t	nwp_slot_A Information for NWP Slot A.
SlotInfo_t	nwp_slot_B Information for NWP Slot B.
uint8_t	current_active_nwp_slot Currently active NWP slot (A or B)

Public Attribute Documentation

nwp_slot_A

```
SlotInfo_t NWP_SlotManagement_t::nwp_slot_A
```

Information for NWP Slot A.

nwp_slot_B

```
SlotInfo_t NWP_SlotManagement_t::nwp_slot_B
```

Information for NWP Slot B.

current_active_nwp_slot

```
uint8_t NWP_SlotManagement_t::current_active_nwp_slot
```

Currently active NWP slot (A or B)

sl_si91x_fw_ab_slot_management_t

Firmware slot management structure.

Public Attributes

uint32_t	slot_magic_word Magic word for slot integrity verification (0xA5A5B5B5)
M4_SlotManagement_t	m4_slot_info Slot information for M4.
NWP_SlotManagement_t	nwp_slot_info Slot information for NWP.
uint32_t	slot_struct_crc CRC for the slot information structure.

Public Attribute Documentation

slot_magic_word

```
uint32_t sl_si91x_fw_ab_slot_management_t::slot_magic_word
```

Magic word for slot integrity verification (0xA5A5B5B5)

m4_slot_info

```
M4_SlotManagement_t sl_si91x_fw_ab_slot_management_t::m4_slot_info
```

Slot information for M4.

nwp_slot_info

```
NWP_SlotManagement_t sl_si91x_fw_ab_slot_management_t::nwp_slot_info
```

Slot information for NWP.

slot_struct_crc

```
uint32_t sl_si91x_fw_ab_slot_management_t::slot_struct_crc
```

CRC for the slot information structure.

Services

Overview

Si91x-Platform's service layer provides APIs for efficient power management, input-output (I/O) operations, memory handling, timers, and more.

The service layer APIs can be used to easily integrate these features into any application, abstracting the complexity of the underlying layers.

Note: For more details about API status codes, see the Status Code Documentation.

Overview of Sleep Modes

- Standby Mode: The system enters a lower power state but retains memory. Some peripherals remain active to allow the system to wake up through interrupts, typically maintaining more functionality than in sleep mode.
- Sleep Mode: The CPU enters a low-power state with limited peripheral activity. This mode offers quicker wake ups than deeper sleep modes but consumes more power than deep sleep. External events or timers typically trigger wake ups.
- Deep Sleep Mode: The system achieves the maximum power saving by shutting down most of its components, including the CPU and peripherals, while preserving the system state in memory. Limited wake up sources are available, including critical peripherals and timers. This mode saves more power than Standby or Sleep Mode, but the wake up time is longer due to the deeper sleep state.

Peripheral Availability Across Sleep Modes

For information on the peripherals available in different sleep modes, see the Wakeup Sources section in the SiWx91x Reference Manual.