SCTimer: SCTimer/PWM (SCT)

Overview

The MCUXpresso SDK provides a driver for the SCTimer Module (SCT) of MCUXpresso SDK devices.

Function groups

The SCTimer driver supports the generation of PWM signals. The driver also supports enabling events in various states of the SCTimer and the actions that will be triggered when an event occurs.

Initialization and deinitialization

The function **SCTIMER_Init()** initializes the SCTimer with specified configurations. The function **SCTIMER_GetDefaultConfig()** gets the default configurations.

The function **SCTIMER_Deinit()** halts the SCTimer counter and turns off the module clock.

PWM Operations

The function **SCTIMER_SetupPwm()** sets up SCTimer channels for PWM output. The function can set up the PWM signal properties duty cycle and level-mode (active low or high) to use. However, the same PWM period and PWM mode (edge or center-aligned) is applied to all channels requesting the PWM output. The signal duty cycle is provided as a percentage of the PWM period. Its value should be between 1 and 100.

The function **SCTIMER_UpdatePwmDutycycle()** updates the PWM signal duty cycle of a particular SCTimer channel.

Status

Provides functions to get and clear the SCTimer status.

Interrupt

Provides functions to enable/disable SCTimer interrupts and get current enabled interrupts.

SCTimer State machine and operations

The SCTimer has 10 states and each state can have a set of events enabled that can trigger a user specified action when the event occurs.

SCTimer event operations

The user can create an event and enable it in the current state using the functions **SCTIMER_CreateAndScheduleEvent()** and **SCTIMER_ScheduleEvent()**. **SCTIMER_CreateAndScheduleEvent()** creates a new event based on the users preference and enables it in the current state. **SCTIMER_ScheduleEvent()** enables an event created earlier in the current state.

SCTimer state operations

The user can get the current state number by calling **SCTIMER_GetCurrentState()**, he can use this state number to set state transitions when a particular event is triggered.

Once the user has created and enabled events for the current state he can go to the next state by calling the function **SCTIMER_IncreaseState()**. The user can then start creating events to be enabled in this new state.

SCTimer action operations

There are a set of functions that decide what action should be taken when an event is triggered. **SCTIMER_SetupCaptureAction()** sets up which counter to capture and which capture register to read on event trigger. **SCTIMER_SetupNextStateAction()** sets up which state the SCTimer state machine should transition to on event trigger. **SCTIMER_SetupOutputSetAction()** sets up which pin to set on event trigger. **SCTIMER_SetupOutputClearAction()** sets up which pin to clear on event trigger. **SCTIMER_SetupOutputToggleAction()** sets up which pin to toggle on event trigger. **SCTIMER_SetupCounterLimitAction()** sets up which counter will be limited on event trigger. **SCTIMER_SetupCounterStopAction()** sets up which counter will be stopped on event trigger. **SCTIMER_SetupCounterStartAction()** sets up which counter will be started on event trigger. **SCTIMER_SetupCounterHaltAction()** sets up which counter will be halted on event trigger. **SCTIMER_SetupDmaTriggerAction()** sets up which DMA request will be activated on event trigger.

16-bit counter mode

The SCTimer is configurable to run as two 16-bit counters via the enableCounterUnify flag that is available in the configuration structure passed in to the **SCTIMER_Init()** function.

When operating in 16-bit mode, it is important the user specify the appropriate counter to use when working with the functions: **SCTIMER_StartTimer()**, **SCTIMER_StopTimer()**, **SCTIMER_CreateAndScheduleEvent()**, **SCTIMER_SetupCaptureAction()**, **SCTIMER_SetupCounterLimitAction()**, **SCTIMER_SetupCounterStopAction()**, **SCTIMER_SetupCounterStartAction()**, **SCTIMER_SetupCounterHaltAction()**.

Typical use case

PWM output

Output a PWM signal on 2 SCTimer channels with different duty cycles.

```
int main(void)
{
    sctimer_config_t sctimerInfo;
    sctimer_pwm_signal_param_t pwmParam;
    uint32_t event;
    uint32_t sctimerClock;

    /* Board pin, clock, debug console init */
    BOARD_InitHardware();

    sctimerClock = CLOCK_GetFreq(kCLOCK_BusClk);

    /* Print a note to terminal */
    PRINTF("\r\nSCTimer example to output 2 center-aligned PWM signals\r\n");
    PRINTF("\r\nYou will see a change in LED brightness if an LED is connected to
       the SCTimer output pins");
    PRINTF("\r\nIf no LED is connected to the pin, then probe the signal using an
       oscilloscope");

    SCTIMER_GetDefaultConfig(&sctimerInfo);

    /* Initialize SCTimer module */
    SCTIMER_Init(SCT0, &sctimerInfo);

    /* Configure first PWM with frequency 24kHZ from output 4 */
    pwmParam.output = kSCTIMER_Out_4;
    pwmParam.level = kSCTIMER_HighTrue;
    pwmParam.dutyCyclePercent = 50;
    if (SCTIMER_SetupPwm(SCT0, &pwmParam, kSCTIMER_CenterAlignedPwm, 24000U,
       sctimerClock, &event) == kStatus_Fail)
    {
        return -1;
    }

    /* Configure second PWM with different duty cycle but same frequency as before
       */
    pwmParam.output = kSCTIMER_Out_2;
    pwmParam.level = kSCTIMER_LowTrue;
    pwmParam.dutyCyclePercent = 20;
    if (SCTIMER_SetupPwm(SCT0, &pwmParam, kSCTIMER_CenterAlignedPwm, 24000U,
       sctimerClock, &event) == kStatus_Fail)
    {
        return -1;
    }

    /* Start the timer */
    SCTIMER_StartTimer(SCT0, kSCTIMER_Counter_L);

    while (1)
    {
    }
}
```

Files

file **fsl_sctimer.h**

Data Structures

struct **sctimer_pwm_signal_param_t**

	Options to configure a SCTimer PWM signal. More...
struct	**sctimer_config_t** SCTimer configuration structure. More...

Typedefs

typedef void(*	**sctimer_event_callback_t**)(void) SCTimer callback typedef. More...

Enumerations

enum	**sctimer_pwm_mode_t** {
	kSCTIMER_EdgeAlignedPwm = 0U,
	kSCTIMER_CenterAlignedPwm
	}
	SCTimer PWM operation modes. More...

enum	**sctimer_counter_t** {
	kSCTIMER_Counter_L = 0U,
	kSCTIMER_Counter_H
	}
	SCTimer counters when working as two independent 16-bit counters. More...

enum	**sctimer_input_t** {
	kSCTIMER_Input_0 = 0U,
	kSCTIMER_Input_1,
	kSCTIMER_Input_2,
	kSCTIMER_Input_3,
	kSCTIMER_Input_4,
	kSCTIMER_Input_5,
	kSCTIMER_Input_6,
	kSCTIMER_Input_7
	}
	List of SCTimer input pins. More...

enum	**sctimer_out_t** { **kSCTIMER_Out_0** = 0U, **kSCTIMER_Out_1**, **kSCTIMER_Out_2**, **kSCTIMER_Out_3**, **kSCTIMER_Out_4**, **kSCTIMER_Out_5**,

	kSCTIMER_Out_6, **kSCTIMER_Out_7** } List of SCTimer output pins. More...

enum	**sctimer_pwm_level_select_t** {
	kSCTIMER_LowTrue = 0U,
	kSCTIMER_HighTrue
	}
	SCTimer PWM output pulse mode: high-true, low-true or no output. More...

enum	**sctimer_clock_mode_t** {
	kSCTIMER_System_ClockMode = 0U,
	kSCTIMER_Sampled_ClockMode,
	kSCTIMER_Input_ClockMode,
	kSCTIMER_Asynchronous_ClockMode
	}
	SCTimer clock mode options. More...

enum	**sctimer_clock_select_t** {
	kSCTIMER_Clock_On_Rise_Input_0 = 0U,
	kSCTIMER_Clock_On_Fall_Input_0,
	kSCTIMER_Clock_On_Rise_Input_1,
	kSCTIMER_Clock_On_Fall_Input_1,
	kSCTIMER_Clock_On_Rise_Input_2,
	kSCTIMER_Clock_On_Fall_Input_2,
	kSCTIMER_Clock_On_Rise_Input_3,
	kSCTIMER_Clock_On_Fall_Input_3,
	kSCTIMER_Clock_On_Rise_Input_4,
	kSCTIMER_Clock_On_Fall_Input_4,
	kSCTIMER_Clock_On_Rise_Input_5,
	kSCTIMER_Clock_On_Fall_Input_5,
	kSCTIMER_Clock_On_Rise_Input_6,
	kSCTIMER_Clock_On_Fall_Input_6,
	kSCTIMER_Clock_On_Rise_Input_7,
	kSCTIMER_Clock_On_Fall_Input_7
	}
	SCTimer clock select options. More...

enum	**sctimer_conflict_resolution_t** { **kSCTIMER_ResolveNone** = 0U, **kSCTIMER_ResolveSet**,

	kSCTIMER_ResolveClear, **kSCTIMER_ResolveToggle** } SCTimer output conflict resolution options. More...
enum	**sctimer_event_t** List of SCTimer event types.

enum	**sctimer_interrupt_enable_t** {
	kSCTIMER_Event0InterruptEnable = (1U << 0),
	kSCTIMER_Event1InterruptEnable = (1U << 1),
	kSCTIMER_Event2InterruptEnable = (1U << 2),
	kSCTIMER_Event3InterruptEnable = (1U << 3),
	kSCTIMER_Event4InterruptEnable = (1U << 4),
	kSCTIMER_Event5InterruptEnable = (1U << 5),
	kSCTIMER_Event6InterruptEnable = (1U << 6),
	kSCTIMER_Event7InterruptEnable = (1U << 7),
	kSCTIMER_Event8InterruptEnable = (1U << 8),
	kSCTIMER_Event9InterruptEnable = (1U << 9),
	kSCTIMER_Event10InterruptEnable = (1U << 10),
	kSCTIMER_Event11InterruptEnable = (1U << 11),
	kSCTIMER_Event12InterruptEnable = (1U << 12)
	}
	List of SCTimer interrupts. More...

enum	**sctimer_status_flags_t** {
	kSCTIMER_Event0Flag = (1U << 0),
	kSCTIMER_Event1Flag = (1U << 1),
	kSCTIMER_Event2Flag = (1U << 2),
	kSCTIMER_Event3Flag = (1U << 3),
	kSCTIMER_Event4Flag = (1U << 4),
	kSCTIMER_Event5Flag = (1U << 5),
	kSCTIMER_Event6Flag = (1U << 6),
	kSCTIMER_Event7Flag = (1U << 7),
	kSCTIMER_Event8Flag = (1U << 8),
	kSCTIMER_Event9Flag = (1U << 9),
	kSCTIMER_Event10Flag = (1U << 10),
	kSCTIMER_Event11Flag = (1U << 11),
	kSCTIMER_Event12Flag = (1U << 12),
	kSCTIMER_BusErrorLFlag,
	kSCTIMER_BusErrorHFlag
	}

List of SCTimer flags. More...

Driver version

#define	**FSL_SCTIMER_DRIVER_VERSION** (**MAKE_VERSION**(2, 0, 0)) Version 2.0.0.

Initialization and deinitialization

status_t	**SCTIMER_Init** (SCT_Type *base, const **sctimer_config_t** *config) Ungates the SCTimer clock and configures the peripheral for basic operation. More...
void	**SCTIMER_Deinit** (SCT_Type *base) Gates the SCTimer clock. More...
void	**SCTIMER_GetDefaultConfig** (**sctimer_config_t** *config) Fills in the SCTimer configuration structure with the default settings. More...

PWM setup operations

status_t	**SCTIMER_SetupPwm** (SCT_Type *base, const **sctimer_pwm_signal_param_t** *pwmParams, **sctimer_pwm_mode_t** mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz, uint32_t *event) Configures the PWM signal parameters. More...
void	**SCTIMER_UpdatePwmDutycycle** (SCT_Type *base, **sctimer_out_t** output, uint8_t dutyCyclePercent, uint32_t event) Updates the duty cycle of an active PWM signal. More...

Interrupt Interface

static void	**SCTIMER_EnableInterrupts** (SCT_Type *base, uint32_t mask) Enables the selected SCTimer interrupts. More...
static void	**SCTIMER_DisableInterrupts** (SCT_Type *base, uint32_t mask) Disables the selected SCTimer interrupts. More...
static uint32_t	**SCTIMER_GetEnabledInterrupts** (SCT_Type *base) Gets the enabled SCTimer interrupts. More...

Status Interface

static uint32_t	**SCTIMER_GetStatusFlags** (SCT_Type *base) Gets the SCTimer status flags. More...

static void	**SCTIMER_ClearStatusFlags** (SCT_Type *base, uint32_t mask) Clears the SCTimer status flags. More...

Counter Start and Stop

static void	**SCTIMER_StartTimer** (SCT_Type *base, **sctimer_counter_t** countertoStart) Starts the SCTimer counter. More...
static void	**SCTIMER_StopTimer** (SCT_Type *base, **sctimer_counter_t** countertoStop) Halts the SCTimer counter. More...

Functions to create a new event and manage the state logic

status_t	**SCTIMER_CreateAndScheduleEvent** (SCT_Type *base, **sctimer_event_t** howToMonitor, uint32_t matchValue, uint32_t whichIO, **sctimer_counter_t** whichCounter, uint32_t *event) Create an event that is triggered on a match or IO and schedule in current state. More...
void	**SCTIMER_ScheduleEvent** (SCT_Type *base, uint32_t event) Enable an event in the current state. More...
status_t	**SCTIMER_IncreaseState** (SCT_Type *base) Increase the state by 1. More...
uint32_t	**SCTIMER_GetCurrentState** (SCT_Type *base) Provides the current state. More...

Actions to take in response to an event

status_t	**SCTIMER_SetupCaptureAction** (SCT_Type *base, **sctimer_counter_t** whichCounter, uint32_t *captureRegister, uint32_t event) Setup capture of the counter value on trigger of a selected event. More...
void	**SCTIMER_SetCallback** (SCT_Type *base, **sctimer_event_callback_t** callback, uint32_t event) Receive noticification when the event trigger an interrupt. More...
static void	**SCTIMER_SetupNextStateAction** (SCT_Type *base, uint32_t nextState, uint32_t event) Transition to the specified state. More...
static void	**SCTIMER_SetupOutputSetAction** (SCT_Type *base, uint32_t whichIO, uint32_t event) Set the Output. More...
static void	**SCTIMER_SetupOutputClearAction** (SCT_Type *base, uint32_t whichIO, uint32_t event) Clear the Output. More...
void	**SCTIMER_SetupOutputToggleAction** (SCT_Type *base, uint32_t whichIO, uint32_t event)

	Toggle the output level. More...
static void	**SCTIMER_SetupCounterLimitAction** (SCT_Type *base, **sctimer_counter_t** whichCounter, uint32_t event) Limit the running counter. More...
static void	**SCTIMER_SetupCounterStopAction** (SCT_Type *base, **sctimer_counter_t** whichCounter, uint32_t event) Stop the running counter. More...
static void	**SCTIMER_SetupCounterStartAction** (SCT_Type *base, **sctimer_counter_t** whichCounter, uint32_t event) Re-start the stopped counter. More...
static void	**SCTIMER_SetupCounterHaltAction** (SCT_Type *base, **sctimer_counter_t** whichCounter, uint32_t event) Halt the running counter. More...
static void	**SCTIMER_SetupDmaTriggerAction** (SCT_Type *base, uint32_t dmaNumber, uint32_t event) Generate a DMA request. More...
void	**SCTIMER_EventHandleIRQ** (SCT_Type *base) SCTimer interrupt handler. More...

Data Structure Documentation

struct sctimer_pwm_signal_param_t

Data Fields

sctimer_out_t	**output** The output pin to use to generate the PWM signal.
sctimer_pwm_level_select_t	**level** PWM output active level select. More...
uint8_t	**dutyCyclePercent** PWM pulse width, value should be between 1 to 100 100 = always active signal (100% duty cycle). More...

Field Documentation

sctimer_pwm_level_select_t sctimer_pwm_signal_param_t::level

uint8_t sctimer_pwm_signal_param_t::dutyCyclePercent

struct sctimer_config_t

This structure holds the configuration settings for the SCTimer peripheral. To initialize this structure to reasonable defaults, call the SCTMR_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

Data Fields

bool	**enableCounterUnify** true: SCT operates as a unified 32-bit counter; false: SCT operates as two 16-bit counters
sctimer_clock_mode_t	**clockMode** SCT clock mode value.
sctimer_clock_select_t	**clockSelect** SCT clock select value.
bool	**enableBidirection_l** true: Up-down count mode for the L or unified counter false: Up count mode only for the L or unified counter
bool	**enableBidirection_h** true: Up-down count mode for the H or unified counter false: Up count mode only for the H or unified counter. More...
uint8_t	**prescale_l** Prescale value to produce the L or unified counter clock.
uint8_t	**prescale_h** Prescale value to produce the H counter clock. More...
uint8_t	**outInitState** Defines the initial output value.

Field Documentation

bool sctimer_config_t::enableBidirection_h

This field is used only if the enableCounterUnify is set to false

uint8_t sctimer_config_t::prescale_h

This field is used only if the enableCounterUnify is set to false

Typedef Documentation

typedef void(* sctimer_event_callback_t)(void)

Enumeration Type Documentation

enum sctimer_pwm_mode_t

Enumerator	
kSCTIMER_EdgeAlignedPwm	Edge-aligned PWM.
kSCTIMER_CenterAlignedPwm	Center-aligned PWM.

enum sctimer_counter_t

Enumerator	
kSCTIMER_Counter_L	Counter L.
kSCTIMER_Counter_H	Counter H.

enum sctimer_input_t

Enumerator	
kSCTIMER_Input_0	SCTIMER input 0.
kSCTIMER_Input_1	SCTIMER input 1.
kSCTIMER_Input_2	SCTIMER input 2.
kSCTIMER_Input_3	SCTIMER input 3.
kSCTIMER_Input_4	SCTIMER input 4.
kSCTIMER_Input_5	SCTIMER input 5.
kSCTIMER_Input_6	SCTIMER input 6.
kSCTIMER_Input_7	SCTIMER input 7.

enum sctimer_out_t

Enumerator	
kSCTIMER_Out_0	SCTIMER output 0.
kSCTIMER_Out_1	SCTIMER output 1.
kSCTIMER_Out_2	SCTIMER output 2.
kSCTIMER_Out_3	SCTIMER output 3.
kSCTIMER_Out_4	SCTIMER output 4.
kSCTIMER_Out_5	SCTIMER output 5.
kSCTIMER_Out_6	SCTIMER output 6.
kSCTIMER_Out_7	SCTIMER output 7.

enum sctimer_pwm_level_select_t

Enumerator	
kSCTIMER_LowTrue	Low true pulses.
kSCTIMER_HighTrue	High true pulses.

enum sctimer_clock_mode_t

Enumerator	
kSCTIMER_System_ClockMode	System Clock Mode.
kSCTIMER_Sampled_ClockMode	Sampled System Clock Mode.
kSCTIMER_Input_ClockMode	SCT Input Clock Mode.
kSCTIMER_Asynchronous_ClockMode	Asynchronous Mode.

enum sctimer_clock_select_t

Enumerator	
kSCTIMER_Clock_On_Rise_Input_0	Rising edges on input 0.
kSCTIMER_Clock_On_Fall_Input_0	Falling edges on input 0.
kSCTIMER_Clock_On_Rise_Input_1	Rising edges on input 1.
kSCTIMER_Clock_On_Fall_Input_1	Falling edges on input 1.
kSCTIMER_Clock_On_Rise_Input_2	Rising edges on input 2.
kSCTIMER_Clock_On_Fall_Input_2	Falling edges on input 2.
kSCTIMER_Clock_On_Rise_Input_3	Rising edges on input 3.
kSCTIMER_Clock_On_Fall_Input_3	Falling edges on input 3.
kSCTIMER_Clock_On_Rise_Input_4	Rising edges on input 4.
kSCTIMER_Clock_On_Fall_Input_4	Falling edges on input 4.
kSCTIMER_Clock_On_Rise_Input_5	Rising edges on input 5.
kSCTIMER_Clock_On_Fall_Input_5	Falling edges on input 5.
kSCTIMER_Clock_On_Rise_Input_6	Rising edges on input 6.
kSCTIMER_Clock_On_Fall_Input_6	Falling edges on input 6.
kSCTIMER_Clock_On_Rise_Input_7	Rising edges on input 7.
kSCTIMER_Clock_On_Fall_Input_7	Falling edges on input 7.

enum sctimer_conflict_resolution_t

Specifies what action should be taken if multiple events dictate that a given output should be both set and cleared at the same time

Enumerator	
kSCTIMER_ResolveNone	No change.
kSCTIMER_ResolveSet	Set output.
kSCTIMER_ResolveClear	Clear output.
kSCTIMER_ResolveToggle	Toggle output.

enum sctimer_interrupt_enable_t

Enumerator	
kSCTIMER_Event0InterruptEnable	Event 0 interrupt.
kSCTIMER_Event1InterruptEnable	Event 1 interrupt.
kSCTIMER_Event2InterruptEnable	Event 2 interrupt.
kSCTIMER_Event3InterruptEnable	Event 3 interrupt.
kSCTIMER_Event4InterruptEnable	Event 4 interrupt.
kSCTIMER_Event5InterruptEnable	Event 5 interrupt.
kSCTIMER_Event6InterruptEnable	Event 6 interrupt.
kSCTIMER_Event7InterruptEnable	Event 7 interrupt.
kSCTIMER_Event8InterruptEnable	Event 8 interrupt.
kSCTIMER_Event9InterruptEnable	Event 9 interrupt.
kSCTIMER_Event10InterruptEnable	Event 10 interrupt.
kSCTIMER_Event11InterruptEnable	Event 11 interrupt.
kSCTIMER_Event12InterruptEnable	Event 12 interrupt.

enum sctimer_status_flags_t

Enumerator	
kSCTIMER_Event0Flag	Event 0 Flag.
kSCTIMER_Event1Flag	Event 1 Flag.
kSCTIMER_Event2Flag	Event 2 Flag.
kSCTIMER_Event3Flag	Event 3 Flag.
kSCTIMER_Event4Flag	Event 4 Flag.
kSCTIMER_Event5Flag	Event 5 Flag.
kSCTIMER_Event6Flag	Event 6 Flag.
kSCTIMER_Event7Flag	Event 7 Flag.
kSCTIMER_Event8Flag	Event 8 Flag.
kSCTIMER_Event9Flag	Event 9 Flag.
kSCTIMER_Event10Flag	Event 10 Flag.
kSCTIMER_Event11Flag	Event 11 Flag.
kSCTIMER_Event12Flag	Event 12 Flag.
kSCTIMER_BusErrorLFlag	Bus error due to write when L counter was not halted.
kSCTIMER_BusErrorHFlag	Bus error due to write when H counter was not halted.

Function Documentation

status_t SCTIMER_Init (SCT_Type * base,
const sctimer_config_t * config
)

Note

This API should be called at the beginning of the application using the SCTimer driver.

Parameters

base	SCTimer peripheral base address
config	Pointer to the user configuration structure.

Returns

kStatus_Success indicates success; Else indicates failure.

void SCTIMER_Deinit (SCT_Type * base)

Parameters

base	SCTimer peripheral base address

void SCTIMER_GetDefaultConfig (sctimer_config_t * config)

The default values are:

```
*   config->enableCounterUnify = true;
*   config->clockMode = kSCTIMER_System_ClockMode;
*   config->clockSelect = kSCTIMER_Clock_On_Rise_Input_0;
*   config->enableBidirection_l = false;
*   config->enableBidirection_h = false;
*   config->prescale_l = 0;
*   config->prescale_h = 0;
*   config->outInitState = 0;
*
```

Parameters

config	Pointer to the user configuration structure.

```
status_t SCTIMER_SetupPwm ( SCT_Type *                          base,
                            const sctimer_pwm_signal_param_t *  pwmParams,
                            sctimer_pwm_mode_t                  mode,
                            uint32_t                            pwmFreq_Hz,
                            uint32_t                            srcClock_Hz,
                            uint32_t *                          event
                          )
```

Call this function to configure the PWM signal period, mode, duty cycle, and edge. This function will create 2 events; one of the events will trigger on match with the pulse value and the other will trigger when the counter matches the PWM period. The PWM period event is also used as a limit event to reset the counter or change direction. Both events are enabled for the same state. The state number can be retrieved by calling the function SCTIMER_GetCurrentStateNumber(). The counter is set to operate as one 32-bit counter (unify bit is set to 1). The counter operates in bi-directional mode when generating a center-aligned PWM.

Note

> When setting PWM output from multiple output pins, they all should use the same PWM mode i.e all PWM's should be either edge-aligned or center-aligned. When using this API, the PWM signal frequency of all the initialized channels must be the same. Otherwise all the initialized channels' PWM signal frequency is equal to the last call to the API's pwmFreq_Hz.

Parameters

base	SCTimer peripheral base address
pwmParams	PWM parameters to configure the output
mode	PWM operation mode, options available in enumeration **sctimer_pwm_mode_t**
pwmFreq_Hz	PWM signal frequency in Hz
srcClock_Hz	SCTimer counter clock in Hz
event	Pointer to a variable where the PWM period event number is stored

Returns

kStatus_Success on success kStatus_Fail If we have hit the limit in terms of number of events created or if an incorrect PWM dutycylce is passed in.

```
void SCTIMER_UpdatePwmDutycycle ( SCT_Type *    base,
                                  sctimer_out_t output,
                                  uint8_t       dutyCyclePercent,
                                  uint32_t      event
                                )
```

Parameters

base	SCTimer peripheral base address
output	The output to configure
dutyCyclePercent	New PWM pulse width; the value should be between 1 to 100
event	Event number associated with this PWM signal. This was returned to the user by the function **SCTIMER_SetupPwm()**.

```
static void SCTIMER_EnableInterrupts ( SCT_Type * base,
                                       uint32_t   mask
                                     )                      inline  static
```

Parameters

base	SCTimer peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration **sctimer_interrupt_enable_t**

```
static void SCTIMER_DisableInterrupts ( SCT_Type * base,
                                        uint32_t   mask
                                      )                     inline  static
```

Parameters

base	SCTimer peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration **sctimer_interrupt_enable_t**

static uint32_t SCTIMER_GetEnabledInterrupts (SCT_Type * base) inline static

Parameters

base	SCTimer peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration **sctimer_interrupt_enable_t**

static uint32_t SCTIMER_GetStatusFlags (SCT_Type * base) inline static

Parameters

base	SCTimer peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration **sctimer_status_flags_t**

static void SCTIMER_ClearStatusFlags (SCT_Type * base, uint32_t mask) inline static

Parameters

base	SCTimer peripheral base address
mask	The status flags to clear. This is a logical OR of members of the enumeration **sctimer_status_flags_t**

```
static void SCTIMER_StartTimer ( SCT_Type *         base,
                                 sctimer_counter_t  countertoStart
                               )
```

inline static

Parameters

base	SCTimer peripheral base address
countertoStart	SCTimer counter to start; if unify mode is set then function always writes to HALT_L bit

```
static void SCTIMER_StopTimer ( SCT_Type *         base,
                                sctimer_counter_t  countertoStop
                              )
```

inline static

Parameters

base	SCTimer peripheral base address
countertoStop	SCTimer counter to stop; if unify mode is set then function always writes to HALT_L bit

```
status_t SCTIMER_CreateAndScheduleEvent ( SCT_Type *          base,
                                          sctimer_event_t     howToMonitor,
                                          uint32_t            matchValue,
                                          uint32_t            whichIO,
                                          sctimer_counter_t   whichCounter,
                                          uint32_t *          event
                                        )
```

This function will configure an event using the options provided by the user. If the event type uses the counter match, then the function will set the user provided match value into a match register and put this match register number into the event control register. The event is enabled for the current state and the event number is increased by one at the end. The function returns the event number; this event number can be used to configure actions to be done when this event is triggered.

Parameters

base	SCTimer peripheral base address
howToMonitor	Event type; options are available in the enumeration **sctimer_interrupt_enable_t**
matchValue	The match value that will be programmed to a match register
whichIO	The input or output that will be involved in event triggering. This field is ignored if the event type is "match only"
whichCounter	SCTimer counter to use when operating in 16-bit mode. In 32-bit mode, this field has no meaning as we have only 1 unified counter; hence ignored.
event	Pointer to a variable where the new event number is stored

Returns

kStatus_Success on success kStatus_Error if we have hit the limit in terms of number of events created or if we have reached the limit in terms of number of match registers

void SCTIMER_ScheduleEvent (SCT_Type * base,
uint32_t event
)

This function will allow the event passed in to trigger in the current state. The event must be created earlier by either calling the function **SCTIMER_SetupPwm()** or function **SCTIMER_CreateAndScheduleEvent()** .

Parameters

base	SCTimer peripheral base address
event	Event number to enable in the current state

status_t SCTIMER_IncreaseState (SCT_Type * base)

All future events created by calling the function **SCTIMER_ScheduleEvent()** will be enabled in this new state.

Parameters

base	SCTimer peripheral base address

Returns

kStatus_Success on success kStatus_Error if we have hit the limit in terms of states used

uint32_t SCTIMER_GetCurrentState (SCT_Type * base)

User can use this to set the next state by calling the function **SCTIMER_SetupNextStateAction()**.

Parameters

base	SCTimer peripheral base address

Returns

The current state

```
status_t SCTIMER_SetupCaptureAction ( SCT_Type *          base,
                                      sctimer_counter_t  whichCounter,
                                      uint32_t *         captureRegister,
                                      uint32_t           event
                                    )
```

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use when operating in 16-bit mode. In 32-bit mode, this field has no meaning as only the Counter_L bits are used.
captureRegister	Pointer to a variable where the capture register number will be returned. User can read the captured value from this register when the specified event is triggered.
event	Event number that will trigger the capture

Returns

kStatus_Success on success kStatus_Error if we have hit the limit in terms of number of match/capture registers available

```
void SCTIMER_SetCallback ( SCT_Type *                base,
                           sctimer_event_callback_t  callback,
                           uint32_t                  event
                         )
```

If the interrupt for the event is enabled by the user, then a callback can be registered which will be invoked when the event is triggered

Parameters

base	SCTimer peripheral base address
event	Event number that will trigger the interrupt
callback	Function to invoke when the event is triggered

```
static void SCTIMER_SetupNextStateAction ( SCT_Type *  base,
                                           uint32_t    nextState,
                                           uint32_t    event
                                         )
```

inline static

This transition will be triggered by the event number that is passed in by the user.

Parameters

base	SCTimer peripheral base address
nextState	The next state SCTimer will transition to
event	Event number that will trigger the state transition

```
static void SCTIMER_SetupOutputSetAction ( SCT_Type *  base,
                                           uint32_t    whichIO,
                                           uint32_t    event
                                         )
```

inline static

This output will be set when the event number that is passed in by the user is triggered.

Parameters

base	SCTimer peripheral base address
whichIO	The output to set
event	Event number that will trigger the output change

```
static void SCTIMER_SetupOutputClearAction ( SCT_Type *  base,
                                             uint32_t    whichIO,
                                             uint32_t    event
                                           )
```

inline static

This output will be cleared when the event number that is passed in by the user is triggered.

Parameters

base	SCTimer peripheral base address
whichIO	The output to clear
event	Event number that will trigger the output change

```
void SCTIMER_SetupOutputToggleAction ( SCT_Type *  base,
                                       uint32_t    whichIO,
                                       uint32_t    event
                                     )
```

This change in the output level is triggered by the event number that is passed in by the user.

Parameters

base	SCTimer peripheral base address
whichIO	The output to toggle
event	Event number that will trigger the output change

```
static void SCTIMER_SetupCounterLimitAction ( SCT_Type *        base,
                                              sctimer_counter_t whichCounter,
                                              uint32_t          event
                                            )
```

inline static

The counter is limited when the event number that is passed in by the user is triggered.

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use when operating in 16-bit mode. In 32-bit mode, this field has no meaning as only the Counter_L bits are used.
event	Event number that will trigger the counter to be limited

```
static void SCTIMER_SetupCounterStopAction ( SCT_Type *        base,
                                             sctimer_counter_t whichCounter,
                                             uint32_t          event
                                           )
```

inline static

The counter is stopped when the event number that is passed in by the user is triggered.

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use when operating in 16-bit mode. In 32-bit mode, this field has no meaning as only the Counter_L bits are used.
event	Event number that will trigger the counter to be stopped

```
static void SCTIMER_SetupCounterStartAction ( SCT_Type *          base,
                                              sctimer_counter_t   whichCounter,
                                              uint32_t            event
                                            )                                   inline static
```

The counter will re-start when the event number that is passed in by the user is triggered.

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use when operating in 16-bit mode. In 32-bit mode, this field has no meaning as only the Counter_L bits are used.
event	Event number that will trigger the counter to re-start

```
static void SCTIMER_SetupCounterHaltAction ( SCT_Type *          base,
                                             sctimer_counter_t   whichCounter,
                                             uint32_t            event
                                           )                                   inline static
```

The counter is disabled (halted) when the event number that is passed in by the user is triggered. When the counter is halted, all further events are disabled. The HALT condition can only be removed by calling the **SCTIMER_StartTimer()** function.

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use when operating in 16-bit mode. In 32-bit mode, this field has no meaning as only the Counter_L bits are used.
event	Event number that will trigger the counter to be halted

```
static void SCTIMER_SetupDmaTriggerAction ( SCT_Type *  base,
                                            uint32_t    dmaNumber,
                                            uint32_t    event
                                          )
```

inline static

DMA request will be triggered by the event number that is passed in by the user.

Parameters

base	SCTimer peripheral base address
dmaNumber	The DMA request to generate
event	Event number that will trigger the DMA request

```
void SCTIMER_EventHandleIRQ ( SCT_Type *  base )
```

Parameters

base	SCTimer peripheral base address.