Timer Driver

Version 1.0.0

Generated by Doxygen 1.8.13

Contents

Chapter 1

Timer Driver: Two part Time-Base/Capture Compare Driver

This driver took a bit more creative thinking to be able to combine both regular timer usage and more complex compare/capture usage into a single interface. Of course it became two interfaces, the timer_interface and timer_↩
cc_interface. An extra challenge was thinking through how to get the exact same driver to work for the three different types of timers on the stm32F411XE. The answer was to work around the weakest ones (TIM10 and TIM11). All fancy advanced features of TIM1 must be accessed the hard way with register reads and writes.

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

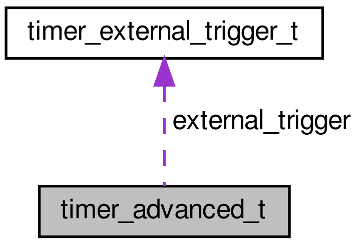
Chapter 4

Data Structure Documentation

4.1 timer_advanced_t Struct Reference

```
#include <timer_stm32f411_config.h>
```

Collaboration diagram for timer_advanced_t:



Data Fields

- timer_alt_clock_source_t clock_source
- timer_opm_t one_pulse_mode
- timer_udis_t update_event_dis
- timer_slave_mode_t slave_mode
- timer_trigger_t trigger_source
- timer_master_slave_mode_t msm
- const timer_external_trigger_t ∗ external_trigger

4.1.1 Detailed Description

Advanced structure contains access to features such as one pulse mode, update event disables, and most importantly, slave modes and exernal clock sources through slave_mode, trigger_surce, and ∗external_trigger

4.1.2 Field Documentation

4.1.2.1 clock_source

```
timer_alt_clock_source_t clock_source
```

Determines alternate clock usage

4.1.2.2 external_trigger

```
const timer_external_trigger_t* external_trigger
```

Struct containing configuration information for trigger sources external to the perpheral

4.1.2.3 msm

```
timer_master_slave_mode_t msm
```

Master/Slave behaviour. Determines whether the master and this timer synchronise themselves

4.1.2.4 one_pulse_mode

```
timer_opm_t one_pulse_mode
```

Determines whether the timer restarts after completing a count cycle

4.1.2.5 slave_mode

```
timer_slave_mode_t slave_mode
```

Determines from where the timer receives its clock and how it responds to it

4.1.2.6 trigger_source

```
timer_trigger_t trigger_source
```

Trigger sources for slave behaviour, works with slave_mode

4.1.2.7 update_event_dis

```
timer_udis_t update_event_dis
```

Determines whether the timer generates update events after count cycles

The documentation for this struct was generated from the following file:

- /home/marko/Documents/embedded_workspace/timer_driver/timer_stm32f411_config.h

4.2 timer_cc_config_t Struct Reference

```
#include <timer_cc_stm32f411_config.h>
```

Data Fields

- timer_cc_mode_t **cc_mode**
- timer_cc_output_polarity_t **output_polarity**
- timer_cc_output_fe_t **output_fast_enable**
- timer_cc_output_pe_t **output_preload_enable**
- timer_cc_output_mode_t **output_mode**
- timer_cc_output_ce_t **output_clear_enable**
- timer_cc_input_prescaler_t **input_event_prescaler**
- timer_cc_input_filter_t **input_event_filter**
- timer_cc_input_polarity_t **input_polarity**

4.2.1 Detailed Description

Dual-purpose CC config structure which is parsed differently depending on the cc_mode member.

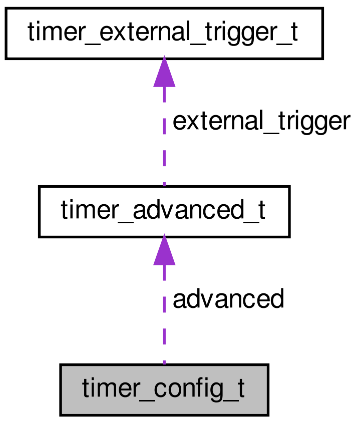
The documentation for this struct was generated from the following file:

- /home/marko/Documents/embedded_workspace/timer_driver/timer_cc_stm32f411_config.h

4.3 timer_config_t Struct Reference

```
#include <timer_stm32f411_config.h>
```

Collaboration diagram for timer_config_t:



Data Fields

- timer_alignment_t alignment
- timer_direction_t direction
- timer_prescaler_t prescaler
- uint32_t auto_reload
- timer_arpe_t auto_reload_preload_en
- const timer_advanced_t ∗ advanced

4.3.1 Detailed Description

Basic config data. Used to configure the timebase of the timer.

4.3.2 Field Documentation

4.3.2.1 advanced

```
const timer_advanced_t* advanced
```

Pointer to a structure containing more advanced options.

4.3.2.2 alignment

```
timer_alignment_t alignment
```

Determines whether the timer counts unidirectionally or bidirectionally

4.3.2.3 auto_reload

```
uint32_t auto_reload
```

The value by which the counter generates an update event and restarts counting

4.3.2.4 auto_reload_preload_en

```
timer_arpe_t auto_reload_preload_en
```

Setting deciding whether new Auto_Reload values are transferred immediately or after next update

4.3.2.5 direction

```
timer_direction_t direction
```

Determines count direction when unidirectional

4.3.2.6 prescaler

```
timer_prescaler_t prescaler
```

Division factor of peripheral input clock

The documentation for this struct was generated from the following file:

- /home/marko/Documents/embedded_workspace/timer_driver/timer_stm32f411_config.h

4.4 timer_external_trigger_t Struct Reference

```
#include <timer_stm32f411_config.h>
```

Data Fields

- timer_digital_filter_clock_div_t dts
- timer_external_trigger_prescaler_t prescaler
- timer_external_trigger_filter_t filter
- timer_external_trigger_polarity_t polarity

4.4.1 Detailed Description

Used to configure the behaviour of the timer when reading external triggers. Embedded within the advanced sub-structure.

4.4.2 Field Documentation

4.4.2.1 dts

```
timer_digital_filter_clock_div_t dts
```

Sampling frequency settings

4.4.2.2 filter

```
timer_external_trigger_filter_t filter
```

Filter settings when using ETR

4.4.2.3 polarity

```
timer_external_trigger_polarity_t polarity
```

ETR polarity settings

4.4.2.4 prescaler

```
timer_external_trigger_prescaler_t prescaler
```

Sampling frequency prescaler settings when using ETR

The documentation for this struct was generated from the following file:

- /home/marko/Documents/embedded_workspace/timer_driver/timer_stm32f411_config.h

Chapter 5

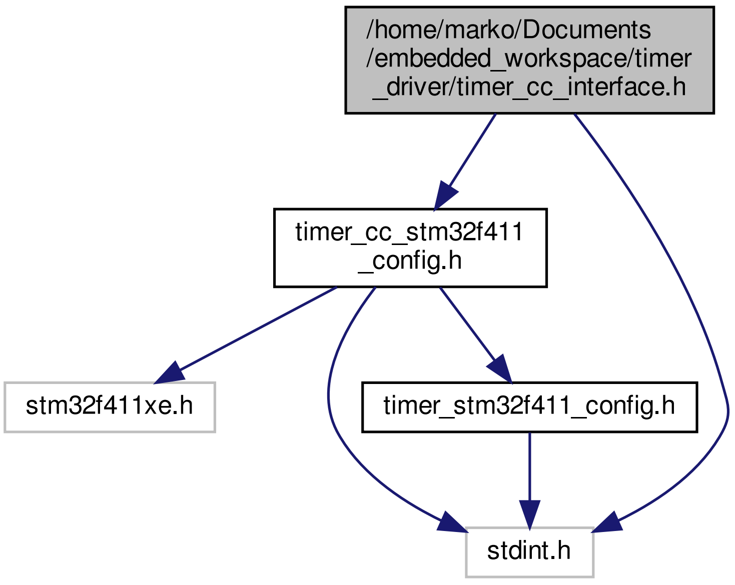
File Documentation

5.1 /home/marko/Documents/embedded_workspace/timer_driver/timer_cc_interface.h File Reference

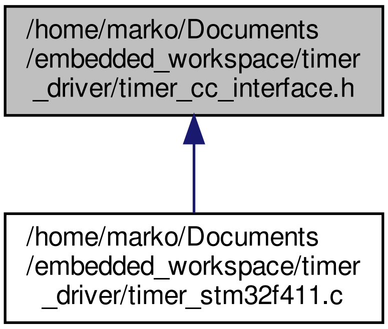
General interface covering user accesses to a timer's capture and compare functionality.

```
#include "timer_cc_stm32f411_config.h"
#include <stdint.h>
```

Include dependency graph for timer_cc_interface.h:



This graph shows which files directly or indirectly include this file:


/home/marko/Documents/embedded_workspace/timer_driver/timer_cc_interface.h
/home/marko/Documents/embedded_workspace/timer_driver/timer_stm32f411.c

Functions

- void **timer_cc_init** (timer_cc_config_t ∗cc_config_table)
- void **timer_cc_control** (timer_cc_t timer_cc, timer_control_t signal)
- uint32_t **timer_cc_read** (timer_cc_t timer_cc)
- void **timer_cc_write** (timer_cc_t timer_cc, uint32_t value)
- void **timer_cc_pwm_duty_cycle_set** (timer_cc_t timer_cc, uint32_t duty_cycle)

5.1.1 Detailed Description

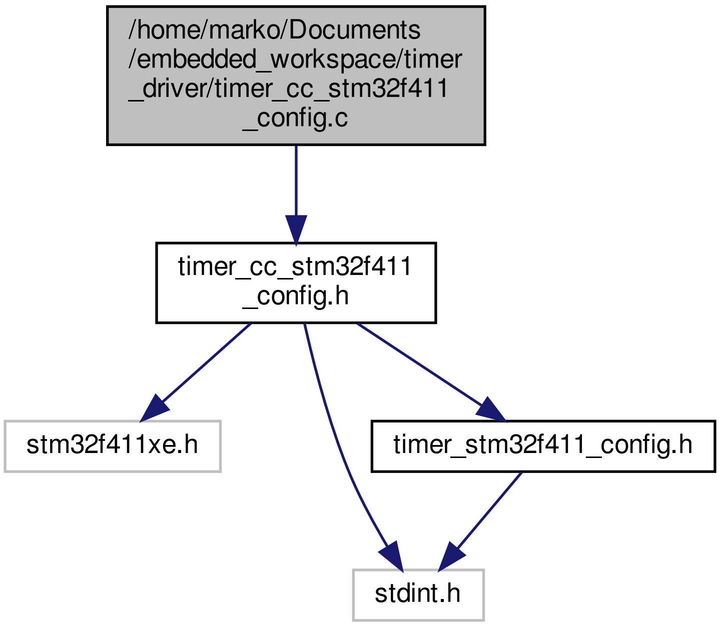
General interface covering user accesses to a timer's capture and compare functionality.

5.2 /home/marko/Documents/embedded_workspace/timer_driver/timer_cc_stm32f411_↩ config.c File Reference

Chip specific configuration table for each planned CC channel.

```
#include "timer_cc_stm32f411_config.h"
```

Include dependency graph for timer_cc_stm32f411_config.c:


/home/marko/Documents/embedded_workspace/timer_driver/timer_cc_stm32f411_config.c
timer_cc_stm32f411_config.h
stm32f411xe.h
timer_stm32f411_config.h
stdint.h

Functions

- const timer_cc_config_t ∗ timer_cc_config_get (void)

5.2.1 Detailed Description

Chip specific configuration table for each planned CC channel.

5.2.2 Function Documentation

5.2.2.1 timer_cc_config_get()

```
const timer_cc_config_t* timer_cc_config_get (
             void  )
```

Description:

This function is used to obtain the configuration data for the CC channels of the timers

PRE-CONDITION: Configuration table needs to populated (sizeof $>$ 0)

POST-CONDITION: The timer CC channels are configured and ready for use.

Returns

const timer_cc_config_t *

Example:

```
  const timer_config_t *timer_config = timer_config_get();
  timer_init(timer_config);
  const timer_cc_config_t *timer_cc_config =
      timer_cc_config_get();
timer_cc_init(timer_cc_config);
```

See also

timer_init
timer_config_get
timer_cc_init
- CHANGE HISTORY -

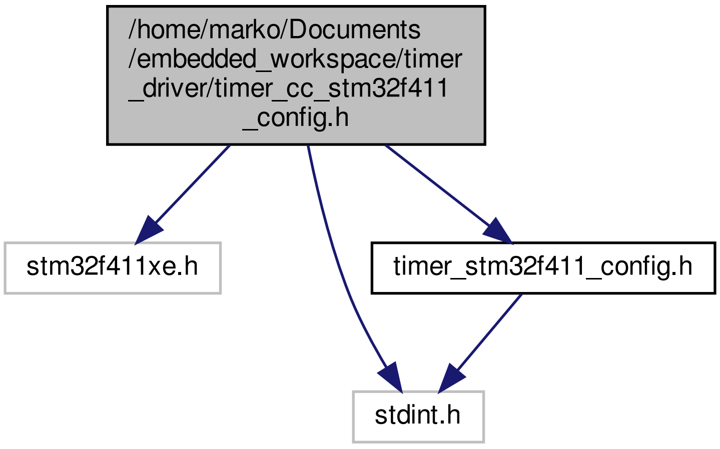
Date	Software Version	Initials	Description

5.3 /home/marko/Documents/embedded_workspace/timer_driver/timer_cc_stm32f411_↩ config.h File Reference

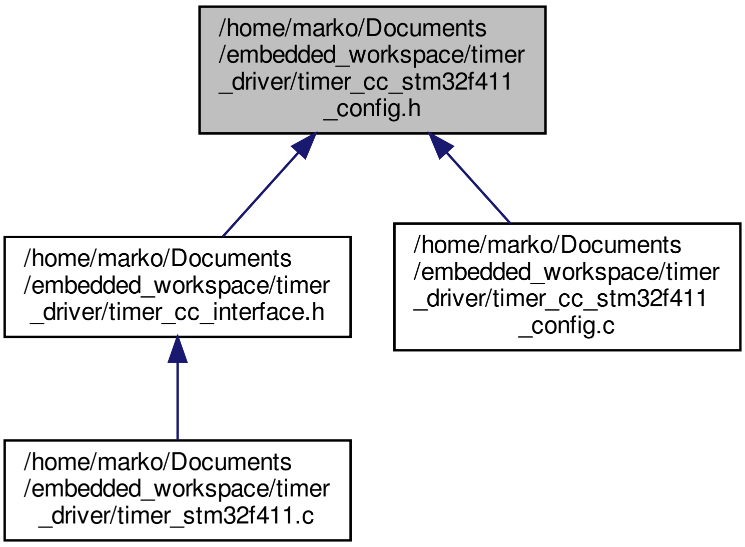
Chip specific header containing requisite enums and structs for proper configuration.

```
#include "stm32f411xe.h"
#include <stdint.h>
#include "timer_stm32f411_config.h"
```

Include dependency graph for timer_cc_stm32f411_config.h:


/home/marko/Documents/embedded_workspace/timer_driver/timer_cc_stm32f411_config.h
stm32f411xe.h
timer_stm32f411_config.h
stdint.h

This graph shows which files directly or indirectly include this file:


/home/marko/Documents/embedded_workspace/timer_driver/timer_cc_stm32f411_config.h
/home/marko/Documents/embedded_workspace/timer_driver/timer_cc_interface.h
/home/marko/Documents/embedded_workspace/timer_driver/timer_cc_stm32f411_config.c
/home/marko/Documents/embedded_workspace/timer_driver/timer_stm32f411.c

Data Structures

- struct timer_cc_config_t

Typedefs

- typedef timer_external_trigger_filter_t timer_cc_input_filter_t

Enumerations

- enum timer_cc_t {
  TIMER1_CCR1, **TIMER1_CCR2**, **TIMER1_CCR3**, **TIMER1_CCR4**,
  TIMER2_CCR1, **TIMER2_CCR2**, **TIMER2_CCR3**, **TIMER2_CCR4**,
  TIMER3_CCR1, **TIMER3_CCR2**, **TIMER3_CCR3**, **TIMER3_CCR4**,
  TIMER4_CCR1, **TIMER4_CCR2**, **TIMER4_CCR3**, **TIMER4_CCR4**,
  TIMER5_CCR1, **TIMER5_CCR2**, **TIMER5_CCR3**, **TIMER5_CCR4**,
  TIMER9_CCR1, **TIMER9_CCR2**, **TIMER10_CCR1**, **TIMER11_CCR1**,
  NUM_CCRS }
- enum timer_cc_mode_t { CC_OUTPUT, CC_INPUT_TI_SELF, CC_INPUT_TI_OPPOSITE, CC_INPUT_T↩
  RC }
- enum timer_cc_output_polarity_t { **OUTPUT_ACTIVE_HIGH**, **OUTPUT_ACTIVE_LOW** }
- enum timer_cc_output_fe_t { **CC_FAST_ENABLE_OFF**, **CC_FAST_ENABLE_ON** }
- enum timer_cc_output_pe_t { **CC_PRELOAD_BUFFER_DISABLED**, **CC_PRELOAD_BUFFER_ENABLED**
  }
- enum timer_cc_output_ce_t { **CC_OUTPUT_CLEAR_DISABLED**, **CC_OUTPUT_CLEAR_EDNABLED** }
- enum timer_cc_output_mode_t {
  FROZEN, ACTIVE_MATCH, INACTIVE_MATCH, TOGGLE_MATCH,
  FORCE_INACTIVE, FORCE_ACTIVE, PWM_MODE_ONE, PWM_MODE_TWO }
- enum timer_cc_input_polarity_t { RISING_EDGE, FALLING_EDGE, BOTH_EDGES }
- enum timer_cc_input_prescaler_t { CAPTURE_DIV_1, CAPTURE_DIV_2, CAPTURE_DIV_4, CAPTURE↩
  _DIV_8 }

Functions

- const timer_cc_config_t ∗ timer_cc_config_get (void)

5.3.1 Detailed Description

Chip specific header containing requisite enums and structs for proper configuration.

5.3.2 Typedef Documentation

5.3.2.1 timer_cc_input_filter_t

```
typedef timer_external_trigger_filter_t timer_cc_input_filter_t
```

Contains the options for the sampling frequency and consecutive event (low pass) filter for the TIy input pin. See

See also

timer_external_trigger_filter_t for the options

5.3.3 Enumeration Type Documentation

5.3.3.1 timer_cc_input_polarity_t

```
enum timer_cc_input_polarity_t
```

Contains the options for the input capture channels edge detection unit.

Enumerator

RISING_EDGE	The TIMx_CCy edge detector (ED) responds to rising edges in TIy
FALLING_EDGE	The TIMx_CCy edge detector (ED) responds to falling edges in TIy
BOTH_EDGES	The TIMx_CCy edge detector (ED) responds to both edges in TIy

5.3.3.2 timer_cc_input_prescaler_t

```
enum timer_cc_input_prescaler_t
```

Contains the options for prescaling of the filtered events

Enumerator

CAPTURE_DIV↩ _1	A capture occurs every detected event
CAPTURE_DIV↩ _2	A capture occurs every second event
CAPTURE_DIV↩ _4	A capture occurs every fourth event
CAPTURE_DIV↩ _8	A capture occurs every eigth event

5.3.3.3 timer_cc_mode_t

```
enum timer_cc_mode_t
```

Contains the modes a cc channel can be in

Enumerator

CC_OUTPUT	The CC channel acts as an output
CC_INPUT_TI_SELF	The CC channel acts an input, which uses the signal from the same channel. e.g TIM3_CH pin (TI3) is processed, and then used within CC3 operations
CC_INPUT_TI_OPPOSITE	The CC channel acts as an input, which uses the signal from its partner channel. e.g. TIM3_CH pin (TI3) is processed, and used by CC4 for its operations
CC_INPUT_TRC	The CC channel acts as an input, and uses the trigger defined in timer_trigger_t

5.3.3.4 timer_cc_output_ce_t

```
enum timer_cc_output_ce_t
```

Contains the options for output clearing. When activated, the output channel immediately drops to its inactive level upon an ETRF input

5.3.3.5 timer_cc_output_fe_t

```
enum timer_cc_output_fe_t
```

Contains options for output fast enable, an option which allows outputs to respond faster (3 vs 5 clock cycles) to a trigger event in PWM mode.

5.3.3.6 timer_cc_output_mode_t

```
enum timer_cc_output_mode_t
```

Contains the various output modes a CC channel can be in.

Enumerator

FROZEN	The cc output pin doesn't respond to matches between TIMx_CNT and TIMx_CCRy
ACTIVE_MATCH	The cc output pin is driven to its active level upon a CNT & CCRy match
INACTIVE_MATCH	The cc output pin is driven to its inactive level upon a CNT & CCRy match
TOGGLE_MATCH	The cc output pin's state is toggled upon CNT & CCRy matches
FORCE_INACTIVE	The cc output pin is forced to its inactive level
FORCE_ACTIVE	The cc output pin is forced to its active level
PWM_MODE_ONE	The cc output pin operates in PWM Non-Inverted Duty Cycles
PWM_MODE_TWO	The cc outptu pin operates in PWM Inverted Duty Cycles

5.3.3.7 timer_cc_output_pe_t

```
enum timer_cc_output_pe_t
```

Contains options for output preload buffering. Works identically to auto-reload preload. When enabled, writes to TIMx_CCRy are only reflected after an update event

5.3.3.8 timer_cc_output_polarity_t

```
enum timer_cc_output_polarity_t
```

Contains self explanatory polarity options for the output of a particular channel

5.3.3.9 timer_cc_t

```
enum timer_cc_t
```

Contains all of the capture and compare channel on chip

5.3.4 Function Documentation

5.3.4.1 timer_cc_config_get()

```
const timer_cc_config_t* timer_cc_config_get (
              void   )
```

Description:

This function is used to obtain the configuration data for the CC channels of the timers

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

POST-CONDITION: The timer CC channels are configured and ready for use.

Returns

const timer_cc_config_t *

Example:

```
  const timer_config_t *timer_config = timer_config_get();
  timer_init(timer_config);
  const timer_cc_config_t *timer_cc_config =
      timer_cc_config_get();
timer_cc_init(timer_cc_config);
```

See also

timer_init
timer_config_get
timer_cc_init
- CHANGE HISTORY -

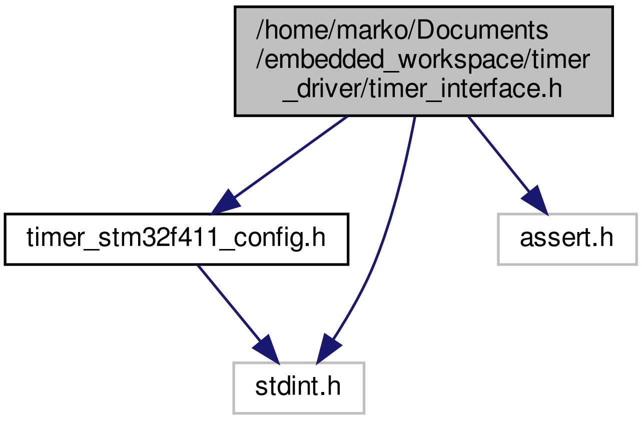
Date	Software Version	Initials	Description

5.4 /home/marko/Documents/embedded_workspace/timer_driver/timer_interface.h File Reference

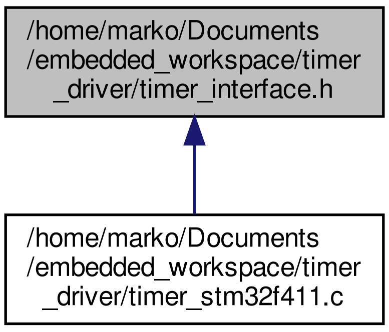
General interface covering user accesses to a timer's timebase functionality.

```
#include "timer_stm32f411_config.h"
#include <stdint.h>
#include <assert.h>
```

Include dependency graph for timer_interface.h:


/home/marko/Documents/embedded_workspace/timer_driver/timer_interface.h
timer_stm32f411_config.h
assert.h
stdint.h

This graph shows which files directly or indirectly include this file:


/home/marko/Documents/embedded_workspace/timer_driver/timer_interface.h
/home/marko/Documents/embedded_workspace/timer_driver/timer_stm32f411.c

Macros

- #define DISABLED 0
- #define ENABLED 1

Enumerations

- enum timer_control_t { TIMER_STOP, TIMER_START }
- enum timer_interrupt_control_t { INTERRUPT_DISABLE, INTERRUPT_ENABLE }

Functions

- void timer_init (timer_config_t ∗config_table)
- void timer_control (timer_t timer, timer_control_t signal)
- uint32_t timer_read (timer_t timer)
- void timer_prescaler_set (timer_t timer, timer_prescaler_t prescaler)
- timer_prescaler_t timer_prescaler_get (timer_t timer)
- void timer_interrupt_control (timer_t timer, timer_interrupt_t interrupt, timer_interrupt_control_t signal)
- void timer_register_write (uint32_t timer_register, uint32_t value)
- uint32_t timer_register_read (uint32_t timer_register)

5.4.1 Detailed Description

General interface covering user accesses to a timer's timebase functionality.

5.4.2 Macro Definition Documentation

5.4.2.1 DISABLED

```
#define DISABLED 0
```

DISABLED and ENABLED macros find their way wherever only a single bit is required to define a mode. Will probalby be phased out for small typedefs in the future.

5.4.2.2 ENABLED

```
#define ENABLED 1
```

DISABLED and ENABLED macros find their way wherever only a single bit is required to define a mode. Will probalby be phased out for small typedefs in the future.

5.4.3 Enumeration Type Documentation

5.4.3.1 timer_control_t

```
enum timer_control_t
```

Universal start/stop signal. Won't change from platform to platform, although the underlying code might

Enumerator

TIMER_STOP	Stops the timer
TIMER_START	Starts the timer

5.4.3.2 timer_interrupt_control_t

```
enum timer_interrupt_control_t
```

Universal interrupt enable/disable signal. Won't change from platform to platform

Enumerator

INTERRUPT_DISABLE	Disables the selected interrupt
INTERRUPT_ENABLE	Enables the selected interrupt

5.4.4 Function Documentation

5.4.4.1 timer_control()

```
void timer_control (
            timer_t timer,
            timer_control_t signal )
```

Description:

This function is used to start or stop the counter.

PRE-CONDITION: The timer has been successfully initiated through timer_init()

POST-CONDITION: The timer has started/stop, as per the signal

Parameters

timer	refers to any timer present on-chip
signal	determines whether the timer stops or starts

Returns

void

Example:

```
timer_init(config);
timer_control(TIMER4, TIMER_START);
```

See also

timer_init
timer_read
timer_interrupt_control
- CHANGE HISTORY -

Date	Software Version	Initials	Description

5.4.4.2 timer_init()

```
void timer_init (
            timer_config_t * config_table )
```

Description:

This function is used to initialise the timer based on the configuration table defined in the timer_stm32f411_config.c

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)
PRE-CONDITION (conditional): If using advanced features, the advanced pointer for the appropriate timer must be non-null PRE-CONDITION (conditional): If using external triggers or TI1/TI2 triggers, the gpio pins must me configured with appropriate AF settings PRE-CONDITION (condtional): If using external triggers, the external_↩ trigger pointer in the advanced structure must be non-null PRE-CONDITION: The RCC clocks for all planned timers must be configured and enabled.

POST-CONDITION: The timers are ready for use.

Parameters

config_table	is a pointer to the configuration table that contains the initialisation structures for each timer.

Returns

void

Example:

```
const timer_config_t *timer_config = timer_config_get();
timer_init(timer_config);
```

See also

timer_config_get
timer_init_external_mode_1
timer_init_external_mode_2
timer_init_slave_mode
- CHANGE HISTORY -

Date	Software Version	Initials	Description

5.4.4.3 timer_interrupt_control()

```
void timer_interrupt_control (
            timer_t timer,
            timer_interrupt_t interrupt,
            timer_interrupt_control_t signal )
```

Description:

```
Activates or deactivates the selected interrupt
```

PRE-CONDITION: The timer has been successfully initiated through timer_init() PRE-CONDITION: The requested interrupt is actually available on the selected timer.

POST-CONDITION: The selected interrupt is enabled/disabled

Parameters

timer	refers to any timer present on-chip
interrupt	refers to the selected interrupt type
signal	decides whether the interrupt is enabled or disabled

Returns

void

Example:

```
timer_init(config);
timer_interrupt_control(TIMER1, UPDATE_INTERRUPT,
     INTERRUPT_ENABLE);
timer_control(TIMER1, TIMER_START);
```

See also

timer_control
- CHANGE HISTORY -

Date	Software Version	Initials	Description

check for invalid combinations

Only timer 1 has BREAK and COM

Timers 9/10/11 don't have CCs 3 and 4

Timers 10/11 don't have CC2 either

Timers 10 and 11 don't have trigger interrrupts

5.4.4.4 timer_prescaler_get()

```
timer_prescaler_t timer_prescaler_get (
             timer_t timer )
```

Description:

```
Gets the prescaler value
```

PRE-CONDITION: The timer has been successfully initiated through timer_init()

POST-CONDITION: The timer's current prescaler value (TIMx_PSC) is returned

Parameters

timer	refers to any timer present on-chip

Returns

timer_prescaler_t (uint16_t)

Example:

```
timer_prescaler_t curr_prescaler_timer3 = timer_prescaler_get(
      TIMER3);
```

See also

timer_control
timer_prescaler_set
timer_interrupt_control
timer_read
- CHANGE HISTORY -

Date	Software Version	Initials	Description

5.4.4.5 timer_prescaler_set()

```
void timer_prescaler_set (
            timer_t timer,
            timer_prescaler_t prescaler )
```

Description:

```
Sets the prescaler value "on the fly"
```

PRE-CONDITION: The timer has been successfully initiated through timer_init()

POST-CONDITION: The timer's clock is divided by the new prescaler

Parameters

timer	refers to any timer present on-chip
prescaler	is a uint16_t value

Returns

void

Example:

```
//On the fly changes of the prescaler
  timer_init(config);
  timer_prescaler_set(TIMER5, 3200);
  timer_control(TIMER5, TIMER_START);
  while (timer_read(TIMER5) < 300);
  timer_prescaler_set(TIMER5, 5000);
```

See also

timer_control
timer_prescaler_get
timer_interrupt_control
timer_read
- CHANGE HISTORY -

Date	Software Version	Initials	Description

5.4.4.6 timer_read()

```
uint32_t timer_read (
            timer_t timer )
```

Description:

This function returns the contents of a timer's CNT register

Note: Technically the function works without starting the timer, but that's useless

PRE-CONDITION: The timer has been successfully initiated through timer_init()

POST-CONDITION: The function has returned the current contents of CNT

Parameters

timer	refers to any timer present on-chip

Returns

uint32_t

Example:

```
uint32_t curr_value = timer_read(TIMER9);
```

See also

timer_control
timer_interrupt_control
timer_prescaler_set
timer_prescaler_get
- CHANGE HISTORY -

Date	Software Version	Initials	Description

5.4.4.7 timer_register_read()

```
uint32_t timer_register_read (
            uint32_t timer_register )
```

Description:

```
Read the current value of the register in timer address. It is the
user's own responisibility to consult the RM0383 to ensure that no reserved bits are
overwritten, etc.
Intended to be used alongside timer_register_write() to create composite advanced user functions
```

PRE-CONDITION: The address does in fact lie in the address space of any timer.

POST-CONDITION: The register's current contents are returned

Parameters

timer_register	is a uint32_t which is cast as a 32bit address

Returns

uint32_t timer_register's contents

Example:

```
uint32_t dier_timer3 = timer_register_read(TIM3_BASE + 0x0C); //get current value
dier_timer3 &= ~(0x01UL << TIM_DIER_CC3DE_Pos); //clear the DMA request on CC3 bit
timer_register_write(TIM3_BASE + 0x0C, dier_timer3);
```

See also

timer_register_write
timer_read
- CHANGE HISTORY -

Date	Software Version	Initials	Description

5.4.4.8 timer_register_write()

```
void timer_register_write (
            uint32_t timer_register,
            uint32_t value )
```

Description:

```
Writes the desired value into the desired timer address space register. It is the
user's own responisibility to consult the RM0383 to ensure that no reserved bits are
overwritten, etc.
Intended to be used alongside timer_register_read() to create composite advanced user functions
```

PRE-CONDITION: The address does in fact lie in the address space of any timer.

POST-CONDITION: The desired register's contents now reflect "value"

Parameters

timer_register	is a uint32_t which is cast as a 32bit address
value	is an (up to) uint32_t value which is written to the desired register

Returns

void

Example:

```
uint32_t dier_timer3 = timer_register_read(TIM3_BASE + 0x0C); //get current value
dier_timer3 &= ~(0x01UL << TIM_DIER_CC3DE_Pos); //clear the DMA request on CC3 bit
timer_register_write(TIM3_BASE + 0x0C, dier_timer3);
```

See also

timer_register_read
timer_read
- CHANGE HISTORY -

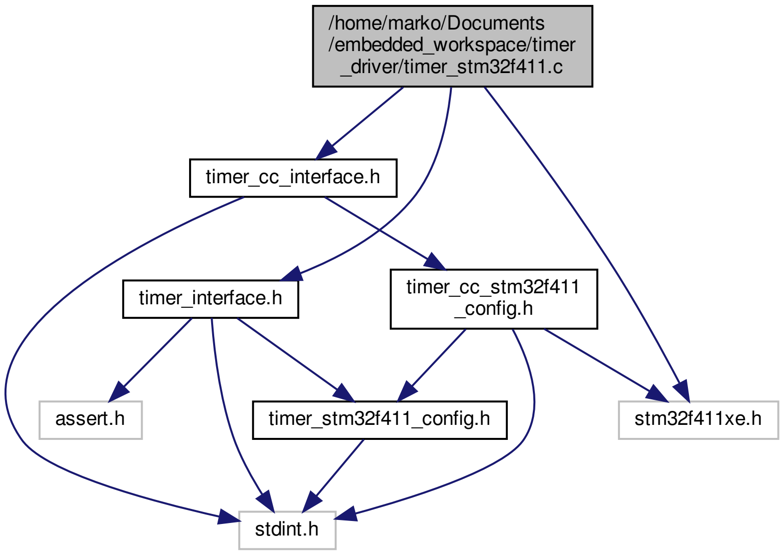
Date	Software Version	Initials	Description

5.5 /home/marko/Documents/embedded_workspace/timer_driver/timer_stm32f411.c File Reference

Microcontroller specific implementation of timer functionality.

```
#include "timer_interface.h"
#include "timer_cc_interface.h"
#include "stm32f411xe.h"
```

Include dependency graph for timer_stm32f411.c:



Functions

- void timer_init (timer_config_t ∗config_table)
- void timer_control (timer_t timer, timer_control_t signal)
- uint32_t timer_read (timer_t timer)
- void timer_interrupt_control (timer_t timer, timer_interrupt_t interrupt, timer_interrupt_control_t signal)
- void timer_prescaler_set (timer_t timer, timer_prescaler_t prescaler)
- timer_prescaler_t timer_prescaler_get (timer_t timer)
- void timer_register_write (uint32_t timer_register, uint32_t value)
- uint32_t timer_register_read (uint32_t timer_register)
- void **timer_cc_init** (timer_cc_config_t ∗config_table)
- void **timer_cc_control** (timer_cc_t timer_cc, timer_control_t signal)
- uint32_t **timer_cc_read** (timer_cc_t timer_cc)
- void **timer_cc_write** (timer_cc_t timer_cc, uint32_t value)
- void **timer_cc_pwm_duty_cycle_set** (timer_cc_t timer_cc, uint32_t duty_cycle_pcnt)

5.5.1 Detailed Description

Microcontroller specific implementation of timer functionality.

5.5.2 Function Documentation

5.5.2.1 timer_control()

```
void timer_control (
            timer_t timer,
            timer_control_t signal )
```

Description:

This function is used to start or stop the counter.

PRE-CONDITION: The timer has been successfully initiated through timer_init()

POST-CONDITION: The timer has started/stop, as per the signal

Parameters

timer	refers to any timer present on-chip
signal	determines whether the timer stops or starts

Returns

void

Example:

```
timer_init(config);
timer_control(TIMER4, TIMER_START);
```

See also

timer_init
timer_read
timer_interrupt_control
- CHANGE HISTORY -

Date	Software Version	Initials	Description

5.5.2.2 timer_init()

```
void timer_init (
            timer_config_t * config_table )
```

Description:

This function is used to initialise the timer based on the configuration table defined in the timer_stm32f411_config.c

PRE-CONDITION: Configuration table needs to populated (sizeof $>$ 0)
PRE-CONDITION (conditional): If using advanced features, the advanced pointer for the appropriate timer must be non-null PRE-CONDITION (conditional): If using external triggers or TI1/TI2 triggers, the gpio pins must me configured with appropriate AF settings PRE-CONDITION (condtional): If using external triggers, the external_↩ trigger pointer in the advanced structure must be non-null PRE-CONDITION: The RCC clocks for all planned timers must be configured and enabled.

POST-CONDITION: The timers are ready for use.

Parameters

config_table	is a pointer to the configuration table that contains the initialisation structures for each timer.

Returns

void

Example:

```
const timer_config_t *timer_config = timer_config_get();
timer_init(timer_config);
```

See also

timer_config_get
timer_init_external_mode_1
timer_init_external_mode_2
timer_init_slave_mode
- CHANGE HISTORY -

Date	Software Version	Initials	Description

5.5.2.3 timer_interrupt_control()

```
void timer_interrupt_control (
            timer_t timer,
            timer_interrupt_t interrupt,
            timer_interrupt_control_t signal )
```

Description:

```
Activates or deactivates the selected interrupt
```

PRE-CONDITION: The timer has been successfully initiated through timer_init() PRE-CONDITION: The requested interrupt is actually available on the selected timer.

POST-CONDITION: The selected interrupt is enabled/disabled

Parameters

timer	refers to any timer present on-chip
interrupt	refers to the selected interrupt type
signal	decides whether the interrupt is enabled or disabled

Returns

void

Example:

```
timer_init(config);
timer_interrupt_control(TIMER1, UPDATE_INTERRUPT,
      INTERRUPT_ENABLE);
timer_control(TIMER1, TIMER_START);
```

See also

timer_control

- CHANGE HISTORY -

Date	Software Version	Initials	Description

check for invalid combinations

Only timer 1 has BREAK and COM

Timers 9/10/11 don't have CCs 3 and 4

Timers 10/11 don't have CC2 either

Timers 10 and 11 don't have trigger interrrupts

5.5.2.4 timer_prescaler_get()

```
timer_prescaler_t timer_prescaler_get (
              timer_t timer )
```

Description:

```
Gets the prescaler value
```

PRE-CONDITION: The timer has been successfully initiated through timer_init()

POST-CONDITION: The timer's current prescaler value (TIMx_PSC) is returned

Parameters

timer	refers to any timer present on-chip

Returns

timer_prescaler_t (uint16_t)

Example:

```
timer_prescaler_t curr_prescaler_timer3 = timer_prescaler_get(
      TIMER3);
```

See also

timer_control
timer_prescaler_set
timer_interrupt_control
timer_read
- CHANGE HISTORY -

Date	Software Version	Initials	Description

5.5.2.5 timer_prescaler_set()

```
void timer_prescaler_set (
            timer_t timer,
            timer_prescaler_t prescaler )
```

Description:

```
Sets the prescaler value "on the fly"
```

PRE-CONDITION: The timer has been successfully initiated through timer_init()

POST-CONDITION: The timer's clock is divided by the new prescaler

Parameters

timer	refers to any timer present on-chip
prescaler	is a uint16_t value

Returns

void

Example:

```
//On the fly changes of the prescaler
  timer_init(config);
  timer_prescaler_set(TIMER5, 3200);
  timer_control(TIMER5, TIMER_START);
  while (timer_read(TIMER5) < 300);
  timer_prescaler_set(TIMER5, 5000);
```

See also

timer_control
timer_prescaler_get
timer_interrupt_control
timer_read
- CHANGE HISTORY -

Date	Software Version	Initials	Description

5.5.2.6 timer_read()

```
uint32_t timer_read (
            timer_t timer )
```

Description:

This function returns the contents of a timer's CNT register

Note: Technically the function works without starting the timer, but that's useless

PRE-CONDITION: The timer has been successfully initiated through timer_init()

POST-CONDITION: The function has returned the current contents of CNT

Parameters

timer	refers to any timer present on-chip

Returns

uint32_t

Example:

```
uint32_t curr_value = timer_read(TIMER9);
```

See also

timer_control
timer_interrupt_control
timer_prescaler_set
timer_prescaler_get
- CHANGE HISTORY -

Date	Software Version	Initials	Description

5.5.2.7 timer_register_read()

```
uint32_t timer_register_read (
            uint32_t timer_register )
```

Description:

```
Read the current value of the register in timer address. It is the
user's own responisibility to consult the RM0383 to ensure that no reserved bits are
overwritten, etc.
Intended to be used alongside timer_register_write() to create composite advanced user functions
```

PRE-CONDITION: The address does in fact lie in the address space of any timer.

POST-CONDITION: The register's current contents are returned

Parameters

timer_register	is a uint32_t which is cast as a 32bit address

Returns

uint32_t timer_register's contents

Example:

```
uint32_t dier_timer3 = timer_register_read(TIM3_BASE + 0x0C); //get current value
dier_timer3 &= ~(0x01UL << TIM_DIER_CC3DE_Pos); //clear the DMA request on CC3 bit
timer_register_write(TIM3_BASE + 0x0C, dier_timer3);
```

See also

timer_register_write
timer_read
- CHANGE HISTORY -

Date	Software Version	Initials	Description

5.5.2.8 timer_register_write()

```
void timer_register_write (
            uint32_t timer_register,
            uint32_t value )
```

Description:

```
Writes the desired value into the desired timer address space register. It is the
user’s own responisibility to consult the RM0383 to ensure that no reserved bits are
overwritten, etc.
Intended to be used alongside timer_register_read() to create composite advanced user functions
```

PRE-CONDITION: The address does in fact lie in the address space of any timer.

POST-CONDITION: The desired register's contents now reflect "value"

Parameters

timer_register	is a uint32_t which is cast as a 32bit address
value	is an (up to) uint32_t value which is written to the desired register

Returns

void

Example:

```
uint32_t dier_timer3 = timer_register_read(TIM3_BASE + 0x0C); //get current value
dier_timer3 &= ~(0x01UL << TIM_DIER_CC3DE_Pos); //clear the DMA request on CC3 bit
timer_register_write(TIM3_BASE + 0x0C, dier_timer3);
```

See also

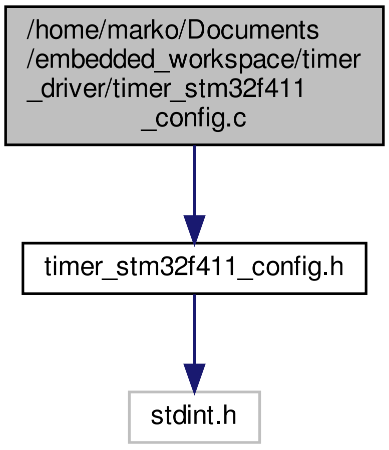
timer_register_read
timer_read
- CHANGE HISTORY -

Date	Software Version	Initials	Description

5.6 /home/marko/Documents/embedded_workspace/timer_driver/timer_stm32f411_↩ config.c File Reference

Collection of configuration tables used to configure a timer. Config_table is the one you'll probably need the most. The _advanced and _trigger tables are only used in their respective contexts, and will remain empty unless using external triggering, one shot mode, or disabling update events.

```
#include "timer_stm32f411_config.h"
```
Include dependency graph for timer_stm32f411_config.c:


/home/marko/Documents/embedded_workspace/timer_driver/timer_stm32f411_config.c
timer_stm32f411_config.h
stdint.h

Functions

- const timer_config_t ∗ timer_config_get (void)

5.6.1 Detailed Description

Collection of configuration tables used to configure a timer. Config_table is the one you'll probably need the most. The _advanced and _trigger tables are only used in their respective contexts, and will remain empty unless using external triggering, one shot mode, or disabling update events.

5.6.2 Function Documentation

5.6.2.1 timer_config_get()

```
const timer_config_t* timer_config_get (
            void  )
```

Description: Retrieves the config table for the timer peripheral, normally hidden statically within the config.c file.

PRE-CONDITION: The config table has been populated/exists with a size greater than 0.

POST-CONDITION: The returned value points to the base of the config table

Returns

const timer_config_t *

Note: To configure a timer, visit the config_table array in timer_stm32f411_conig.c and set all the elements of the corresponding structure. e.g.

```
static const timer_config_t config_table[NUM_TIMERS] =
{
...
TIM4 {EXTERNAL_MODE1, ENCODER_MODE_1, CENTER_ALIGNED_3, 0, 15934, 775,
      ENABLED, 0},
...
}
```

Example:

```
const timer_config_t *timer_config_table = timer_config_get(void);
timer_init(timer_config_table);
```

See also

timer_init

- CHANGE HISTORY -

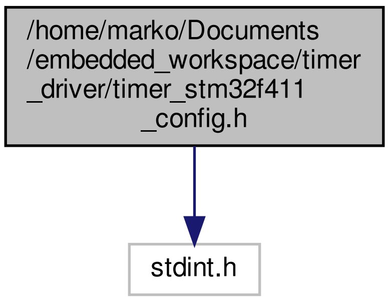
Date	Software Version	Initials	Description

5.7 /home/marko/Documents/embedded_workspace/timer_driver/timer_stm32f411_config.h File Reference

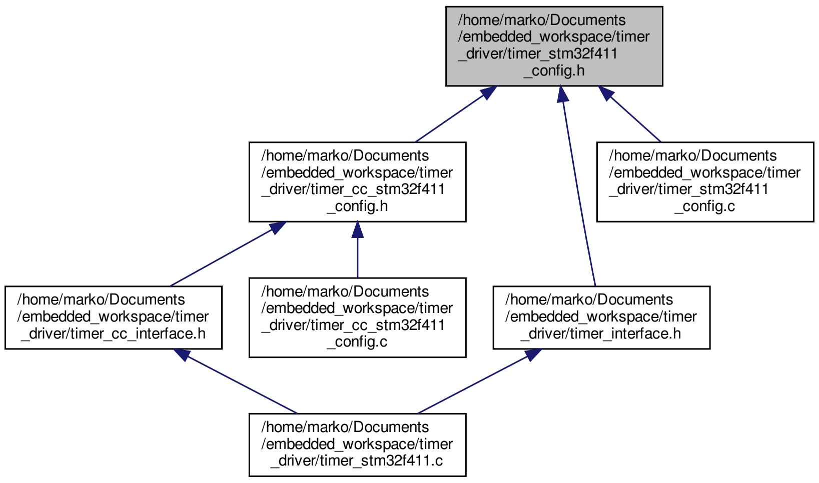
Microcontroller specific header containing typedefs for all relevant config options.

```
#include <stdint.h>
```

Include dependency graph for timer_stm32f411_config.h:



This graph shows which files directly or indirectly include this file:


/home/marko/Documents
/embedded_workspace/timer
_driver/timer_stm32f411
_config.h
/home/marko/Documents
/embedded_workspace/timer
_driver/timer_cc_stm32f411
_config.h
/home/marko/Documents
/embedded_workspace/timer
_driver/timer_stm32f411
_config.c
/home/marko/Documents
/embedded_workspace/timer
_driver/timer_cc_interface.h
/home/marko/Documents
/embedded_workspace/timer
_driver/timer_cc_stm32f411
_config.c
/home/marko/Documents
/embedded_workspace/timer
_driver/timer_interface.h
/home/marko/Documents
/embedded_workspace/timer
_driver/timer_stm32f411.c

Data Structures

- struct timer_external_trigger_t
- struct timer_advanced_t
- struct timer_config_t

Macros

- #define **NULL** (void ∗)0

Typedefs

- typedef uint16_t timer_prescaler_t

Enumerations

- enum timer_t {
  TIMER1, TIMER2, TIMER3, TIMER4,
  TIMER5, TIMER9, TIMER10, TIMER11,
  NUM_TIMERS }
- enum timer_interrupt_t {
  UPDATE_INTERRUPT, CC1_INTERRUPT, CC2_INTERRUPT, CC3_INTERRUPT,
  CC4_INTERRUPT, COM_INTERRUPT, TRIGGER_INTERRUPT, BREAK_INTERRUPT }
- enum timer_alt_clock_source_t { INTERNAL_CLOCK, EXTERNAL_MODE_1, EXTERNAL_MODE_2 }
- enum timer_slave_mode_t {
  SLAVE_MODE_DISABLED, ENCODER_MODE_1, ENCODER_MODE_2, ENCODER_MODE_3,
  RESET_MODE, GATED_MODE, TRIGGER_MODE }
- enum timer_trigger_t {
  ITR0, ITR1, ITR2, ITR3,
  TI1_EDGE_DETECTOR, TI1_FP1, TI2_FP2, ETRF }
- enum timer_external_trigger_prescaler_t { TRIGGER_DIV_1, TRIGGER_DIV_2, TRIGGER_DIV_4, TRIG←
  GER_DIV_8 }
- enum timer_digital_filter_clock_div_t { CK_INT_TIMES_1, CK_INT_TIMES_2, CK_INT_TIMES_4 }
- enum timer_external_trigger_filter_t {
  DTS, CK_INT_N_2, CK_INT_N_4, CK_INT_N_8,
  DTS_DIV_2_N_6, DTS_DIV_2_N_8, DTS_DIV_4_N_6, DTS_DIV_4_N_8,
  DTS_DIV_8_N_6, DTS_DIV_8_N_8, DTS_DIV_16_N_5, DTS_DIV_16_N_6,
  DTS_DIV_16_N_8, DTS_DIV_32_N_5, DTS_DIV_32_N_6, DTS_DIV_32_N_8 }
- enum timer_master_slave_mode_t { MASTER_SLAVE_DISABLED, MASTER_SLAVE_ENABLED }
- enum timer_alignment_t { EDGE_ALIGNED, CENTER_ALIGNED_1, CENTER_ALIGNED_2, CENTER_A←
  LIGNED_3 }
- enum timer_direction_t { UPCOUNTER, DOWNCOUNTER }
- enum timer_opm_t { TIMER_OPM_DISABLED, TIMER_OPM_ENABLED }
- enum timer_arpe_t { TIMER_ARPE_DISABLED, TIMER_ARPE_ENABLED }
- enum timer_udis_t { TIMER_UDIS_DISABLED, TIMER_UDIS_ENABLED }
- enum timer_external_trigger_polarity_t { TRIGGER_POLARITY_NON_INVERTED, TRIGGER_POLARIT←
  Y_INVERTED }

Functions

- const timer_config_t ∗ timer_config_get (void)

5.7.1 Detailed Description

Microcontroller specific header containing typedefs for all relevant config options.

5.7.2 Typedef Documentation

5.7.2.1 timer_prescaler_t

```
typedef uint16_t timer_prescaler_t
```

The main prescaler must be a 16bit number

5.7.3 Enumeration Type Documentation

5.7.3.1 timer_alignment_t

```
enum timer_alignment_t
```

Contains the options for counter alignment

Enumerator

EDGE_ALIGNED	The counter counts purely up or down, depending on the selected direciton **See also** timer_direction_t
CENTER_ALIGNED↩_1	The counter counts up, then down. Capture Compare match events only occur on the way down
CENTER_ALIGNED↩_2	The counter counts up, then down. Capture Compare match events only occur on the way up
CENTER_ALIGNED↩_3	The counter counts up, then down. Capture Compare match events occur on both ways

5.7.3.2 timer_alt_clock_source_t

```
enum timer_alt_clock_source_t
```

Defines the available clock sources. For full detail

See also

RM0383 chapters 12, 13, or 14.

Enumerator

INTERNAL_CLOCK	Default option. Timer is clocked by CK_INT. Prevents alternative configurations
EXTERNAL_MODE↩_1	Timer is clocked by an internal trigger source TRGI. See
EXTERNAL_MODE↩_2	Timer is clocked by the external trigger source. Identical to selecting ETRF as the trigger in External Mode 1

5.7.3.3 timer_arpe_t

```
enum timer_arpe_t
```

Auto Reload Preload Buffer options. When enabled, new ARR values are only active after the next update event. When disabled, new ARR values are immediately transferred

Enumerator

TIMER_ARPE_DISABLED	Disabled ARR buffer
TIMER_ARPE_ENABLED	Enabled ARR buffer

5.7.3.4 timer_digital_filter_clock_div_t

```
enum timer_digital_filter_clock_div_t
```

Contains the frequency of the digital event filter.

Enumerator

CK_INT_TIMES↩ _1	fDTS = fCK_INT
CK_INT_TIMES↩ _2	fDTS = 2∗fCK_INT
CK_INT_TIMES↩ _4	fDTS = 4∗fCK_INT

5.7.3.5 timer_direction_t

```
enum timer_direction_t
```

The options for counter direction. Only relevant in edge-aligned mode

Enumerator

UPCOUNTER	The counter counts from 0 up to the reload value
DOWNCOUNTER	The counter counts from the reload value down to 0

5.7.3.6 timer_external_trigger_filter_t

```
enum timer_external_trigger_filter_t
```

Contains the options for digital filtration of external trigger events. The selected sampling frequency is (fDTS or fCK_INT) divided by a factor of 2- 32, and N consecutive events must occur before they are acknowledged

Enumerator

DTS	Input sampled at fDTS (fs = fDTS), 1 input triggers the event (N = 1)
CK_INT_N_2	fs = CK_INT, N = 2
CK_INT_N_4	fs = fCK_INT, N = 4
CK_INT_N_8	fs = fCK_INT, N = 8
DTS_DIV_2_N_6	fs = fDTS/2, N = 6
DTS_DIV_2_N_8	fs = fDTS/2, N = 8
DTS_DIV_4_N_6	fs = fDTS/4, N = 6
DTS_DIV_4_N_8	fs = fDTS/4, N = 8
DTS_DIV_8_N_6	fs = fDTS/8, N = 6
DTS_DIV_8_N_8	fs = fDTS/8, N = 8
DTS_DIV_16_N↵ _5	fs = fDTS/16, N = 5
DTS_DIV_16_N↵ _6	fs = fDTS/16, N = 6
DTS_DIV_16_N↵ _8	fs = fDTS/16, N = 8
DTS_DIV_32_N↵ _5	fs = fDTS/32, N = 5
DTS_DIV_32_N↵ _6	fs = fDTS/32, N = 6
DTS_DIV_32_N↵ _8	fs = fDTS/32, N = 8

5.7.3.7 timer_external_trigger_polarity_t

```
enum timer_external_trigger_polarity_t
```

Options for the external trigger's polarity.

Enumerator

TRIGGER_POLARITY_NON_INVERTED	External trigger is active high and/or rising edge
TRIGGER_POLARITY_INVERTED	External trigger is active low and/or falling edge

5.7.3.8 timer_external_trigger_prescaler_t

```
enum timer_external_trigger_prescaler_t
```

Contains the prescaler options on the raw external trigger input

Enumerator

TRIGGER_DIV↩_1	ETR is purely sampled
TRIGGER_DIV↩_2	Every second ETR event is sampled
TRIGGER_DIV↩_4	Every fourth ETR event is sampled
TRIGGER_DIV↩_8	Every eighth ETR event is sampled

5.7.3.9 timer_interrupt_t

```
enum timer_interrupt_t
```

Defines all types of interrupts supported on the timers. Not all timers support all interrupts.

Enumerator

UPDATE_INTERRUPT	Generate interrupt on update event (timer overflow/underflow, etc). All Timers
CC1_INTERRUPT	Generate interrupt on CC1 event. All Timers
CC2_INTERRUPT	Generate interrupt on CC2 event. All Timers but 10 and 11
CC3_INTERRUPT	Generate interrupt on CC3 event. Timers 1-5
CC4_INTERRUPT	Generate interrupt on CC4 event. Timers 1-5
COM_INTERRUPT	Generate interrupt on COM event. **See also** RM0383 pg 295, Bit 5. Timer 1 only.
TRIGGER_INTERRUPT	Generate interrupt on Trigger event. **See also** TIF bit in RM0383, TIMx_SR. All timers but 10 and 11
BREAK_INTERRUPT	M Generate interrupt on Break event. **See also** RM0383 pg 295, Bit 7. Timer 1 only

5.7.3.10 timer_master_slave_mode_t

```
enum timer_master_slave_mode_t
```

Master/Slave mode is used for multiple timer synchronisation. With MSM ON, the master device will delay its counting by one cycle, giving the slave time to receive and parse the start command.

Enumerator

MASTER_SLAVE_DISABLED	Master/Slave mode is not on
MASTER_SLAVE_ENABLED	Master/Slave mode is activated

5.7.3.11 timer_opm_t

```
enum timer_opm_t
```

Contains the option for one pulse mode. A timer in one pulse mode must be restarted after every update event. It can also be reconfigured inbetween starts.

Enumerator

TIMER_OPM_DISABLED	One Pulse Mode is off
TIMER_OPM_ENABLED	One Pulse Mode is on

5.7.3.12 timer_slave_mode_t

```
enum timer_slave_mode_t
```

Contains all the waves a timer can be controller from an outside source. For detailed information on encoder modes, see RM0383 Chapter 12.3.16

Enumerator

SLAVE_MODE_DISABLED	Timer is clocked internally from Pclk
ENCODER_MODE_1	Timer is clocked by edge transitions on TI2 (TIMx_CH2) depending on the level of TI1 (TIMx_CH1)
ENCODER_MODE_2	Timer is clocked by edge transitions on TI1 (TIMx_CH1) depending on the level of TI2 (TIMx_CH2)
ENCODER_MODE_3	Timer is clocked by transitions on TI1 (TIMx_CH1) and TI2 (TIMx_CH2) depending on the level f the opposite
RESET_MODE	Timer is reset on rising edge of TRGI (timer_trigger_t)
GATED_MODE	Timer only counts while TRGI is (timer_trigger_t) is high
TRIGGER_MODE	Timer starts counting upon rising edge of TRGI (timer_trigger_t)

5.7.3.13 timer_t

```
enum timer_t
```

Contains all on-chip timers. These are the handle by which the user interfaces with the timers.

Enumerator

TIMER1	Timer 1: Advanced-control 16-bit Timer - Chapter 12 RM0383
TIMER2	Timer 2: General Purpose 32-bit Timer - Chapter 13 RM0383
TIMER3	Timer 3: General Purpose 16-bit Timer - Chapter 13 RM0383
TIMER4	Timer 4: General Purpose 16-bit Timer - Chapter 13 RM0383
TIMER5	Timer 5: General Purpose 32-bit Timer - Chapter 13 RM0383
TIMER9	Timer 9: High Speed APB2 General Purpose 16-bit Timer - Chapter 14 RM0383
TIMER10	Timer 10: High Speed APB2 General Purpose 16-bit Timer - Chapter 14 RM0383
TIMER11	Timer 11: High Speed APB2 General Purpose 16-bit Timer - Chapter 14 RM0383
NUM_TIMERS	total number of timers used in the generation of register arrays, init looping, etc.

5.7.3.14 timer_trigger_t

```
enum timer_trigger_t
```

Contains all of the possible internal trigger sources (TRGI), used to control various slave modes. Internal Trigger Sources (ITR0-3) vary from timer to timer. See Table 49 and similar for examples

Enumerator

ITR0	TRGI is controlled by the trigger output (TRGO) of the first timer in the table
ITR1	TRGI is controlled by the trigger output (TRGO) of the second timer in the table
ITR2	TRGI is controlled by the trigger output (TRGO) of the third timer in the table
ITR3	TRGI is controlled by the trigger output (TRGO) of the fourth timer in the table
TI1_EDGE_DETECTOR	TRGI takes on the output of the raw edge detector, without filtering
TI1_FP1	TRGI takes on the value of the filtered edge detection on input TI1 (TIMx_CH1)
TI2_FP2	TRGI takes on the value of the filtered edge detection on input TI2 (TIMx_CH2)
ETRF	TRGI takes on the value of the external trigger source

5.7.3.15 timer_udis_t

```
enum timer_udis_t
```

Updated Disable options. When UDIS = 1, update events are not generated under any circumstances

Enumerator

TIMER_UDIS_DISABLED	Update events are generated
TIMER_UDIS_ENABLED	Update events are not generated

5.7.4 Function Documentation

5.7.4.1 timer_config_get()

```
const timer_config_t* timer_config_get (
            void  )
```

Description: Retrieves the config table for the timer peripheral, normally hidden statically within the config.c file.

PRE-CONDITION: The config table has been populated/exists with a size greater than 0.

POST-CONDITION: The returned value points to the base of the config table

Returns

const timer_config_t ∗

Note: To configure a timer, visit the config_table array in timer_stm32f411_conig.c and set all the elements of the corresponding structure. e.g.

```
static const timer_config_t config_table[NUM_TIMERS] =
{
...
TIM4 {EXTERNAL_MODE1, ENCODER_MODE_1, CENTER_ALIGNED_3, 0, 15934, 775,
      ENABLED, 0},
...
}
```

Example:

```
const timer_config_t *timer_config_table = timer_config_get(void);
timer_init(timer_config_table);
```

See also

timer_init

- CHANGE HISTORY -

Date	Software Version	Initials	Description