SWC_GPTMR

Version v1.0
10/16/2023 10:08:00 AM

Table of Contents

Data Structure Index

Data Structures

Here are the data structures with brief descriptions:

File Index

File List

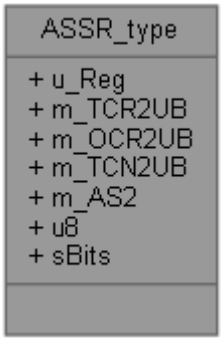
Here is a list of all files with brief descriptions:

Data Structure Documentation

ASSR_type Union Reference

: Type define of Union bit field "Asynchronous Status Register"

`#include <TMR_priv.h>`
Collaboration diagram for ASSR_type:



Data Fields

- u8 u_Reg
- struct {
- __I u8 m_TCR2UB: 1
- __I u8 m_OCR2UB: 1
- __I u8 m_TCN2UB: 1
- __IO u8 m_AS2: 1
- __IO u8: 4
- } sBits

Detailed Description

: Type define of Union bit field "Asynchronous Status Register"

Type : Union **Unit** : None

Field Documentation

__IO u8 m_AS2

Asynchronous Timer/Counter2

__I u8 m_OCR2UB

Output Compare Register2 Update Busy

__I u8 m_TCN2UB

Timer/Counter2 Update Busy

__I u8 **m_TCR2UB**

Timer/Counter Control Register2 Update Busy

struct { ... } sBits

__IO u8

Reversed

u8 **u_Reg**

Byte

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

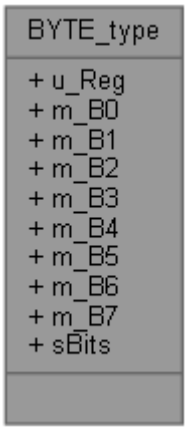
TMR_priv.h

BYTE_type Union Reference

: Type define of Union bit field of Single Byte"byte bits exchange"

```
#include <TMR_priv.h>
```

Collaboration diagram for BYTE_type:



Data Fields

- u8 u_Reg
- struct {
- __IO u8 m_B0: 1
- __IO u8 m_B1: 1
- __IO u8 m_B2: 1
- __IO u8 m_B3: 1
- __IO u8 m_B4: 1
- __IO u8 m_B5: 1
- __IO u8 m_B6: 1
- __IO u8 m_B7: 1
- } sBits

Detailed Description

: Type define of Union bit field of Single Byte"byte bits exchange"

Type : Union **Unit** : None

Field Documentation

__IO u8 m_B0

Bit 0 "LSB"

__IO u8 m_B1

Bit 1

__IO u8 m_B2

Bit 2

__IO u8 **m_B3**

Bit 3

__IO u8 **m_B4**

Bit 4

__IO u8 **m_B5**

Bit 5

__IO u8 **m_B6**

Bit 6

__IO u8 **m_B7**

Bit 7 "MSB"

struct { ... } sBits

u8 **u_Reg**

Byte

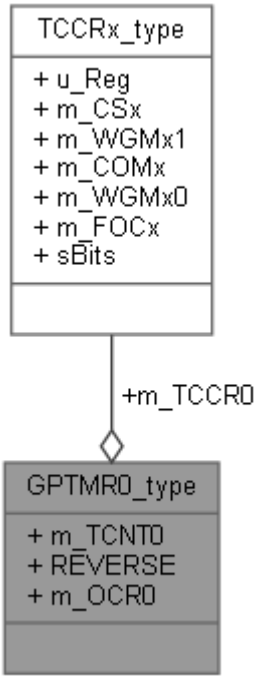
The documentation for this union was generated from the following file:

TMR_priv.h

GPTMR0_type Struct Reference

: General Purpose Input Output Registers

`#include <TMR_priv.h>`
Collaboration diagram for GPTMR0_type:



Data Fields

- __IO u8 m_TCNT0
- __IO TCCRx_type m_TCCR0
- __I u8 REVERSE [8]
- __IO u8 m_OCR0

Detailed Description

: General Purpose Input Output Registers

Type : Struct **Unit** : None

Field Documentation

__IO u8 m_OCR0

Output Compare Register

__IO TCCRx_type m_TCCR0

Timer/Counter Control Register

__IO u8 **m_TCNT0**

Timer/Counter Register

__I u8 **REVERSE[8]**

Reversed

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

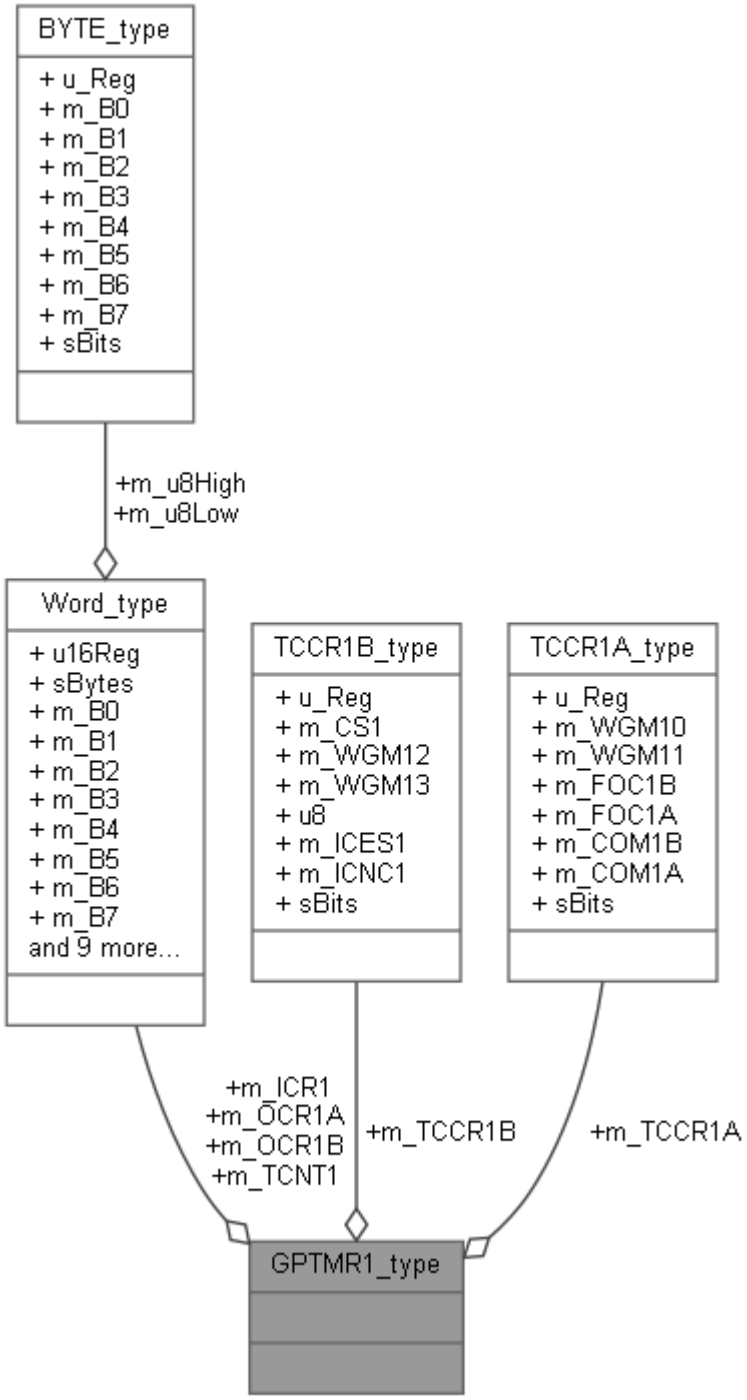
TMR_priv.h

GPTMR1_type Struct Reference

: General Purpose Input Output Registers

`#include <TMR_priv.h>`

Collaboration diagram for GPTMR1_type:



Data Fields

- __IO Word_type m_ICR1

- __IO Word_type m_OCR1B
- __IO Word_type m_OCR1A
- __IO Word_type m_TCNT1
- __IO TCCR1B_type m_TCCR1B
- __IO TCCR1A_type m_TCCR1A

Detailed Description

: General Purpose Input Output Registers

Type : Struct **Unit** : None

Field Documentation

__IO Word_type m_ICR1

Input Compare Register

__IO Word_type m_OCR1A

Output Compare Register

__IO Word_type m_OCR1B

Output Compare Register

__IO TCCR1A_type m_TCCR1A

Timer/Counter Control Register

__IO TCCR1B_type m_TCCR1B

Timer/Counter Control Register

__IO Word_type m_TCNT1

Output Compare Register

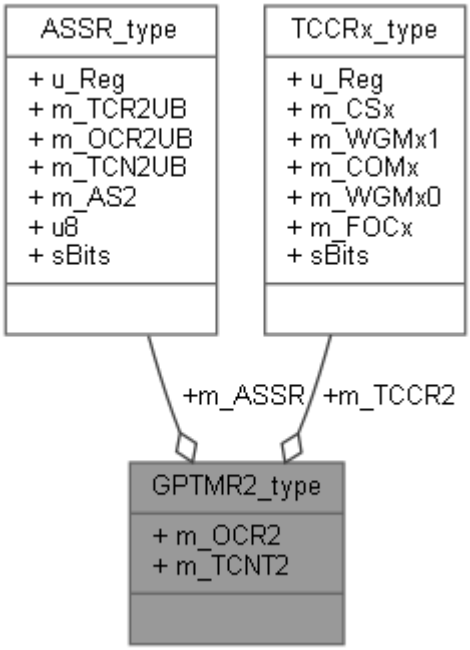
The documentation for this struct was generated from the following file:

TMR_priv.h

GPTMR2_type Struct Reference

: General Purpose Input Output Registers

`#include <TMR_priv.h>`
Collaboration diagram for GPTMR2_type:



Data Fields

- __IO ASSR_type m_ASSR
- __IO u8 m_OCR2
- __IO u8 m_TCNT2
- __IO TCCRx_type m_TCCR2

Detailed Description

: General Purpose Input Output Registers

Type : Struct **Unit** : None

Field Documentation

__IO ASSR_type m_ASSR

Asynchronous Status Register

__IO u8 m_OCR2

Output Compare Register

__IO TCCRx_type m_TCCR2

Timer/Counter Control Register

__IO u8 m_TCNT2

Timer/Counter Register

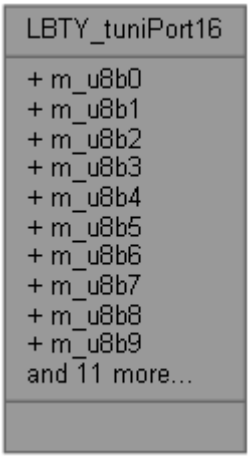
The documentation for this struct was generated from the following file:

TMR_priv.h

LBTY_tuniPort16 Union Reference

`#include <LBTY_int.h>`

Collaboration diagram for LBTY_tuniPort16:



Data Fields

- struct {
- u8 m_u8b0:1
- u8 m_u8b1:1
- u8 m_u8b2:1
- u8 m_u8b3:1
- u8 m_u8b4:1
- u8 m_u8b5:1
- u8 m_u8b6:1
- u8 m_u8b7:1
- u8 m_u8b8:1
- u8 m_u8b9:1
- u8 m_u8b10:1
- u8 m_u8b11:1
- u8 m_u8b12:1
- u8 m_u8b13:1
- u8 m_u8b14:1
- u8 m_u8b15:1
- } sBits
- struct {
- u8 m_u8low
- u8 m_u8high
- } sBytes
- u16 u_u16Word

Field Documentation

u8 m_u8b0

u8 m_u8b1

u8 m_u8b10

u8 m_u8b11

u8 m_u8b12

u8 m_u8b13

u8 m_u8b14

u8 m_u8b15

u8 m_u8b2

u8 m_u8b3

u8 m_u8b4

u8 m_u8b5

u8 m_u8b6

u8 m_u8b7

u8 m_u8b8

u8 m_u8b9

u8 m_u8high

u8 m_u8low

struct { ... } sBits

struct { ... } sBytes

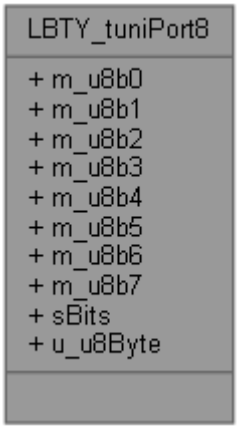
u16 u_u16Word

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

- H:/0/Workspaces/MCU Drivers/ATmega32/MCAL/SWC_BSW/LBTY_int.h

LBTY_tuniPort8 Union Reference

`#include <LBTY_int.h>`
Collaboration diagram for LBTY_tuniPort8:



Data Fields

- struct {
- u8 m_u8b0:1
- u8 m_u8b1:1
- u8 m_u8b2:1
- u8 m_u8b3:1
- u8 m_u8b4:1
- u8 m_u8b5:1
- u8 m_u8b6:1
- u8 m_u8b7:1
- } sBits
- u8 u_u8Byte

Detailed Description

Union Byte bit by bit

Field Documentation

u8 m_u8b0

u8 m_u8b1

u8 m_u8b2

u8 m_u8b3

u8 m_u8b4

u8 m_u8b5

u8 m_u8b6

u8 m_u8b7

struct { ... } sBits

u8 u_u8Byte

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

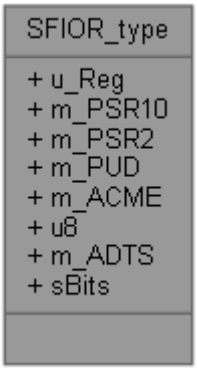
- H:/0/Workspaces/MCU Drivers/ATmega32/MCAL/SWC_BSW/LBTY_int.h

SFIOR_type Union Reference

: Type define of Union bit field "Special Function I/O Register"

```
#include <TMR_priv.h>
```

Collaboration diagram for SFIOR_type:



Data Fields

- u8 u_Reg
- struct {
- __IO u8 m_PSR10: 1
- __IO u8 m_PSR2: 1
- __IO u8 m_PUD: 1
- __IO u8 m_ACME: 1
- __IO u8: 1
- __IO u8 m_ADTS: 3
- } sBits

Detailed Description

: Type define of Union bit field "Special Function I/O Register"

Type : Union **Unit** : None

Field Documentation

__IO u8 m_ACME

Analog Comparator Multiplexer Enable

__IO u8 m_ADTS

ADC Auto Trigger Source

__IO u8 m_PSR10

Prescaler Reset Timer/Counter1 and Timer/Counter0

__IO u8 **m_PSR2**

Prescaler Reset Timer/Counter2

__IO u8 **m_PUD**

Pull-up disable

struct { ... } sBits

__IO u8

Reversed

u8 **u_Reg**

Byte

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

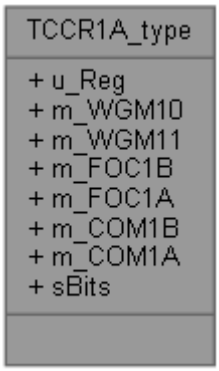
TMR_priv.h

TCCR1A_type Union Reference

: Type define of Union bit field "Timer/Counter Control Register A"

```
#include <TMR_priv.h>
```

Collaboration diagram for TCCR1A_type:



Data Fields

- u8 u_Reg
- struct {
- __IO u8 m_WGM10: 1
- __IO u8 m_WGM11: 1
- __IO u8 m_FOC1B: 1
- __IO u8 m_FOC1A: 1
- __IO u8 m_COM1B: 2
- __IO u8 m_COM1A: 2
- } sBits

Detailed Description

: Type define of Union bit field "Timer/Counter Control Register A"

Type : Union **Unit** : None

Field Documentation

__IO u8 m_COM1A

Compare Match Output Mode

__IO u8 m_COM1B

Compare Match Output Mode

__IO u8 m_FOC1A

Force Output Compare

__IO u8 m_FOC1B

Force Output Compare

__IO u8 **m_WGM10**

Waveform Generation Mode

__IO u8 **m_WGM11**

Waveform Generation Mode

struct { ... } sBits

u8 **u_Reg**

Byte

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

TMR_priv.h

TCCR1B_type Union Reference

: Type define of Union bit field "Timer/Counter Control Register B"

```
#include <TMR_priv.h>
```

Collaboration diagram for TCCR1B_type:

Data Fields

- u8 u_Reg
- struct {
- __IO u8 m_CS1: 3
- __IO u8 m_WGM12: 1
- __IO u8 m_WGM13: 1
- __IO u8: 1
- __IO u8 m_ICES1: 1
- __IO u8 m_ICNC1: 1
- } sBits

Detailed Description

: Type define of Union bit field "Timer/Counter Control Register B"

Type : Union **Unit** : None

Field Documentation

__IO u8 m_CS1

Clock Select

__IO u8 m_ICES1

Input Capture Edge Select

__IO u8 m_ICNC1

Input Capture Noise Canceler

__IO u8 m_WGM12

Waveform Generation Mode

__IO u8 **m_WGM13**

Waveform Generation Mode

struct { ... } sBits

__IO u8

Reversed

u8 **u_Reg**

Byte

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

TMR_priv.h

TCCRx_type Union Reference

: Type define of Union bit field "Timer/Counter Control Register"

`#include <TMR_priv.h>`
Collaboration diagram for TCCRx_type:

Data Fields

- u8 u_Reg
- struct {
- __IO u8 m_CSx: 3
- __IO u8 m_WGMx1: 1
- __IO u8 m_COMx: 2
- __IO u8 m_WGMx0: 1
- __IO u8 m_FOCx: 1
- } sBits

Detailed Description

: Type define of Union bit field "Timer/Counter Control Register"

Type : Union **Unit** : None

Field Documentation

__IO u8 m_COMx

Compare Match Output Mode

__IO u8 m_CSx

Clock Select

__IO u8 m_FOCx

Force Output Compare

__IO u8 **m_WGMx0**

Waveform Generation Mode

__IO u8 **m_WGMx1**

Waveform Generation Mode

struct { ... } sBits

u8 **u_Reg**

Byte

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

TMR_priv.h

TIFR_type Union Reference

: Type define of Union bit field "Timer/Counter Interrupt Flag Register Reg"

`#include <TMR_priv.h>`
Collaboration diagram for TIFR_type:

Data Fields

- u8 u_Reg
- struct {
- __IO u8 m_TOV0: 1
- __IO u8 m_OCF0: 1
- __IO u8 m_TOV1: 1
- __IO u8 m_OCF1B: 1
- __IO u8 m_OCF1A: 1
- __IO u8 m_ICF1: 1
- __IO u8 m_TOV2: 1
- __IO u8 m_OCF2: 1
- } sBits

Detailed Description

: Type define of Union bit field "Timer/Counter Interrupt Flag Register Reg"

Type : Union **Unit** : None

Field Documentation

__IO u8 m_ICF1

Timer/Counter1, Input Capture Flag

__IO u8 m_OCF0

Timer/Counter1 Output Compare Match Flag

__IO u8 **m_OCF1A**

Timer/Counter1 Output Compare Match Flag

__IO u8 **m_OCF1B**

Timer/Counter1 Output Compare Match Flag

__IO u8 **m_OCF2**

Timer/Counter2 Output Compare Match Flag

__IO u8 **m_TOV0**

Timer/Counter1 Overflow Flag

__IO u8 **m_TOV1**

Timer/Counter1 Overflow Flag

__IO u8 **m_TOV2**

Timer/Counter2 Overflow Flag

struct { ... } sBits

u8 **u_Reg**

Byte

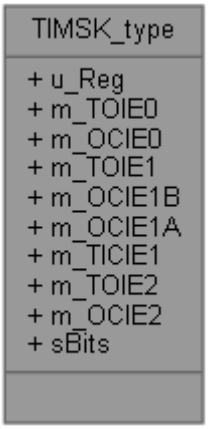
The documentation for this union was generated from the following file:

TMR_priv.h

TIMSK_type Union Reference

: Type define of Union bit field "Timer/Counter Control Register"

`#include <TMR_priv.h>`
Collaboration diagram for TIMSK_type:



Data Fields

- u8 u_Reg
- struct {
- __IO u8 m_TOIE0: 1
- __IO u8 m_OCIE0: 1
- __IO u8 m_TOIE1: 1
- __IO u8 m_OCIE1B: 1
- __IO u8 m_OCIE1A: 1
- __IO u8 m_TICIE1: 1
- __IO u8 m_TOIE2: 1
- __IO u8 m_OCIE2: 1
- } sBits

Detailed Description

: Type define of Union bit field "Timer/Counter Control Register"

Type : Union **Unit** : None

Field Documentation

__IO u8 m_OCIE0

Timer/Counter1 Output Compare Match Interrupt Enable

__IO u8 m_OCIE1A

Timer/Counter1 Output Compare Match Interrupt Enable

__IO u8 **m_OCIE1B**

Timer/Counter1 Output Compare Match Interrupt Enable

__IO u8 **m_OCIE2**

Timer/Counter2 Output Compare Match Interrupt Enable

__IO u8 **m_TICIE1**

Timer/Counter1, Input Capture Interrupt Enable

__IO u8 **m_TOIE0**

Timer/Counter1 Overflow Interrupt Enable

__IO u8 **m_TOIE1**

Timer/Counter1 Overflow Interrupt Enable

__IO u8 **m_TOIE2**

Timer/Counter2 Overflow Interrupt Enable

struct { ... } sBits

u8 **u_Reg**

Byte

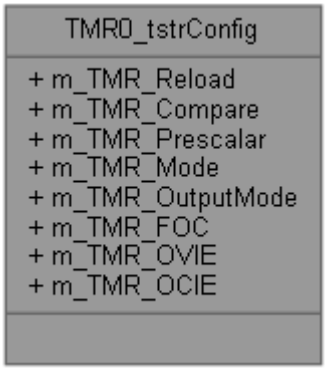
The documentation for this union was generated from the following file:

TMR_priv.h

TMR0_tstrConfig Struct Reference

`#include <TMR_int.h>`

Collaboration diagram for TMR0_tstrConfig:



Data Fields

- u8 m_TMR_Reload
- u8 m_TMR_Compare
- TMR0_tenuClockSource m_TMR_Prescalar
- TMRx_u8_tenuWaveGenerationMode m_TMR_Mode
- TMRx_u8_tenuCompareOutputMode m_TMR_OutputMode
- LBTY_tenuFlagStatus m_TMR_FOC
- LBTY_tenuFlagStatus m_TMR_OVIE
- LBTY_tenuFlagStatus m_TMR_OCIE

Field Documentation

u8 m_TMR_Compare

LBTY_tenuFlagStatus m_TMR_FOC

TMRx_u8_tenuWaveGenerationMode m_TMR_Mode

LBTY_tenuFlagStatus m_TMR_OCIE

TMRx_u8_tenuCompareOutputMode m_TMR_OutputMode

LBTY_tenuFlagStatus m_TMR_OVIE

TMR0_tenuClockSource m_TMR_Prescalar

u8 m_TMR_Reload

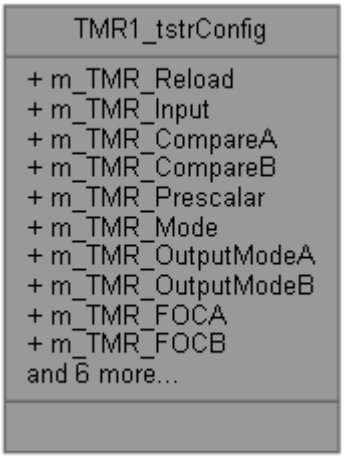
The documentation for this struct was generated from the following file:

TMR_int.h

TMR1_tstrConfig Struct Reference

`#include <TMR_int.h>`

Collaboration diagram for TMR1_tstrConfig:



Data Fields

- u16 m_TMR_Reload
- u16 m_TMR_Input
- u16 m_TMR_CompareA
- u16 m_TMR_CompareB
- TMR1_tenuClockSource m_TMR_Prescalar
- TMR1_tenuWaveGenerationMode m_TMR_Mode
- TMR1_tenuCompareOutputMode m_TMR_OutputModeA
- TMR1_tenuCompareOutputMode m_TMR_OutputModeB
- LBTY_tenuFlagStatus m_TMR_FOCA
- LBTY_tenuFlagStatus m_TMR_FOCB
- LBTY_tenuFlagStatus m_TMR_TICIE
- LBTY_tenuFlagStatus m_TMR_OCIEA
- LBTY_tenuFlagStatus m_TMR_OCIEB
- LBTY_tenuFlagStatus m_TMR_TOIE
- LBTY_tenuFlagStatus m_TMR_InputNoise
- TMR1_tenuInputCaptureEdgeSelect m_TMR_InputEdge

Field Documentation

u16 m_TMR_CompareA

u16 m_TMR_CompareB

LBTY_tenuFlagStatus m_TMR_FOCA

LBTY_tenuFlagStatus m_TMR_FOCB

u16 m_TMR_Input

TMR1_tenuInputCaptureEdgeSelect m_TMR_InputEdge

LBTY_tenuFlagStatus m_TMR_InputNoise

TMR1_tenuWaveGenerationMode m_TMR_Mode

LBTY_tenuFlagStatus m_TMR_OCIEA

LBTY_tenuFlagStatus m_TMR_OCIEB

TMR1_tenuCompareOutputMode m_TMR_OutputModeA

TMR1_tenuCompareOutputMode m_TMR_OutputModeB

TMR1_tenuClockSource m_TMR_Prescalar

u16 m_TMR_Reload

LBTY_tenuFlagStatus m_TMR_TICIE

LBTY_tenuFlagStatus m_TMR_TOIE

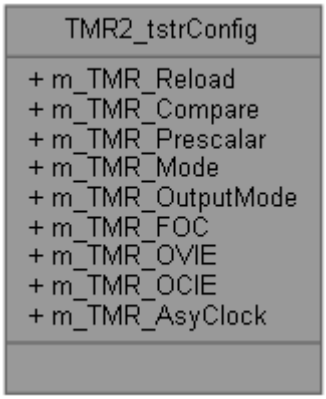
The documentation for this struct was generated from the following file:

TMR_int.h

TMR2_tstrConfig Struct Reference

`#include <TMR_int.h>`

Collaboration diagram for TMR2_tstrConfig:



Data Fields

- u8 m_TMR_Reload
- u8 m_TMR_Compare
- TMR2_tenuClockSource m_TMR_Prescalar
- TMRx_u8_tenuWaveGenerationMode m_TMR_Mode
- TMRx_u8_tenuCompareOutputMode m_TMR_OutputMode
- LBTY_tenuFlagStatus m_TMR_FOC
- LBTY_tenuFlagStatus m_TMR_OVIE
- LBTY_tenuFlagStatus m_TMR_OCIE
- TMR2_tenuInputCaptureEdgeSelect m_TMR_AsyClock

Field Documentation

TMR2_tenuInputCaptureEdgeSelect m_TMR_AsyClock

u8 m_TMR_Compare

LBTY_tenuFlagStatus m_TMR_FOC

TMRx_u8_tenuWaveGenerationMode m_TMR_Mode

LBTY_tenuFlagStatus m_TMR_OCIE

TMRx_u8_tenuCompareOutputMode m_TMR_OutputMode

LBTY_tenuFlagStatus m_TMR_OVIE

TMR2_tenuClockSource m_TMR_Prescalar

u8 m_TMR_Reload

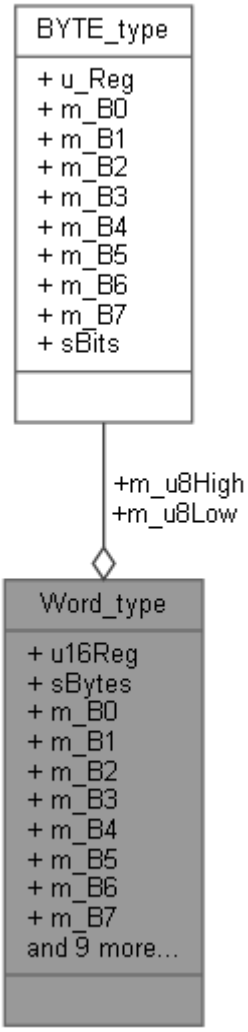
The documentation for this struct was generated from the following file:

TMR_int.h

Word_type Union Reference

: Type define of Union bit field of Half Word "bits exchange"

`#include <TMR_priv.h>`
Collaboration diagram for Word_type:



Data Fields

- u16 u16Reg
- struct {
- BYTE_type m_u8Low
- BYTE_type m_u8High
- } sBytes
- struct {
- __IO u8 m_B0: 1
- __IO u8 m_B1: 1
- __IO u8 m_B2: 1
- __IO u8 m_B3: 1
- __IO u8 m_B4: 1
- __IO u8 m_B5: 1
- __IO u8 m_B6: 1

- __IO u8 m_B7: 1
- __IO u8 m_B8: 1
- __IO u8 m_B9: 1
- __IO u8 m_B10: 1
- __IO u8 m_B11: 1
- __IO u8 m_B12: 1
- __IO u8 m_B13: 1
- __IO u8 m_B14: 1
- __IO u8 m_B15: 1
- } sBits

Detailed Description

: Type define of Union bit field of Half Word "bits exchange"

Type : Union **Unit** : None

Field Documentation

__IO u8 m_B0

Bit 0 "LSB"

__IO u8 m_B1

Bit 1

__IO u8 m_B10

Bit 10

__IO u8 m_B11

Bit 11

__IO u8 m_B12

Bit 12

__IO u8 m_B13

Bit 13

__IO u8 m_B14

Bit 14

__IO u8 m_B15

Bit 15 "MSB"

__IO u8 m_B2

Bit 2

__IO u8 **m_B3**

Bit 3

__IO u8 **m_B4**

Bit 4

__IO u8 **m_B5**

Bit 5

__IO u8 **m_B6**

Bit 6

__IO u8 **m_B7**

Bit 7

__IO u8 **m_B8**

Bit 8

__IO u8 **m_B9**

Bit 9

BYTE_type **m_u8High**

High Byte

BYTE_type **m_u8Low**

Low Byte

struct { ... } sBits

struct { ... } sBytes

u16 **u16Reg**

half Word

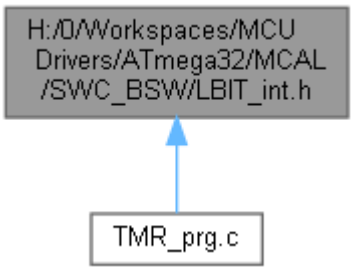
The documentation for this union was generated from the following file:

TMR_priv.h

File Documentation

H:/0/Workspaces/MCU Drivers/ATmega32/MCAL/SWC_BSW/LBIT_int.h File Reference

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



Macros

- #define _BV(bit) (1u<<(bit))
- #define SET_BIT(REG, bit) ((REG) |= (1u<<(bit)))
- #define CLR_BIT(REG, bit) ((REG) &= ~(1u<<(bit)))
- #define TOG_BIT(REG, bit) ((REG) ^= (1u<<(bit)))
- #define SET_BYTE(REG, bit) ((REG) |= (0xFFu<<(bit)))
- #define CLR_BYTE(REG, bit) ((REG) &= ~(0xFFu<<(bit)))
- #define TOG_BYTE(REG, bit) ((REG) ^= (0xFFu<<(bit)))
- #define SET_MASK(REG, MASK) ((REG) |= (MASK))
- #define CLR_MASK(REG, MASK) ((REG) &= ~(MASK))
- #define TOG_MASK(REG, MASK) ((REG) ^= (MASK))
- #define GET_MASK(REG, MASK) ((REG) & (MASK))
- #define SET_REG(REG) ((REG) = ~(0u))
- #define CLR_REG(REG) ((REG) = (0u))
- #define TOG_REG(REG) ((REG) ^= ~(0u))
- #define GET_BIT(REG, bit) (((REG)>>(bit)) & 0x01u)
- #define GET_NIB(REG, bit) (((REG)>>(bit)) & 0x0Fu)
- #define GET_BYTE(REG, bit) (((REG)>>(bit)) & 0xFFu)
- #define ASSIGN_BIT(REG, bit, value) ((REG) = ((REG) & ~(0x01u<<(bit))) | (((value) & 0x01u)<<(bit)))
- #define ASSIGN_NIB(REG, bit, value) ((REG) = ((REG) & ~(0x0Fu<<(bit))) | (((value) & 0x0Fu)<<(bit)))
- #define ASSIGN_BYTE(REG, bit, value) ((REG) = ((REG) & ~(0xFFu<<(bit))) | (((value) & 0xFFu)<<(bit)))
- #define CON_u8Bits(b7, b6, b5, b4, b3, b2, b1, b0)

  (0b##b7##b6##b5##b4##b3##b2##b1##b0)
- #define CON_u16Bits(b15, b14, b13, b12, b11, b10, b9, b8, b7, b6, b5, b4, b3, b2, b1, b0)

  (0b##b15##b14##b13##b12##b11##b10##b9##b8##b7##b6##b5##b4##b3##b2##b1##b0)

Macro Definition Documentation

#define _BV(bit) (1u<<(bit))

#define ASSIGN_BIT(REG, bit, value) ((REG) = ((REG) & ~(0x01u<<(bit))) | (((value) & 0x01u)<<(bit)))

#define ASSIGN_BYTE(REG, bit, value) ((REG) = ((REG) & ~(0xFFu<<(bit))) | (((value) & 0xFFu)<<(bit)))

#define ASSIGN_NIB(REG, bit, value) ((REG) = ((REG) & ~(0x0Fu<<(bit))) | (((value) & 0x0Fu)<<(bit)))

#define CLR_BIT(REG, bit) ((REG) &= ~(1u<<(bit)))

#define CLR_BYTE(REG, bit) ((REG) &= ~(0xFFu<<(bit)))

#define CLR_MASK(REG, MASK) ((REG) &= ~(MASK))

#define CLR_REG(REG) ((REG) = (0u))

#define CON_u16Bits(b15, b14, b13, b12, b11, b10, b9, b8, b7, b6, b5, b4, b3, b2, b1, b0)

(0b##b15##b14##b13##b12##b11##b10##b9##b8##b7##b6##b5##b4##b3##b2##b1##b0)

#define CON_u8Bits(b7, b6, b5, b4, b3, b2, b1, b0)

(0b##b7##b6##b5##b4##b3##b2##b1##b0)

#define GET_BIT(REG, bit) (((REG)>>(bit)) & 0x01u)

#define GET_BYTE(REG, bit) (((REG)>>(bit)) & 0xFFu)

#define GET_MASK(REG, MASK) ((REG) & (MASK))

#define GET_NIB(REG, bit) (((REG)>>(bit)) & 0x0Fu)

#define SET_BIT(REG, bit) ((REG) |= (1u<<(bit)))

Bitwise Operation

```
#define SET_BYTE( REG,   bit)   ((REG) |=   (0xFFu<<(bit)))

#define SET_MASK( REG,   MASK)   ((REG) |=   (MASK))

#define SET_REG( REG)   ((REG)   = ~(0u))

#define TOG_BIT( REG,   bit)   ((REG) ^=   (1u<<(bit)))

#define TOG_BYTE( REG,   bit)   ((REG) ^=   (0xFFu<<(bit)))

#define TOG_MASK( REG,   MASK)   ((REG) ^=   (MASK))

#define TOG_REG( REG)   ((REG) ^= ~(0u))
```

LBIT_int.h

Go to the documentation of this file.

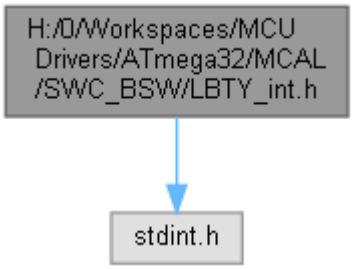
```
1 /*
******************************************************************************* */
2 /* ********************** FILE DEFINITION SECTION ************************** */
3 /* ******************************************************************************* */
4 /* File Name   : LBIT_int.h                                                    */
5 /* Author      : MAAM                                                          */
6 /* Version     : v01                                                           */
7 /* date        : Mar 24, 2023                                                  */
8 /* description : Bitwise Library                                               */
9 /* ******************************************************************************* */
10 /* ************************ HEADER FILES INCLUDES ************************  */
11 /* ******************************************************************************* */
12
13 #ifndef LBIT_INT_H_
14 #define LBIT_INT_H_
15
16 /* ******************************************************************************* */
17 /* ********************** TYPE_DEF/STRUCT/ENUM SECTION ********************** */
18 /* ******************************************************************************* */
19
20 /* ******************************************************************************* */
21 /* ************************** MACRO/DEFINE SECTION ************************** */
22 /* ******************************************************************************* */
23
24 #define _BV(bit)                                          (1u<<(bit))
25
27 #define SET_BIT(REG, bit)                                 ((REG) |=  (1u<<(bit)))
28 #define CLR_BIT(REG, bit)                                 ((REG) &= ~(1u<<(bit)))
29 #define TOG_BIT(REG, bit)                                 ((REG) ^=  (1u<<(bit)))
30
31 #define SET_BYTE(REG, bit)                                ((REG) |=  (0xFFu<<(bit)))
32 #define CLR_BYTE(REG, bit)                                ((REG) &= ~(0xFFu<<(bit)))
33 #define TOG_BYTE(REG, bit)                                ((REG) ^=  (0xFFu<<(bit)))
34
35 #define SET_MASK(REG, MASK)                               ((REG) |=  (MASK))
36 #define CLR_MASK(REG, MASK)                               ((REG) &= ~(MASK))
37 #define TOG_MASK(REG, MASK)                               ((REG) ^=  (MASK))
38 #define GET_MASK(REG, MASK)                               ((REG) &   (MASK))
39
40 #define SET_REG(REG)                                      ((REG)  = ~(0u))
41 #define CLR_REG(REG)                                      ((REG)  =  (0u))
42 #define TOG_REG(REG)                                      ((REG) ^= ~(0u))
43
44 #define GET_BIT(REG, bit)                                 (((REG)>>(bit)) & 0x01u)
45 #define GET_NIB(REG, bit)                                 (((REG)>>(bit)) & 0x0Fu)
46 #define GET_BYTE(REG, bit)                                (((REG)>>(bit)) & 0xFFu)
47
48 #define ASSIGN_BIT(REG, bit, value)                      ((REG) = ((REG) & ~(0x01u<<(bit)))
| (((value) & 0x01u)<<(bit)))
49 #define ASSIGN_NIB(REG, bit, value)                      ((REG) = ((REG) & ~(0x0Fu<<(bit)))
| (((value) & 0x0Fu)<<(bit)))
50 #define ASSIGN_BYTE(REG, bit, value)                     ((REG) = ((REG) & ~(0xFFu<<(bit)))
| (((value) & 0xFFu)<<(bit)))
51
52 /*
53 #define ASSIGN_BIT(REG,bit,value)                         do{
\
54                                                            REG &= ~(0x01u<<bit);
\
55                                                            REG |=  ((value & 0x01u)<<bit);
\
56                                                           }while(0)
57 */
58
59 /*           bits together in an u8 register           */
60 #define CON_u8Bits(b7, b6, b5, b4, b3, b2, b1, b0)
\
61
(0b##b7##b6##b5##b4##b3##b2##b1##b0)
62
63 /*           bits together in an u16 register          */
64 #define CON_u16Bits(b15, b14, b13, b12, b11, b10, b9, b8, b7, b6, b5, b4, b3, b2, b1,
b0)   \
```

```
65
(0b##b15##b14##b13##b12##b11##b10##b9##b8##b7##b6##b5##b4##b3##b2##b1##b0)
66
67 /* ************************************************************************** */
68 /* ***************************** CONST SECTION ****************************** */
69 /* ************************************************************************** */
70
71 /* ************************************************************************** */
72 /* ***************************** VARIABLE SECTION ***************************** */
73 /* ************************************************************************** */
74
75 /* ************************************************************************** */
76 /* ***************************** FUNCTION SECTION ***************************** */
77 /* ************************************************************************** */
78
79
80 #endif /* LBIT_INT_H_ */
81 /***************************** E N D (LBIT_int.h) ******************************/
```

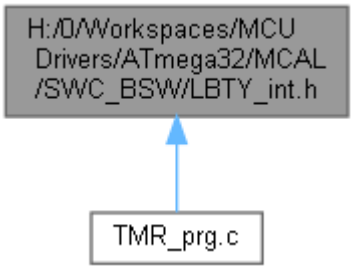
H:/0/Workspaces/MCU Drivers/ATmega32/MCAL/SWC_BSW/LBTY_int.h File Reference

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

Include dependency graph for LBTY_int.h:



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



Data Structures

- union LBTY_tuniPort8union LBTY_tuniPort16

Macros

- #define __IO volatile
- #define __O volatile
- #define __I volatile const
- #define LBTY_u8vidNOP()
- #define LBTY_NULL ((void *) 0U)
- #define LBTY_u8ZERO ((u8)0x00U)
- #define LBTY_u8MAX ((u8)0xFFU)
- #define LBTY_s8MAX ((s8)0x7F)
- #define LBTY_s8MIN ((s8)0x80)
- #define LBTY_u16ZERO ((u16)0x0000U)
- #define LBTY_u16MAX ((u16)0xFFFFU)
- #define LBTY_s16MAX ((u16)0x7FFF)
- #define LBTY_s16MIN ((u16)0x8000)
- #define LBTY_u32ZERO ((u32)0x00000000UL)
- #define LBTY_u32MAX ((u32)0xFFFFFFFFUL)
- #define LBTY_s32MAX ((u32)0x7FFFFFFFL)
- #define LBTY_s32MIN ((u32)0x80000000L)
- #define LBTY_u64ZERO ((u64)0x0000000000000000ULL)
- #define LBTY_u64MAX ((u64)0xFFFFFFFFFFFFFFFFULL)
- #define LBTY_s64MAX ((u64)0x7FFFFFFFFFFFFFFFLL)
- #define LBTY_s64MIN ((u64)0x8000000000000000LL)

Typedefs

- typedef uint8_t u8
- typedef uint16_t u16
- typedef uint32_t u32
- typedef uint64_t u64
- typedef int8_t s8
- typedef int16_t s16
- typedef int32_t s32
- typedef int64_t s64
- typedef float f32
- typedef double f64
- typedef u8 * pu8
- typedef u16 * pu16
- typedef u32 * pu32
- typedef u64 * pu64
- typedef s8 * ps8
- typedef s16 * ps16
- typedef s32 * ps32
- typedef s64 * ps64

Enumerations

- enum LBTY_tenuFlagStatus { LBTY_RESET = 0, LBTY_SET = !LBTY_RESET }
- enum LBTY_tenuBoolean { LBTY_TRUE = 0x55, LBTY_FALSE = 0xAA }
- enum LBTY_tenuErrorStatus { LBTY_OK = (u16)0, LBTY_NOK, LBTY_NULL_POINTER, LBTY_INDEX_OUT_OF_RANGE, LBTY_NO_MASTER_CHANNEL, LBTY_READ_ERROR, LBTY_WRITE_ERROR, LBTY_UNDEFINED_ERROR, LBTY_IN_PROGRESS }

Macro Definition Documentation

#define __I volatile const

#define __IO volatile

#define __O volatile

#define LBTY_NULL ((void *) 0U)

#define LBTY_s16MAX ((u16)0x7FFF)

#define LBTY_s16MIN ((u16)0x8000)

#define LBTY_s32MAX ((u32)0x7FFFFFFFL)

#define LBTY_s32MIN ((u32)0x80000000L)

#define LBTY_s64MAX ((u64)0x7FFFFFFFFFFFFFFFLL)

#define LBTY_s64MIN ((u64)0x8000000000000000LL)

#define LBTY_s8MAX ((s8)0x7F)

#define LBTY_s8MIN ((s8)0x80)

#define LBTY_u16MAX ((u16)0xFFFFU)

#define LBTY_u16ZERO ((u16)0x0000U)

#define LBTY_u32MAX ((u32)0xFFFFFFFFUL)

#define LBTY_u32ZERO ((u32)0x00000000UL)

#define LBTY_u64MAX ((u64)0xFFFFFFFFFFFFFFFFULL)

#define LBTY_u64ZERO ((u64)0x0000000000000000ULL)

#define LBTY_u8MAX ((u8)0xFFU)

#define LBTY_u8vidNOP()

#define LBTY_u8ZERO ((u8)0x00U)

Data Types Limitation

Typedef Documentation

typedef float f32

Standard Real Decimal number

typedef double f64

typedef s16* ps16

typedef s32* ps32

typedef s64* ps64

typedef s8* ps8

Standard Pointer to Signed Byte/Word/Long_Word

typedef u16* pu16

typedef u32* pu32

typedef u64* pu64

typedef u8* pu8

Standard Pointer to Unsigned Byte/Word/Long_Word

typedef int16_t s16

typedef int32_t s32

typedef int64_t s64

typedef int8_t s8

Standard Signed Byte/Word/Long_Word

typedef uint16_t u16

typedef uint32_t u32

typedef uint64_t u64

typedef uint8_t u8

Data Types New Definitions Standard Unsigned Byte/Word/Long_Word

---

Enumeration Type Documentation

enum LBTY_tenuBoolean

Boolean type

Enumerator:

LBTY_TRUE	
LBTY_FALSE	

```
96              {
97   LBTY_TRUE = 0x55,
98   LBTY_FALSE = 0xAA
99 } LBTY_tenuBoolean;
```

enum LBTY_tenuErrorStatus

Error Return type

Enumerator:

LBTY_OK	
LBTY_NOK	
LBTY_NULL_POINTER	
LBTY_INDEX_OUT_OF_RANGE	
LBTY_NO_MASTER_CHANNEL	
LBTY_READ_ERROR	
LBTY_WRITE_ERROR	
LBTY_UNDEFINED_ERROR	
LBTY_IN_PROGRESS	

```
102             {
103   LBTY_OK = (u16)0,
104   LBTY_NOK,
105   LBTY_NULL_POINTER,
106   LBTY_INDEX_OUT_OF_RANGE,
107   LBTY_NO_MASTER_CHANNEL,
108   LBTY_READ_ERROR,
109   LBTY_WRITE_ERROR,
110   LBTY_UNDEFINED_ERROR,
111   LBTY_IN_PROGRESS        /* Error is not available, wait for availability */
112 } LBTY_tenuErrorStatus;
```

enum LBTY_tenuFlagStatus

Flag Status type

Enumerator:

LBTY_RESET	
LBTY_SET	

```
90              {
91    LBTY_RESET = 0,
92    LBTY_SET = !LBTY_RESET
93 } LBTY_tenuFlagStatus;
```

LBTY_int.h

Go to the documentation of this file.

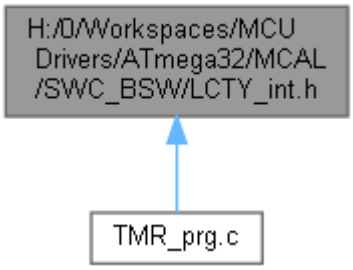
```
1 /*
2 /* ********************** FILE DEFINITION SECTION ************************** */
3 /* ************************************************************************* */
4 /* File Name   : LBTY_int.h                                                  */
5 /* Author      : MAAM                                                        */
6 /* Version     : v01                                                         */
7 /* date        : Mar 23, 2023                                                */
8 /* description : Basic Library                                               */
9 /* ************************************************************************* */
10 /* ************************ HEADER FILES INCLUDES ************************  */
11 /* ************************************************************************* */
12
13 #ifndef _LBTY_INT_H_
14 #define _LBTY_INT_H_
15
16 #include <stdint.h>
17
18 /* ************************************************************************* */
19 /* ****************************** TYPE_DEF SECTION ************************* */
20 /* ************************************************************************* */
21
24 typedef uint8_t     u8 ;
25 typedef uint16_t    u16;
26 typedef uint32_t    u32;
27 typedef uint64_t    u64;
28
30 typedef int8_t      s8 ;
31 typedef int16_t     s16;
32 typedef int32_t     s32;
33 typedef int64_t     s64;
34
36 typedef float       f32;
37 typedef double      f64;
38
40 typedef u8*         pu8 ;
41 typedef u16*        pu16;
42 typedef u32*        pu32;
43 typedef u64*        pu64;
44
46 typedef s8*         ps8 ;
47 typedef s16*        ps16;
48 typedef s32*        ps32;
49 typedef s64*        ps64;
50
51 /* ************************************************************************* */
52 /* ************************** MACRO/DEFINE SECTION ************************* */
53 /* ************************************************************************* */
54
55 /*****************************************************************************/
56 #define __IO    volatile
57 #define __O     volatile
58 #define __I     volatile const
59 /*****************************************************************************/
60
61 #define LBTY_u8vidNOP()
62 #define LBTY_NULL       ((void *) 0U)
63
65 #define LBTY_u8ZERO     ((u8)0x00U)
66 #define LBTY_u8MAX      ((u8)0xFFU)
67 #define LBTY_s8MAX      ((s8)0x7F )
68 #define LBTY_s8MIN      ((s8)0x80 )
69
70 #define LBTY_u16ZERO    ((u16)0x0000U)
71 #define LBTY_u16MAX     ((u16)0xFFFFU)
72 #define LBTY_s16MAX     ((u16)0x7FFF )
73 #define LBTY_s16MIN     ((u16)0x8000 )
74
75 #define LBTY_u32ZERO    ((u32)0x00000000UL)
76 #define LBTY_u32MAX     ((u32)0xFFFFFFFFUL)
77 #define LBTY_s32MAX     ((u32)0x7FFFFFFFL )
78 #define LBTY_s32MIN     ((u32)0x80000000L )
79
```

```
80 #define LBTY_u64ZERO     ((u64)0x0000000000000000ULL)
81 #define LBTY_u64MAX      ((u64)0xFFFFFFFFFFFFFFFFULL)
82 #define LBTY_s64MAX      ((u64)0x7FFFFFFFFFFFFFFFLL )
83 #define LBTY_s64MIN      ((u64)0x8000000000000000LL )
84
85 /* ********************************************************************************* */
86 /* ****************************** ENUM SECTION ************************************* */
87 /* ********************************************************************************* */
88
90 typedef enum {
91     LBTY_RESET = 0,
92     LBTY_SET = !LBTY_RESET
93 } LBTY_tenuFlagStatus;
94
96 typedef enum {
97   LBTY_TRUE = 0x55,
98   LBTY_FALSE = 0xAA
99 } LBTY_tenuBoolean;
100
102 typedef enum {
103   LBTY_OK = (u16)0,
104   LBTY_NOK,
105   LBTY_NULL_POINTER,
106   LBTY_INDEX_OUT_OF_RANGE,
107   LBTY_NO_MASTER_CHANNEL,
108   LBTY_READ_ERROR,
109   LBTY_WRITE_ERROR,
110   LBTY_UNDEFINED_ERROR,
111   LBTY_IN_PROGRESS         /* Error is not available, wait for availability */
112 } LBTY_tenuErrorStatus;
113
114 /* ********************************************************************************* */
115 /* ****************************** STRUCT SECTION ************************************* */
116 /* ********************************************************************************* */
117
119 typedef union {
120   struct {
121     u8 m_u8b0 :1;       // LSB
122     u8 m_u8b1 :1;
123     u8 m_u8b2 :1;
124     u8 m_u8b3 :1;
125     u8 m_u8b4 :1;
126     u8 m_u8b5 :1;
127     u8 m_u8b6 :1;
128     u8 m_u8b7 :1;       // MSB
129   } sBits;
130   u8 u_u8Byte;
131 } LBTY_tuniPort8;
132
133 typedef union {
134   struct  {
135     u8 m_u8b0  :1;      // LSB
136     u8 m_u8b1  :1;
137     u8 m_u8b2  :1;
138     u8 m_u8b3  :1;
139     u8 m_u8b4  :1;
140     u8 m_u8b5  :1;
141     u8 m_u8b6  :1;
142     u8 m_u8b7  :1;
143     u8 m_u8b8  :1;
144     u8 m_u8b9  :1;
145     u8 m_u8b10 :1;
146     u8 m_u8b11 :1;
147     u8 m_u8b12 :1;
148     u8 m_u8b13 :1;
149     u8 m_u8b14 :1;
150     u8 m_u8b15 :1;      // MSB
151   } sBits;
152   struct {
153     u8 m_u8low;
154     u8 m_u8high;
155   } sBytes;
156   u16 u_u16Word;
157 } LBTY_tuniPort16;
158
159 /* ********************************************************************************* */
160 /* ***************************** FUNCTION SECTION ************************************ */
```

```
161 /* ************************************************************************** */
162
163
164 #endif /* _LBTY_INT_H_ */
165 /*************************** E N D (LBTY_int.h) ****************************/
```

H:/0/Workspaces/MCU Drivers/ATmega32/MCAL/SWC_BSW/LCTY_int.h File Reference

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



Macros

- #define LCTY_PROGMEM __attribute__((__progmem__))
- #define LCTY_PURE __attribute__((__pure__))
- #define LCTY_INLINE __attribute__((always_inline)) static inline
- #define LCTY_INTERRUPT __attribute__((interrupt))
- #define CTY_PACKED __attribute__((__packed__))
- #define LCTY_CONST __attribute__((__const__))
- #define LCTY_DPAGE __attribute__((dp))
- #define LCTY_NODPAGE __attribute__((nodp))
- #define LCTY_SECTION(section) __attribute__((section(# section)))
- #define LCTY_ASM(cmd) __asm__ __volatile__ (# cmd ::)

Macro Definition Documentation

#define CTY_PACKED __attribute__((__packed__))

#define LCTY_ASM(cmd) __asm__ __volatile__ (# cmd ::)

#define LCTY_CONST __attribute__((__const__))

#define LCTY_DPAGE __attribute__((dp))

#define LCTY_INLINE __attribute__((always_inline)) static inline

#define LCTY_INTERRUPT __attribute__((interrupt))

#define LCTY_NODPAGE __attribute__((nodp))

#define LCTY_PROGMEM __attribute__((__progmem__))

#define LCTY_PURE __attribute__((__pure__))

#define LCTY_SECTION(section) __attribute__((section(# section)))

LCTY_int.h

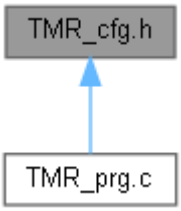
Go to the documentation of this file.

```
1 /*
2 /* ********************** FILE DEFINITION SECTION ************************** */
3 /* ************************************************************************* */
4 /* File Name   : LCTY_int.h                                                  */
5 /* Author      : MAAM                                                        */
6 /* Version     : v00                                                         */
7 /* date        : Apr 26, 2023                                                */
8 /* description : Compiler Library                                            */
9 /* ************************************************************************* */
10 /* ************************ HEADER FILES INCLUDES ************************  */
11 /* ************************************************************************* */
12
13 #ifndef LCTY_INT_H_
14 #define LCTY_INT_H_
15
16 /* ************************************************************************* */
17 /* ********************** TYPE_DEF/STRUCT/ENUM SECTION ********************** */
18 /* ************************************************************************* */
19
20 /* ************************************************************************* */
21 /* ************************** MACRO/DEFINE SECTION ************************** */
22 /* ************************************************************************* */
23
24 /* prog memory attribute */
25 #define LCTY_PROGMEM                __attribute__((__progmem__))
26
27 /* pure attribute */
28 #define LCTY_PURE                   __attribute__((__pure__))
29
30 /* Abstraction for inlining */
31 //#define LCTY_INLINE               static inline
32 #define LCTY_INLINE                 __attribute__((always_inline)) static inline
33
34 /* define function as interrupt handler */
35 #define LCTY_INTERRUPT              __attribute__((interrupt))
36
37 /* Memory packed to pass Memory padding */
38 #define CTY_PACKED                  __attribute__((__packed__))
39
40 /* Const attribute */
41 #define LCTY_CONST                  __attribute__((__const__))
42
43 /* place variable in direct page */
44 #define LCTY_DPAGE                  __attribute__((dp))
45
46 /* do not place variable in direct page */
47 #define LCTY_NODPAGE                __attribute__((nodp))
48
49 /* Sections */
50 #define LCTY_SECTION(section)       __attribute__((section( # section)))
51
52 /* Abstraction for assembly command */
53 # define LCTY_ASM(cmd)              __asm__ __volatile__ ( # cmd ::)
54
55 /* ************************************************************************* */
56 /* ***************************** CONST SECTION ***************************** */
57 /* ************************************************************************* */
58
59 /* ************************************************************************* */
60 /* ***************************** VARIABLE SECTION ************************** */
61 /* ************************************************************************* */
62
63 /* ************************************************************************* */
64 /* ***************************** FUNCTION SECTION ************************** */
65 /* ************************************************************************* */
66
67
68 #endif /* LCTY_INT_H_ */
69 /*************************** E N D (LCTY_int.h) ****************************/
```

main.c File Reference

TMR_cfg.h File Reference

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



TMR_cfg.h

Go to the documentation of this file.

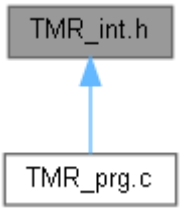
```
1 /*
2 /* ********************** FILE DEFINITION SECTION ************************** */
3 /* ************************************************************************** */
4 /* File Name   : TMR_cfg.h                                                    */
5 /* Author      : MAAM                                                         */
6 /* Version     : v01.2                                                        */
7 /* date        : Apr 5, 2023                                                  */
8 /* ************************************************************************** */
9 /* ************************ HEADER FILES INCLUDES ************************** */
10 /* ************************************************************************** */
11
12 #ifndef TMR_CFG_H_
13 #define TMR_CFG_H_
14
15 /* ************************************************************************** */
16 /* ********************** TYPE_DEF/STRUCT/ENUM SECTION ********************** */
17 /* ************************************************************************** */
18
19 #if defined(TMR0)
20
21 #define TMR0_CLOCK_SOURCE                       TMR0_Fosc_Prescaler_1
22 #define TMR0_MODE_INIT                          TMRx_u8_CTC_Mode_Mode
23 #define TMR0_COMPARE_OUTPUT_MODE                TMRx_u8_COM_Disconnected
24
25 #define TMR0_COMPARE_MATCH_INTERRUPT_INIT_STATE LBTY_SET
26 #define TMR0_OVERFLOW_INTERRUPT_INIT_STATE      LBTY_RESET
27
28 #define TMR0_COUNTER_INIT                       0x00u
29 #define TMR0_OUTPUT_COMPARE_INIT                0x9Fu
30
31 #elif defined(PWM0)
32
33 #define TMR0_CLOCK_SOURCE                       TMR0_Fosc_Prescaler_64
34 #define TMR0_MODE_INIT                          TMRx_u8_PWM_Fase_Mode
35 #define TMR0_COMPARE_OUTPUT_MODE                TMRx_u8_FastPWM_Clear_on_Match
36
37 #define TMR0 COMPARE MATCH INTERRUPT INIT STATE LBTY RESET
38 #define TMR0_OVERFLOW_INTERRUPT_INIT_STATE      LBTY_SET
39
40 // Resolution_PWM = log(Top + 1) / log(2)
41 // F_P_PWM = F_clk / (Prescaler *  2 * Top )
42 // F_F_PWM = F_clk / (Prescaler * (1 + Top))
43 // counts  = top - F CPU / (Freq * Prescaler)
44 #define PWM0_FREQ_INIT                          1000u
45 #define PWM0_DUTY_INIT                          50u
46
47 #define TMR0_COUNTER_INIT                       0x00u
48 #define TMR0_OUTPUT_COMPARE_INIT                0xFFu
49
50 #endif
51
52
/*****************************************************************************************
*******************************/
53
54 #if defined(TMR2)
55
56 #define TMR2_ASYNCHRONOUS_CLOCK                 TMR2_TOSC_Clock
57 #define TMR2_CLOCK_SOURCE                       TMR2_Fosc_Prescaler_1
58 #define TMR2 MODE INIT                          TMRx u8 Normal Mode
59 #define TMR2_COMPARE_OUTPUT_MODE                TMRx_u8_COM_Disconnected
60
61 #define TMR2_COMPARE_MATCH_INTERRUPT_INIT_STATE LBTY_RESET
62 #define TMR2_OVERFLOW_INTERRUPT_INIT_STATE      LBTY_SET
63
64 #define TMR2 COUNTER INIT                       0x00u
65 #define TMR2_OUTPUT_COMPARE_INIT                0xFFu
66
67 #elif defined(PWM2)
68
69 #define TMR2_ASYNCHRONOUS_CLOCK                 TMR2_IO_Clock
70 #define TMR2_CLOCK_SOURCE                       TMR2_Fosc_Prescaler_8
```

```
71 #define TMR2_MODE_INIT                          TMRx_u8_PWM_Fase_Mode
72 #define TMR2_COMPARE_OUTPUT_MODE                TMRx_u8_FastPWM_Clear_on_Match
73
74 #define TMR2_COMPARE_MATCH_INTERRUPT_INIT_STATE LBTY_RESET
75 #define TMR2_OVERFLOW_INTERRUPT_INIT_STATE      LBTY_SET
76
77 // Resolution_PWM = log(Top + 1) / log(2)
78 // F_P_PWM = F_clk / (Prescaler *  2 * Top )
79 // F_F_PWM = F_clk / (Prescaler * (1 + Top))
80 // counts  = top - F_CPU / (Freq * Prescaler)
81 #define PWM2_FREQ_INIT                          10000u
82 #define PWM2_DUTY_INIT                          50u
83
84 #define TMR2_COUNTER_INIT                       0x00u
85 #define TMR2_OUTPUT_COMPARE_INIT                0xFFu
86
87 #endif
88
89
/*****************************************************************************
*****************************/
90
91 #if defined(TMR1)
92
93 #define TMR1_CLOCK_SOURCE                       TMR1_Fosc_Prescaler_1
94
95 #define TMR1_MODE_INIT                          TMR1_Normal_Mode
96 #define TMR1_COMPARE_OUTPUT_A_MODE              TMR1_COM_Disconnected
97 #define TMR1_COMPARE_OUTPUT_B_MODE              TMR1_COM_Disconnected
98
99 #define TMR1_INPUT_CAPTURE_INTERRUPT_STATE      LBTY_RESET
100 #define TMR1_OVERFLOW_INTERRUPT_STATE           LBTY_RESET
101 #define TMR1_COMPARE_A_MATCH_INTERRUPT_STATE    LBTY_RESET
102 #define TMR1_COMPARE_B_MATCH_INTERRUPT_STATE    LBTY_RESET
103
104 #define TMR1_INPUT_CAPTURE_NOISE_CANCELER       LBTY_SET
105 #define TMR1_INPUT_CAPTURE_EDGE_SELECT          TMR1_Capture_Rising_Edge
106
107 #define TMR1_COUNTER_INIT                       0x00u
108 #define TMR1_INPUT_CAPTURE_INIT                 0x00u
109 #define TMR1_OUTPUT_COMPARE_A_INIT              0xFFu
110 #define TMR1_OUTPUT_COMPARE_B_INIT              0xFFu
111
112 #elif defined(PWM1)
113
114 #define TMR1_CLOCK_SOURCE                       TMR1_Fosc_Prescaler_8
115 #define TMR1_MODE_INIT                          TMR1_PWM_Fase_Mode_ICR1
116 #define TMR1_COMPARE_OUTPUT_A_MODE              TMR1_COM_Disconnected
117 #define TMR1_COMPARE_OUTPUT_B_MODE              TMR1_FastPWM_Clear_on_Match
118
119 #define TMR1_INPUT_CAPTURE_INTERRUPT_STATE      LBTY_RESET
120 #define TMR1_OVERFLOW_INTERRUPT_STATE           LBTY_SET
121 #define TMR1_COMPARE_A_MATCH_INTERRUPT_STATE    LBTY_RESET
122 #define TMR1_COMPARE_B_MATCH_INTERRUPT_STATE    LBTY_RESET
123
124 #define TMR1_INPUT_CAPTURE_NOISE_CANCELER       LBTY_SET
125 #define TMR1_INPUT_CAPTURE_EDGE_SELECT          TMR1_Capture_Rising_Edge
126
127 // Resolution_PWM = log(Top + 1) / log(2)
128 // F_P_PWM = F_clk / (Prescaler *  2 * Top )
129 // F_F_PWM = F_clk / (Prescaler * (1 + Top))
130 // counts  = top - F_CPU / (Freq * Prescaler)
131 #define PWM1_FREQ_INIT                          10000u
132 #define PWM1A_DUTY_INIT                         50u
133 #define PWM1B_DUTY_INIT                         50u
134
135 #define TMR1_COUNTER_INIT                       0x00u
136 #define TMR1_INPUT_CAPTURE_INIT                 0x00u
137 #define TMR1_OUTPUT_COMPARE_A_INIT              0xFFu
138 #define TMR1_OUTPUT_COMPARE_B_INIT              0xFFu
139
140 #endif
141
142
/*****************************************************************************
*****************************/
143
```

```
144 /* ********************************************************************** */
145 /* ************************ MACRO/DEFINE SECTION ************************ */
146 /* ********************************************************************** */
147
148 /* ********************************************************************** */
149 /* ***************************** CONST SECTION **************************** */
150 /* ********************************************************************** */
151
152 /* ********************************************************************** */
153 /* ***************************** VARIABLE SECTION ************************* */
154 /* ********************************************************************** */
155
156 /* ********************************************************************** */
157 /* ***************************** FUNCTION SECTION ************************* */
158 /* ********************************************************************** */
159
160
161 #endif /* TMR_CFG_H_ */
162 /***************************** E N D (TMR_cfg.h) ******************************/
```

TMR_int.h File Reference

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



Data Structures

- struct TMR0_tstrConfigstruct TMR2_tstrConfig
- struct TMR1_tstrConfig

Macros

- #define TMR0
- #define TMR2
- #define TMR1
- #define TMR_TICK_US (1.0f/(F_CPU/1000000))

Enumerations

- enum TMR0_tenuClockSource { TMR0_NoClockSource_Disable = (u8)0u, TMR0_Fosc_Prescaler_1, TMR0_Fosc_Prescaler_8, TMR0_Fosc_Prescaler_64, TMR0_Fosc_Prescaler_256, TMR0_Fosc_Prescaler_1024, TMR0_ExternalClock_FallingEdge, TMR0_ExternalClock_RisinfEdge }
- enum TMR1_tenuClockSource { TMR1_NoClockSource_Disable = (u8)0u, TMR1_Fosc_Prescaler_1, TMR1_Fosc_Prescaler_8, TMR1_Fosc_Prescaler_64, TMR1_Fosc_Prescaler_256, TMR1_Fosc_Prescaler_1024, TMR1_ExternalClock_FallingEdge, TMR1_ExternalClock_RisinfEdge }
- enum TMR2_tenuClockSource { TMR2_NoClockSource_Disable = (u8)0u, TMR2_Fosc_Prescaler_1, TMR2_Fosc_Prescaler_8, TMR2_Fosc_Prescaler_32, TMR2_Fosc_Prescaler_64, TMR2_Fosc_Prescaler_128, TMR2_Fosc_Prescaler_256, TMR2_Fosc_Prescaler_1024 }
- enum TMRx_u8_tenuWaveGenerationMode { TMRx_u8_Normal_Mode = (u8)0u, TMRx_u8_PWM_PhaseCorrect_Mode, TMRx_u8_CTC_Mode_Mode, TMRx_u8_PWM_Fase_Mode }
- enum TMR1_tenuWaveGenerationMode { TMR1_Normal_Mode = (u8)0u, TMR1_PWM_PhaseCorrect_Mode_8bit, TMR1_PWM_PhaseCorrect_Mode_9bit, TMR1_PWM_PhaseCorrect_Mode_10bit, TMR1_CTC_Mode_Mode_ICR1, TMR1_PWM_Fase_Mode_8bit, TMR1_PWM_Fase_Mode_9bit, TMR1_PWM_Fase_Mode_10bit, TMR1_PWM_Phase_Freq_Correct_Mode_ICR1, TMR1_PWM_Phase_Freq_Correct_Mode_ICR1A, TMR1_PWM_Phase_Correct_Mode_ICR1, TMR1_PWM_Phase_Correct_Mode_ICR1A, TMR1_CTC_Mode_Mode_ICR1A, TMR1_Reserved, TMR1_PWM_Fase_Mode_ICR1, TMR1_PWM_Fase_Mode_ICR1A }
- enum TMRx_u8_tenuCompareOutputMode { TMRx_u8_COM_Disconnected = (u8)0u, TMRx_u8_COM_Toggle_on_Match, TMRx_u8_COM_Clear_on_Match, TMRx_u8_COM_Set_on_Match, TMRx_u8_FastPWM_Clear_on_Match = (u8)2u, TMRx_u8_FastPWM_Set_on_Match, TMRx_u8_PhasePWM_Clear_on_Match = (u8)2u, TMRx_u8_PhasePWM_Set_on_Match }
- enum TMR2_tenuInputCaptureEdgeSelect { TMR2_IO_Clock = (u8)0u, TMR2_TOSC_Clock }
- enum TMR1_tenuCompareOutputMode { TMR1_COM_Disconnected = (u8)0u, TMR1_COM_Toggle_on_Match, TMR1_COM_Clear_on_Match, TMR1_COM_Set_on_Match, TMR1_FastPWM_ToggleA_on_Match_Mode15 = (u8)1u, TMR1_FastPWM_Clear_on_Match = (u8)2u, TMR1_FastPWM_Set_on_Match, TMR1_PhasePWM_ToggleA_on_Match_Mode15 = (u8)1u, TMR1_PhasePWM_Clear_on_Match = (u8)2u, TMR1_PhasePWM_Set_on_Match }

- enum TMR1_tenuInputCaptureEdgeSelect { TMR1_Capture_Falling_Edge = (u8)0u, TMR1_Capture_Rising_Edge, TMR1_Capture_Off }

Functions

- void TMR0_vidSetConfig (TMR0_tstrConfig const *const pstrConfig)
- void TMR0_vidSRestConfig (TMR0_tstrConfig *const pstrConfig)
- void TMR0_vidInit (void)
- void TMR0_vidEnable (void)
- void TMR0_vidDisable (void)
- void TMR0_vidSetForceOutputCompare (void)
- void TMR0_vidResetForceOutputCompare (void)
- LBTY_tenuErrorStatus TMR2_u8Async (TMR2_tenuInputCaptureEdgeSelect u8Async)
- LBTY_tenuErrorStatus TMR0_u8SetMode (TMRx_u8_tenuWaveGenerationMode u8Mode)
- LBTY_tenuErrorStatus TMR0_u8SetOutputMode (TMRx_u8_tenuCompareOutputMode u8OutMode)
- LBTY_tenuErrorStatus TMR0_u8SetOutputCompare (u8 u8Reload)
- LBTY_tenuErrorStatus TMR0_u8SetCounter (u8 u8Reload)
- LBTY_tenuErrorStatus TMR0_u8GetOutputCompare (u8 *pu8Reload)
- LBTY_tenuErrorStatus TMR0_u8GetCounter (u8 *pu8Reload)
- void TMR0_vidSetCompareNum (u16 u16Num)
- void TMR0_vidGetCompareNum (u16 *pu16Num)
- void TMR0_vidSetOverflowNum (u16 u16Num)
- void TMR0_vidGetOverflowNum (u16 *pu16Num)
- void TMR0_vidGetTicks (u32 *pu32Tick)
- void TMR0_vidCompareMatch_Enable (void)
- void TMR0_vidCompareMatch_Disable (void)
- void TMR0_vidSetCompareMatch_Flag (void)
- void TMR0_vidClrCompareMatch_Flag (void)
- void TMR0_vidOverFlow_Enable (void)
- void TMR0_vidOverFlow_Disable (void)
- void TMR0_vidSetOverFlow_Flag (void)
- void TMR0_vidClrOverFlow_Flag (void)
- void TMR0_vidSetCallBack_CompareMatch (void(*pCallBack)(void))
- void TMR0_vidSetCallBack_OverFlow (void(*pCallBack)(void))
- void TMR2_vidSetConfig (TMR2_tstrConfig const *const pstrConfig)
- void TMR2_vidSRestConfig (TMR2_tstrConfig *const pstrConfig)
- void TMR2_vidInit (void)
- void TMR2_vidEnable (void)
- void TMR2_vidDisable (void)
- void TMR2_vidSetForceOutputCompare (void)
- void TMR2_vidResetForceOutputCompare (void)
- LBTY_tenuErrorStatus TMR2_u8SetMode (TMRx_u8_tenuWaveGenerationMode u8Mode)
- LBTY_tenuErrorStatus TMR2_u8SetOutputMode (TMRx_u8_tenuCompareOutputMode u8OutMode)
- LBTY_tenuErrorStatus TMR2_u8SetOutputCompare (u8 u8Reload)
- LBTY_tenuErrorStatus TMR2_u8SetCounter (u8 u8Reload)
- LBTY_tenuErrorStatus TMR2_u8GetOutputCompare (u8 *pu8Reload)
- LBTY_tenuErrorStatus TMR2_u8GetCounter (u8 *pu8Reload)
- void TMR2_vidSetCompareNum (u16 u16Num)
- void TMR2_vidGetCompareNum (u16 *pu16Num)
- void TMR2_vidSetOverflowNum (u16 u16Num)
- void TMR2_vidGetOverflowNum (u16 *pu16Num)
- void TMR2_vidGetTicks (u32 *pu32Tick)
- void TMR2_vidCompareMatch_Enable (void)
- void TMR2_vidCompareMatch_Disable (void)
- void TMR2_vidSetCompareMatch_Flag (void)

- void TMR2_vidClrCompareMatch_Flag (void)
- void TMR2_vidOverFlow_Enable (void)
- void TMR2_vidOverFlow_Disable (void)
- void TMR2_vidSetOverFlow_Flag (void)
- void TMR2_vidClrOverFlow_Flag (void)
- void TMR2_vidSetCallBack_CompareMatch (void(*pCallBack)(void))
- void TMR2_vidSetCallBack_OverFlow (void(*pCallBack)(void))
- void TMR1_vidSetConfig (TMR1_tstrConfig const *const pstrConfig)
- void TMR1_vidSRestConfig (TMR1_tstrConfig *const pstrConfig)
- void TMR1_vidInit (void)
- void TMR1_vidEnable (void)
- void TMR1_vidDisable (void)
- void TMR1_vidInitInputCapture (void)
- void TMR1_vidSetForceOutputCompareA (void)
- void TMR1_vidResetForceOutputCompareA (void)
- void TMR1_vidSetForceOutputCompareB (void)
- void TMR1_vidResetForceOutputCompareB (void)
- LBTY_tenuErrorStatus TMR1_u8SetMode (TMR1_tenuWaveGenerationMode u8Mode)
- LBTY_tenuErrorStatus TMR1_u8SetOutputModeA (TMR1_tenuCompareOutputMode u8OutMode)
- LBTY_tenuErrorStatus TMR1_u8SetOutputModeB (TMR1_tenuCompareOutputMode u8OutMode)
- LBTY_tenuErrorStatus TMR1_u8SetInputCapture (u16 u16Reload)
- LBTY_tenuErrorStatus TMR1_u8SetOutputCompare_A (u16 u16Reload)
- LBTY_tenuErrorStatus TMR1_u8SetOutputCompare_B (u16 u16Reload)
- LBTY_tenuErrorStatus TMR1_u8SetCounter (u16 u16Reload)
- LBTY_tenuErrorStatus TMR1_u8GetInputCapture (u16 *pu16Reload)
- LBTY_tenuErrorStatus TMR1_u8GetOutputCompare_A (u16 *pu16Reload)
- LBTY_tenuErrorStatus TMR1_u8GetOutputCompare_B (u16 *pu16Reload)
- LBTY_tenuErrorStatus TMR1_u8GetCounter (u16 *pu16Reload)
- void TMR1_vidSetOverflowNum (u16 u16Num)
- void TMR1_vidGetOverflowNum (u16 *pu16Num)
- void TMR1_vidGetTicks (u32 *pu32Tick)
- void TMR1_vidInputCapture_Enable (void)
- void TMR1_vidInputCapture_Disable (void)
- void TMR1_vidSetInputCapture_Flag (void)
- void TMR1_vidClrInputCapture_Flag (void)
- void TMR1_vidCompareMatch_A_Enable (void)
- void TMR1_vidCompareMatch_A_Disable (void)
- void TMR1_vidSetCompareMatch_A_Flag (void)
- void TMR1_vidClrCompareMatch_A_Flag (void)
- void TMR1_vidCompareMatch_B_Enable (void)
- void TMR1_vidCompareMatch_B_Disable (void)
- void TMR1_vidSetCompareMatch_B_Flag (void)
- void TMR1_vidClrCompareMatch_B_Flag (void)
- void TMR1_vidOverFlow_Enable (void)
- void TMR1_vidOverFlow_Disable (void)
- void TMR1_vidSetOverFlow_Flag (void)
- void TMR1_vidClrOverFlow_Flag (void)
- void TMR1_vidSetCallBack_CaptureEvent (void(*pCallBack)(void))
- void TMR1_vidSetCallBack_CompareMatch_A (void(*pCallBack)(void))
- void TMR1_vidSetCallBack_CompareMatch_B (void(*pCallBack)(void))
- void TMR1_vidSetCallBack_OverFlow (void(*pCallBack)(void))

Macro Definition Documentation

#define TMR0

#define TMR1

#define TMR2

#define TMR_TICK_US (1.0f/(F_CPU/1000000))

Enumeration Type Documentation

enum TMR0_tenuClockSource

Enumerator:

TMR0_NoClockSource_Disable	
TMR0_Fosc_Prescaler_1	
TMR0_Fosc_Prescaler_8	
TMR0_Fosc_Prescaler_64	
TMR0_Fosc_Prescaler_256	
TMR0_Fosc_Prescaler_1024	
TMR0_ExternalClock_FallingEdge	
TMR0_ExternalClock_RisinfEdge	

```
29              {
30      TMR0_NoClockSource_Disable = (u8)0u,
31      TMR0_Fosc_Prescaler_1,
32      TMR0_Fosc_Prescaler_8,
33      TMR0_Fosc_Prescaler_64,
34      TMR0_Fosc_Prescaler_256,
35      TMR0_Fosc_Prescaler_1024,
36      TMR0_ExternalClock_FallingEdge,
37      TMR0_ExternalClock_RisinfEdge
38  }TMR0_tenuClockSource;
```

enum TMR1_tenuClockSource

Enumerator:

TMR1_NoClockSource_Disable	
TMR1_Fosc_Prescaler_1	
TMR1_Fosc_Prescaler_8	
TMR1_Fosc_Prescaler_64	
TMR1_Fosc_Prescaler_256	
TMR1_Fosc_Presc	

aler_1024	
TMR1_ExternalClock_FallingEdge	
TMR1_ExternalClock_RisinfEdge	

```
40              {
41      TMR1_NoClockSource_Disable = (u8)0u,
42      TMR1_Fosc_Prescaler_1,
43      TMR1_Fosc_Prescaler_8,
44      TMR1_Fosc_Prescaler_64,
45      TMR1_Fosc_Prescaler_256,
46      TMR1_Fosc_Prescaler_1024,
47      TMR1_ExternalClock_FallingEdge,
48      TMR1_ExternalClock_RisinfEdge
49  }TMR1_tenuClockSource;
```

enum TMR1_tenuCompareOutputMode

Enumerator:

TMR1_COM_Disconnected	Non PWM Mode
TMR1_COM_Toggle_on_Match	
TMR1_COM_Clear_on_Match	
TMR1_COM_Set_on_Match	
TMR1_FastPWM_ToggleA_on_Match_Mode15	Fast PWM Mode
TMR1_FastPWM_Clear_on_Match	
TMR1_FastPWM_Set_on_Match	
TMR1_PhasePWM_ToggleA_on_Match_Mode15	Phase PWM Mode
TMR1_PhasePWM_Clear_on_Match	
TMR1_PhasePWM_Set_on_Match	

```
110             {
112     TMR1_COM_Disconnected = (u8)0u,
113     TMR1_COM_Toggle_on_Match,
114     TMR1_COM_Clear_on_Match,
115     TMR1_COM_Set_on_Match,
116
118     TMR1_FastPWM_ToggleA_on_Match_Mode15 = (u8)1u,
119     TMR1_FastPWM_Clear_on_Match = (u8)2u,   // Non Inverting Mode
120     TMR1_FastPWM_Set_on_Match,              // Inverting Mode
121
123     TMR1_PhasePWM_ToggleA_on_Match_Mode15 = (u8)1u,
124     TMR1_PhasePWM_Clear_on_Match = (u8)2u,  // Low Pulse
125     TMR1_PhasePWM_Set_on_Match,             // High Pulse
126
127 }TMR1_tenuCompareOutputMode;
```

enum TMR1_tenuInputCaptureEdgeSelect

Enumerator:

TMR1_Capture_F	

alling_Edge	
TMR1_Capture_Rising_Edge	
TMR1_Capture_Off	

```
129             {
130     TMR1_Capture_Falling_Edge = (u8)0u,
131     TMR1_Capture_Rising_Edge,
132     TMR1_Capture_Off
133 }TMR1_tenuInputCaptureEdgeSelect;
```

enum TMR1_tenuWaveGenerationMode

Enumerator:

TMR1_Normal_Mode	
TMR1_PWM_PhaseCorrect_Mode_8bit	
TMR1_PWM_PhaseCorrect_Mode_9bit	
TMR1_PWM_PhaseCorrect_Mode_10bit	
TMR1_CTC_Mode_Mode_ICR1	
TMR1_PWM_Fase_Mode_8bit	
TMR1_PWM_Fase_Mode_9bit	
TMR1_PWM_Fase_Mode_10bit	
TMR1_PWM_Phase_Freq_Correct_Mode_ICR1	
TMR1_PWM_Phase_Freq_Correct_Mode_ICR1A	
TMR1_PWM_Phase_Correct_Mode_ICR1	
TMR1_PWM_Phase_Correct_Mode_ICR1A	
TMR1_CTC_Mode_Mode_ICR1A	
TMR1_Reserved	
TMR1_PWM_Fase_Mode_ICR1	
TMR1_PWM_Fase_Mode_ICR1A	

```
69              {
70      TMR1_Normal_Mode = (u8)0u,
71      TMR1_PWM_PhaseCorrect_Mode_8bit,
72      TMR1_PWM_PhaseCorrect_Mode_9bit,
73      TMR1_PWM_PhaseCorrect_Mode_10bit,
74      TMR1_CTC_Mode_Mode_ICR1,            //Clear Timer on Compare Match
75      TMR1_PWM_Fase_Mode_8bit,
76      TMR1_PWM_Fase_Mode_9bit,
77      TMR1_PWM_Fase_Mode_10bit,
```

```
78      TMR1_PWM_Phase_Freq_Correct_Mode_ICR1,
79      TMR1_PWM_Phase_Freq_Correct_Mode_ICR1A,
80      TMR1_PWM_Phase_Correct_Mode_ICR1,
81      TMR1_PWM_Phase_Correct_Mode_ICR1A,
82      TMR1_CTC_Mode_Mode_ICR1A,           //Clear Timer on Compare Match
83      TMR1_Reserved,
84      TMR1_PWM_Fase_Mode_ICR1,
85      TMR1_PWM_Fase_Mode_ICR1A,
86  }TMR1_tenuWaveGenerationMode;
```

enum TMR2_tenuClockSource

Enumerator:

TMR2_NoClockSource_Disable	
TMR2_Fosc_Prescaler_1	
TMR2_Fosc_Prescaler_8	
TMR2_Fosc_Prescaler_32	
TMR2_Fosc_Prescaler_64	
TMR2_Fosc_Prescaler_128	
TMR2_Fosc_Prescaler_256	
TMR2_Fosc_Prescaler_1024	

```
51              {
52      TMR2_NoClockSource_Disable = (u8)0u,
53      TMR2_Fosc_Prescaler_1,
54      TMR2_Fosc_Prescaler_8,
55      TMR2_Fosc_Prescaler_32,
56      TMR2_Fosc_Prescaler_64,
57      TMR2_Fosc_Prescaler_128,
58      TMR2_Fosc_Prescaler_256,
59      TMR2_Fosc_Prescaler_1024
60  }TMR2_tenuClockSource;
```

enum TMR2_tenuInputCaptureEdgeSelect

Enumerator:

TMR2_IO_Clock	
TMR2_TOSC_Clock	

```
105             {
106      TMR2_IO_Clock = (u8)0u,
107      TMR2_TOSC_Clock
108  }TMR2_tenuInputCaptureEdgeSelect;
```

enum TMRx_u8_tenuCompareOutputMode

Enumerator:

TMRx_u8_COM_Disconnected	Non PWM Mode
TMRx_u8_COM_Toggle_on_Match	
TMRx_u8_COM_Clear_on_Match	

TMRx_u8_COM_Set_on_Match	
TMRx_u8_FastPWM_Clear_on_Match	Fast PWM Mode
TMRx_u8_FastPWM_Set_on_Match	
TMRx_u8_PhasePWM_Clear_on_Match	Phase PWM Mode
TMRx_u8_PhasePWM_Set_on_Match	

```
88              {
90      TMRx_u8_COM_Disconnected = (u8)0u,
91      TMRx_u8_COM_Toggle_on_Match,
92      TMRx_u8_COM_Clear_on_Match,
93      TMRx_u8_COM_Set_on_Match,
94
96      TMRx_u8_FastPWM_Clear_on_Match = (u8)2u,     // Non Inverting Mode
97      TMRx_u8_FastPWM_Set_on_Match,                // Inverting Mode
98
100      TMRx_u8_PhasePWM_Clear_on_Match = (u8)2u,    // Low Pulse
101      TMRx_u8_PhasePWM_Set_on_Match,               // High Pulse
102
103 }TMRx_u8_tenuCompareOutputMode;
```

enum TMRx_u8_tenuWaveGenerationMode

Enumerator:

TMRx_u8_Normal_Mode	
TMRx_u8_PWM_PhaseCorrect_Mode	
TMRx_u8_CTC_Mode_Mode	
TMRx_u8_PWM_Fase_Mode	

```
62              {
63      TMRx_u8_Normal_Mode = (u8)0u,
64      TMRx_u8_PWM_PhaseCorrect_Mode,
65      TMRx_u8_CTC_Mode_Mode,          //Clear Timer on Compare Match
66      TMRx_u8_PWM_Fase_Mode
67 }TMRx_u8_tenuWaveGenerationMode;
```

Function Documentation

LBTY_tenuErrorStatus TMR0_u8GetCounter (u8 * *pu8Reload*)

```
319                                                         {
320     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
321     if(pu8Reload != LBTY_NULL){
322         *pu8Reload = S_TMR0->m_TCNT0;
323     }else{
324         u8RetErrorState = LBTY_NULL_POINTER;
325     }
326     return u8RetErrorState;
327 }
```

LBTY_tenuErrorStatus TMR0_u8GetOutputCompare (u8 * *pu8Reload*)

```
309                                                             {
310     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
```

```
311     if(pu8Reload != LBTY_NULL){
312         *pu8Reload = S_TMR0->m_OCR0;
313     }else{
314         u8RetErrorState = LBTY_NULL_POINTER;
315     }
316     return u8RetErrorState;
317 }
```

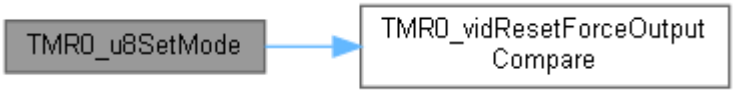
LBTY_tenuErrorStatus TMR0_u8SetCounter (u8 *u8Reload*)

```
299                                                     {
300     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
301     if(u8Reload <= LBTY_u8MAX){
302         S_TMR0->m_TCNT0 = strTMR0_Config_GLB.m_TMR_Reload = u8Reload;
303     }else{
304         u8RetErrorState = LBTY_WRITE_ERROR;
305     }
306     return u8RetErrorState;
307 }
```

LBTY_tenuErrorStatus TMR0_u8SetMode (TMRx_u8_tenuWaveGenerationMode *u8Mode*)

```
217                                                                                 {
218     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
219     switch(u8Mode){
220         case TMRx_u8_Normal_Mode:
221             S_TMR0->m_TCCR0.sBits.m_WGMx0 = GET_BIT(TMRx_u8_Normal_Mode,
TMRx_WGMx0_MASK);
222             S_TMR0->m_TCCR0.sBits.m_WGMx1 = GET_BIT(TMRx_u8_Normal_Mode,
TMRx_WGMx1_MASK);
223             break;
224         case TMRx_u8_PWM_PhaseCorrect_Mode:
225             TMR0_vidResetForceOutputCompare();
226             S_TMR0->m_TCCR0.sBits.m_WGMx0 =
GET_BIT(TMRx_u8_PWM_PhaseCorrect_Mode, TMRx_WGMx0_MASK);
227             S_TMR0->m_TCCR0.sBits.m_WGMx1 =
GET_BIT(TMRx_u8_PWM_PhaseCorrect_Mode, TMRx_WGMx1_MASK);
228             break;
229         case TMRx_u8_CTC_Mode_Mode:
230             S_TMR0->m_TCCR0.sBits.m_WGMx0 = GET_BIT(TMRx_u8_CTC_Mode_Mode,
TMRx_WGMx0_MASK);
231             S_TMR0->m_TCCR0.sBits.m_WGMx1 = GET_BIT(TMRx_u8_CTC_Mode_Mode,
TMRx_WGMx1_MASK);
232             break;
233         case TMRx_u8_PWM_Fase_Mode:
234             TMR0_vidResetForceOutputCompare();
235             S_TMR0->m_TCCR0.sBits.m_WGMx0 = GET_BIT(TMRx_u8_PWM_Fase_Mode,
TMRx_WGMx0_MASK);
236             S_TMR0->m_TCCR0.sBits.m_WGMx1 = GET_BIT(TMRx_u8_PWM_Fase_Mode,
TMRx_WGMx1_MASK);
237             break;
238         default:
239             u8RetErrorState = LBTY_WRITE_ERROR;
240             break;
241     }
242     if(u8RetErrorState == LBTY_OK){
243         strTMR0_Config_GLB.m_TMR_Mode = u8Mode;
244     }
245     return u8RetErrorState;
246 }
```

Here is the call graph for this function:



LBTY_tenuErrorStatus TMR0_u8SetOutputCompare (u8 *u8Reload*)

```
289                                                     {
290     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
291     if(u8Reload <= LBTY_u8MAX){
292         S_TMR0->m_OCR0 = strTMR0_Config_GLB.m_TMR_Compare = u8Reload;
293     }else{
294         u8RetErrorState = LBTY_WRITE_ERROR;
295     }
```

```
296     return u8RetErrorState;
297 }
```

LBTY_tenuErrorStatus TMR0_u8SetOutputMode (TMRx_u8_tenuCompareOutputMode *u8OutMode*)

```
248
{
249     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
250     TMRx_u8_tenuWaveGenerationMode u8Mode =
251             (S_TMR0->m_TCCR0.sBits.m_WGMx0<<TMRx_WGMx0_MASK) |
(S_TMR0->m_TCCR0.sBits.m_WGMx1<<TMRx_WGMx1_MASK);
252     switch(u8Mode){
253         case TMRx_u8_Normal_Mode:
254         case TMRx_u8_CTC_Mode_Mode:
255             switch(u8OutMode){
256                 case TMRx_u8_COM_Disconnected:
S_TMR0->m_TCCR0.sBits.m_COMx = TMRx_u8_COM_Disconnected;            break;
257                 case TMRx_u8_COM_Toggle_on_Match:
S_TMR0->m_TCCR0.sBits.m_COMx = TMRx_u8_COM_Toggle_on_Match;         break;
258                 case TMRx_u8_COM_Clear_on_Match:
S_TMR0->m_TCCR0.sBits.m_COMx = TMRx_u8_COM_Clear_on_Match;          break;
259                 case TMRx_u8_COM_Set_on_Match:
S_TMR0->m_TCCR0.sBits.m_COMx = TMRx_u8_COM_Set_on_Match;            break;
260                 default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
261             }
262             break;
263         case TMRx_u8_PWM_PhaseCorrect_Mode:
264             switch(u8OutMode){
265                 case
TMRx_u8_PhasePWM_Clear_on_Match:S_TMR0->m_TCCR0.sBits.m_COMx =
TMRx_u8_PhasePWM_Clear_on_Match;    break;
266                 case TMRx_u8_PhasePWM_Set_on_Match:
S_TMR0->m_TCCR0.sBits.m_COMx = TMRx_u8_PhasePWM_Set_on_Match;      break;
267                 default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
268             }
269             break;
270         case TMRx_u8_PWM_Fase_Mode:
271             switch(u8OutMode){
272                 case
TMRx_u8_FastPWM_Clear_on_Match:S_TMR0->m_TCCR0.sBits.m_COMx =
TMRx_u8_FastPWM_Clear_on_Match;      break;
273                 case TMRx_u8_FastPWM_Set_on_Match:
S_TMR0->m_TCCR0.sBits.m_COMx = TMRx_u8_FastPWM_Set_on_Match;        break;
274                 default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
275             }
276             break;
277         default:
278             u8RetErrorState = LBTY_WRITE_ERROR;
279             break;
280     }
281     if(u8RetErrorState == LBTY_OK){
282         if(u8OutMode != TMRx_u8_COM_Disconnected)
283             GPIO_u8SetPinDirection(TMR_OC0_PORT, TMR_OC0_PIN, PIN_OUTPUT);
284         strTMR0_Config_GLB.m_TMR_OutputMode = u8OutMode;
285     }
286     return u8RetErrorState;
287 }
```

void TMR0_vidClrCompareMatch_Flag (void)

```
406 {S_TIFR->sBits.m_OCF0   = LBTY_RESET;}
```

void TMR0_vidClrOverFlow_Flag (void)

```
412 {S_TIFR->sBits.m_TOV0   = LBTY_RESET;}
```

void TMR0_vidCompareMatch_Disable (void)

```
403 {S_TIMSK->sBits.m_OCIE0 = LBTY_RESET;}
```

void TMR0_vidCompareMatch_Enable (void)

```
402 {S_TIMSK->sBits.m_OCIE0  = LBTY_SET;}
```

void TMR0_vidDisable (void)

```
205                                 {
206     S_TMR0->m_TCCR0.sBits.m_CSx = TMR0_NoClockSource_Disable;
207 }
```

void TMR0_vidEnable (void)

```
198                                {
199     S_TMR0->m_TCCR0.sBits.m_CSx = strTMR0_Config_GLB.m_TMR_Prescalar;
200     if(strTMR0_Config_GLB.m_TMR_Prescalar == TMR0_ExternalClock_FallingEdge ||
201        strTMR0_Config_GLB.m_TMR_Prescalar == TMR0_ExternalClock_RisinfEdge){
202         GPIO_u8SetPinDirection(TMR_EXT0_PORT, TMR_EXT0_PIN, PIN_INPUT);
203     }
204 }
```

void TMR0_vidGetCompareNum (u16 * *pu16Num*)

```
385                                                {
386     *pu16Num = TMR0_u8CompareNum_GLB;
387 }
```

void TMR0_vidGetOverflowNum (u16 * *pu16Num*)

```
392                                                 {
393     *pu16Num = TMR0_u8OverflewNum_GLB;
394 }
```

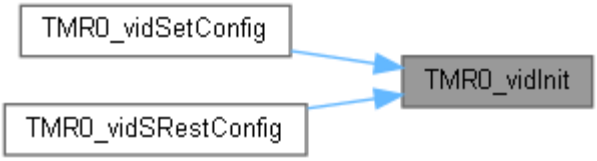
void TMR0_vidGetTicks (u32 * *pu32Tick*)

```
396                                          {
397     *pu32Tick = (u32)TMR_u8MAX * TMR0_u8OverflewNum_GLB + S_TMR0->m_TCNT0;
398 }
```

void TMR0_vidInit (void)

```
160                          {
161     //S_SFIOR->sBits.m_PSR10 = LBTY_SET;
162
163     //TMR0_vidEnable();
164     S_TMR0->m_TCCR0.sBits.m_CSx   = strTMR0_Config_GLB.m_TMR_Prescalar;
165     if(strTMR0_Config_GLB.m_TMR_Prescalar == TMR0_ExternalClock_FallingEdge ||
166        strTMR0_Config_GLB.m_TMR_Prescalar == TMR0_ExternalClock_RisinfEdge){
167         GPIO_u8SetPinDirection(TMR_EXT0_PORT, TMR_EXT0_PIN, PIN_INPUT);
168     }
169     //TMR0_u8SetMode(TMR0_MODE_INIT);
170     S_TMR0->m_TCCR0.sBits.m_WGMx0 = GET_BIT(strTMR0_Config_GLB.m_TMR_Mode,
TMRx_WGMx0_MASK);
171     S_TMR0->m_TCCR0.sBits.m_WGMx1 = GET_BIT(strTMR0_Config_GLB.m_TMR_Mode,
TMRx_WGMx1_MASK);
172     //TMR0_u8SetOutputMode(TMR0_COMPARE_OUTPUT_MODE);
173     S_TMR0->m_TCCR0.sBits.m_COMx  = strTMR0_Config_GLB.m_TMR_OutputMode;
174     if(S_TMR0->m_TCCR0.sBits.m_COMx != TMRx_u8_COM_Disconnected)
175         GPIO_u8SetPinDirection(TMR_OC0_PORT, TMR_OC0_PIN, PIN_OUTPUT);
176     //TMR0_vidResetForceOutputCompare();
177     S_TMR0->m_TCCR0.sBits.m_FOCx  = strTMR0_Config_GLB.m_TMR_FOC;
178
179 #if defined(TMR0)
180     //TMR0_u8SetOutputCompare(TMR0_OUTPUT_COMPARE_INIT);
181     S_TMR0->m_OCR0  = strTMR0_Config_GLB.m_TMR_Compare;
182     //TMR0_u8SetCounter(TMR0_COUNTER_INIT);
183     S_TMR0->m_TCNT0 = strTMR0_Config_GLB.m_TMR_Reload;
184 #elif defined(PWM0)
185     PWM_vidDisable_OC0();
186     PWM_u8SetFreq_OC0(strTMR0_Config_GLB.m_TMR_Freq);
187     PWM_u8SetDuty_OC0(strTMR0_Config_GLB.m_TMR_Duty);
188     TMR0_Reload_Delay = TMRx_RELOAD_DELAY[strTMR0_Config_GLB.m_TMR_Prescalar];
189 #endif
190
191     S_TIMSK->sBits.m_OCIE0 = strTMR0_Config_GLB.m_TMR_OCIE;
192     S_TIMSK->sBits.m_TOIE0 = strTMR0_Config_GLB.m_TMR_OVIE;
193
194     S_TIFR->sBits.m_OCF0   = LBTY_RESET;
195     S_TIFR->sBits.m_TOV0   = LBTY_RESET;
196 }
```

Here is the caller graph for this function:



void TMR0_vidOverFlow_Disable (void)

```
409 {S_TIMSK->sBits.m_TOIE0 = LBTY_RESET;}
```

void TMR0_vidOverFlow_Enable (void)

```
408 {S_TIMSK->sBits.m_TOIE0 = LBTY_SET;}
```

void TMR0_vidResetForceOutputCompare (void)

```
213                                                 {
214     S_TMR0->m_TCCR0.sBits.m_FOCx  = strTMR0_Config_GLB.m_TMR_FOC = LBTY_RESET;
215 }
```

Here is the caller graph for this function:

void TMR0_vidSetCallBack_CompareMatch (void(*)(void) *pCallBack*)

```
414                                                                         {
415     if(*pCallBack == LBTY_NULL)     return;
416     pFuncCallBack_TMR0_CompareMatch = pCallBack;
417 }
```

void TMR0_vidSetCallBack_OverFlow (void(*)(void) *pCallBack*)

```
418                                                                     {
419     if(*pCallBack == LBTY_NULL)     return;
420     pFuncCallBack_TMR0_OverFlow = pCallBack;
421 }
```

void TMR0_vidSetCompareMatch_Flag (void)

```
405 {S_TIFR->sBits.m_OCF0   = LBTY_SET;}
```

void TMR0_vidSetCompareNum (u16 *u16Num*)

```
382                                         {
383     TMR0_u8CompareNum_GLB = u16Num;
384 }
```

void TMR0_vidSetConfig (TMR0_tstrConfig const *const *pstrConfig*)

```
133                                                                     {
134     if(pstrConfig != LBTY_NULL){
135         strTMR0_Config_GLB = *pstrConfig;
136     }
137     TMR0_vidInit();
138 }
```

Here is the call graph for this function:

void TMR0_vidSetForceOutputCompare (void)

```
209                                                 {
210     S_TMR0->m_TCCR0.sBits.m_FOCx  = strTMR0_Config_GLB.m_TMR_FOC = LBTY_SET;
211 }
```

void TMR0_vidSetOverFlow_Flag (void)

```
411 {S_TIFR->sBits.m_TOV0   = LBTY_SET;}
```

void TMR0_vidSetOverflowNum (u16 *u16Num*)

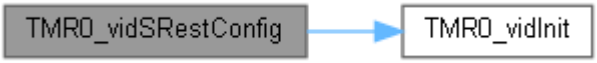
```
389                                         {
```

```
390     TMR0_u8OverflewNum_GLB = u16Num;
391 }
```

void TMR0_vidSRestConfig (TMR0_tstrConfig *const *pstrConfig*)

```
140                                                                 {
141 #if defined(PWM0)
142     strTMR0_Config_GLB.m_TMR_Freq       = PWM0_FREQ_INIT;
143     strTMR0_Config_GLB.m_TMR_Duty       = PWM0_DUTY_INIT;
144 #endif
145     strTMR0_Config_GLB.m_TMR_Reload     = TMR0_COUNTER_INIT;
146     strTMR0_Config_GLB.m_TMR_Compare    = TMR0_OUTPUT_COMPARE_INIT;
147     strTMR0_Config_GLB.m_TMR_Prescalar  = TMR0_CLOCK_SOURCE;
148     strTMR0_Config_GLB.m_TMR_Mode       = TMR0_MODE_INIT;
149     strTMR0_Config_GLB.m_TMR_OutputMode = TMR0_COMPARE_OUTPUT_MODE;
150     strTMR0_Config_GLB.m_TMR_FOC        = LBTY_RESET;
151     strTMR0_Config_GLB.m_TMR_OVIE       = TMR0_OVERFLOW_INTERRUPT_INIT_STATE;
152     strTMR0_Config_GLB.m_TMR_OCIE       =
TMR0_COMPARE_MATCH_INTERRUPT_INIT_STATE;
153
154     if(pstrConfig != LBTY_NULL){
155         *pstrConfig = strTMR0_Config_GLB;
156     }
157     TMR0_vidInit();
158 }
```

Here is the call graph for this function:



LBTY_tenuErrorStatus TMR1_u8GetCounter (u16 * *pu16Reload*)

```
1170                                                        {
1171     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
1172     if(pu16Reload != LBTY_NULL){
1173         *pu16Reload = S_TMR1->m_TCNT1.u16Reg;
1174     }else{
1175         u8RetErrorState = LBTY_NULL_POINTER;
1176     }
1177     return u8RetErrorState;
1178 }
```

LBTY_tenuErrorStatus TMR1_u8GetInputCapture (u16 * *pu16Reload*)

```
1140                                                            {
1141     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
1142     if(pu16Reload != LBTY_NULL){
1143         *pu16Reload = S_TMR1->m_ICR1.u16Reg;
1144     }else{
1145         u8RetErrorState = LBTY_NULL_POINTER;
1146     }
1147     return u8RetErrorState;
1148 }
```

LBTY_tenuErrorStatus TMR1_u8GetOutputCompare_A (u16 * *pu16Reload*)

```
1150                                                                {
1151     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
1152     if(pu16Reload != LBTY_NULL){
1153         *pu16Reload = S_TMR1->m_OCR1A.u16Reg;
1154     }else{
1155         u8RetErrorState = LBTY_NULL_POINTER;
1156     }
1157     return u8RetErrorState;
1158 }
```

LBTY_tenuErrorStatus TMR1_u8GetOutputCompare_B (u16 * *pu16Reload*)

```
1160                                                                {
1161     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
1162     if(pu16Reload != LBTY_NULL){
1163         *pu16Reload = S_TMR1->m_OCR1B.u16Reg;
1164     }else{
1165         u8RetErrorState = LBTY_NULL_POINTER;
1166     }
1167     return u8RetErrorState;
1168 }
```

LBTY_tenuErrorStatus TMR1_u8SetCounter (u16 *u16Reload*)

```
1130                                                                {
1131     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
1132     if(u16Reload <= LBTY_u16MAX){
1133         S_TMR1->m_TCNT1.u16Reg = strTMR1_Config_GLB.m_TMR_Reload = u16Reload;
1134     }else{
1135         u8RetErrorState = LBTY_WRITE_ERROR;
1136     }
1137     return u8RetErrorState;
1138 }
```

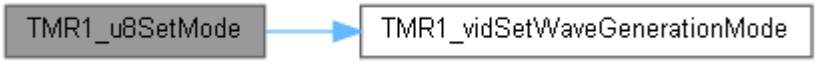
LBTY_tenuErrorStatus TMR1_u8SetInputCapture (u16 *u16Reload*)

```
1100                                                                {
1101     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
1102     if(u16Reload <= LBTY_u16MAX){
1103         S_TMR1->m_ICR1.u16Reg = strTMR1_Config_GLB.m_TMR_Input = u16Reload;
1104     }else{
1105         u8RetErrorState = LBTY_WRITE_ERROR;
1106     }
1107     return u8RetErrorState;
1108 }
```

LBTY_tenuErrorStatus TMR1_u8SetMode (TMR1_tenuWaveGenerationMode *u8Mode*)

```
961                                                                 {
962     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
963     switch(u8Mode){
964         case TMRx_u8_Normal_Mode:
965         case TMR1_PWM_PhaseCorrect_Mode_8bit:
966         case TMR1_PWM_PhaseCorrect_Mode_9bit:
967         case TMR1_PWM_PhaseCorrect_Mode_10bit:
968         case TMR1_CTC_Mode_Mode_ICR1:
969         case TMR1_PWM_Fase_Mode_8bit:
970         case TMR1_PWM_Fase_Mode_9bit:
971         case TMR1_PWM_Fase_Mode_10bit:
972         case TMR1_PWM_Phase_Freq_Correct_Mode_ICR1:
973         case TMR1_PWM_Phase_Freq_Correct_Mode_ICR1A:
974         case TMR1_PWM_Phase_Correct_Mode_ICR1:
975         case TMR1_PWM_Phase_Correct_Mode_ICR1A:
976         case TMR1_CTC_Mode_Mode_ICR1A:
977         case TMR1_PWM_Fase_Mode_ICR1:
978         case TMR1_PWM_Fase_Mode_ICR1A:
979             TMR1_vidSetWaveGenerationMode(u8Mode);
980             break;
981         default:
982             u8RetErrorState = LBTY_WRITE_ERROR;
983             break;
984     }
985     return u8RetErrorState;
986 }
```

Here is the call graph for this function:



LBTY_tenuErrorStatus TMR1_u8SetOutputCompare_A (u16 *u16Reload*)

```
1110                                                                {
1111     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
1112     if(u16Reload <= LBTY_u16MAX){
1113         S_TMR1->m_OCR1A.u16Reg = strTMR1_Config_GLB.m_TMR_CompareA = u16Reload;
1114     }else{
1115         u8RetErrorState = LBTY_WRITE_ERROR;
1116     }
1117     return u8RetErrorState;
1118 }
```

LBTY_tenuErrorStatus TMR1_u8SetOutputCompare_B (u16 *u16Reload*)

```
1120                                                                {
1121     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
1122     if(u16Reload <= LBTY_u16MAX){
1123         S_TMR1->m_OCR1B.u16Reg = strTMR1_Config_GLB.m_TMR_CompareB = u16Reload;
1124     }else{
1125         u8RetErrorState = LBTY_WRITE_ERROR;
```

```
1126     }
1127     return u8RetErrorState;
1128 }
```

LBTY_tenuErrorStatus TMR1_u8SetOutputModeA (TMR1_tenuCompareOutputMode *u8OutMode*)

```
988
{
989     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
990     TMR1_tenuWaveGenerationMode u8Mode =
991             (S_TMR1->m_TCCR1A.sBits.m_WGM10<<TMRx_WGMx0_MASK) |
(S_TMR1->m_TCCR1A.sBits.m_WGM11<<TMRx_WGMx1_MASK) |
992             (S_TMR1->m_TCCR1B.sBits.m_WGM12<<TMRx_WGMx2_MASK) |
(S_TMR1->m_TCCR1B.sBits.m_WGM13<<TMRx_WGMx3_MASK);
993
994     switch(u8Mode){
995         case TMRx_u8_Normal_Mode:
996         case TMR1_CTC_Mode_Mode_ICR1:
997         case TMR1_CTC_Mode_Mode_ICR1A:
998             switch(u8OutMode){
999                 case TMR1_COM_Disconnected:     S_TMR1->m_TCCR1A.sBits.m_COM1A
= TMR1_COM_Disconnected;         break;
1000                case TMR1_COM_Toggle_on_Match:  S_TMR1->m_TCCR1A.sBits.m_COM1A
= TMR1_COM_Toggle_on_Match;      break;
1001                case TMR1_COM_Clear_on_Match:   S_TMR1->m_TCCR1A.sBits.m_COM1A
= TMR1_COM_Clear_on_Match;       break;
1002                case TMR1_COM_Set_on_Match:     S_TMR1->m_TCCR1A.sBits.m_COM1A
= TMR1_COM_Set_on_Match;         break;
1003                default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
1004            }
1005            break;
1006        case TMR1_PWM_PhaseCorrect_Mode_8bit:
1007        case TMR1_PWM_PhaseCorrect_Mode_9bit:
1008        case TMR1_PWM_PhaseCorrect_Mode_10bit:
1009        case TMR1_PWM_Phase_Freq_Correct_Mode_ICR1:
1010        case TMR1_PWM_Phase_Freq_Correct_Mode_ICR1A:
1011        case TMR1_PWM_Phase_Correct_Mode_ICR1:
1012        case TMR1_PWM_Phase_Correct_Mode_ICR1A:
1013            switch(u8OutMode){
1014                case
TMR1_PhasePWM_ToggleA_on_Match_Mode15:S_TMR1->m_TCCR1A.sBits.m_COM1A =
TMR1_PhasePWM_ToggleA_on_Match_Mode15;  break;
1015                case TMR1_PhasePWM_Clear_on_Match:
S_TMR1->m_TCCR1A.sBits.m_COM1A = TMR1_PhasePWM_Clear_on_Match;          break;
1016                case TMR1_PhasePWM_Set_on_Match:
S_TMR1->m_TCCR1A.sBits.m_COM1A = TMR1_PhasePWM_Set_on_Match;            break;
1017                default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
1018            }
1019            break;
1020        case TMR1_PWM_Fase_Mode_8bit:
1021        case TMR1_PWM_Fase_Mode_9bit:
1022        case TMR1_PWM_Fase_Mode_10bit:
1023        case TMR1_PWM_Fase_Mode_ICR1:
1024        case TMR1_PWM_Fase_Mode_ICR1A:
1025            switch(u8OutMode){
1026                case
TMR1_FastPWM_ToggleA_on_Match_Mode15:S_TMR1->m_TCCR1A.sBits.m_COM1A =
TMR1_FastPWM_ToggleA_on_Match_Mode15;    break;
1027                case TMR1_FastPWM_Clear_on_Match:
S_TMR1->m_TCCR1A.sBits.m_COM1A = TMR1_FastPWM_Clear_on_Match;            break;
1028                case TMR1_FastPWM_Set_on_Match:
S_TMR1->m_TCCR1A.sBits.m_COM1A = TMR1_FastPWM_Set_on_Match;              break;
1029                default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
1030            }
1031            break;
1032        default:
1033            u8RetErrorState = LBTY_WRITE_ERROR;
1034            break;
1035    }
1036    if(u8RetErrorState == LBTY_OK){
1037        if(u8OutMode != TMR1_COM_Disconnected)
1038            GPIO_u8SetPinDirection(TMR_OC1A_PORT, TMR_OC1A_PIN, PIN_OUTPUT);
1039        strTMR1_Config_GLB.m_TMR_OutputModeA = u8OutMode;
1040    }
1041    return u8RetErrorState;
```

```
1042 }
```

LBTY_tenuErrorStatus TMR1_u8SetOutputModeB (TMR1_tenuCompareOutputMode *u8OutMode*)

```
1044
{
1045     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
1046     TMR1_tenuWaveGenerationMode u8Mode =
1047              (S_TMR1->m_TCCR1A.sBits.m_WGM10<<TMRx_WGMx0_MASK) |
(S_TMR1->m_TCCR1A.sBits.m_WGM11<<TMRx_WGMx1_MASK) |
1048              (S_TMR1->m_TCCR1B.sBits.m_WGM12<<TMRx_WGMx2_MASK) |
(S_TMR1->m_TCCR1B.sBits.m_WGM13<<TMRx_WGMx3_MASK);
1049
1050     switch(u8Mode){
1051         case TMRx_u8_Normal_Mode:
1052         case TMR1_CTC_Mode_Mode_ICR1:
1053         case TMR1_CTC_Mode_Mode_ICR1A:
1054             switch(u8OutMode){
1055                 case TMR1_COM_Disconnected:     S_TMR1->m_TCCR1A.sBits.m_COM1B
= TMR1_COM_Disconnected;         break;
1056                 case TMR1_COM_Toggle_on_Match:  S_TMR1->m_TCCR1A.sBits.m_COM1B
= TMR1_COM_Toggle_on_Match;      break;
1057                 case TMR1_COM_Clear_on_Match:   S_TMR1->m_TCCR1A.sBits.m_COM1B
= TMR1_COM_Clear_on_Match;       break;
1058                 case TMR1_COM_Set_on_Match:     S_TMR1->m_TCCR1A.sBits.m_COM1B
= TMR1_COM_Set_on_Match;         break;
1059                 default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
1060             }
1061             break;
1062         case TMR1_PWM_PhaseCorrect_Mode_8bit:
1063         case TMR1_PWM_PhaseCorrect_Mode_9bit:
1064         case TMR1_PWM_PhaseCorrect_Mode_10bit:
1065         case TMR1_PWM_Phase_Freq_Correct_Mode_ICR1:
1066         case TMR1_PWM_Phase_Freq_Correct_Mode_ICR1A:
1067         case TMR1_PWM_Phase_Correct_Mode_ICR1:
1068         case TMR1_PWM_Phase_Correct_Mode_ICR1A:
1069             switch(u8OutMode){
1070                 case
TMR1_PhasePWM_ToggleA_on_Match_Mode15:S_TMR1->m_TCCR1A.sBits.m_COM1B =
TMR1_PhasePWM_ToggleA_on_Match_Mode15;  break;
1071                 case TMR1_PhasePWM_Clear_on_Match:
S_TMR1->m_TCCR1A.sBits.m_COM1B = TMR1_PhasePWM_Clear_on_Match;           break;
1072                 case TMR1_PhasePWM_Set_on_Match:
S_TMR1->m_TCCR1A.sBits.m_COM1B = TMR1_PhasePWM_Set_on_Match;             break;
1073                 default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
1074             }
1075             break;
1076         case TMR1_PWM_Fase_Mode_8bit:
1077         case TMR1_PWM_Fase_Mode_9bit:
1078         case TMR1_PWM_Fase_Mode_10bit:
1079         case TMR1_PWM_Fase_Mode_ICR1:
1080         case TMR1_PWM_Fase_Mode_ICR1A:
1081             switch(u8OutMode){
1082                 case
TMR1_FastPWM_ToggleA_on_Match_Mode15:S_TMR1->m_TCCR1A.sBits.m_COM1B =
TMR1_FastPWM_ToggleA_on_Match_Mode15;    break;
1083                 case TMR1_FastPWM_Clear_on_Match:
S_TMR1->m_TCCR1A.sBits.m_COM1B = TMR1_FastPWM_Clear_on_Match;             break;
1084                 case TMR1_FastPWM_Set_on_Match:
S_TMR1->m_TCCR1A.sBits.m_COM1B = TMR1_FastPWM_Set_on_Match;               break;
1085                 default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
1086             }
1087             break;
1088         default:
1089             u8RetErrorState = LBTY_WRITE_ERROR;
1090             break;
1091     }
1092     if(u8RetErrorState == LBTY_OK){
1093         if(u8OutMode != TMR1_COM_Disconnected)
1094             GPIO_u8SetPinDirection(TMR_OC1B_PORT, TMR_OC1B_PIN, PIN_OUTPUT);
1095         strTMR1_Config_GLB.m_TMR_OutputModeB = u8OutMode;
1096     }
1097     return u8RetErrorState;
1098 }
```

void TMR1_vidClrCompareMatch_A_Flag (void)

```
1348 {S_TIFR->sBits.m_OCF1A    = LBTY_RESET;}
```

void TMR1_vidClrCompareMatch_B_Flag (void)

```
1354 {S_TIFR->sBits.m_OCF1B    = LBTY_RESET;}
```

void TMR1_vidClrInputCapture_Flag (void)

```
1342 {S_TIFR->sBits.m_ICF1    = LBTY_RESET;}
```

void TMR1_vidClrOverFlow_Flag (void)

```
1360 {S_TIFR->sBits.m_TOV1    = LBTY_RESET;}
```

void TMR1_vidCompareMatch_A_Disable (void)

```
1345 {S_TIMSK->sBits.m_OCIE1A = LBTY_RESET;}
```

void TMR1_vidCompareMatch_A_Enable (void)

```
1344 {S_TIMSK->sBits.m_OCIE1A = LBTY_SET;}
```

void TMR1_vidCompareMatch_B_Disable (void)

```
1351 {S_TIMSK->sBits.m_OCIE1B = LBTY_RESET;}
```

void TMR1_vidCompareMatch_B_Enable (void)

```
1350 {S_TIMSK->sBits.m_OCIE1B = LBTY_SET;}
```

void TMR1_vidDisable (void)

```
925                              {
926     S_TMR1->m_TCCR1B.sBits.m_CS1 = TMR1_NoClockSource_Disable;
927 }
```

void TMR1_vidEnable (void)

```
916                             {
917     S_TMR1->m_TCCR1B.sBits.m_CS1 = strTMR1_Config_GLB.m_TMR_Prescalar;
918     if(strTMR1_Config_GLB.m_TMR_Prescalar == TMR1_ExternalClock_FallingEdge ||
919        strTMR1_Config_GLB.m_TMR_Prescalar == TMR1_ExternalClock_RisinfEdge){
920         GPIO_u8SetPinDirection(TMR_EXT0_PORT, TMR_EXT0_PIN, PIN_INPUT);
921         GPIO_u8SetPinDirection(TMR_EXT1_PORT, TMR_EXT1_PIN, PIN_INPUT);
922     }
923 }
```

void TMR1_vidGetOverflowNum (u16 * *pu16Num*)

```
1328                                                {
1329     *pu16Num = TMR1_u8OverflewNum_GLB;
1330 }
```

void TMR1_vidGetTicks (u32 * *pu32Tick*)

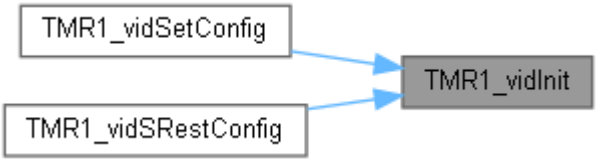
```
1332                                        {
1333     *pu32Tick = (u32)TMR_u16MAX * TMR1_u8OverflewNum_GLB +
S_TMR1->m_TCNT1.u16Reg;
1334 }
```

void TMR1_vidInit (void)

```
848                          {
849
850     //S_SFIOR->sBits.m_PSR10 = LBTY_SET;
851
852     // TMR1_u8SetMode(TMR1_MODE_INIT);
853     S_TMR1->m_TCCR1A.sBits.m_WGM10 = GET_BIT(strTMR1_Config_GLB.m_TMR_Mode,
TMRx_WGMx0_MASK);
854     S_TMR1->m_TCCR1A.sBits.m_WGM11 = GET_BIT(strTMR1_Config_GLB.m_TMR_Mode,
TMRx_WGMx1_MASK);
855     S_TMR1->m_TCCR1B.sBits.m_WGM12 = GET_BIT(strTMR1_Config_GLB.m_TMR_Mode,
TMRx_WGMx2_MASK);
856     S_TMR1->m_TCCR1B.sBits.m_WGM13 = GET_BIT(strTMR1_Config_GLB.m_TMR_Mode,
TMRx_WGMx3_MASK);
857
```

```
858     // TMR1_u8SetOutputModeA(TMR1_COMPARE_OUTPUT_A_MODE);
859     // TMR1_u8SetOutputModeB(TMR1_COMPARE_OUTPUT_B_MODE);
860     S_TMR1->m_TCCR1A.sBits.m_FOC1A = strTMR1_Config_GLB.m_TMR_FOCA;
861     S_TMR1->m_TCCR1A.sBits.m_FOC1B = strTMR1_Config_GLB.m_TMR_FOCB;
862     S_TMR1->m_TCCR1A.sBits.m_COM1A = strTMR1_Config_GLB.m_TMR_OutputModeA;
863     S_TMR1->m_TCCR1A.sBits.m_COM1B = strTMR1_Config_GLB.m_TMR_OutputModeB;
864
865     if(strTMR1_Config_GLB.m_TMR_OutputModeA != TMR1_COM_Disconnected)
866         GPIO_u8SetPinDirection(TMR_OC1A_PORT, TMR_OC1A_PIN, PIN_OUTPUT);
867     if(strTMR1_Config_GLB.m_TMR_OutputModeB != TMR1_COM_Disconnected)
868         GPIO_u8SetPinDirection(TMR_OC1B_PORT, TMR_OC1B_PIN, PIN_OUTPUT);
869
870     //TMR1_vidEnable();
871     S_TMR1->m_TCCR1B.sBits.m_CS1   = strTMR1_Config_GLB.m_TMR_Prescalar;
872     if(strTMR1_Config_GLB.m_TMR_Prescalar == TMR1_ExternalClock_FallingEdge ||
873        strTMR1_Config_GLB.m_TMR_Prescalar == TMR1_ExternalClock_RisinfEdge){
874         GPIO_u8SetPinDirection(TMR_EXT0_PORT, TMR_EXT0_PIN, PIN_INPUT);
875         GPIO_u8SetPinDirection(TMR_EXT1_PORT, TMR_EXT1_PIN, PIN_INPUT);
876     }
877     //TMR1_vidInitInputCapture();
878     S_TMR1->m_TCCR1B.sBits.m_ICNC1 = strTMR1_Config_GLB.m_TMR_InputNoise;
879     S_TMR1->m_TCCR1B.sBits.m_ICES1 = strTMR1_Config_GLB.m_TMR_InputEdge;
880     if(strTMR1_Config_GLB.m_TMR_InputEdge != TMR1_Capture_Off){
881         GPIO_u8SetPinDirection(TMR_ICP1_PORT, TMR_ICP1_PIN, PIN_INPUT);
882     }
883
884 #if defined(TMR1)
885     //TMR1_u8SetInputCapture(TMR1_INPUT_CAPTURE_INIT);
886     S_TMR1->m_ICR1.u16Reg  = strTMR1_Config_GLB.m_TMR_Input;
887     //TMR1_u8SetOutputCompare_A(TMR1_OUTPUT_COMPARE_A_INIT);
888     S_TMR1->m_OCR1A.u16Reg = strTMR1_Config_GLB.m_TMR_CompareA;
889     //TMR1_u8SetOutputCompare_B(TMR1_OUTPUT_COMPARE_B_INIT);
890     S_TMR1->m_OCR1B.u16Reg = strTMR1_Config_GLB.m_TMR_CompareB;
891     //TMR1_u8SetCounter(TMR1_COUNTER_INIT);
892     S_TMR1->m_TCNT1.u16Reg = strTMR1_Config_GLB.m_TMR_Reload;
893 #elif defined(PWM1)
894     PWM_vidDisable_OC1x();
895     PWM_u8SetFreq_OC1x(strTMR1_Config_GLB.m_TMR_Freq);
896
897     if(strTMR1_Config_GLB.m_TMR_OutputModeA != TMR1_COM_Disconnected)
898         PWM_u8SetDuty_OC1A(strTMR1_Config_GLB.m_TMR_Duty_A);
899     if(strTMR1_Config_GLB.m_TMR_OutputModeB != TMR1_COM_Disconnected)
900         PWM_u8SetDuty_OC1B(strTMR1_Config_GLB.m_TMR_Duty_B);
901
902     TMR1_Reload_Delay = TMRx_RELOAD_DELAY[strTMR1_Config_GLB.m_TMR_Prescalar];
903 #endif
904
905     S_TIMSK->sBits.m_TICIE1 = strTMR1_Config_GLB.m_TMR_TICIE;
906     S_TIMSK->sBits.m_OCIE1A = strTMR1_Config_GLB.m_TMR_OCIEA;
907     S_TIMSK->sBits.m_OCIE1B = strTMR1_Config_GLB.m_TMR_OCIEB;
908     S_TIMSK->sBits.m_TOIE1  = strTMR1_Config_GLB.m_TMR_TOIE;
909
910     S_TIFR->sBits.m_ICF1    = LBTY_RESET;
911     S_TIFR->sBits.m_OCF1A   = LBTY_RESET;
912     S_TIFR->sBits.m_OCF1B   = LBTY_RESET;
913     S_TIFR->sBits.m_TOV1    = LBTY_RESET;
914 }
```

Here is the caller graph for this function:



void TMR1_vidInitInputCapture (void)

```
929                                 {
930     S_TMR1->m_TCCR1B.sBits.m_ICNC1 = strTMR1_Config_GLB.m_TMR_InputNoise;
931     S_TMR1->m_TCCR1B.sBits.m_ICES1 = strTMR1_Config_GLB.m_TMR_InputEdge;
932     if(strTMR1_Config_GLB.m_TMR_InputEdge != TMR1_Capture_Off){
933         GPIO_u8SetPinDirection(TMR_ICP1_PORT, TMR_ICP1_PIN, PIN_INPUT);
934     }
935 }
```

void TMR1_vidInputCapture_Disable (void)

```
1339 {S_TIMSK->sBits.m_TICIE1 = LBTY_RESET;}
```

void TMR1_vidInputCapture_Enable (void)

```
1338 {S_TIMSK->sBits.m_TICIE1 = LBTY_SET;}
```

void TMR1_vidOverFlow_Disable (void)

```
1357 {S_TIMSK->sBits.m_TOIE1 = LBTY_RESET;}
```

void TMR1_vidOverFlow_Enable (void)

```
1356 {S_TIMSK->sBits.m_TOIE1 = LBTY_SET;}
```

void TMR1_vidResetForceOutputCompareA (void)

```
941                                             {
942     S_TMR1->m_TCCR1A.sBits.m_FOC1A  = strTMR1_Config_GLB.m_TMR_FOCA =
LBTY_RESET;
943 }
```

void TMR1_vidResetForceOutputCompareB (void)

```
949                                             {
950     S_TMR1->m_TCCR1A.sBits.m_FOC1B  = strTMR1_Config_GLB.m_TMR_FOCB =
LBTY_RESET;
951 }
```

void TMR1_vidSetCallBack_CaptureEvent (void(*)(void) *pCallBack*)

```
1362                                                                    {
1363     if(*pCallBack == LBTY_NULL)     return;
1364     pFuncCallBack_TMR1_CaptureEven = pCallBack;
1365 }
```

void TMR1_vidSetCallBack_CompareMatch_A (void(*)(void) *pCallBack*)

```
1366                                                                      {
1367     if(*pCallBack == LBTY_NULL)     return;
1368     pFuncCallBack_TMR1_CompareMatch_A = pCallBack;
1369 }
```

void TMR1_vidSetCallBack_CompareMatch_B (void(*)(void) *pCallBack*)

```
1370                                                                      {
1371     if(*pCallBack == LBTY_NULL)     return;
1372     pFuncCallBack_TMR1_CompareMatch_B = pCallBack;
1373 }
```

void TMR1_vidSetCallBack_OverFlow (void(*)(void) *pCallBack*)

```
1374                                                                  {
1375     if(*pCallBack == LBTY_NULL)     return;
1376     pFuncCallBack_TMR1_OverFlow = pCallBack;
1377 }
```

void TMR1_vidSetCompareMatch_A_Flag (void)

```
1347 {S_TIFR->sBits.m_OCF1A    = LBTY_SET;}
```

void TMR1_vidSetCompareMatch_B_Flag (void)

```
1353 {S_TIFR->sBits.m_OCF1B    = LBTY_SET;}
```

void TMR1_vidSetConfig (TMR1_tstrConfig const *const *pstrConfig*)

```
812                                                                     {
813     if(pstrConfig != LBTY_NULL){
814         strTMR1_Config_GLB = *pstrConfig;
815     }
816     TMR1_vidInit();
817 }
```

Here is the call graph for this function:

void TMR1_vidSetForceOutputCompareA (void)

```
937                                                 {
938     S_TMR1->m_TCCR1A.sBits.m_FOC1A  = strTMR1_Config_GLB.m_TMR_FOCA = LBTY_SET;
939 }
```

void TMR1_vidSetForceOutputCompareB (void)

```
945                                                 {
946     S_TMR1->m_TCCR1A.sBits.m_FOC1B  = strTMR1_Config_GLB.m_TMR_FOCB = LBTY_SET;
947 }
```

void TMR1_vidSetInputCapture_Flag (void)

```
1341 {S_TIFR->sBits.m_ICF1    = LBTY_SET;}
```

void TMR1_vidSetOverFlow_Flag (void)

```
1359 {S_TIFR->sBits.m_TOV1    = LBTY_SET;}
```

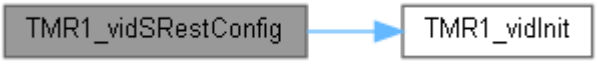
void TMR1_vidSetOverflowNum (u16 *u16Num*)

```
1325                                                {
1326     TMR1_u8OverflewNum_GLB = u16Num;
1327 }
```

void TMR1_vidSRestConfig (TMR1_tstrConfig *const *pstrConfig*)

```
819                                                                 {
820 #if defined(PWM1)
821     strTMR1_Config_GLB.m_TMR_Freq        = PWM1_FREQ_INIT;
822     strTMR1_Config_GLB.m_TMR_Duty_A      = PWM1A_DUTY_INIT;
823     strTMR1_Config_GLB.m_TMR_Duty_B      = PWM1B_DUTY_INIT;
824 #endif
825     strTMR1_Config_GLB.m_TMR_Reload      = TMR1_COUNTER_INIT;
826     strTMR1_Config_GLB.m_TMR_Input       = TMR1_INPUT_CAPTURE_INIT;
827     strTMR1_Config_GLB.m_TMR_CompareA    = TMR1_OUTPUT_COMPARE_A_INIT;
828     strTMR1_Config_GLB.m_TMR_CompareB    = TMR1_OUTPUT_COMPARE_B_INIT;
829     strTMR1_Config_GLB.m_TMR_Prescalar   = TMR1_CLOCK_SOURCE;
830     strTMR1_Config_GLB.m_TMR_Mode        = TMR1_MODE_INIT;
831     strTMR1_Config_GLB.m_TMR_OutputModeA = TMR1_COMPARE_OUTPUT_A_MODE;
832     strTMR1_Config_GLB.m_TMR_OutputModeB = TMR1_COMPARE_OUTPUT_B_MODE;
833     strTMR1_Config_GLB.m_TMR_FOCA        = LBTY_RESET;
834     strTMR1_Config_GLB.m_TMR_FOCB        = LBTY_RESET;
835     strTMR1_Config_GLB.m_TMR_TICIE       = TMR1_INPUT_CAPTURE_INTERRUPT_STATE;
836     strTMR1_Config_GLB.m_TMR_OCIEA       =
TMR1_COMPARE_A_MATCH_INTERRUPT_STATE;
837     strTMR1_Config_GLB.m_TMR_OCIEB       =
TMR1_COMPARE_B_MATCH_INTERRUPT_STATE;
838     strTMR1_Config_GLB.m_TMR_TOIE        = TMR1_OVERFLOW_INTERRUPT_STATE;
839     strTMR1_Config_GLB.m_TMR_InputNoise  = TMR1_INPUT_CAPTURE_NOISE_CANCELER;
840     strTMR1_Config_GLB.m_TMR_InputEdge   = TMR1_INPUT_CAPTURE_EDGE_SELECT;
841
842     if(pstrConfig != LBTY_NULL){
843         *pstrConfig = strTMR1_Config_GLB;
844     }
845     TMR1_vidInit();
846 }
```

Here is the call graph for this function:



LBTY_tenuErrorStatus TMR2_u8Async (TMR2_tenuInputCaptureEdgeSelect *u8Async*)

```
554                                                                                 {
555     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
556
557     switch(u8Async){
558         case TMR2_IO_Clock:
559         case TMR2_TOSC_Clock:
560             S_TIMSK->sBits.m_OCIE2 = LBTY_RESET;
561             S_TIMSK->sBits.m_TOIE2 = LBTY_RESET;
562             S_TMR2->m_ASSR.sBits.m_AS2 = strTMR2_Config_GLB.m_TMR_AsyClock =
u8Async;
563             if(u8Async == TMR2_TOSC_Clock){
```

```
564                 GPIO_u8SetPinDirection(TMR_OSC1_PORT, TMR_OSC1_PIN, PIN_INPUT);
565                 GPIO_u8SetPinDirection(TMR_OSC2_PORT, TMR_OSC2_PIN, PIN_INPUT);
566             }
567             break;
568         default:
569             u8RetErrorState = LBTY_WRITE_ERROR;
570             break;
571     }
572
573     return u8RetErrorState;
574 }
```

LBTY_tenuErrorStatus TMR2_u8GetCounter (u8 * *pu8Reload*)

```
680                                                                 {
681     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
682     if(pu8Reload != LBTY_NULL){
683         *pu8Reload = S_TMR2->m_TCNT2;
684     }else{
685         u8RetErrorState = LBTY_NULL_POINTER;
686     }
687     return u8RetErrorState;
688 }
```

LBTY_tenuErrorStatus TMR2_u8GetOutputCompare (u8 * *pu8Reload*)

```
670                                                                 {
671     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
672     if(pu8Reload != LBTY_NULL){
673         *pu8Reload = S_TMR2->m_OCR2;
674     }else{
675         u8RetErrorState = LBTY_NULL_POINTER;
676     }
677     return u8RetErrorState;
678 }
```

LBTY_tenuErrorStatus TMR2_u8SetCounter (u8 *u8Reload*)

```
659                                                             {
660     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
661     if(u8Reload <= LBTY_u8MAX){
662         TMR2_vidTimerUpdateBusy();
663         S_TMR2->m_TCNT2 = strTMR2_Config_GLB.m_TMR_Reload = u8Reload;
664     }else{
665         u8RetErrorState = LBTY_WRITE_ERROR;
666     }
667     return u8RetErrorState;
668 }
```

Here is the call graph for this function:

LBTY_tenuErrorStatus TMR2_u8SetMode (TMRx_u8_tenuWaveGenerationMode *u8Mode*)

```
576                                                                                 {
577     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
578     TMR2_vidControlUpdateBusy();
579     switch(u8Mode){
580         case TMRx_u8_Normal_Mode:
581             S_TMR2->m_TCCR2.sBits.m_WGMx0 = GET_BIT(TMRx_u8_Normal_Mode,
TMRx_WGMx0_MASK);
582             S_TMR2->m_TCCR2.sBits.m_WGMx1 = GET_BIT(TMRx_u8_Normal_Mode,
TMRx_WGMx1_MASK);
583             break;
584         case TMRx_u8_PWM_PhaseCorrect_Mode:
585             S_TMR2->m_TCCR2.sBits.m_WGMx0 =
GET_BIT(TMRx_u8_PWM_PhaseCorrect_Mode, TMRx_WGMx0_MASK);
586             S_TMR2->m_TCCR2.sBits.m_WGMx1 =
GET_BIT(TMRx_u8_PWM_PhaseCorrect_Mode, TMRx_WGMx1_MASK);
587             break;
588         case TMRx_u8_CTC_Mode_Mode:
589             S_TMR2->m_TCCR2.sBits.m_WGMx0 = GET_BIT(TMRx_u8_CTC_Mode_Mode,
TMRx_WGMx0_MASK);
```

```
590             S_TMR2->m_TCCR2.sBits.m_WGMx1 = GET_BIT(TMRx_u8_CTC_Mode_Mode,
TMRx_WGMx1_MASK);
591             break;
592         case TMRx_u8_PWM_Fase_Mode:
593             S_TMR2->m_TCCR2.sBits.m_WGMx0 = GET_BIT(TMRx_u8_PWM_Fase_Mode,
TMRx_WGMx0_MASK);
594             S_TMR2->m_TCCR2.sBits.m_WGMx1 = GET_BIT(TMRx_u8_PWM_Fase_Mode,
TMRx_WGMx1_MASK);
595             break;
596         default:
597             u8RetErrorState = LBTY_WRITE_ERROR;
598             break;
599     }
600     if(u8RetErrorState == LBTY_OK){
601         strTMR2_Config_GLB.m_TMR_Mode = u8Mode;
602     }
603     return u8RetErrorState;
604 }
```

Here is the call graph for this function:

LBTY_tenuErrorStatus TMR2_u8SetOutputCompare (u8 *u8Reload*)

```
648                                                     {
649     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
650     if(u8Reload <= LBTY_u8MAX){
651         TMR2_vidCompareUpdateBusy();
652         S_TMR2->m_OCR2 = strTMR2_Config_GLB.m_TMR_Compare = u8Reload;
653     }else{
654         u8RetErrorState = LBTY_WRITE_ERROR;
655     }
656     return u8RetErrorState;
657 }
```

Here is the call graph for this function:

LBTY_tenuErrorStatus TMR2_u8SetOutputMode (TMRx_u8_tenuCompareOutputMode *u8OutMode*)

```
606
{
607     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
608     TMRx_u8_tenuWaveGenerationMode u8Mode =
609             (S_TMR2->m_TCCR2.sBits.m_WGMx0<<TMRx_WGMx0_MASK) |
(S_TMR2->m_TCCR2.sBits.m_WGMx1<<TMRx_WGMx1_MASK);
610     TMR2_vidControlUpdateBusy();
611     switch(u8Mode){
612         case TMRx_u8_Normal_Mode:
613         case TMRx_u8_CTC_Mode_Mode:
614             switch(u8OutMode){
615                 case TMRx_u8_COM_Disconnected:
S_TMR2->m_TCCR2.sBits.m_COMx = TMRx_u8_COM_Disconnected;            break;
616                 case TMRx_u8_COM_Toggle_on_Match:
S_TMR2->m_TCCR2.sBits.m_COMx = TMRx_u8_COM_Toggle_on_Match;         break;
617                 case TMRx_u8_COM_Clear_on_Match:
S_TMR2->m_TCCR2.sBits.m_COMx = TMRx_u8_COM_Clear_on_Match;          break;
618                 case TMRx_u8_COM_Set_on_Match:
S_TMR2->m_TCCR2.sBits.m_COMx = TMRx_u8_COM_Set_on_Match;            break;
619                 default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
620             }
621             break;
622         case TMRx_u8_PWM_PhaseCorrect_Mode:
623             switch(u8OutMode){
624                 case
TMRx_u8_PhasePWM_Clear_on_Match:S_TMR2->m_TCCR2.sBits.m_COMx =
TMRx_u8_PhasePWM_Clear_on_Match;    break;
625                 case TMRx_u8_PhasePWM_Set_on_Match:
S_TMR2->m_TCCR2.sBits.m_COMx = TMRx_u8_PhasePWM_Set_on_Match;      break;
626                 default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
627             }
628             break;
629         case TMRx_u8_PWM_Fase_Mode:
```

```
630             switch(u8OutMode){
631                 case
TMRx_u8_FastPWM_Clear_on_Match:S_TMR2->m_TCCR2.sBits.m_COMx =
TMRx_u8_FastPWM_Clear_on_Match;      break;
632                 case TMRx_u8_FastPWM_Set_on_Match:
S_TMR2->m_TCCR2.sBits.m_COMx = TMRx_u8_FastPWM_Set_on_Match;        break;
633                 default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
634             }
635             break;
636         default:
637             u8RetErrorState = LBTY_WRITE_ERROR;
638             break;
639     }
640     if(u8RetErrorState == LBTY_OK){
641         if(u8OutMode != TMRx_u8_COM_Disconnected)
642             GPIO_u8SetPinDirection(TMR_OC2_PORT, TMR_OC2_PIN, PIN_OUTPUT);
643         strTMR2_Config_GLB.m_TMR_OutputMode = u8OutMode;
644     }
645     return u8RetErrorState;
646 }
```

Here is the call graph for this function:

void TMR2_vidClrCompareMatch_Flag (void)

```
769 {S_TIFR->sBits.m_OCF2    = LBTY_RESET;}
```

void TMR2_vidClrOverFlow_Flag (void)

```
775 {S_TIFR->sBits.m_TOV2    = LBTY_RESET;}
```

void TMR2_vidCompareMatch_Disable (void)

```
766 {S_TIMSK->sBits.m_OCIE2 = LBTY_RESET;}
```

void TMR2_vidCompareMatch_Enable (void)

```
765 {S_TIMSK->sBits.m_OCIE2 = LBTY_SET;}
```

void TMR2_vidDisable (void)

```
541                             {
542     TMR2_vidControlUpdateBusy();
543     S_TMR2->m_TCCR2.sBits.m_CSx = TMR2_NoClockSource_Disable;
544 }
```

Here is the call graph for this function:

void TMR2_vidEnable (void)

```
536                             {
537     TMR2_vidControlUpdateBusy();
538     S_TMR2->m_TCCR2.sBits.m_CSx = strTMR2_Config_GLB.m_TMR_Prescalar;
539 }
```

Here is the call graph for this function:

void TMR2_vidGetCompareNum (u16 * *pu16Num*)

```
748                                         {
749     *pu16Num = TMR2_u8CompareNum_GLB;
750 }
```

void TMR2_vidGetOverflowNum (u16 * *pu16Num*)

```
755                                          {
756     *pu16Num = TMR2_u8OverflewNum_GLB;
757 }
```

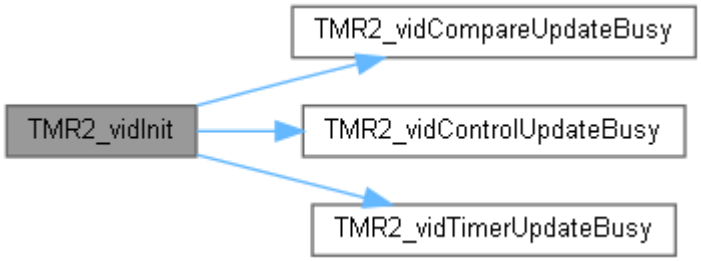
void TMR2_vidGetTicks (u32 * *pu32Tick*)

```
759                                         {
760     *pu32Tick = (u32)TMR_u8MAX * TMR2_u8OverflewNum_GLB + S_TMR2->m_TCNT2;
761 }
```

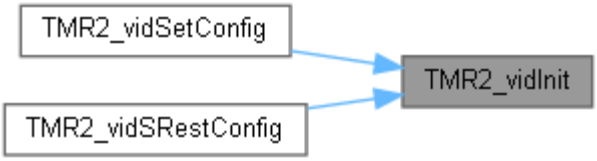
void TMR2_vidInit (void)

```
487                             {
488     //S_SFIOR->sBits.m_PSR2 = LBTY_SET;
489
490     // TMR2_vidAsync(TMR2_ASYNCHRONOUS_CLOCK);
491     S_TIMSK->sBits.m_OCIE2 = LBTY_RESET;
492     S_TIMSK->sBits.m_TOIE2 = LBTY_RESET;
493     S_TMR2->m_ASSR.sBits.m_AS2 = strTMR2_Config_GLB.m_TMR_AsyClock;
494
495     if(strTMR2_Config_GLB.m_TMR_AsyClock == TMR2_TOSC_Clock){
496         GPIO_u8SetPinDirection(TMR_OSC1_PORT, TMR_OSC1_PIN, PIN_INPUT);
497         GPIO_u8SetPinDirection(TMR_OSC2_PORT, TMR_OSC2_PIN, PIN_INPUT);
498     }
499
500     //TMR2_vidEnable();
501     TMR2_vidControlUpdateBusy();
502     S_TMR2->m_TCCR2.sBits.m_CSx   = strTMR2_Config_GLB.m_TMR_Prescalar;
503     //TMR2_u8SetMode(TMR2_MODE_INIT);
504     TMR2_vidControlUpdateBusy();
505     S_TMR2->m_TCCR2.sBits.m_WGMx0 = GET_BIT(strTMR2_Config_GLB.m_TMR_Mode,
TMRx_WGMx0_MASK);
506     S_TMR2->m_TCCR2.sBits.m_WGMx1 = GET_BIT(strTMR2_Config_GLB.m_TMR_Mode,
TMRx_WGMx1_MASK);
507     //TMR2_u8SetOutputMode(TMR2_COMPARE_OUTPUT_MODE);
508     TMR2_vidControlUpdateBusy();
509     S_TMR2->m_TCCR2.sBits.m_COMx  = strTMR2_Config_GLB.m_TMR_OutputMode;
510     if(S_TMR2->m_TCCR2.sBits.m_COMx != TMRx_u8_COM_Disconnected)
511         GPIO_u8SetPinDirection(TMR_OC2_PORT, TMR_OC2_PIN, PIN_OUTPUT);
512     //TMR2_vidSetForceOutputCompare();
513     S_TMR2->m_TCCR2.sBits.m_FOCx  = strTMR2_Config_GLB.m_TMR_FOC;
514
515 #if defined(TMR2)
516     //TMR2_u8SetOutputCompare(TMR2_OUTPUT_COMPARE_INIT);
517     TMR2_vidCompareUpdateBusy();
518     S_TMR2->m_OCR2 = strTMR2_Config_GLB.m_TMR_Compare;
519     //TMR2_u8SetCounter(TMR2_COUNTER_INIT);
520     TMR2_vidTimerUpdateBusy();
521     S_TMR2->m_TCNT2 = strTMR2_Config_GLB.m_TMR_Reload;
522 #elif defined(PWM2)
523     PWM_vidDisable_OC2();
524     PWM_u8SetFreq_OC2(strTMR2_Config_GLB.m_TMR_Freq);
525     PWM_u8SetDuty_OC2(strTMR2_Config_GLB.m_TMR_Duty);
526     TMR2_Reload_Delay = TMRx_RELOAD_DELAY[strTMR2_Config_GLB.m_TMR_Prescalar];
527 #endif
528
529     S_TIMSK->sBits.m_OCIE2 = strTMR2_Config_GLB.m_TMR_OCIE;
530     S_TIMSK->sBits.m_TOIE2 = strTMR2_Config_GLB.m_TMR_OVIE;
531
532     S_TIFR->sBits.m_OCF2   = LBTY_RESET;
533     S_TIFR->sBits.m_TOV2   = LBTY_RESET;
534 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void TMR2_vidOverFlow_Disable (void)

```
772 {S_TIMSK->sBits.m_TOIE2 = LBTY_RESET;}
```

void TMR2_vidOverFlow_Enable (void)

```
771 {S_TIMSK->sBits.m_TOIE2 = LBTY_SET;}
```

void TMR2_vidResetForceOutputCompare (void)

```
550                                         {
551     S_TMR2->m_TCCR2.sBits.m_FOCx  = strTMR2_Config_GLB.m_TMR_FOC = LBTY_RESET;
552 }
```

void TMR2_vidSetCallBack_CompareMatch (void(*)(void) *pCallBack*)

```
777                                                                 {
778     pFuncCallBack_TMR2_CompareMatch = pCallBack;
779 }
```

void TMR2_vidSetCallBack_OverFlow (void(*)(void) *pCallBack*)

```
780                                                                 {
781     pFuncCallBack_TMR2_OverFlow = pCallBack;
782 }
```

void TMR2_vidSetCompareMatch_Flag (void)

```
768 {S_TIFR->sBits.m_OCF2   = LBTY_SET;}
```

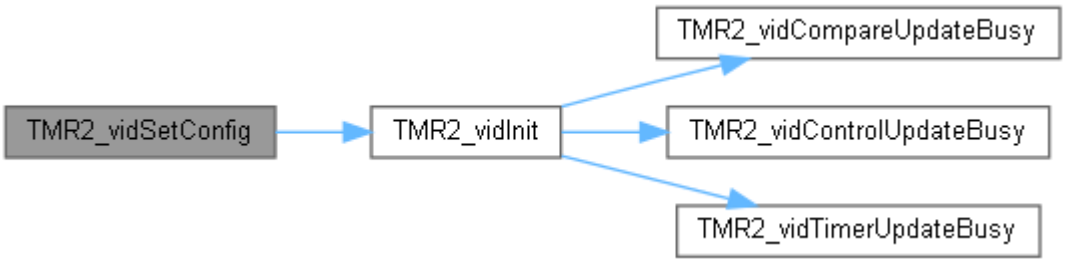
void TMR2_vidSetCompareNum (u16 *u16Num*)

```
745                                         {
746     TMR2_u8CompareNum_GLB = u16Num;
747 }
```

void TMR2_vidSetConfig (TMR2_tstrConfig const *const *pstrConfig*)

```
459                                                                 {
460     if(pstrConfig != LBTY_NULL){
461         strTMR2_Config_GLB = *pstrConfig;
462     }
463     TMR2_vidInit();
464 }
```

Here is the call graph for this function:



void TMR2_vidSetForceOutputCompare (void)

```
546                                         {
547     S_TMR2->m_TCCR2.sBits.m_FOCx  = strTMR2_Config_GLB.m_TMR_FOC = LBTY_SET;
548 }
```

void TMR2_vidSetOverFlow_Flag (void)

```
774 {S_TIFR->sBits.m_TOV2   = LBTY_SET;}
```

void TMR2_vidSetOverflowNum (u16 *u16Num*)

```
752                                             {
753     TMR2_u8OverflewNum_GLB = u16Num;
754 }
```

void TMR2_vidSRestConfig (TMR2_tstrConfig *const *pstrConfig*)

```
466                                                             {
467 #if defined(PWM2)
468     strTMR2_Config_GLB.m_TMR_Freq       = PWM2_FREQ_INIT;
469     strTMR2_Config_GLB.m_TMR_Duty       = PWM2_DUTY_INIT;
470 #endif
471     strTMR2_Config_GLB.m_TMR_Reload     = TMR2_COUNTER_INIT;
472     strTMR2_Config_GLB.m_TMR_Compare    = TMR2_OUTPUT_COMPARE_INIT;
473     strTMR2_Config_GLB.m_TMR_Prescalar  = TMR2_CLOCK_SOURCE;
474     strTMR2_Config_GLB.m_TMR_Mode       = TMR2_MODE_INIT;
475     strTMR2_Config_GLB.m_TMR_OutputMode= TMR2_COMPARE_OUTPUT_MODE;
476     strTMR2_Config_GLB.m_TMR_FOC        = LBTY_RESET;
477     strTMR2_Config_GLB.m_TMR_OVIE       = TMR2_OVERFLOW_INTERRUPT_INIT_STATE;
478     strTMR2_Config_GLB.m_TMR_OCIE       =
TMR2_COMPARE_MATCH_INTERRUPT_INIT_STATE;
479     strTMR2_Config_GLB.m_TMR_AsyClock   = TMR2_ASYNCHRONOUS_CLOCK;
480
481     if(pstrConfig != LBTY_NULL){
482         *pstrConfig = strTMR2_Config_GLB;
483     }
484     TMR2_vidInit();
485 }
```

Here is the call graph for this function:

TMR_int.h

Go to the documentation of this file.

```
1 /*
2 /* ********************** FILE DEFINITION SECTION ************************** */
3 /* ************************************************************************** */
4 /* File Name    : TMR_int.h                                                   */
5 /* Author       : MAAM                                                        */
6 /* Version      : v01.2                                                       */
7 /* date         : Apr 5, 2023                                                 */
8 /* ************************************************************************** */
9 /* ************************ HEADER FILES INCLUDES ************************  */
10 /* ************************************************************************** */
11
12 #ifndef TMR_INT_H_
13 #define TMR_INT_H_
14
15 #if !defined(TMR0) && !defined(PWM0)
16 #define TMR0
17 #endif
18 #if !defined(TMR2) && !defined(PWM2)
19 #define TMR2
20 #endif
21 #if !defined(TMR1) && !defined(PWM1)
22 #define TMR1
23 #endif
24
25 /* ************************************************************************** */
26 /* ********************** TYPE_DEF/STRUCT/ENUM SECTION ********************** */
27 /* ************************************************************************** */
28
29 typedef enum{
30     TMR0_NoClockSource_Disable = (u8)0u,
31     TMR0_Fosc_Prescaler_1,
32     TMR0_Fosc_Prescaler_8,
33     TMR0_Fosc_Prescaler_64,
34     TMR0_Fosc_Prescaler_256,
35     TMR0_Fosc_Prescaler_1024,
36     TMR0_ExternalClock_FallingEdge,
37     TMR0_ExternalClock_RisinfEdge
38 }TMR0_tenuClockSource;
39
40 typedef enum{
41     TMR1_NoClockSource_Disable = (u8)0u,
42     TMR1_Fosc_Prescaler_1,
43     TMR1_Fosc_Prescaler_8,
44     TMR1_Fosc_Prescaler_64,
45     TMR1_Fosc_Prescaler_256,
46     TMR1_Fosc_Prescaler_1024,
47     TMR1_ExternalClock_FallingEdge,
48     TMR1_ExternalClock_RisinfEdge
49 }TMR1_tenuClockSource;
50
51 typedef enum{
52     TMR2_NoClockSource_Disable = (u8)0u,
53     TMR2_Fosc_Prescaler_1,
54     TMR2_Fosc_Prescaler_8,
55     TMR2_Fosc_Prescaler_32,
56     TMR2_Fosc_Prescaler_64,
57     TMR2_Fosc_Prescaler_128,
58     TMR2_Fosc_Prescaler_256,
59     TMR2_Fosc_Prescaler_1024
60 }TMR2_tenuClockSource;
61
62 typedef enum{
63     TMRx_u8_Normal_Mode = (u8)0u,
64     TMRx_u8_PWM_PhaseCorrect_Mode,
65     TMRx_u8_CTC_Mode_Mode,          //Clear Timer on Compare Match
66     TMRx_u8_PWM_Fase_Mode
67 }TMRx_u8_tenuWaveGenerationMode;
68
69 typedef enum{
70     TMR1_Normal_Mode = (u8)0u,
71     TMR1_PWM_PhaseCorrect_Mode_8bit,
72     TMR1_PWM_PhaseCorrect_Mode_9bit,
```

```
73      TMR1_PWM_PhaseCorrect_Mode_10bit,
74      TMR1_CTC_Mode_Mode_ICR1,              //Clear Timer on Compare Match
75      TMR1_PWM_Fase_Mode_8bit,
76      TMR1_PWM_Fase_Mode_9bit,
77      TMR1_PWM_Fase_Mode_10bit,
78      TMR1_PWM_Phase_Freq_Correct_Mode_ICR1,
79      TMR1_PWM_Phase_Freq_Correct_Mode_ICR1A,
80      TMR1_PWM_Phase_Correct_Mode_ICR1,
81      TMR1_PWM_Phase_Correct_Mode_ICR1A,
82      TMR1_CTC_Mode_Mode_ICR1A,             //Clear Timer on Compare Match
83      TMR1_Reserved,
84      TMR1_PWM_Fase_Mode_ICR1,
85      TMR1_PWM_Fase_Mode_ICR1A,
86  }TMR1_tenuWaveGenerationMode;
87
88  typedef enum{
90      TMRx_u8_COM_Disconnected = (u8)0u,
91      TMRx_u8_COM_Toggle_on_Match,
92      TMRx_u8_COM_Clear_on_Match,
93      TMRx_u8_COM_Set_on_Match,
94
96      TMRx_u8_FastPWM_Clear_on_Match = (u8)2u,     // Non Inverting Mode
97      TMRx_u8_FastPWM_Set_on_Match,                // Inverting Mode
98
100      TMRx_u8_PhasePWM_Clear_on_Match = (u8)2u,    // Low Pulse
101      TMRx_u8_PhasePWM_Set_on_Match,               // High Pulse
102
103  }TMRx_u8_tenuCompareOutputMode;
104
105  typedef enum{
106      TMR2_IO_Clock = (u8)0u,
107      TMR2_TOSC_Clock
108  }TMR2_tenuInputCaptureEdgeSelect;
109
110  typedef enum{
112      TMR1_COM_Disconnected = (u8)0u,
113      TMR1_COM_Toggle_on_Match,
114      TMR1_COM_Clear_on_Match,
115      TMR1_COM_Set_on_Match,
116
118      TMR1_FastPWM_ToggleA_on_Match_Mode15 = (u8)1u,
119      TMR1_FastPWM_Clear_on_Match = (u8)2u,    // Non Inverting Mode
120      TMR1_FastPWM_Set_on_Match,               // Inverting Mode
121
123      TMR1_PhasePWM_ToggleA_on_Match_Mode15 = (u8)1u,
124      TMR1_PhasePWM_Clear_on_Match = (u8)2u,   // Low Pulse
125      TMR1_PhasePWM_Set_on_Match,              // High Pulse
126
127  }TMR1_tenuCompareOutputMode;
128
129  typedef enum{
130      TMR1_Capture_Falling_Edge = (u8)0u,
131      TMR1_Capture_Rising_Edge,
132      TMR1_Capture_Off
133  }TMR1_tenuInputCaptureEdgeSelect;
134
135
/*****************************************************************************************
*******************************/
136  #if defined(TMR0) || defined(PWM0)
137  typedef struct{
138  #if defined(PWM0)
139      u32                             m_TMR_Freq;
140      u16                             m_TMR_Duty;
141  #endif
142      u8                              m_TMR_Reload;
143      u8                              m_TMR_Compare;
144      TMR0_tenuClockSource            m_TMR_Prescalar;
145      TMRx_u8_tenuWaveGenerationMode  m_TMR_Mode;
146      TMRx_u8_tenuCompareOutputMode   m_TMR_OutputMode;
147      LBTY_tenuFlagStatus             m_TMR_FOC;
148      LBTY_tenuFlagStatus             m_TMR_OVIE;
149      LBTY_tenuFlagStatus             m_TMR_OCIE;
150  }TMR0_tstrConfig;
151  #endif
152  #if defined(TMR2) || defined(PWM2)
153  typedef struct{
```

```
#if defined(PWM2)
    u32                                 m_TMR_Freq;
    u16                                 m_TMR_Duty;
#endif
    u8                                  m_TMR_Reload;
    u8                                  m_TMR_Compare;
    TMR2_tenuClockSource                m_TMR_Prescalar;
    TMRx_u8_tenuWaveGenerationMode      m_TMR_Mode;
    TMRx_u8_tenuCompareOutputMode       m_TMR_OutputMode;
    LBTY_tenuFlagStatus                 m_TMR_FOC;
    LBTY_tenuFlagStatus                 m_TMR_OVIE;
    LBTY_tenuFlagStatus                 m_TMR_OCIE;
    TMR2_tenuInputCaptureEdgeSelect     m_TMR_AsyClock;
}TMR2_tstrConfig;
#endif
#if defined(TMR1) || defined(PWM1)
typedef struct{
#if defined(PWM1)
    u32                                 m_TMR_Freq;
    u16                                 m_TMR_Duty_A;
    u16                                 m_TMR_Duty_B;
#endif
    u16                                 m_TMR_Reload;
    u16                                 m_TMR_Input;
    u16                                 m_TMR_CompareA;
    u16                                 m_TMR_CompareB;
    TMR1_tenuClockSource                m_TMR_Prescalar;
    TMR1_tenuWaveGenerationMode         m_TMR_Mode;
    TMR1_tenuCompareOutputMode          m_TMR_OutputModeA;
    TMR1_tenuCompareOutputMode          m_TMR_OutputModeB;
    LBTY_tenuFlagStatus                 m_TMR_FOCA;
    LBTY_tenuFlagStatus                 m_TMR_FOCB;
    LBTY_tenuFlagStatus                 m_TMR_TICIE;
    LBTY_tenuFlagStatus                 m_TMR_OCIEA;
    LBTY_tenuFlagStatus                 m_TMR_OCIEB;
    LBTY_tenuFlagStatus                 m_TMR_TOIE;
    LBTY_tenuFlagStatus                 m_TMR_InputNoise;
    TMR1_tenuInputCaptureEdgeSelect     m_TMR_InputEdge;
}TMR1_tstrConfig;
#endif
/* ********************************************************************************* */
/* ************************* MACRO/DEFINE SECTION ************************** */
/* ********************************************************************************* */

#define TMR_TICK_US         (1.0f/(F_CPU/1000000))

/* ********************************************************************************* */
/* ***************************** CONST SECTION ***************************** */
/* ********************************************************************************* */

/* ********************************************************************************* */
/* ***************************** VARIABLE SECTION ************************** */
/* ********************************************************************************* */

/* ********************************************************************************* */
/* ***************************** FUNCTION SECTION ************************** */
/* ********************************************************************************* */

extern void TMR0_vidSetConfig(TMR0_tstrConfig const* const pstrConfig);
extern void TMR0_vidSRestConfig(TMR0_tstrConfig* const pstrConfig);

extern void TMR0_vidInit(void);

extern void TMR0_vidEnable(void);
extern void TMR0_vidDisable(void);

extern void TMR0_vidSetForceOutputCompare(void);
extern void TMR0_vidResetForceOutputCompare(void);

extern LBTY_tenuErrorStatus TMR2_u8Async(TMR2_tenuInputCaptureEdgeSelect u8Async);
extern LBTY_tenuErrorStatus TMR0_u8SetMode(TMRx_u8_tenuWaveGenerationMode u8Mode);
extern LBTY_tenuErrorStatus TMR0_u8SetOutputMode(TMRx_u8_tenuCompareOutputMode
u8OutMode);

extern LBTY_tenuErrorStatus TMR0_u8SetOutputCompare(u8 u8Reload);
extern LBTY_tenuErrorStatus TMR0_u8SetCounter(u8 u8Reload);

```

```
230 extern LBTY_tenuErrorStatus TMR0_u8GetOutputCompare(u8* pu8Reload);
231 extern LBTY_tenuErrorStatus TMR0_u8GetCounter(u8* pu8Reload);
232
233 #if defined(PWM0)
234 extern LBTY_tenuErrorStatus PWM_u8SetFreq_OC0(u32 u32Freq);
235 extern LBTY_tenuErrorStatus PWM_u8SetDuty_OC0(u16 u16Duty);
236
237 static inline void PWM_vidEnable_OC0(void) {TMR0_vidEnable(); }
238 static inline void PWM_vidDisable_OC0(void){TMR0_vidDisable();}
239 #endif
240
241 extern void TMR0_vidSetCompareNum(u16 u16Num);
242 extern void TMR0_vidGetCompareNum(u16* pu16Num);
243
244 extern void TMR0_vidSetOverflowNum(u16 u16Num);
245 extern void TMR0_vidGetOverflowNum(u16* pu16Num);
246 extern void TMR0_vidGetTicks(u32* pu32Tick);
247
248 extern void TMR0_vidCompareMatch_Enable(void);
249 extern void TMR0_vidCompareMatch_Disable(void);
250
251 extern void TMR0_vidSetCompareMatch_Flag(void);
252 extern void TMR0_vidClrCompareMatch_Flag(void);
253
254 extern void TMR0_vidOverFlow_Enable(void);
255 extern void TMR0_vidOverFlow_Disable(void);
256
257 extern void TMR0_vidSetOverFlow_Flag(void);
258 extern void TMR0_vidClrOverFlow_Flag(void);
259
260 extern void TMR0_vidSetCallBack_CompareMatch(void (*pCallBack)(void));
261 extern void TMR0_vidSetCallBack_OverFlow(void (*pCallBack)(void));
262
263
/*******************************************************************************
*******************************/
264
265 extern void TMR2_vidSetConfig(TMR2_tstrConfig const* const pstrConfig);
266 extern void TMR2_vidSRestConfig(TMR2_tstrConfig* const pstrConfig);
267
268 extern void TMR2_vidInit(void);
269
270 extern void TMR2_vidEnable(void);
271 extern void TMR2_vidDisable(void);
272
273 extern void TMR2_vidSetForceOutputCompare(void);
274 extern void TMR2_vidResetForceOutputCompare(void);
275
276 extern LBTY_tenuErrorStatus TMR2_u8SetMode(TMRx_u8_tenuWaveGenerationMode u8Mode);
277 extern LBTY_tenuErrorStatus TMR2_u8SetOutputMode(TMRx_u8_tenuCompareOutputMode
u8OutMode);
278
279 extern LBTY_tenuErrorStatus TMR2_u8SetOutputCompare(u8 u8Reload);
280 extern LBTY_tenuErrorStatus TMR2_u8SetCounter(u8 u8Reload);
281
282 extern LBTY_tenuErrorStatus TMR2_u8GetOutputCompare(u8* pu8Reload);
283 extern LBTY_tenuErrorStatus TMR2_u8GetCounter(u8* pu8Reload);
284
285 #if defined(PWM2)
286 extern LBTY_tenuErrorStatus PWM_u8SetFreq_OC2(u32 u32Freq);
287 extern LBTY_tenuErrorStatus PWM u8SetDuty OC2(u16 u16Duty);
288
289 static inline void PWM_vidEnable_OC2(void) {TMR2_vidEnable(); }
290 static inline void PWM_vidDisable_OC2(void){TMR2_vidDisable();}
291 #endif
292
293 extern void TMR2_vidSetCompareNum(u16 u16Num);
294 extern void TMR2_vidGetCompareNum(u16* pu16Num);
295
296 extern void TMR2_vidSetOverflowNum(u16 u16Num);
297 extern void TMR2_vidGetOverflowNum(u16* pu16Num);
298 extern void TMR2_vidGetTicks(u32* pu32Tick);
299
300 extern void TMR2_vidCompareMatch_Enable(void);
301 extern void TMR2_vidCompareMatch_Disable(void);
302
303 extern void TMR2_vidSetCompareMatch_Flag(void);
```

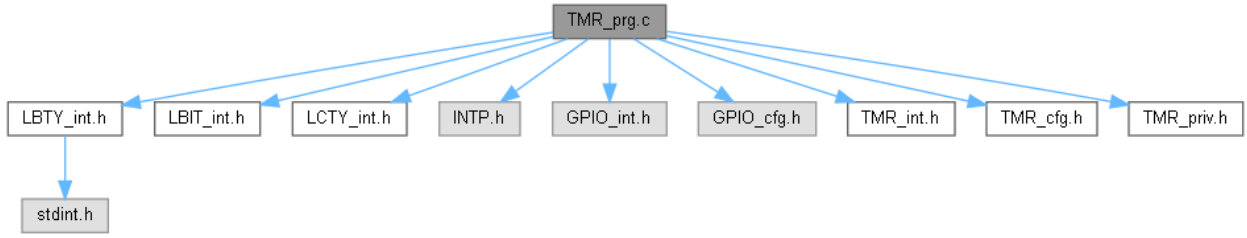
```
304 extern void TMR2_vidClrCompareMatch_Flag(void);
305
306 extern void TMR2_vidOverFlow_Enable(void);
307 extern void TMR2_vidOverFlow_Disable(void);
308
309 extern void TMR2_vidSetOverFlow_Flag(void);
310 extern void TMR2_vidClrOverFlow_Flag(void);
311
312 extern void TMR2_vidSetCallBack_CompareMatch(void (*pCallBack)(void));
313 extern void TMR2_vidSetCallBack_OverFlow(void (*pCallBack)(void));
314
315
/*******************************************************************************
*******************************/
316
317 void TMR1_vidSetConfig(TMR1_tstrConfig const* const pstrConfig);
318 void TMR1_vidSRestConfig(TMR1_tstrConfig* const pstrConfig);
319
320 void TMR1_vidInit(void);
321
322 void TMR1_vidEnable(void);
323 void TMR1_vidDisable(void);
324 void TMR1_vidInitInputCapture(void);
325
326 void TMR1_vidSetForceOutputCompareA(void);
327 void TMR1_vidResetForceOutputCompareA(void);
328 void TMR1_vidSetForceOutputCompareB(void);
329 void TMR1_vidResetForceOutputCompareB(void);
330
331 LBTY_tenuErrorStatus TMR1_u8SetMode(TMR1_tenuWaveGenerationMode u8Mode);
332 LBTY_tenuErrorStatus TMR1_u8SetOutputModeA(TMR1_tenuCompareOutputMode u8OutMode);
333 LBTY_tenuErrorStatus TMR1_u8SetOutputModeB(TMR1_tenuCompareOutputMode u8OutMode);
334
335 LBTY_tenuErrorStatus TMR1_u8SetInputCapture(u16 u16Reload);
336 LBTY_tenuErrorStatus TMR1_u8SetOutputCompare_A(u16 u16Reload);
337 LBTY_tenuErrorStatus TMR1_u8SetOutputCompare_B(u16 u16Reload);
338 LBTY_tenuErrorStatus TMR1_u8SetCounter(u16 u16Reload);
339
340 LBTY_tenuErrorStatus TMR1_u8GetInputCapture(u16* pu16Reload);
341 LBTY_tenuErrorStatus TMR1_u8GetOutputCompare_A(u16* pu16Reload);
342 LBTY_tenuErrorStatus TMR1_u8GetOutputCompare_B(u16* pu16Reload);
343 LBTY_tenuErrorStatus TMR1_u8GetCounter(u16* pu16Reload);
344
345 #if defined(PWM1)
346 LBTY_tenuErrorStatus PWM_u8SetFreq_OC1x(u32 u32Freq);
347 LBTY_tenuErrorStatus PWM_u8SetDuty_OC1A(u16 u16Duty);
348 LBTY_tenuErrorStatus PWM_u8SetDuty_OC1B(u16 u16Duty);
349
350 static inline void PWM_vidEnable_OC1x(void) {TMR1_vidEnable(); }
351 static inline void PWM_vidDisable_OC1x(void){TMR1_vidDisable();}
352 #endif
353
354 void TMR1_vidSetOverflowNum(u16 u16Num);
355 void TMR1_vidGetOverflowNum(u16* pu16Num);
356 void TMR1_vidGetTicks(u32* pu32Tick);
357
358 void TMR1_vidInputCapture_Enable(void);
359 void TMR1_vidInputCapture_Disable(void);
360
361 void TMR1_vidSetInputCapture_Flag(void);
362 void TMR1_vidClrInputCapture_Flag(void);
363
364 void TMR1_vidCompareMatch_A_Enable(void);
365 void TMR1_vidCompareMatch_A_Disable(void);
366
367 void TMR1_vidSetCompareMatch_A_Flag(void);
368 void TMR1_vidClrCompareMatch_A_Flag(void);
369
370 void TMR1_vidCompareMatch_B_Enable(void);
371 void TMR1_vidCompareMatch_B_Disable(void);
372
373 void TMR1_vidSetCompareMatch_B_Flag(void);
374 void TMR1_vidClrCompareMatch_B_Flag(void);
375
376 void TMR1_vidOverFlow_Enable(void);
377 void TMR1_vidOverFlow_Disable(void);
378
```

```
379 void TMR1_vidSetOverFlow_Flag(void);
380 void TMR1_vidClrOverFlow_Flag(void);
381
382 void TMR1_vidSetCallBack_CaptureEvent(void (*pCallBack)(void));
383 void TMR1_vidSetCallBack_CompareMatch_A(void (*pCallBack)(void));
384 void TMR1_vidSetCallBack_CompareMatch_B(void (*pCallBack)(void));
385 void TMR1_vidSetCallBack_OverFlow(void (*pCallBack)(void));
386
387 #endif /* TMR_INT_H_ */
388 /************************** E N D (TMR_int.h) ****************************/
```

TMR_prg.c File Reference

```
#include "LBTY_int.h"
#include "LBIT_int.h"
#include "LCTY_int.h"
#include "INTP.h"
#include "GPIO_int.h"
#include "GPIO_cfg.h"
#include "TMR_int.h"
#include "TMR_cfg.h"
#include "TMR_priv.h"
```

Include dependency graph for TMR_prg.c:



Functions

- void TMR0_vidSetConfig (TMR0_tstrConfig const *const pstrConfig)
- void TMR0_vidSRestConfig (TMR0_tstrConfig *const pstrConfig)
- void TMR0_vidInit (void)
- void TMR0_vidEnable (void)
- void TMR0_vidDisable (void)
- void TMR0_vidSetForceOutputCompare (void)
- void TMR0_vidResetForceOutputCompare (void)
- LBTY_tenuErrorStatus TMR0_u8SetMode (TMRx_u8_tenuWaveGenerationMode u8Mode)
- LBTY_tenuErrorStatus TMR0_u8SetOutputMode (TMRx_u8_tenuCompareOutputMode u8OutMode)
- LBTY_tenuErrorStatus TMR0_u8SetOutputCompare (u8 u8Reload)
- LBTY_tenuErrorStatus TMR0_u8SetCounter (u8 u8Reload)
- LBTY_tenuErrorStatus TMR0_u8GetOutputCompare (u8 *pu8Reload)
- LBTY_tenuErrorStatus TMR0_u8GetCounter (u8 *pu8Reload)
- void TMR0_vidSetCompareNum (u16 u16Num)
- void TMR0_vidGetCompareNum (u16 *pu16Num)
- void TMR0_vidSetOverflowNum (u16 u16Num)
- void TMR0_vidGetOverflowNum (u16 *pu16Num)
- void TMR0_vidGetTicks (u32 *pu32Tick)
- void TMR0_vidCompareMatch_Enable (void)
- void TMR0_vidCompareMatch_Disable (void)
- void TMR0_vidSetCompareMatch_Flag (void)
- void TMR0_vidClrCompareMatch_Flag (void)
- void TMR0_vidOverFlow_Enable (void)
- void TMR0_vidOverFlow_Disable (void)
- void TMR0_vidSetOverFlow_Flag (void)
- void TMR0_vidClrOverFlow_Flag (void)
- void TMR0_vidSetCallBack_CompareMatch (void(*pCallBack)(void))
- void TMR0_vidSetCallBack_OverFlow (void(*pCallBack)(void))
- LCTY_INLINE void TMR2_vidControlUpdateBusy (void)
- LCTY_INLINE void TMR2_vidCompareUpdateBusy (void)
- LCTY_INLINE void TMR2_vidTimerUpdateBusy (void)
- void TMR2_vidSetConfig (TMR2_tstrConfig const *const pstrConfig)

- void TMR2_vidSRestConfig (TMR2_tstrConfig *const pstrConfig)
- void TMR2_vidInit (void)
- void TMR2_vidEnable (void)
- void TMR2_vidDisable (void)
- void TMR2_vidSetForceOutputCompare (void)
- void TMR2_vidResetForceOutputCompare (void)
- LBTY_tenuErrorStatus TMR2_u8Async (TMR2_tenuInputCaptureEdgeSelect u8Async)
- LBTY_tenuErrorStatus TMR2_u8SetMode (TMRx_u8_tenuWaveGenerationMode u8Mode)
- LBTY_tenuErrorStatus TMR2_u8SetOutputMode (TMRx_u8_tenuCompareOutputMode u8OutMode)
- LBTY_tenuErrorStatus TMR2_u8SetOutputCompare (u8 u8Reload)
- LBTY_tenuErrorStatus TMR2_u8SetCounter (u8 u8Reload)
- LBTY_tenuErrorStatus TMR2_u8GetOutputCompare (u8 *pu8Reload)
- LBTY_tenuErrorStatus TMR2_u8GetCounter (u8 *pu8Reload)
- void TMR2_vidSetCompareNum (u16 u16Num)
- void TMR2_vidGetCompareNum (u16 *pu16Num)
- void TMR2_vidSetOverflowNum (u16 u16Num)
- void TMR2_vidGetOverflowNum (u16 *pu16Num)
- void TMR2_vidGetTicks (u32 *pu32Tick)
- void TMR2_vidCompareMatch_Enable (void)
- void TMR2_vidCompareMatch_Disable (void)
- void TMR2_vidSetCompareMatch_Flag (void)
- void TMR2_vidClrCompareMatch_Flag (void)
- void TMR2_vidOverFlow_Enable (void)
- void TMR2_vidOverFlow_Disable (void)
- void TMR2_vidSetOverFlow_Flag (void)
- void TMR2_vidClrOverFlow_Flag (void)
- void TMR2_vidSetCallBack_CompareMatch (void(*pCallBack)(void))
- void TMR2_vidSetCallBack_OverFlow (void(*pCallBack)(void))
- void TMR1_vidSetConfig (TMR1_tstrConfig const *const pstrConfig)
- void TMR1_vidSRestConfig (TMR1_tstrConfig *const pstrConfig)
- void TMR1_vidInit (void)
- void TMR1_vidEnable (void)
- void TMR1_vidDisable (void)
- void TMR1_vidInitInputCapture (void)
- void TMR1_vidSetForceOutputCompareA (void)
- void TMR1_vidResetForceOutputCompareA (void)
- void TMR1_vidSetForceOutputCompareB (void)
- void TMR1_vidResetForceOutputCompareB (void)
- LCTY_INLINE void TMR1_vidSetWaveGenerationMode (TMR1_tenuWaveGenerationMode u8Mode)
- LBTY_tenuErrorStatus TMR1_u8SetMode (TMR1_tenuWaveGenerationMode u8Mode)
- LBTY_tenuErrorStatus TMR1_u8SetOutputModeA (TMR1_tenuCompareOutputMode u8OutMode)
- LBTY_tenuErrorStatus TMR1_u8SetOutputModeB (TMR1_tenuCompareOutputMode u8OutMode)
- LBTY_tenuErrorStatus TMR1_u8SetInputCapture (u16 u16Reload)
- LBTY_tenuErrorStatus TMR1_u8SetOutputCompare_A (u16 u16Reload)
- LBTY_tenuErrorStatus TMR1_u8SetOutputCompare_B (u16 u16Reload)
- LBTY_tenuErrorStatus TMR1_u8SetCounter (u16 u16Reload)
- LBTY_tenuErrorStatus TMR1_u8GetInputCapture (u16 *pu16Reload)
- LBTY_tenuErrorStatus TMR1_u8GetOutputCompare_A (u16 *pu16Reload)
- LBTY_tenuErrorStatus TMR1_u8GetOutputCompare_B (u16 *pu16Reload)
- LBTY_tenuErrorStatus TMR1_u8GetCounter (u16 *pu16Reload)
- void TMR1_vidSetOverflowNum (u16 u16Num)
- void TMR1_vidGetOverflowNum (u16 *pu16Num)
- void TMR1_vidGetTicks (u32 *pu32Tick)

- void TMR1_vidInputCapture_Enable (void)
- void TMR1_vidInputCapture_Disable (void)
- void TMR1_vidSetInputCapture_Flag (void)
- void TMR1_vidClrInputCapture_Flag (void)
- void TMR1_vidCompareMatch_A_Enable (void)
- void TMR1_vidCompareMatch_A_Disable (void)
- void TMR1_vidSetCompareMatch_A_Flag (void)
- void TMR1_vidClrCompareMatch_A_Flag (void)
- void TMR1_vidCompareMatch_B_Enable (void)
- void TMR1_vidCompareMatch_B_Disable (void)
- void TMR1_vidSetCompareMatch_B_Flag (void)
- void TMR1_vidClrCompareMatch_B_Flag (void)
- void TMR1_vidOverFlow_Enable (void)
- void TMR1_vidOverFlow_Disable (void)
- void TMR1_vidSetOverFlow_Flag (void)
- void TMR1_vidClrOverFlow_Flag (void)
- void TMR1_vidSetCallBack_CaptureEvent (void(*pCallBack)(void))
- void TMR1_vidSetCallBack_CompareMatch_A (void(*pCallBack)(void))
- void TMR1_vidSetCallBack_CompareMatch_B (void(*pCallBack)(void))
- void TMR1_vidSetCallBack_OverFlow (void(*pCallBack)(void))

Variables

- const u8 TMRx_RELOAD_DELAY [] = {0, 47, 6, 1, 1, 1, 0, 0}
- static void(* pFuncCallBack_TMR0_CompareMatch)(void) = INTP_vidCallBack
- static void(* pFuncCallBack_TMR0_OverFlow)(void) = INTP_vidCallBack
- static void(* pFuncCallBack_TMR2_CompareMatch)(void) = INTP_vidCallBack
- static void(* pFuncCallBack_TMR2_OverFlow)(void) = INTP_vidCallBack
- static void(* pFuncCallBack_TMR1_CaptureEven)(void) = INTP_vidCallBack
- static void(* pFuncCallBack_TMR1_CompareMatch_A)(void) = INTP_vidCallBack
- static void(* pFuncCallBack_TMR1_CompareMatch_B)(void) = INTP_vidCallBack
- static void(* pFuncCallBack_TMR1_OverFlow)(void) = INTP_vidCallBack
- static volatile TMR0_tstrConfig strTMR0_Config_GLB
- static volatile TMR2_tstrConfig strTMR2_Config_GLB
- static volatile TMR1_tstrConfig strTMR1_Config_GLB
- static volatile u16 TMR0_u8CompareNum_GLB = LBTY_u8ZERO
- static volatile u16 TMR2_u8CompareNum_GLB = LBTY_u8ZERO
- static volatile u16 TMR0_u8OverflewNum_GLB = LBTY_u8ZERO
- static volatile u16 TMR2_u8OverflewNum_GLB = LBTY_u8ZERO
- static volatile u16 TMR1_u8OverflewNum_GLB = LBTY_u8ZERO

Function Documentation

LBTY_tenuErrorStatus TMR0_u8GetCounter (u8 * *pu8Reload*)

```
319                                                     {
320     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
321     if(pu8Reload != LBTY_NULL){
322         *pu8Reload = S_TMR0->m_TCNT0;
323     }else{
324         u8RetErrorState = LBTY_NULL_POINTER;
325     }
326     return u8RetErrorState;
327 }
```

LBTY_tenuErrorStatus TMR0_u8GetOutputCompare (u8 * *pu8Reload*)

```
309                                                     {
310     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
311     if(pu8Reload != LBTY_NULL){
312         *pu8Reload = S_TMR0->m_OCR0;
```

```
313     }else{
314         u8RetErrorState = LBTY_NULL_POINTER;
315     }
316     return u8RetErrorState;
317 }
```

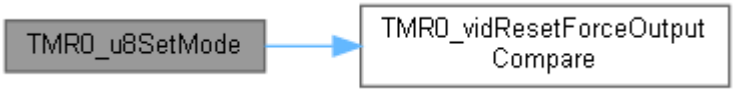
LBTY_tenuErrorStatus TMR0_u8SetCounter (u8 *u8Reload*)

```
299                                                 {
300     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
301     if(u8Reload <= LBTY_u8MAX){
302         S_TMR0->m_TCNT0 = strTMR0_Config_GLB.m_TMR_Reload = u8Reload;
303     }else{
304         u8RetErrorState = LBTY_WRITE_ERROR;
305     }
306     return u8RetErrorState;
307 }
```

LBTY_tenuErrorStatus TMR0_u8SetMode (TMRx_u8_tenuWaveGenerationMode *u8Mode*)

```
217                                                                                 {
218     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
219     switch(u8Mode){
220         case TMRx_u8_Normal_Mode:
221             S_TMR0->m_TCCR0.sBits.m_WGMx0 = GET_BIT(TMRx_u8_Normal_Mode,
TMRx_WGMx0_MASK);
222             S_TMR0->m_TCCR0.sBits.m_WGMx1 = GET_BIT(TMRx_u8_Normal_Mode,
TMRx_WGMx1_MASK);
223             break;
224         case TMRx_u8_PWM_PhaseCorrect_Mode:
225             TMR0_vidResetForceOutputCompare();
226             S_TMR0->m_TCCR0.sBits.m_WGMx0 =
GET_BIT(TMRx_u8_PWM_PhaseCorrect_Mode, TMRx_WGMx0_MASK);
227             S_TMR0->m_TCCR0.sBits.m_WGMx1 =
GET_BIT(TMRx_u8_PWM_PhaseCorrect_Mode, TMRx_WGMx1_MASK);
228             break;
229         case TMRx_u8_CTC_Mode_Mode:
230             S_TMR0->m_TCCR0.sBits.m_WGMx0 = GET_BIT(TMRx_u8_CTC_Mode_Mode,
TMRx_WGMx0_MASK);
231             S_TMR0->m_TCCR0.sBits.m_WGMx1 = GET_BIT(TMRx_u8_CTC_Mode_Mode,
TMRx_WGMx1_MASK);
232             break;
233         case TMRx_u8_PWM_Fase_Mode:
234             TMR0_vidResetForceOutputCompare();
235             S_TMR0->m_TCCR0.sBits.m_WGMx0 = GET_BIT(TMRx_u8_PWM_Fase_Mode,
TMRx_WGMx0_MASK);
236             S_TMR0->m_TCCR0.sBits.m_WGMx1 = GET_BIT(TMRx_u8_PWM_Fase_Mode,
TMRx_WGMx1_MASK);
237             break;
238         default:
239             u8RetErrorState = LBTY_WRITE_ERROR;
240             break;
241     }
242     if(u8RetErrorState == LBTY_OK){
243         strTMR0_Config_GLB.m_TMR_Mode = u8Mode;
244     }
245     return u8RetErrorState;
246 }
```

Here is the call graph for this function:



LBTY_tenuErrorStatus TMR0_u8SetOutputCompare (u8 *u8Reload*)

```
289                                                 {
290     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
291     if(u8Reload <= LBTY_u8MAX){
292         S_TMR0->m_OCR0 = strTMR0_Config_GLB.m_TMR_Compare = u8Reload;
293     }else{
294         u8RetErrorState = LBTY_WRITE_ERROR;
295     }
296     return u8RetErrorState;
297 }
```

LBTY_tenuErrorStatus TMR0_u8SetOutputMode (TMRx_u8_tenuCompareOutputMode *u8OutMode*)

```
248
{
249     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
250     TMRx_u8_tenuWaveGenerationMode u8Mode =
251             (S_TMR0->m_TCCR0.sBits.m_WGMx0<<TMRx_WGMx0_MASK) |
(S_TMR0->m_TCCR0.sBits.m_WGMx1<<TMRx_WGMx1_MASK);
252     switch(u8Mode){
253         case TMRx_u8_Normal_Mode:
254         case TMRx_u8_CTC_Mode_Mode:
255             switch(u8OutMode){
256                 case TMRx_u8_COM_Disconnected:
S_TMR0->m_TCCR0.sBits.m_COMx = TMRx_u8_COM_Disconnected;            break;
257                 case TMRx_u8_COM_Toggle_on_Match:
S_TMR0->m_TCCR0.sBits.m_COMx = TMRx_u8_COM_Toggle_on_Match;         break;
258                 case TMRx_u8_COM_Clear_on_Match:
S_TMR0->m_TCCR0.sBits.m_COMx = TMRx_u8_COM_Clear_on_Match;          break;
259                 case TMRx_u8_COM_Set_on_Match:
S_TMR0->m_TCCR0.sBits.m_COMx = TMRx_u8_COM_Set_on_Match;            break;
260                 default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
261             }
262             break;
263         case TMRx_u8_PWM_PhaseCorrect_Mode:
264             switch(u8OutMode){
265                 case
TMRx_u8_PhasePWM_Clear_on_Match:S_TMR0->m_TCCR0.sBits.m_COMx =
TMRx_u8_PhasePWM_Clear_on_Match;    break;
266                 case TMRx_u8_PhasePWM_Set_on_Match:
S_TMR0->m_TCCR0.sBits.m_COMx = TMRx_u8_PhasePWM_Set_on_Match;      break;
267                 default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
268             }
269             break;
270         case TMRx_u8_PWM_Fase_Mode:
271             switch(u8OutMode){
272                 case
TMRx_u8_FastPWM_Clear_on_Match:S_TMR0->m_TCCR0.sBits.m_COMx =
TMRx_u8_FastPWM_Clear_on_Match;     break;
273                 case TMRx_u8_FastPWM_Set_on_Match:
S_TMR0->m_TCCR0.sBits.m_COMx = TMRx_u8_FastPWM_Set_on_Match;        break;
274                 default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
275             }
276             break;
277         default:
278             u8RetErrorState = LBTY_WRITE_ERROR;
279             break;
280     }
281     if(u8RetErrorState == LBTY_OK){
282         if(u8OutMode != TMRx_u8_COM_Disconnected)
283             GPIO_u8SetPinDirection(TMR_OC0_PORT, TMR_OC0_PIN, PIN_OUTPUT);
284         strTMR0_Config_GLB.m_TMR_OutputMode = u8OutMode;
285     }
286     return u8RetErrorState;
287 }
```

void TMR0_vidClrCompareMatch_Flag (void)

```
406 {S_TIFR->sBits.m_OCF0   = LBTY_RESET;}
```

void TMR0_vidClrOverFlow_Flag (void)

```
412 {S_TIFR->sBits.m_TOV0   = LBTY_RESET;}
```

void TMR0_vidCompareMatch_Disable (void)

```
403 {S_TIMSK->sBits.m_OCIE0 = LBTY_RESET;}
```

void TMR0_vidCompareMatch_Enable (void)

```
402 {S_TIMSK->sBits.m_OCIE0  = LBTY_SET;}
```

void TMR0_vidDisable (void)

```
205                         {
206     S_TMR0->m_TCCR0.sBits.m_CSx = TMR0_NoClockSource_Disable;
207 }
```

void TMR0_vidEnable (void)

```
198                             {
199     S_TMR0->m_TCCR0.sBits.m_CSx = strTMR0_Config_GLB.m_TMR_Prescalar;
200     if(strTMR0_Config_GLB.m_TMR_Prescalar == TMR0_ExternalClock_FallingEdge ||
201        strTMR0_Config_GLB.m_TMR_Prescalar == TMR0_ExternalClock_RisinfEdge){
202         GPIO_u8SetPinDirection(TMR_EXT0_PORT, TMR_EXT0_PIN, PIN_INPUT);
203     }
204 }
```

void TMR0_vidGetCompareNum (u16 * *pu16Num*)

```
385                                             {
386     *pu16Num = TMR0_u8CompareNum_GLB;
387 }
```

void TMR0_vidGetOverflowNum (u16 * *pu16Num*)

```
392                                               {
393     *pu16Num = TMR0_u8OverflewNum_GLB;
394 }
```

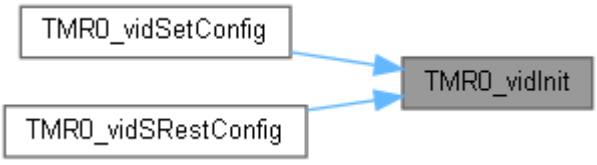
void TMR0_vidGetTicks (u32 * *pu32Tick*)

```
396                                           {
397     *pu32Tick = (u32)TMR_u8MAX * TMR0_u8OverflewNum_GLB + S_TMR0->m_TCNT0;
398 }
```

void TMR0_vidInit (void)

```
160                         {
161     //S_SFIOR->sBits.m_PSR10 = LBTY_SET;
162 
163     //TMR0_vidEnable();
164     S_TMR0->m_TCCR0.sBits.m_CSx   = strTMR0_Config_GLB.m_TMR_Prescalar;
165     if(strTMR0_Config_GLB.m_TMR_Prescalar == TMR0_ExternalClock_FallingEdge ||
166        strTMR0_Config_GLB.m_TMR_Prescalar == TMR0_ExternalClock_RisinfEdge){
167         GPIO_u8SetPinDirection(TMR_EXT0_PORT, TMR_EXT0_PIN, PIN_INPUT);
168     }
169     //TMR0_u8SetMode(TMR0_MODE_INIT);
170     S_TMR0->m_TCCR0.sBits.m_WGMx0 = GET_BIT(strTMR0_Config_GLB.m_TMR_Mode,
TMRx_WGMx0_MASK);
171     S_TMR0->m_TCCR0.sBits.m_WGMx1 = GET_BIT(strTMR0_Config_GLB.m_TMR_Mode,
TMRx_WGMx1_MASK);
172     //TMR0_u8SetOutputMode(TMR0_COMPARE_OUTPUT_MODE);
173     S_TMR0->m_TCCR0.sBits.m_COMx  = strTMR0_Config_GLB.m_TMR_OutputMode;
174     if(S_TMR0->m_TCCR0.sBits.m_COMx != TMRx_u8_COM_Disconnected)
175         GPIO_u8SetPinDirection(TMR_OC0_PORT, TMR_OC0_PIN, PIN_OUTPUT);
176     //TMR0_vidResetForceOutputCompare();
177     S_TMR0->m_TCCR0.sBits.m_FOCx  = strTMR0_Config_GLB.m_TMR_FOC;
178 
179 #if defined(TMR0)
180     //TMR0_u8SetOutputCompare(TMR0_OUTPUT_COMPARE_INIT);
181     S_TMR0->m_OCR0  = strTMR0_Config_GLB.m_TMR_Compare;
182     //TMR0_u8SetCounter(TMR0_COUNTER_INIT);
183     S_TMR0->m_TCNT0 = strTMR0_Config_GLB.m_TMR_Reload;
184 #elif defined(PWM0)
185     PWM_vidDisable_OC0();
186     PWM_u8SetFreq_OC0(strTMR0_Config_GLB.m_TMR_Freq);
187     PWM_u8SetDuty_OC0(strTMR0_Config_GLB.m_TMR_Duty);
188     TMR0_Reload_Delay = TMRx_RELOAD_DELAY[strTMR0_Config_GLB.m_TMR_Prescalar];
189 #endif
190 
191     S_TIMSK->sBits.m_OCIE0 = strTMR0_Config_GLB.m_TMR_OCIE;
192     S_TIMSK->sBits.m_TOIE0 = strTMR0_Config_GLB.m_TMR_OVIE;
193 
194     S_TIFR->sBits.m_OCF0   = LBTY_RESET;
195     S_TIFR->sBits.m_TOV0   = LBTY_RESET;
196 }
```

Here is the caller graph for this function:



void TMR0_vidOverFlow_Disable (void)

```
409 {S_TIMSK->sBits.m_TOIE0 = LBTY_RESET;}
```

void TMR0_vidOverFlow_Enable (void)

```
408 {S_TIMSK->sBits.m_TOIE0 = LBTY_SET;}
```

void TMR0_vidResetForceOutputCompare (void)

```
213                                                 {
214     S_TMR0->m_TCCR0.sBits.m_FOCx  = strTMR0_Config_GLB.m_TMR_FOC = LBTY_RESET;
215 }
```

Here is the caller graph for this function:

void TMR0_vidSetCallBack_CompareMatch (void(*)(void) *pCallBack*)

```
414                                                                         {
415     if(*pCallBack == LBTY_NULL)     return;
416     pFuncCallBack_TMR0_CompareMatch = pCallBack;
417 }
```

void TMR0_vidSetCallBack_OverFlow (void(*)(void) *pCallBack*)

```
418                                                                     {
419     if(*pCallBack == LBTY_NULL)     return;
420     pFuncCallBack_TMR0_OverFlow = pCallBack;
421 }
```

void TMR0_vidSetCompareMatch_Flag (void)

```
405 {S_TIFR->sBits.m_OCF0   = LBTY_SET;}
```

void TMR0_vidSetCompareNum (u16 *u16Num*)

```
382                                         {
383     TMR0_u8CompareNum_GLB = u16Num;
384 }
```

void TMR0_vidSetConfig (TMR0_tstrConfig const *const *pstrConfig*)

```
133                                                                         {
134     if(pstrConfig != LBTY_NULL){
135         strTMR0_Config_GLB = *pstrConfig;
136     }
137     TMR0_vidInit();
138 }
```

Here is the call graph for this function:

void TMR0_vidSetForceOutputCompare (void)

```
209                                             {
210     S_TMR0->m_TCCR0.sBits.m_FOCx  = strTMR0_Config_GLB.m_TMR_FOC = LBTY_SET;
211 }
```

void TMR0_vidSetOverFlow_Flag (void)

```
411 {S_TIFR->sBits.m_TOV0   = LBTY_SET;}
```

void TMR0_vidSetOverflowNum (u16 *u16Num*)

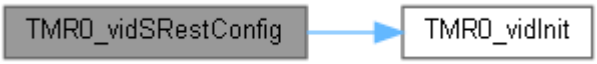
```
389                                         {
390     TMR0_u8OverflewNum_GLB = u16Num;
391 }
```

void TMR0_vidSRestConfig (TMR0_tstrConfig *const *pstrConfig*)

```
140                                                                     {
141 #if defined(PWM0)
142     strTMR0_Config_GLB.m_TMR_Freq       = PWM0_FREQ_INIT;
```

```
143     strTMR0_Config_GLB.m_TMR_Duty       = PWM0_DUTY_INIT;
144 #endif
145     strTMR0_Config_GLB.m_TMR_Reload     = TMR0_COUNTER_INIT;
146     strTMR0_Config_GLB.m_TMR_Compare    = TMR0_OUTPUT_COMPARE_INIT;
147     strTMR0_Config_GLB.m_TMR_Prescalar  = TMR0_CLOCK_SOURCE;
148     strTMR0_Config_GLB.m_TMR_Mode       = TMR0_MODE_INIT;
149     strTMR0_Config_GLB.m_TMR_OutputMode = TMR0_COMPARE_OUTPUT_MODE;
150     strTMR0_Config_GLB.m_TMR_FOC        = LBTY_RESET;
151     strTMR0_Config_GLB.m_TMR_OVIE       = TMR0_OVERFLOW_INTERRUPT_INIT_STATE;
152     strTMR0_Config_GLB.m_TMR_OCIE       =
TMR0_COMPARE_MATCH_INTERRUPT_INIT_STATE;
153
154     if(pstrConfig != LBTY_NULL){
155         *pstrConfig = strTMR0_Config_GLB;
156     }
157     TMR0_vidInit();
158 }
```

Here is the call graph for this function:



LBTY_tenuErrorStatus TMR1_u8GetCounter (u16 * *pu16Reload*)

```
1170                                                                {
1171     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
1172     if(pu16Reload != LBTY_NULL){
1173         *pu16Reload = S_TMR1->m_TCNT1.u16Reg;
1174     }else{
1175         u8RetErrorState = LBTY_NULL_POINTER;
1176     }
1177     return u8RetErrorState;
1178 }
```

LBTY_tenuErrorStatus TMR1_u8GetInputCapture (u16 * *pu16Reload*)

```
1140                                                                    {
1141     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
1142     if(pu16Reload != LBTY_NULL){
1143         *pu16Reload = S_TMR1->m_ICR1.u16Reg;
1144     }else{
1145         u8RetErrorState = LBTY_NULL_POINTER;
1146     }
1147     return u8RetErrorState;
1148 }
```

LBTY_tenuErrorStatus TMR1_u8GetOutputCompare_A (u16 * *pu16Reload*)

```
1150                                                                        {
1151     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
1152     if(pu16Reload != LBTY_NULL){
1153         *pu16Reload = S_TMR1->m_OCR1A.u16Reg;
1154     }else{
1155         u8RetErrorState = LBTY_NULL_POINTER;
1156     }
1157     return u8RetErrorState;
1158 }
```

LBTY_tenuErrorStatus TMR1_u8GetOutputCompare_B (u16 * *pu16Reload*)

```
1160                                                                        {
1161     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
1162     if(pu16Reload != LBTY_NULL){
1163         *pu16Reload = S_TMR1->m_OCR1B.u16Reg;
1164     }else{
1165         u8RetErrorState = LBTY_NULL_POINTER;
1166     }
1167     return u8RetErrorState;
1168 }
```

LBTY_tenuErrorStatus TMR1_u8SetCounter (u16 *u16Reload*)

```
1130                                                                {
1131     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
1132     if(u16Reload <= LBTY_u16MAX){
1133         S_TMR1->m_TCNT1.u16Reg = strTMR1_Config_GLB.m_TMR_Reload = u16Reload;
1134     }else{
```

```
1135         u8RetErrorState = LBTY_WRITE_ERROR;
1136     }
1137     return u8RetErrorState;
1138 }
```

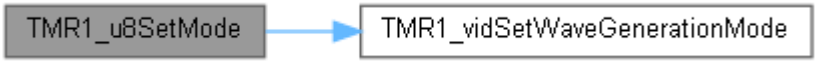
LBTY_tenuErrorStatus TMR1_u8SetInputCapture (u16 *u16Reload*)

```
1100                                                                    {
1101     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
1102     if(u16Reload <= LBTY_u16MAX){
1103         S_TMR1->m_ICR1.u16Reg = strTMR1_Config_GLB.m_TMR_Input = u16Reload;
1104     }else{
1105         u8RetErrorState = LBTY_WRITE_ERROR;
1106     }
1107     return u8RetErrorState;
1108 }
```

LBTY_tenuErrorStatus TMR1_u8SetMode (TMR1_tenuWaveGenerationMode *u8Mode*)

```
961                                                                                 {
962     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
963     switch(u8Mode){
964         case TMRx_u8_Normal_Mode:
965         case TMR1_PWM_PhaseCorrect_Mode_8bit:
966         case TMR1_PWM_PhaseCorrect_Mode_9bit:
967         case TMR1_PWM_PhaseCorrect_Mode_10bit:
968         case TMR1_CTC_Mode_Mode_ICR1:
969         case TMR1_PWM_Fase_Mode_8bit:
970         case TMR1_PWM_Fase_Mode_9bit:
971         case TMR1_PWM_Fase_Mode_10bit:
972         case TMR1_PWM_Phase_Freq_Correct_Mode_ICR1:
973         case TMR1_PWM_Phase_Freq_Correct_Mode_ICR1A:
974         case TMR1_PWM_Phase_Correct_Mode_ICR1:
975         case TMR1_PWM_Phase_Correct_Mode_ICR1A:
976         case TMR1_CTC_Mode_Mode_ICR1A:
977         case TMR1_PWM_Fase_Mode_ICR1:
978         case TMR1_PWM_Fase_Mode_ICR1A:
979             TMR1_vidSetWaveGenerationMode(u8Mode);
980             break;
981         default:
982             u8RetErrorState = LBTY_WRITE_ERROR;
983             break;
984     }
985     return u8RetErrorState;
986 }
```

Here is the call graph for this function:



LBTY_tenuErrorStatus TMR1_u8SetOutputCompare_A (u16 *u16Reload*)

```
1110                                                                    {
1111     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
1112     if(u16Reload <= LBTY_u16MAX){
1113         S_TMR1->m_OCR1A.u16Reg = strTMR1_Config_GLB.m_TMR_CompareA = u16Reload;
1114     }else{
1115         u8RetErrorState = LBTY_WRITE_ERROR;
1116     }
1117     return u8RetErrorState;
1118 }
```

LBTY_tenuErrorStatus TMR1_u8SetOutputCompare_B (u16 *u16Reload*)

```
1120                                                                    {
1121     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
1122     if(u16Reload <= LBTY_u16MAX){
1123         S_TMR1->m_OCR1B.u16Reg = strTMR1_Config_GLB.m_TMR_CompareB = u16Reload;
1124     }else{
1125         u8RetErrorState = LBTY_WRITE_ERROR;
1126     }
1127     return u8RetErrorState;
1128 }
```

LBTY_tenuErrorStatus TMR1_u8SetOutputModeA (TMR1_tenuCompareOutputMode *u8OutMode*)

```
988
{
989     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
990     TMR1_tenuWaveGenerationMode u8Mode =
991             (S_TMR1->m_TCCR1A.sBits.m_WGM10<<TMRx_WGMx0_MASK) |
(S_TMR1->m_TCCR1A.sBits.m_WGM11<<TMRx_WGMx1_MASK) |
992             (S_TMR1->m_TCCR1B.sBits.m_WGM12<<TMRx_WGMx2_MASK) |
(S_TMR1->m_TCCR1B.sBits.m_WGM13<<TMRx_WGMx3_MASK);
993
994     switch(u8Mode){
995         case TMRx_u8_Normal_Mode:
996         case TMR1_CTC_Mode_Mode_ICR1:
997         case TMR1_CTC_Mode_Mode_ICR1A:
998             switch(u8OutMode){
999                 case TMR1_COM_Disconnected:     S_TMR1->m_TCCR1A.sBits.m_COM1A
= TMR1_COM_Disconnected;         break;
1000                case TMR1_COM_Toggle_on_Match:  S_TMR1->m_TCCR1A.sBits.m_COM1A
= TMR1_COM_Toggle_on_Match;      break;
1001                case TMR1_COM_Clear_on_Match:   S_TMR1->m TCCR1A.sBits.m COM1A
= TMR1_COM_Clear_on_Match;       break;
1002                case TMR1_COM_Set_on_Match:     S_TMR1->m_TCCR1A.sBits.m_COM1A
= TMR1_COM_Set_on_Match;         break;
1003                default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
1004            }
1005            break;
1006        case TMR1_PWM_PhaseCorrect_Mode_8bit:
1007        case TMR1_PWM_PhaseCorrect_Mode_9bit:
1008        case TMR1_PWM_PhaseCorrect_Mode_10bit:
1009        case TMR1_PWM_Phase_Freq_Correct_Mode_ICR1:
1010        case TMR1_PWM_Phase_Freq_Correct_Mode_ICR1A:
1011        case TMR1_PWM_Phase_Correct_Mode_ICR1:
1012        case TMR1_PWM_Phase_Correct_Mode_ICR1A:
1013            switch(u8OutMode){
1014                case
TMR1_PhasePWM_ToggleA_on_Match_Mode15:S_TMR1->m_TCCR1A.sBits.m_COM1A =
TMR1_PhasePWM_ToggleA_on_Match_Mode15;  break;
1015                case TMR1_PhasePWM_Clear_on_Match:
S_TMR1->m_TCCR1A.sBits.m_COM1A = TMR1_PhasePWM_Clear_on_Match;          break;
1016                case TMR1_PhasePWM_Set_on_Match:
S_TMR1->m_TCCR1A.sBits.m_COM1A = TMR1_PhasePWM_Set_on_Match;            break;
1017                default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
1018            }
1019            break;
1020        case TMR1_PWM_Fase_Mode_8bit:
1021        case TMR1_PWM_Fase_Mode_9bit:
1022        case TMR1_PWM_Fase_Mode_10bit:
1023        case TMR1_PWM_Fase_Mode_ICR1:
1024        case TMR1_PWM_Fase_Mode_ICR1A:
1025            switch(u8OutMode){
1026                case
TMR1_FastPWM_ToggleA_on_Match_Mode15:S_TMR1->m_TCCR1A.sBits.m_COM1A =
TMR1_FastPWM_ToggleA_on_Match_Mode15;    break;
1027                case TMR1_FastPWM_Clear_on_Match:
S_TMR1->m_TCCR1A.sBits.m_COM1A = TMR1_FastPWM_Clear_on_Match;            break;
1028                case TMR1_FastPWM_Set_on_Match:
S_TMR1->m_TCCR1A.sBits.m_COM1A = TMR1_FastPWM_Set_on_Match;              break;
1029                default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
1030            }
1031            break;
1032        default:
1033            u8RetErrorState = LBTY_WRITE_ERROR;
1034            break;
1035    }
1036    if(u8RetErrorState == LBTY_OK){
1037        if(u8OutMode != TMR1_COM_Disconnected)
1038            GPIO_u8SetPinDirection(TMR_OC1A_PORT, TMR_OC1A_PIN, PIN_OUTPUT);
1039        strTMR1_Config_GLB.m_TMR_OutputModeA = u8OutMode;
1040    }
1041    return u8RetErrorState;
1042 }
```

LBTY_tenuErrorStatus TMR1_u8SetOutputModeB (TMR1_tenuCompareOutputMode *u8OutMode*)

```
1044
{
1045     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
1046     TMR1_tenuWaveGenerationMode u8Mode =
1047              (S_TMR1->m_TCCR1A.sBits.m_WGM10<<TMRx_WGMx0_MASK) |
(S_TMR1->m_TCCR1A.sBits.m_WGM11<<TMRx_WGMx1_MASK) |
1048              (S_TMR1->m_TCCR1B.sBits.m_WGM12<<TMRx_WGMx2_MASK) |
(S_TMR1->m_TCCR1B.sBits.m_WGM13<<TMRx_WGMx3_MASK);
1049
1050     switch(u8Mode){
1051         case TMRx_u8_Normal_Mode:
1052         case TMR1_CTC_Mode_Mode_ICR1:
1053         case TMR1_CTC_Mode_Mode_ICR1A:
1054             switch(u8OutMode){
1055                 case TMR1_COM_Disconnected:     S_TMR1->m_TCCR1A.sBits.m_COM1B
= TMR1_COM_Disconnected;         break;
1056                 case TMR1_COM_Toggle_on_Match:  S_TMR1->m_TCCR1A.sBits.m_COM1B
= TMR1_COM_Toggle_on_Match;      break;
1057                 case TMR1_COM_Clear_on_Match:   S_TMR1->m_TCCR1A.sBits.m_COM1B
= TMR1_COM_Clear_on_Match;       break;
1058                 case TMR1_COM_Set_on_Match:     S_TMR1->m_TCCR1A.sBits.m_COM1B
= TMR1_COM_Set_on_Match;         break;
1059                 default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
1060             }
1061             break;
1062         case TMR1_PWM_PhaseCorrect_Mode_8bit:
1063         case TMR1_PWM_PhaseCorrect_Mode_9bit:
1064         case TMR1_PWM_PhaseCorrect_Mode_10bit:
1065         case TMR1_PWM_Phase_Freq_Correct_Mode_ICR1:
1066         case TMR1_PWM_Phase_Freq_Correct_Mode_ICR1A:
1067         case TMR1_PWM_Phase_Correct_Mode_ICR1:
1068         case TMR1_PWM_Phase_Correct_Mode_ICR1A:
1069             switch(u8OutMode){
1070                 case
TMR1_PhasePWM_ToggleA_on_Match_Mode15:S_TMR1->m_TCCR1A.sBits.m_COM1B =
TMR1_PhasePWM_ToggleA_on_Match_Mode15;  break;
1071                 case TMR1_PhasePWM_Clear_on_Match:
S_TMR1->m_TCCR1A.sBits.m_COM1B = TMR1_PhasePWM_Clear_on_Match;           break;
1072                 case TMR1_PhasePWM_Set_on_Match:
S_TMR1->m_TCCR1A.sBits.m_COM1B = TMR1_PhasePWM_Set_on_Match;             break;
1073                 default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
1074             }
1075             break;
1076         case TMR1_PWM_Fase_Mode_8bit:
1077         case TMR1_PWM_Fase_Mode_9bit:
1078         case TMR1_PWM_Fase_Mode_10bit:
1079         case TMR1_PWM_Fase_Mode_ICR1:
1080         case TMR1_PWM_Fase_Mode_ICR1A:
1081             switch(u8OutMode){
1082                 case
TMR1_FastPWM_ToggleA_on_Match_Mode15:S_TMR1->m_TCCR1A.sBits.m_COM1B =
TMR1_FastPWM_ToggleA_on_Match_Mode15;    break;
1083                 case TMR1_FastPWM_Clear_on_Match:
S_TMR1->m_TCCR1A.sBits.m_COM1B = TMR1_FastPWM_Clear_on_Match;             break;
1084                 case TMR1_FastPWM_Set_on_Match:
S_TMR1->m_TCCR1A.sBits.m_COM1B = TMR1_FastPWM_Set_on_Match;               break;
1085                 default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
1086             }
1087             break;
1088         default:
1089             u8RetErrorState = LBTY_WRITE_ERROR;
1090             break;
1091     }
1092     if(u8RetErrorState == LBTY_OK){
1093         if(u8OutMode != TMR1_COM_Disconnected)
1094             GPIO_u8SetPinDirection(TMR_OC1B_PORT, TMR_OC1B_PIN, PIN_OUTPUT);
1095         strTMR1_Config_GLB.m_TMR_OutputModeB = u8OutMode;
1096     }
1097     return u8RetErrorState;
1098 }
```

void TMR1_vidClrCompareMatch_A_Flag (void)

```
1348 {S_TIFR->sBits.m_OCF1A   = LBTY_RESET;}
```

void TMR1_vidClrCompareMatch_B_Flag (void)

```
1354 {S_TIFR->sBits.m_OCF1B    = LBTY_RESET;}
```

void TMR1_vidClrInputCapture_Flag (void)

```
1342 {S_TIFR->sBits.m_ICF1    = LBTY_RESET;}
```

void TMR1_vidClrOverFlow_Flag (void)

```
1360 {S_TIFR->sBits.m_TOV1    = LBTY_RESET;}
```

void TMR1_vidCompareMatch_A_Disable (void)

```
1345 {S_TIMSK->sBits.m_OCIE1A = LBTY_RESET;}
```

void TMR1_vidCompareMatch_A_Enable (void)

```
1344 {S_TIMSK->sBits.m_OCIE1A = LBTY_SET;}
```

void TMR1_vidCompareMatch_B_Disable (void)

```
1351 {S_TIMSK->sBits.m_OCIE1B = LBTY_RESET;}
```

void TMR1_vidCompareMatch_B_Enable (void)

```
1350 {S_TIMSK->sBits.m_OCIE1B = LBTY_SET;}
```

void TMR1_vidDisable (void)

```
925                             {
926     S_TMR1->m_TCCR1B.sBits.m_CS1 = TMR1_NoClockSource_Disable;
927 }
```

void TMR1_vidEnable (void)

```
916                             {
917     S_TMR1->m_TCCR1B.sBits.m_CS1 = strTMR1_Config_GLB.m_TMR_Prescalar;
918     if(strTMR1_Config_GLB.m_TMR_Prescalar == TMR1_ExternalClock_FallingEdge ||
919        strTMR1_Config_GLB.m_TMR_Prescalar == TMR1_ExternalClock_RisinfEdge){
920         GPIO_u8SetPinDirection(TMR_EXT0_PORT, TMR_EXT0_PIN, PIN_INPUT);
921         GPIO_u8SetPinDirection(TMR_EXT1_PORT, TMR_EXT1_PIN, PIN_INPUT);
922     }
923 }
```

void TMR1_vidGetOverflowNum (u16 * *pu16Num*)

```
1328                                            {
1329     *pu16Num = TMR1_u8OverflewNum_GLB;
1330 }
```

void TMR1_vidGetTicks (u32 * *pu32Tick*)

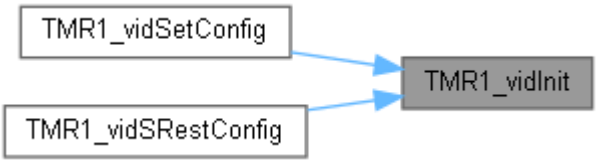
```
1332                                        {
1333     *pu32Tick = (u32)TMR_u16MAX * TMR1_u8OverflewNum_GLB +
S_TMR1->m_TCNT1.u16Reg;
1334 }
```

void TMR1_vidInit (void)

```
848                         {
849
850     //S_SFIOR->sBits.m_PSR10 = LBTY_SET;
851
852     // TMR1_u8SetMode(TMR1_MODE_INIT);
853     S_TMR1->m_TCCR1A.sBits.m_WGM10 = GET_BIT(strTMR1_Config_GLB.m_TMR_Mode,
TMRx_WGMx0_MASK);
854     S_TMR1->m_TCCR1A.sBits.m_WGM11 = GET_BIT(strTMR1_Config_GLB.m_TMR_Mode,
TMRx_WGMx1_MASK);
855     S_TMR1->m_TCCR1B.sBits.m_WGM12 = GET_BIT(strTMR1_Config_GLB.m_TMR_Mode,
TMRx_WGMx2_MASK);
856     S_TMR1->m_TCCR1B.sBits.m_WGM13 = GET_BIT(strTMR1_Config_GLB.m_TMR_Mode,
TMRx_WGMx3_MASK);
857
858     // TMR1_u8SetOutputModeA(TMR1_COMPARE_OUTPUT_A_MODE);
859     // TMR1_u8SetOutputModeB(TMR1_COMPARE_OUTPUT_B_MODE);
860     S_TMR1->m_TCCR1A.sBits.m_FOC1A = strTMR1_Config_GLB.m_TMR_FOCA;
861     S_TMR1->m_TCCR1A.sBits.m_FOC1B = strTMR1_Config_GLB.m_TMR_FOCB;
```

```
862     S_TMR1->m_TCCR1A.sBits.m_COM1A = strTMR1_Config_GLB.m_TMR_OutputModeA;
863     S_TMR1->m_TCCR1A.sBits.m_COM1B = strTMR1_Config_GLB.m_TMR_OutputModeB;
864
865     if(strTMR1_Config_GLB.m_TMR_OutputModeA != TMR1_COM_Disconnected)
866         GPIO_u8SetPinDirection(TMR_OC1A_PORT, TMR_OC1A_PIN, PIN_OUTPUT);
867     if(strTMR1_Config_GLB.m_TMR_OutputModeB != TMR1_COM_Disconnected)
868         GPIO_u8SetPinDirection(TMR_OC1B_PORT, TMR_OC1B_PIN, PIN_OUTPUT);
869
870     //TMR1_vidEnable();
871     S_TMR1->m_TCCR1B.sBits.m_CS1   = strTMR1_Config_GLB.m_TMR_Prescalar;
872     if(strTMR1_Config_GLB.m_TMR_Prescalar == TMR1_ExternalClock_FallingEdge ||
873        strTMR1_Config_GLB.m_TMR_Prescalar == TMR1_ExternalClock_RisinfEdge){
874         GPIO_u8SetPinDirection(TMR_EXT0_PORT, TMR_EXT0_PIN, PIN_INPUT);
875         GPIO_u8SetPinDirection(TMR_EXT1_PORT, TMR_EXT1_PIN, PIN_INPUT);
876     }
877     //TMR1_vidInitInputCapture();
878     S_TMR1->m_TCCR1B.sBits.m_ICNC1 = strTMR1_Config_GLB.m_TMR_InputNoise;
879     S_TMR1->m_TCCR1B.sBits.m_ICES1 = strTMR1_Config_GLB.m_TMR_InputEdge;
880     if(strTMR1_Config_GLB.m_TMR_InputEdge != TMR1_Capture_Off){
881         GPIO_u8SetPinDirection(TMR_ICP1_PORT, TMR_ICP1_PIN, PIN_INPUT);
882     }
883
884 #if defined(TMR1)
885     //TMR1_u8SetInputCapture(TMR1_INPUT_CAPTURE_INIT);
886     S_TMR1->m_ICR1.u16Reg  = strTMR1_Config_GLB.m_TMR_Input;
887     //TMR1_u8SetOutputCompare_A(TMR1_OUTPUT_COMPARE_A_INIT);
888     S_TMR1->m_OCR1A.u16Reg = strTMR1_Config_GLB.m_TMR_CompareA;
889     //TMR1_u8SetOutputCompare_B(TMR1_OUTPUT_COMPARE_B_INIT);
890     S_TMR1->m_OCR1B.u16Reg = strTMR1_Config_GLB.m_TMR_CompareB;
891     //TMR1_u8SetCounter(TMR1_COUNTER_INIT);
892     S_TMR1->m_TCNT1.u16Reg = strTMR1_Config_GLB.m_TMR_Reload;
893 #elif defined(PWM1)
894     PWM_vidDisable_OC1x();
895     PWM_u8SetFreq_OC1x(strTMR1_Config_GLB.m_TMR_Freq);
896
897     if(strTMR1_Config_GLB.m_TMR_OutputModeA != TMR1_COM_Disconnected)
898         PWM_u8SetDuty_OC1A(strTMR1_Config_GLB.m_TMR_Duty_A);
899     if(strTMR1_Config_GLB.m_TMR_OutputModeB != TMR1_COM_Disconnected)
900         PWM_u8SetDuty_OC1B(strTMR1_Config_GLB.m_TMR_Duty_B);
901
902     TMR1_Reload_Delay = TMRx_RELOAD_DELAY[strTMR1_Config_GLB.m_TMR_Prescalar];
903 #endif
904
905     S_TIMSK->sBits.m_TICIE1 = strTMR1_Config_GLB.m_TMR_TICIE;
906     S_TIMSK->sBits.m_OCIE1A = strTMR1_Config_GLB.m_TMR_OCIEA;
907     S_TIMSK->sBits.m_OCIE1B = strTMR1_Config_GLB.m_TMR_OCIEB;
908     S_TIMSK->sBits.m_TOIE1  = strTMR1_Config_GLB.m_TMR_TOIE;
909
910     S_TIFR->sBits.m_ICF1    = LBTY_RESET;
911     S_TIFR->sBits.m_OCF1A   = LBTY_RESET;
912     S_TIFR->sBits.m_OCF1B   = LBTY_RESET;
913     S_TIFR->sBits.m_TOV1    = LBTY_RESET;
914 }
```

Here is the caller graph for this function:



void TMR1_vidInitInputCapture (void)

```
929                                 {
930     S_TMR1->m_TCCR1B.sBits.m_ICNC1 = strTMR1_Config_GLB.m_TMR_InputNoise;
931     S_TMR1->m_TCCR1B.sBits.m_ICES1 = strTMR1_Config_GLB.m_TMR_InputEdge;
932     if(strTMR1_Config_GLB.m_TMR_InputEdge != TMR1_Capture_Off){
933         GPIO_u8SetPinDirection(TMR_ICP1_PORT, TMR_ICP1_PIN, PIN_INPUT);
934     }
935 }
```

void TMR1_vidInputCapture_Disable (void)

```
1339 {S_TIMSK->sBits.m_TICIE1 = LBTY_RESET;}
```

void TMR1_vidInputCapture_Enable (void)

```
1338 {S_TIMSK->sBits.m_TICIE1 = LBTY_SET;}
```

void TMR1_vidOverFlow_Disable (void)

```
1357 {S_TIMSK->sBits.m_TOIE1 = LBTY_RESET;}
```

void TMR1_vidOverFlow_Enable (void)

```
1356 {S_TIMSK->sBits.m_TOIE1 = LBTY_SET;}
```

void TMR1_vidResetForceOutputCompareA (void)

```
941                                                 {
942     S_TMR1->m_TCCR1A.sBits.m_FOC1A  = strTMR1_Config_GLB.m_TMR_FOCA =
LBTY_RESET;
943 }
```

void TMR1_vidResetForceOutputCompareB (void)

```
949                                                 {
950     S_TMR1->m_TCCR1A.sBits.m_FOC1B  = strTMR1_Config_GLB.m_TMR_FOCB =
LBTY_RESET;
951 }
```

void TMR1_vidSetCallBack_CaptureEvent (void(*)(void) *pCallBack*)

```
1362                                                                    {
1363     if(*pCallBack == LBTY_NULL)     return;
1364     pFuncCallBack_TMR1_CaptureEven = pCallBack;
1365 }
```

void TMR1_vidSetCallBack_CompareMatch_A (void(*)(void) *pCallBack*)

```
1366                                                                      {
1367     if(*pCallBack == LBTY_NULL)     return;
1368     pFuncCallBack_TMR1_CompareMatch_A = pCallBack;
1369 }
```

void TMR1_vidSetCallBack_CompareMatch_B (void(*)(void) *pCallBack*)

```
1370                                                                      {
1371     if(*pCallBack == LBTY_NULL)     return;
1372     pFuncCallBack_TMR1_CompareMatch_B = pCallBack;
1373 }
```

void TMR1_vidSetCallBack_OverFlow (void(*)(void) *pCallBack*)

```
1374                                                                  {
1375     if(*pCallBack == LBTY_NULL)     return;
1376     pFuncCallBack_TMR1_OverFlow = pCallBack;
1377 }
```

void TMR1_vidSetCompareMatch_A_Flag (void)

```
1347 {S_TIFR->sBits.m_OCF1A    = LBTY_SET;}
```

void TMR1_vidSetCompareMatch_B_Flag (void)

```
1353 {S_TIFR->sBits.m_OCF1B    = LBTY_SET;}
```

void TMR1_vidSetConfig (TMR1_tstrConfig const *const *pstrConfig*)

```
812                                                                     {
813     if(pstrConfig != LBTY_NULL){
814         strTMR1_Config_GLB = *pstrConfig;
815     }
816     TMR1_vidInit();
817 }
```

Here is the call graph for this function:

void TMR1_vidSetForceOutputCompareA (void)

```
937                                             {
```

```
938     S_TMR1->m_TCCR1A.sBits.m_FOC1A  = strTMR1_Config_GLB.m_TMR_FOCA = LBTY_SET;
939 }
```

void TMR1_vidSetForceOutputCompareB (void)

```
945                                             {
946     S_TMR1->m_TCCR1A.sBits.m_FOC1B  = strTMR1_Config_GLB.m_TMR_FOCB = LBTY_SET;
947 }
```

void TMR1_vidSetInputCapture_Flag (void)

```
1341 {S_TIFR->sBits.m_ICF1   = LBTY_SET;}
```

void TMR1_vidSetOverFlow_Flag (void)

```
1359 {S_TIFR->sBits.m_TOV1   = LBTY_SET;}
```

void TMR1_vidSetOverflowNum (u16 *u16Num*)

```
1325                                            {
1326     TMR1_u8OverflewNum_GLB = u16Num;
1327 }
```

LCTY_INLINE void TMR1_vidSetWaveGenerationMode (TMR1_tenuWaveGenerationMode *u8Mode*)

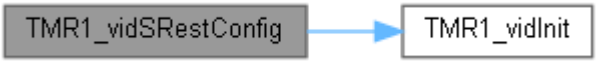
```
953
{
954     S_TMR1->m_TCCR1A.sBits.m_WGM10 = GET_BIT(u8Mode, TMRx_WGMx0_MASK);
955     S_TMR1->m_TCCR1A.sBits.m_WGM11 = GET_BIT(u8Mode, TMRx_WGMx1_MASK);
956     S_TMR1->m_TCCR1B.sBits.m_WGM12 = GET_BIT(u8Mode, TMRx_WGMx2_MASK);
957     S_TMR1->m_TCCR1B.sBits.m_WGM13 = GET_BIT(u8Mode, TMRx_WGMx3_MASK);
958     strTMR1_Config_GLB.m_TMR_Mode = u8Mode;
959 }
```

Here is the caller graph for this function:

void TMR1_vidSRestConfig (TMR1_tstrConfig *const *pstrConfig*)

```
819                                                             {
820 #if defined(PWM1)
821     strTMR1_Config_GLB.m_TMR_Freq        = PWM1_FREQ_INIT;
822     strTMR1_Config_GLB.m_TMR_Duty_A      = PWM1A_DUTY_INIT;
823     strTMR1_Config_GLB.m_TMR_Duty_B      = PWM1B_DUTY_INIT;
824 #endif
825     strTMR1_Config_GLB.m_TMR_Reload      = TMR1_COUNTER_INIT;
826     strTMR1_Config_GLB.m_TMR_Input       = TMR1_INPUT_CAPTURE_INIT;
827     strTMR1_Config_GLB.m_TMR_CompareA    = TMR1_OUTPUT_COMPARE_A_INIT;
828     strTMR1_Config_GLB.m_TMR_CompareB    = TMR1_OUTPUT_COMPARE_B_INIT;
829     strTMR1_Config_GLB.m_TMR_Prescalar   = TMR1_CLOCK_SOURCE;
830     strTMR1_Config_GLB.m_TMR_Mode        = TMR1_MODE_INIT;
831     strTMR1_Config_GLB.m_TMR_OutputModeA = TMR1_COMPARE_OUTPUT_A_MODE;
832     strTMR1_Config_GLB.m_TMR_OutputModeB = TMR1_COMPARE_OUTPUT_B_MODE;
833     strTMR1_Config_GLB.m_TMR_FOCA        = LBTY_RESET;
834     strTMR1_Config_GLB.m_TMR_FOCB        = LBTY_RESET;
835     strTMR1_Config_GLB.m_TMR_TICIE       = TMR1_INPUT_CAPTURE_INTERRUPT_STATE;
836     strTMR1_Config_GLB.m_TMR_OCIEA       =
TMR1_COMPARE_A_MATCH_INTERRUPT_STATE;
837     strTMR1_Config_GLB.m_TMR_OCIEB       =
TMR1_COMPARE_B_MATCH_INTERRUPT_STATE;
838     strTMR1_Config_GLB.m_TMR_TOIE        = TMR1_OVERFLOW_INTERRUPT_STATE;
839     strTMR1_Config_GLB.m_TMR_InputNoise  = TMR1_INPUT_CAPTURE_NOISE_CANCELER;
840     strTMR1_Config_GLB.m_TMR_InputEdge   = TMR1_INPUT_CAPTURE_EDGE_SELECT;
841
842     if(pstrConfig != LBTY_NULL){
843         *pstrConfig = strTMR1_Config_GLB;
844     }
845     TMR1_vidInit();
846 }
```

Here is the call graph for this function:



LBTY_tenuErrorStatus TMR2_u8Async (TMR2_tenuInputCaptureEdgeSelect *u8Async*)

```
554                                                                                 {
555     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
556
557     switch(u8Async){
558         case TMR2_IO_Clock:
559         case TMR2_TOSC_Clock:
560             S_TIMSK->sBits.m_OCIE2 = LBTY_RESET;
561             S_TIMSK->sBits.m_TOIE2 = LBTY_RESET;
562             S_TMR2->m_ASSR.sBits.m_AS2 = strTMR2_Config_GLB.m_TMR_AsyClock =
u8Async;
563             if(u8Async == TMR2_TOSC_Clock){
564                 GPIO_u8SetPinDirection(TMR_OSC1_PORT, TMR_OSC1_PIN, PIN_INPUT);
565                 GPIO_u8SetPinDirection(TMR_OSC2_PORT, TMR_OSC2_PIN, PIN_INPUT);
566             }
567             break;
568         default:
569             u8RetErrorState = LBTY_WRITE_ERROR;
570             break;
571     }
572
573     return u8RetErrorState;
574 }
```

LBTY_tenuErrorStatus TMR2_u8GetCounter (u8 * *pu8Reload*)

```
680                                                         {
681     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
682     if(pu8Reload != LBTY_NULL){
683         *pu8Reload = S_TMR2->m_TCNT2;
684     }else{
685         u8RetErrorState = LBTY_NULL_POINTER;
686     }
687     return u8RetErrorState;
688 }
```

LBTY_tenuErrorStatus TMR2_u8GetOutputCompare (u8 * *pu8Reload*)

```
670                                                                 {
671     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
672     if(pu8Reload != LBTY_NULL){
673         *pu8Reload = S_TMR2->m_OCR2;
674     }else{
675         u8RetErrorState = LBTY_NULL_POINTER;
676     }
677     return u8RetErrorState;
678 }
```

LBTY_tenuErrorStatus TMR2_u8SetCounter (u8 *u8Reload*)

```
659                                                     {
660     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
661     if(u8Reload <= LBTY_u8MAX){
662         TMR2_vidTimerUpdateBusy();
663         S_TMR2->m_TCNT2 = strTMR2_Config_GLB.m_TMR_Reload = u8Reload;
664     }else{
665         u8RetErrorState = LBTY_WRITE_ERROR;
666     }
667     return u8RetErrorState;
668 }
```

Here is the call graph for this function:

LBTY_tenuErrorStatus TMR2_u8SetMode (TMRx_u8_tenuWaveGenerationMode *u8Mode*)

```
576                                                                                 {
577     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
578     TMR2_vidControlUpdateBusy();
579     switch(u8Mode){
580         case TMRx_u8_Normal_Mode:
```

```
581             S_TMR2->m_TCCR2.sBits.m_WGMx0 = GET_BIT(TMRx_u8_Normal_Mode,
TMRx_WGMx0_MASK);
582             S_TMR2->m_TCCR2.sBits.m_WGMx1 = GET_BIT(TMRx_u8_Normal_Mode,
TMRx_WGMx1_MASK);
583             break;
584         case TMRx_u8_PWM_PhaseCorrect_Mode:
585             S_TMR2->m_TCCR2.sBits.m_WGMx0 =
GET_BIT(TMRx_u8_PWM_PhaseCorrect_Mode, TMRx_WGMx0_MASK);
586             S_TMR2->m_TCCR2.sBits.m_WGMx1 =
GET_BIT(TMRx_u8_PWM_PhaseCorrect_Mode, TMRx_WGMx1_MASK);
587             break;
588         case TMRx_u8_CTC_Mode_Mode:
589             S_TMR2->m_TCCR2.sBits.m_WGMx0 = GET_BIT(TMRx_u8_CTC_Mode_Mode,
TMRx_WGMx0_MASK);
590             S_TMR2->m_TCCR2.sBits.m_WGMx1 = GET_BIT(TMRx_u8_CTC_Mode_Mode,
TMRx_WGMx1_MASK);
591             break;
592         case TMRx_u8_PWM_Fase_Mode:
593             S_TMR2->m_TCCR2.sBits.m_WGMx0 = GET_BIT(TMRx_u8_PWM_Fase_Mode,
TMRx_WGMx0_MASK);
594             S_TMR2->m_TCCR2.sBits.m_WGMx1 = GET_BIT(TMRx_u8_PWM_Fase_Mode,
TMRx_WGMx1_MASK);
595             break;
596         default:
597             u8RetErrorState = LBTY_WRITE_ERROR;
598             break;
599     }
600     if(u8RetErrorState == LBTY_OK){
601         strTMR2_Config_GLB.m_TMR_Mode = u8Mode;
602     }
603     return u8RetErrorState;
604 }
```

Here is the call graph for this function:

LBTY_tenuErrorStatus TMR2_u8SetOutputCompare (u8 *u8Reload*)

```
648                                                         {
649     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
650     if(u8Reload <= LBTY_u8MAX){
651         TMR2_vidCompareUpdateBusy();
652         S_TMR2->m_OCR2 = strTMR2_Config_GLB.m_TMR_Compare = u8Reload;
653     }else{
654         u8RetErrorState = LBTY_WRITE_ERROR;
655     }
656     return u8RetErrorState;
657 }
```

Here is the call graph for this function:

LBTY_tenuErrorStatus TMR2_u8SetOutputMode (TMRx_u8_tenuCompareOutputMode *u8OutMode*)

```
606
{
607     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
608     TMRx_u8_tenuWaveGenerationMode u8Mode =
609             (S_TMR2->m_TCCR2.sBits.m_WGMx0<<TMRx_WGMx0_MASK) |
(S_TMR2->m_TCCR2.sBits.m_WGMx1<<TMRx_WGMx1_MASK);
610     TMR2_vidControlUpdateBusy();
611     switch(u8Mode){
612         case TMRx_u8_Normal_Mode:
613         case TMRx_u8_CTC_Mode_Mode:
614             switch(u8OutMode){
615                 case TMRx_u8_COM_Disconnected:
S_TMR2->m_TCCR2.sBits.m_COMx = TMRx_u8_COM_Disconnected;            break;
616                 case TMRx_u8_COM_Toggle_on_Match:
S_TMR2->m_TCCR2.sBits.m_COMx = TMRx_u8_COM_Toggle_on_Match;         break;
617                 case TMRx_u8_COM_Clear_on_Match:
S_TMR2->m_TCCR2.sBits.m_COMx = TMRx_u8_COM_Clear_on_Match;          break;
618                 case TMRx_u8_COM_Set_on_Match:
S_TMR2->m_TCCR2.sBits.m_COMx = TMRx_u8_COM_Set_on_Match;            break;
```

```
619                 default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
620             }
621             break;
622         case TMRx_u8_PWM_PhaseCorrect_Mode:
623             switch(u8OutMode){
624                 case
TMRx_u8_PhasePWM_Clear_on_Match:S_TMR2->m_TCCR2.sBits.m_COMx =
TMRx_u8_PhasePWM_Clear_on_Match;    break;
625                 case TMRx_u8_PhasePWM_Set_on_Match:
S_TMR2->m_TCCR2.sBits.m_COMx = TMRx_u8_PhasePWM_Set_on_Match;      break;
626                 default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
627             }
628             break;
629         case TMRx_u8_PWM_Fase_Mode:
630             switch(u8OutMode){
631                 case
TMRx_u8_FastPWM_Clear_on_Match:S_TMR2->m_TCCR2.sBits.m_COMx =
TMRx_u8_FastPWM_Clear_on_Match;      break;
632                 case TMRx_u8_FastPWM_Set_on_Match:
S_TMR2->m_TCCR2.sBits.m_COMx = TMRx_u8_FastPWM_Set_on_Match;        break;
633                 default:    u8RetErrorState = LBTY_WRITE_ERROR;     break;
634             }
635             break;
636         default:
637             u8RetErrorState = LBTY_WRITE_ERROR;
638             break;
639     }
640     if(u8RetErrorState == LBTY_OK){
641         if(u8OutMode != TMRx_u8_COM_Disconnected)
642             GPIO_u8SetPinDirection(TMR_OC2_PORT, TMR_OC2_PIN, PIN_OUTPUT);
643         strTMR2_Config_GLB.m_TMR_OutputMode = u8OutMode;
644     }
645     return u8RetErrorState;
646 }
```

Here is the call graph for this function:

void TMR2_vidClrCompareMatch_Flag (void)

```
769 {S_TIFR->sBits.m_OCF2   = LBTY_RESET;}
```

void TMR2_vidClrOverFlow_Flag (void)

```
775 {S_TIFR->sBits.m_TOV2   = LBTY_RESET;}
```

void TMR2_vidCompareMatch_Disable (void)

```
766 {S_TIMSK->sBits.m_OCIE2 = LBTY_RESET;}
```

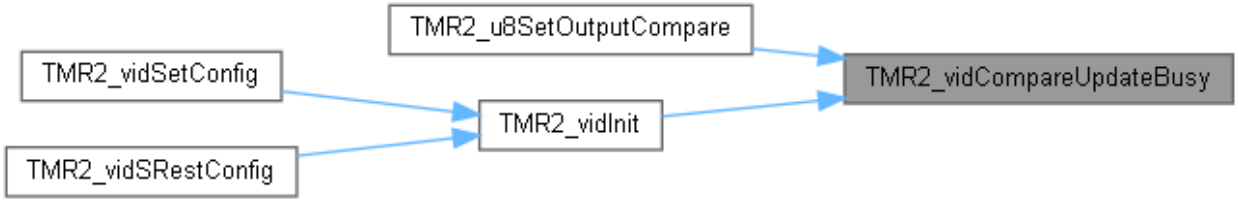
void TMR2_vidCompareMatch_Enable (void)

```
765 {S_TIMSK->sBits.m_OCIE2 = LBTY_SET;}
```

LCTY_INLINE void TMR2_vidCompareUpdateBusy (void)

```
454 {while(S_TMR2->m_ASSR.sBits.m_OCR2UB);}
```

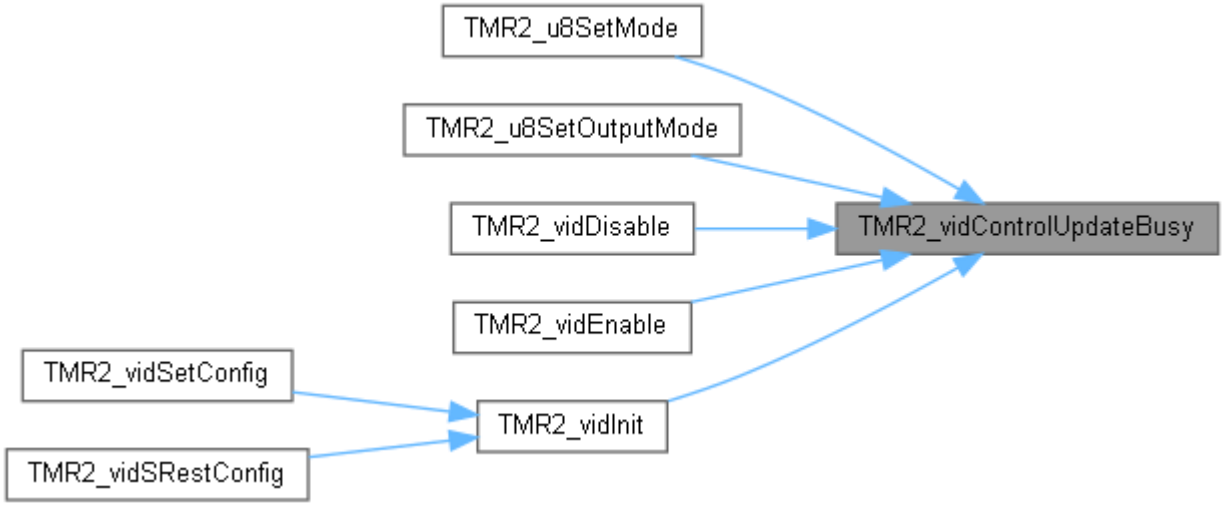
Here is the caller graph for this function:



LCTY_INLINE void TMR2_vidControlUpdateBusy (void)

```
451 {while(S_TMR2->m_ASSR.sBits.m_TCR2UB);}
```

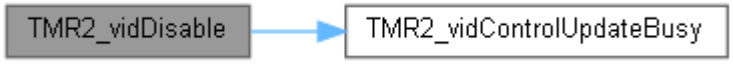
Here is the caller graph for this function:



void TMR2_vidDisable (void)

```
541                                 {
542     TMR2_vidControlUpdateBusy();
543     S_TMR2->m_TCCR2.sBits.m_CSx = TMR2_NoClockSource_Disable;
544 }
```

Here is the call graph for this function:



void TMR2_vidEnable (void)

```
536                                 {
537     TMR2_vidControlUpdateBusy();
538     S_TMR2->m_TCCR2.sBits.m_CSx = strTMR2_Config_GLB.m_TMR_Prescalar;
539 }
```

Here is the call graph for this function:

void TMR2_vidGetCompareNum (u16 * *pu16Num*)

```
748                                             {
749     *pu16Num = TMR2_u8CompareNum_GLB;
750 }
```

void TMR2_vidGetOverflowNum (u16 * *pu16Num*)

```
755                                             {
756     *pu16Num = TMR2_u8OverflewNum_GLB;
757 }
```

void TMR2_vidGetTicks (u32 * *pu32Tick*)

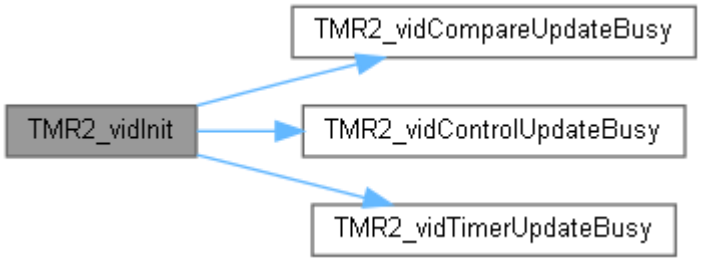
```
759                                             {
760     *pu32Tick = (u32)TMR_u8MAX * TMR2_u8OverflewNum_GLB + S_TMR2->m_TCNT2;
761 }
```

void TMR2_vidInit (void)

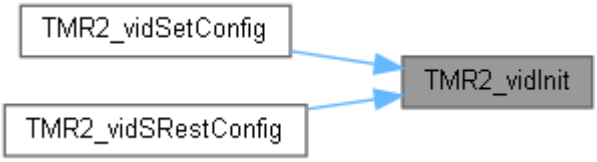
```
487                                 {
488     //S_SFIOR->sBits.m_PSR2 = LBTY_SET;
489
490     // TMR2_vidAsync(TMR2_ASYNCHRONOUS_CLOCK);
491     S_TIMSK->sBits.m_OCIE2 = LBTY_RESET;
492     S_TIMSK->sBits.m_TOIE2 = LBTY_RESET;
493     S_TMR2->m_ASSR.sBits.m_AS2 = strTMR2_Config_GLB.m_TMR_AsyClock;
494
495     if(strTMR2_Config_GLB.m_TMR_AsyClock == TMR2_TOSC_Clock){
496         GPIO_u8SetPinDirection(TMR_OSC1_PORT, TMR_OSC1_PIN, PIN_INPUT);
497         GPIO_u8SetPinDirection(TMR_OSC2_PORT, TMR_OSC2_PIN, PIN_INPUT);
498     }
```

```
499
500     //TMR2_vidEnable();
501     TMR2_vidControlUpdateBusy();
502     S_TMR2->m_TCCR2.sBits.m_CSx   = strTMR2_Config_GLB.m_TMR_Prescalar;
503     //TMR2_u8SetMode(TMR2_MODE_INIT);
504     TMR2_vidControlUpdateBusy();
505     S_TMR2->m_TCCR2.sBits.m_WGMx0 = GET_BIT(strTMR2_Config_GLB.m_TMR_Mode,
TMRx_WGMx0_MASK);
506     S_TMR2->m_TCCR2.sBits.m_WGMx1 = GET_BIT(strTMR2_Config_GLB.m_TMR_Mode,
TMRx_WGMx1_MASK);
507     //TMR2_u8SetOutputMode(TMR2_COMPARE_OUTPUT_MODE);
508     TMR2_vidControlUpdateBusy();
509     S_TMR2->m_TCCR2.sBits.m_COMx  = strTMR2_Config_GLB.m_TMR_OutputMode;
510     if(S_TMR2->m_TCCR2.sBits.m_COMx != TMRx_u8_COM_Disconnected)
511         GPIO_u8SetPinDirection(TMR_OC2_PORT, TMR_OC2_PIN, PIN_OUTPUT);
512     //TMR2_vidSetForceOutputCompare();
513     S_TMR2->m_TCCR2.sBits.m_FOCx  = strTMR2_Config_GLB.m_TMR_FOC;
514
515 #if defined(TMR2)
516     //TMR2_u8SetOutputCompare(TMR2_OUTPUT_COMPARE_INIT);
517     TMR2_vidCompareUpdateBusy();
518     S_TMR2->m_OCR2 = strTMR2_Config_GLB.m_TMR_Compare;
519     //TMR2_u8SetCounter(TMR2_COUNTER_INIT);
520     TMR2_vidTimerUpdateBusy();
521     S_TMR2->m_TCNT2 = strTMR2_Config_GLB.m_TMR_Reload;
522 #elif defined(PWM2)
523     PWM_vidDisable_OC2();
524     PWM_u8SetFreq_OC2(strTMR2_Config_GLB.m_TMR_Freq);
525     PWM_u8SetDuty_OC2(strTMR2_Config_GLB.m_TMR_Duty);
526     TMR2_Reload_Delay = TMRx_RELOAD_DELAY[strTMR2_Config_GLB.m_TMR_Prescalar];
527 #endif
528
529     S_TIMSK->sBits.m_OCIE2 = strTMR2_Config_GLB.m_TMR_OCIE;
530     S_TIMSK->sBits.m_TOIE2 = strTMR2_Config_GLB.m_TMR_OVIE;
531
532     S_TIFR->sBits.m_OCF2   = LBTY_RESET;
533     S_TIFR->sBits.m_TOV2   = LBTY_RESET;
534 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void TMR2_vidOverFlow_Disable (void)

```
772 {S_TIMSK->sBits.m_TOIE2 = LBTY_RESET;}
```

void TMR2_vidOverFlow_Enable (void)

```
771 {S_TIMSK->sBits.m_TOIE2 = LBTY_SET;}
```

void TMR2_vidResetForceOutputCompare (void)

```
550                                         {
551     S_TMR2->m_TCCR2.sBits.m_FOCx  = strTMR2_Config_GLB.m_TMR_FOC = LBTY_RESET;
552 }
```

void TMR2_vidSetCallBack_CompareMatch (void(*)(void) *pCallBack*)

```
777                                                                 {
```

```
778     pFuncCallBack_TMR2_CompareMatch = pCallBack;
779 }
```

void TMR2_vidSetCallBack_OverFlow (void(*)(void) *pCallBack*)

```
780                                                         {
781     pFuncCallBack_TMR2_OverFlow = pCallBack;
782 }
```

void TMR2_vidSetCompareMatch_Flag (void)

```
768 {S_TIFR->sBits.m_OCF2    = LBTY_SET;}
```

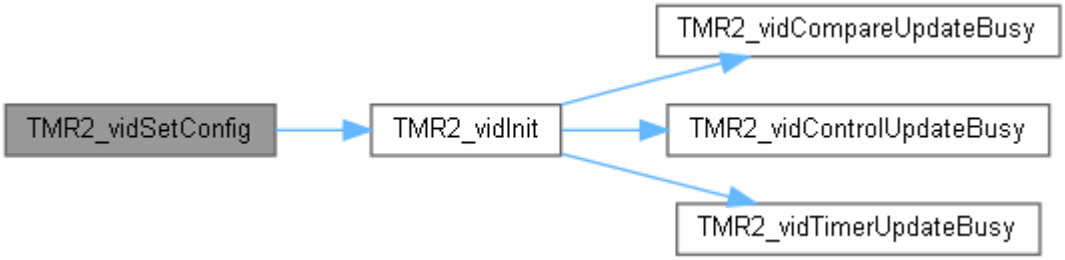
void TMR2_vidSetCompareNum (u16 *u16Num*)

```
745                                         {
746     TMR2_u8CompareNum_GLB = u16Num;
747 }
```

void TMR2_vidSetConfig (TMR2_tstrConfig const *const *pstrConfig*)

```
459                                                             {
460     if(pstrConfig != LBTY_NULL){
461         strTMR2_Config_GLB = *pstrConfig;
462     }
463     TMR2_vidInit();
464 }
```

Here is the call graph for this function:



void TMR2_vidSetForceOutputCompare (void)

```
546                                             {
547     S_TMR2->m_TCCR2.sBits.m_FOCx  = strTMR2_Config_GLB.m_TMR_FOC = LBTY_SET;
548 }
```

void TMR2_vidSetOverFlow_Flag (void)

```
774 {S_TIFR->sBits.m_TOV2    = LBTY_SET;}
```

void TMR2_vidSetOverflowNum (u16 *u16Num*)

```
752                                         {
753     TMR2_u8OverflewNum_GLB = u16Num;
754 }
```

void TMR2_vidSRestConfig (TMR2_tstrConfig *const *pstrConfig*)

```
466                                                             {
467 #if defined(PWM2)
468     strTMR2_Config_GLB.m_TMR_Freq      = PWM2_FREQ_INIT;
469     strTMR2_Config_GLB.m_TMR_Duty      = PWM2_DUTY_INIT;
470 #endif
471     strTMR2_Config_GLB.m_TMR_Reload    = TMR2_COUNTER_INIT;
472     strTMR2_Config_GLB.m_TMR_Compare   = TMR2_OUTPUT_COMPARE_INIT;
473     strTMR2_Config_GLB.m_TMR_Prescalar = TMR2_CLOCK_SOURCE;
474     strTMR2_Config_GLB.m_TMR_Mode      = TMR2_MODE_INIT;
475     strTMR2_Config_GLB.m_TMR_OutputMode= TMR2_COMPARE_OUTPUT_MODE;
476     strTMR2_Config_GLB.m_TMR_FOC       = LBTY_RESET;
477     strTMR2_Config_GLB.m_TMR_OVIE      = TMR2_OVERFLOW_INTERRUPT_INIT_STATE;
478     strTMR2_Config_GLB.m_TMR_OCIE      =
TMR2_COMPARE_MATCH_INTERRUPT_INIT_STATE;
479     strTMR2_Config_GLB.m_TMR_AsyClock  = TMR2_ASYNCHRONOUS_CLOCK;
480
481     if(pstrConfig != LBTY_NULL){
482         *pstrConfig = strTMR2_Config_GLB;
483     }
```

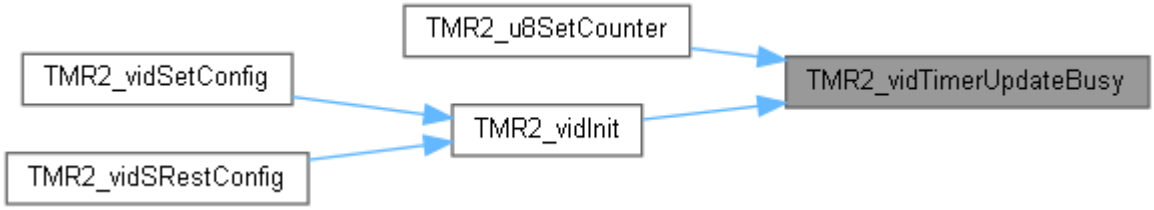
```
484     TMR2_vidInit();
485 }
```

Here is the call graph for this function:

LCTY_INLINE void TMR2_vidTimerUpdateBusy (void)

```
457 {while(S_TMR2->m_ASSR.sBits.m_TCN2UB);}
```

Here is the caller graph for this function:



Variable Documentation

void(* pFuncCallBack_TMR0_CompareMatch) (void) (void) = INTP_vidCallBack [static]

void(* pFuncCallBack_TMR0_OverFlow) (void) (void) = INTP_vidCallBack [static]

void(* pFuncCallBack_TMR1_CaptureEven) (void) (void) = INTP_vidCallBack [static]

void(* pFuncCallBack_TMR1_CompareMatch_A) (void) (void) = INTP_vidCallBack [static]

void(* pFuncCallBack_TMR1_CompareMatch_B) (void) (void) = INTP_vidCallBack [static]

void(* pFuncCallBack_TMR1_OverFlow) (void) (void) = INTP_vidCallBack [static]

void(* pFuncCallBack_TMR2_CompareMatch) (void) (void) = INTP_vidCallBack [static]

void(* pFuncCallBack_TMR2_OverFlow) (void) (void) = INTP_vidCallBack [static]

volatile TMR0_tstrConfig strTMR0_Config_GLB [static]

```
Initial value:= {




    .m_TMR_Reload     = TMR0_COUNTER_INIT,
    .m_TMR_Compare    = TMR0_OUTPUT_COMPARE_INIT,
    .m_TMR_Prescalar  = TMR0_CLOCK_SOURCE,
    .m_TMR_Mode       = TMR0_MODE_INIT,
    .m_TMR_OutputMode = TMR0_COMPARE_OUTPUT_MODE,
    .m_TMR_FOC        = LBTY_RESET,
```

```
    .m_TMR_OVIE       = TMR0_OVERFLOW_INTERRUPT_INIT_STATE,
    .m_TMR_OCIE       = TMR0_COMPARE_MATCH_INTERRUPT_INIT_STATE
}
```

volatile TMR1_tstrConfig strTMR1_Config_GLB [static]

```
Initial value:={




    .m_TMR_Reload      = TMR1_COUNTER_INIT,
    .m_TMR_Input       = TMR1_INPUT_CAPTURE_INIT,
    .m_TMR_CompareA    = TMR1_OUTPUT_COMPARE_A_INIT,
    .m_TMR_CompareB    = TMR1_OUTPUT_COMPARE_B_INIT,
    .m_TMR_Prescalar   = TMR1_CLOCK_SOURCE,
    .m_TMR_Mode        = TMR1_MODE_INIT,
    .m_TMR_OutputModeA = TMR1_COMPARE_OUTPUT_A_MODE,
    .m_TMR_OutputModeB = TMR1_COMPARE_OUTPUT_B_MODE,
    .m_TMR_FOCA        = LBTY_RESET,
    .m_TMR_FOCB        = LBTY_RESET,
    .m_TMR_TICIE       = TMR1_INPUT_CAPTURE_INTERRUPT_STATE,
    .m_TMR_OCIEA       = TMR1_COMPARE_A_MATCH_INTERRUPT_STATE,
    .m_TMR_OCIEB       = TMR1_COMPARE_B_MATCH_INTERRUPT_STATE,
    .m_TMR_TOIE        = TMR1_OVERFLOW_INTERRUPT_STATE,
    .m_TMR_InputNoise  = TMR1_INPUT_CAPTURE_NOISE_CANCELER,
    .m_TMR_InputEdge   = TMR1_INPUT_CAPTURE_EDGE_SELECT,
}
```

volatile TMR2_tstrConfig strTMR2_Config_GLB [static]

```
Initial value:= {




    .m_TMR_Reload     = TMR2_COUNTER_INIT,
    .m_TMR_Compare    = TMR2_OUTPUT_COMPARE_INIT,
    .m_TMR_Prescalar  = TMR2_CLOCK_SOURCE,
    .m_TMR_Mode       = TMR2_MODE_INIT,
    .m_TMR_OutputMode = TMR2_COMPARE_OUTPUT_MODE,
    .m_TMR_FOC        = LBTY_RESET,
    .m_TMR_OVIE       = TMR2_OVERFLOW_INTERRUPT_INIT_STATE,
    .m_TMR_OCIE       = TMR2_COMPARE_MATCH_INTERRUPT_INIT_STATE,
    .m_TMR_AsyClock   = TMR2_ASYNCHRONOUS_CLOCK
}
```

volatile u16 TMR0_u8CompareNum_GLB = LBTY_u8ZERO [static]

volatile u16 TMR0_u8OverflewNum_GLB = LBTY_u8ZERO [static]

volatile u16 TMR1_u8OverflewNum_GLB = LBTY_u8ZERO [static]

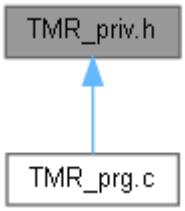
volatile u16 TMR2_u8CompareNum_GLB = LBTY_u8ZERO [static]

volatile u16 TMR2_u8OverflewNum_GLB = LBTY_u8ZERO [static]

const u8 TMRx_RELOAD_DELAY[] = {0, 47, 6, 1, 1, 1, 0, 0}

TMR_priv.h File Reference

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



Data Structures

union ASSR_type*: Type define of Union bit field "Asynchronous Status Register"*

union TCCRx_type*: Type define of Union bit field "Timer/Counter Control Register"*

struct GPTMR2_type*: General Purpose Input Output Registers*

struct GPTMR0_type*: General Purpose Input Output Registers*

union TCCR1B_type*: Type define of Union bit field "Timer/Counter Control Register B"*
union TCCR1A_type*: Type define of Union bit field "Timer/Counter Control Register A"*
union BYTE_type*: Type define of Union bit field of Single Byte"byte bits exchange"*
union Word_type*: Type define of Union bit field of Half Word "bits exchange"*

struct GPTMR1_type*: General Purpose Input Output Registers*

union SFIOR_type*: Type define of Union bit field "Special Function I/O Register"*

union TIFR_type*: Type define of Union bit field "Timer/Counter Interrupt Flag Register Reg"*

union TIMSK_type*: Type define of Union bit field "Timer/Counter Control Register"*

Macros

- #define S_TMR2 ((GPTMR2_type* const)0x42U)
- #define ASSR (*(volatile u8* const)0x42U)
- #define OCR2 (*(volatile u8* const)0x43U)
- #define TCNT2 (*(volatile u8* const)0x44U)
- #define TCCR2 (*(volatile u8* const)0x45U)
- #define S_TMR1 ((GPTMR1_type* const)0x46U)
- #define ICR1L (*(volatile u8* const)0x46U)
- #define ICR1H (*(volatile u8* const)0x47U)
- #define OCR1BL (*(volatile u8* const)0x48U)
- #define OCR1BH (*(volatile u8* const)0x49U)
- #define OCR1AL (*(volatile u8* const)0x4AU)

- #define OCR1AH (*(volatile u8* const)0x4BU)
- #define TCNT1L (*(volatile u8* const)0x4CU)
- #define TCNT1H (*(volatile u8* const)0x4DU)
- #define TCCR1B (*(volatile u8* const)0x4EU)
- #define TCCR1A (*(volatile u8* const)0x4FU)
- #define S_SFIOR ((SFIOR_type* const)0x50U)
- #define SFIOR (*(volatile u8* const)0x50U)
- #define S_TMR0 ((GPTMR0_type* const)0x52U)
- #define TCNT0 (*(volatile u8* const)0x52U)
- #define TCCR0 (*(volatile u8* const)0x53U)
- #define OCR0 (*(volatile u8* const)0x5CU)
- #define S_TIFR ((TIFR_type* const)0x58U)
- #define TIFR (*(volatile u8* const)0x58U)
- #define S_TIMSK ((TIMSK_type* const)0x59U)
- #define TIMSK (*(volatile u8* const)0x59U)
- #define TMRx_WGMx0_MASK 0x0u
- #define TMRx_WGMx1_MASK 0x1u
- #define TMRx_WGMx2_MASK 0x2u
- #define TMRx_WGMx3_MASK 0x3u
- #define TMR_EXT0_PORT B
- #define TMR_EXT0_PIN GPIO_TMR_EXT0_IN
- #define TMR_EXT1_PORT B
- #define TMR_EXT1_PIN GPIO_TMR_EXT1_IN
- #define TMR_OC0_PORT B
- #define TMR_OC0_PIN GPIO_TMR_OC0
- #define TMR_OC2_PORT D
- #define TMR_OC2_PIN GPIO_TMR_OC2
- #define TMR_OSC1_PORT C
- #define TMR_OSC1_PIN GPIO_TMR_OSC1
- #define TMR_OSC2_PORT C
- #define TMR_OSC2_PIN GPIO_TMR_OSC2
- #define TMR_ICP1_PORT D
- #define TMR_ICP1_PIN GPIO_TMR_ICP1
- #define TMR_OC1A_PORT D
- #define TMR_OC1A_PIN GPIO_TMR_OC1A
- #define TMR_OC1B_PORT D
- #define TMR_OC1B_PIN GPIO_TMR_OC1B
- #define TMR_u8MAX (0x00FF + 1u)
- #define TMR_u9MAX (0x01FF + 1u)
- #define TMR_u10MAX (0x03FF + 1u)
- #define TMR_u16MAX (0xFFFF + 1u)

Macro Definition Documentation

#define ASSR (*(volatile u8* const)0x42U)

#define ICR1H (*(volatile u8* const)0x47U)

#define ICR1L (*(volatile u8* const)0x46U)

#define OCR0 (*(volatile u8* const)0x5CU)

#define OCR1AH (*(volatile u8* const)0x4BU)

#define OCR1AL (*(volatile u8* const)0x4AU)

#define OCR1BH (*(volatile u8* const)0x49U)

#define OCR1BL (*(volatile u8* const)0x48U)

#define OCR2 (*(volatile u8* const)0x43U)

#define S_SFIOR ((SFIOR_type* const)0x50U)

Special Function I/O Register

#define S_TIFR ((TIFR_type* const)0x58U)

Timer/Counter Interrupt Flag Register

#define S_TIMSK ((TIMSK_type* const)0x59U)

Timer/Counter Interrupt Mask Register

#define S_TMR0 ((GPTMR0_type* const)0x52U)

Timer/Counter 0 Register

#define S_TMR1 ((GPTMR1_type* const)0x46U)

Timer/Counter 1 Register

#define S_TMR2 ((GPTMR2_type* const)0x42U)

Timer/Counter 0 Register

```
#define SFIOR   (*(volatile u8* const)0x50U)

#define TCCR0   (*(volatile u8* const)0x53U)

#define TCCR1A   (*(volatile u8* const)0x4FU)

#define TCCR1B   (*(volatile u8* const)0x4EU)

#define TCCR2   (*(volatile u8* const)0x45U)

#define TCNT0   (*(volatile u8* const)0x52U)

#define TCNT1H   (*(volatile u8* const)0x4DU)

#define TCNT1L   (*(volatile u8* const)0x4CU)

#define TCNT2   (*(volatile u8* const)0x44U)

#define TIFR   (*(volatile u8* const)0x58U)

#define TIMSK   (*(volatile u8* const)0x59U)

#define TMR_EXT0_PIN   GPIO_TMR_EXT0_IN

#define TMR_EXT0_PORT   B

#define TMR_EXT1_PIN   GPIO_TMR_EXT1_IN

#define TMR_EXT1_PORT   B

#define TMR_ICP1_PIN   GPIO_TMR_ICP1

#define TMR_ICP1_PORT   D

#define TMR_OC0_PIN   GPIO_TMR_OC0

#define TMR_OC0_PORT   B

#define TMR_OC1A_PIN   GPIO_TMR_OC1A

#define TMR_OC1A_PORT   D

#define TMR_OC1B_PIN   GPIO_TMR_OC1B

#define TMR_OC1B_PORT   D

#define TMR_OC2_PIN   GPIO_TMR_OC2

#define TMR_OC2_PORT   D

#define TMR_OSC1_PIN   GPIO_TMR_OSC1
```

```
#define TMR_OSC1_PORT   C

#define TMR_OSC2_PIN   GPIO_TMR_OSC2

#define TMR_OSC2_PORT   C

#define TMR_u10MAX   (0x03FF + 1u)

#define TMR_u16MAX   (0xFFFF + 1u)

#define TMR_u8MAX   (0x00FF + 1u)

#define TMR_u9MAX   (0x01FF + 1u)

#define TMRx_WGMx0_MASK   0x0u

#define TMRx_WGMx1_MASK   0x1u

#define TMRx_WGMx2_MASK   0x2u

#define TMRx_WGMx3_MASK   0x3u
```

TMR_priv.h

Go to the documentation of this file.

```
1 /*
2 /* ********************** FILE DEFINITION SECTION ************************** */
3 /* ************************************************************************** */
4 /* File Name    : TMR_priv.h                                                  */
5 /* Author       : MAAM                                                        */
6 /* Version      : v01.2                                                       */
7 /* date         : Apr 5, 2023                                                 */
8 /* ************************************************************************** */
9 /* ************************* HEADER FILES INCLUDES ************************  */
10 /* ************************************************************************** */
11
12 #ifndef TMR_PRIV_H_
13 #define TMR_PRIV_H_
14
15 /* ************************************************************************** */
16 /* ********************** TYPE_DEF/STRUCT/ENUM SECTION ********************** */
17 /* ************************************************************************** */
18
21 typedef union{
22     u8 u_Reg;
23     struct {
24         __I  u8 m_TCR2UB : 1;
25         __I  u8 m_OCR2UB : 1;
26         __I  u8 m_TCN2UB : 1;
27         __IO u8 m_AS2    : 1;
28         __IO u8          : 4;
29     }sBits;
30 }ASSR_type;
31
32 /*****************************************************************************/
33
36 typedef union{
37     u8 u_Reg;
38     struct {
39         __IO u8 m_CSx  : 3;
40         __IO u8 m_WGMx1: 1;
41         __IO u8 m_COMx : 2;
42         __IO u8 m_WGMx0: 1;
43         __IO u8 m_FOCx : 1;
44     }sBits;
45 }TCCRx_type;
46
47 /*****************************************************************************/
48
51 typedef struct{
52     __IO ASSR_type  m_ASSR;
53     __IO u8         m_OCR2;
54     __IO u8         m_TCNT2;
55     __IO TCCRx_type m_TCCR2;
56 }GPTMR2_type;
57
58 /*****************************************************************************/
59
62 typedef struct{
63     __IO u8         m_TCNT0;
64     __IO TCCRx_type m_TCCR0;
65     __I  u8         REVERSE[8];
66     __IO u8         m_OCR0;
67 }GPTMR0_type;
68
69 /*****************************************************************************/
70
73 typedef union{
74     u8 u_Reg;
75     struct {
76         __IO u8 m_CS1  : 3;
77         __IO u8 m_WGM12: 1;
78         __IO u8 m_WGM13: 1;
79         __IO u8        : 1;
80         __IO u8 m_ICES1: 1;
81         __IO u8 m_ICNC1: 1;
82     }sBits;
```

```
83 }TCCR1B_type;
84
85 /*****************************************************************************/
86
89 typedef union{
90     u8 u_Reg;
91     struct {
92          __IO u8 m_WGM10: 1;
93          __IO u8 m_WGM11: 1;
94          __IO u8 m_FOC1B: 1;
95          __IO u8 m_FOC1A: 1;
96          __IO u8 m_COM1B: 2;
97          __IO u8 m_COM1A: 2;
98     }sBits;
99 }TCCR1A_type;
100
101 /*****************************************************************************/
102
105 typedef union{
106     u8 u_Reg;
107     struct {
108          __IO u8 m_B0 : 1;
109          __IO u8 m_B1 : 1;
110          __IO u8 m_B2 : 1;
111          __IO u8 m_B3 : 1;
112          __IO u8 m_B4 : 1;
113          __IO u8 m_B5 : 1;
114          __IO u8 m_B6 : 1;
115          __IO u8 m_B7 : 1;
116     }sBits;
117 }BYTE_type; // byte bit exchange
118
119 /*****************************************************************************/
120
123 typedef union{
124     u16 u16Reg;
125     struct {
126         BYTE_type m_u8Low;
127         BYTE_type m_u8High;
128     }sBytes;
129     struct {
130          __IO u8 m_B0 : 1;
131          __IO u8 m_B1 : 1;
132          __IO u8 m_B2 : 1;
133          __IO u8 m_B3 : 1;
134          __IO u8 m_B4 : 1;
135          __IO u8 m_B5 : 1;
136          __IO u8 m_B6 : 1;
137          __IO u8 m_B7 : 1;
138          __IO u8 m_B8 : 1;
139          __IO u8 m_B9 : 1;
140          __IO u8 m_B10: 1;
141          __IO u8 m_B11: 1;
142          __IO u8 m_B12: 1;
143          __IO u8 m_B13: 1;
144          __IO u8 m_B14: 1;
145          __IO u8 m_B15: 1;
146     }sBits;
147 }Word_type;
148
149 /*****************************************************************************/
150
153 typedef struct{
154      __IO Word_type   m_ICR1;
155      __IO Word_type   m_OCR1B;
156      __IO Word_type   m_OCR1A;
157      __IO Word_type   m_TCNT1;
158      __IO TCCR1B_type m_TCCR1B;
159      __IO TCCR1A_type m_TCCR1A;
160 }GPTMR1_type;
161
162 /*****************************************************************************/
163
166 typedef union{
167     u8 u_Reg;
168     struct {
169          __IO u8 m_PSR10: 1;
```

```
170             __IO u8 m_PSR2 : 1;
171             __IO u8 m_PUD  : 1;
172             __IO u8 m_ACME : 1;
173             __IO u8        : 1;
174             __IO u8 m_ADTS : 3;
175         }sBits;
176 }SFIOR_type;
177
178 /*****************************************************************************/
179
182 typedef union{
183     u8 u_Reg;
184     struct {
185             __IO u8 m_TOV0 : 1;
186             __IO u8 m_OCF0 : 1;
187             __IO u8 m_TOV1 : 1;
188             __IO u8 m_OCF1B: 1;
189             __IO u8 m_OCF1A: 1;
190             __IO u8 m_ICF1 : 1;
191             __IO u8 m_TOV2 : 1;
192             __IO u8 m_OCF2 : 1;
193     }sBits;
194 }TIFR_type;
195
196 /*****************************************************************************/
197
200 typedef union{
201     u8 u_Reg;
202     struct {
203             __IO u8 m_TOIE0 : 1;
204             __IO u8 m_OCIE0 : 1;
205             __IO u8 m_TOIE1 : 1;
206             __IO u8 m_OCIE1B: 1;
207             __IO u8 m_OCIE1A: 1;
208             __IO u8 m_TICIE1: 1;
209             __IO u8 m_TOIE2 : 1;
210             __IO u8 m_OCIE2 : 1;
211     }sBits;
212 }TIMSK_type;    // Timer/Counter Interrupt Mask Register
213
214 /* ************************************************************************** */
215 /* ************************* MACRO/DEFINE SECTION ************************** */
216 /* ************************************************************************** */
217
219 #define S_TMR2          ((GPTMR2_type* const)0x42U)
220 #define ASSR            (*(volatile u8* const)0x42U)
221 #define OCR2            (*(volatile u8* const)0x43U)
222 #define TCNT2           (*(volatile u8* const)0x44U)
223 #define TCCR2           (*(volatile u8* const)0x45U)
224
226 #define S_TMR1          ((GPTMR1_type* const)0x46U)
227 #define ICR1L           (*(volatile u8* const)0x46U)
228 #define ICR1H           (*(volatile u8* const)0x47U)
229 #define OCR1BL          (*(volatile u8* const)0x48U)
230 #define OCR1BH          (*(volatile u8* const)0x49U)
231 #define OCR1AL          (*(volatile u8* const)0x4AU)
232 #define OCR1AH          (*(volatile u8* const)0x4BU)
233 #define TCNT1L          (*(volatile u8* const)0x4CU)
234 #define TCNT1H          (*(volatile u8* const)0x4DU)
235 #define TCCR1B          (*(volatile u8* const)0x4EU)
236 #define TCCR1A          (*(volatile u8* const)0x4FU)
237
239 #define S_SFIOR         ((SFIOR_type* const)0x50U)
240 #define SFIOR           (*(volatile u8* const)0x50U)
241
243 #define S_TMR0          ((GPTMR0_type* const)0x52U)
244 #define TCNT0           (*(volatile u8* const)0x52U)
245 #define TCCR0           (*(volatile u8* const)0x53U)
246
247 #define OCR0            (*(volatile u8* const)0x5CU)
248
250 #define S_TIFR          ((TIFR_type* const)0x58U)
251 #define TIFR            (*(volatile u8* const)0x58U)
252
254 #define S_TIMSK         ((TIMSK_type* const)0x59U)
255 #define TIMSK           (*(volatile u8* const)0x59U)
256
```

```
257 /* ************************************************************************** */
258
259 #define TMRx_WGMx0_MASK     0x0u
260 #define TMRx_WGMx1_MASK     0x1u
261 #define TMRx_WGMx2_MASK     0x2u
262 #define TMRx_WGMx3_MASK     0x3u
263
264 #define TMR_EXT0_PORT       B
265 #define TMR_EXT0_PIN        GPIO_TMR_EXT0_IN
266 #define TMR_EXT1_PORT       B
267 #define TMR_EXT1_PIN        GPIO_TMR_EXT1_IN
268
269 #define TMR_OC0_PORT        B
270 #define TMR_OC0_PIN         GPIO_TMR_OC0
271
272 #define TMR_OC2_PORT        D
273 #define TMR_OC2_PIN         GPIO_TMR_OC2
274
275 #define TMR_OSC1_PORT       C
276 #define TMR_OSC1_PIN        GPIO_TMR_OSC1
277 #define TMR_OSC2_PORT       C
278 #define TMR_OSC2_PIN        GPIO_TMR_OSC2
279
280 #define TMR_ICP1_PORT       D
281 #define TMR_ICP1_PIN        GPIO_TMR_ICP1
282 #define TMR_OC1A_PORT       D
283 #define TMR_OC1A_PIN        GPIO_TMR_OC1A
284 #define TMR_OC1B_PORT       D
285 #define TMR_OC1B_PIN        GPIO_TMR_OC1B
286
287 #define TMR_u8MAX           (0x00FF + 1u)
288 #define TMR_u9MAX           (0x01FF + 1u)
289 #define TMR_u10MAX          (0x03FF + 1u)
290 #define TMR_u16MAX          (0xFFFF + 1u)
291
292 /* ************************************************************************** */
293 /* ***************************** CONST SECTION ****************************** */
294 /* ************************************************************************** */
295
296 /* ************************************************************************** */
297 /* ***************************** VARIABLE SECTION *************************** */
298 /* ************************************************************************** */
299
300 /* ************************************************************************** */
301 /* ***************************** FUNCTION SECTION *************************** */
302 /* ************************************************************************** */
303
304
305 #endif /* TMR_PRIV_H_ */
306 /***************************** E N D (TMR_priv.h) ****************************/
```