SWC_I2C

Version v1.0
9/2/2023 5:21:00 PM

Table of Contents

Data Structure Index

Data Structures

Here are the data structures with brief descriptions:

File Index

File List

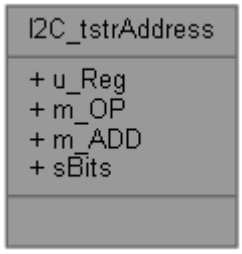
Here is a list of all files with brief descriptions:

Data Structure Documentation

I2C_tstrAddress Union Reference

: I2C Address

`#include <I2C_priv.h>`
Collaboration diagram for I2C_tstrAddress:



Data Fields

- u8 u_Reg
- struct {
- __IO u8 m_OP: 1
- __IO u8 m_ADD: 7
- } sBits

Detailed Description

: I2C Address

Type : Union **Unit** : None

Field Documentation

__IO u8 m_ADD

Slave Address

__IO u8 m_OP

Operation R/W

struct { ... } sBits

u8 u_Reg

Byte

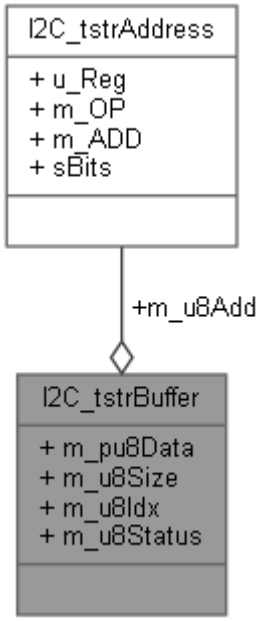
The documentation for this union was generated from the following file:

I2C_priv.h

I2C_tstrBuffer Struct Reference

: I2C TX/RX Buffer

`#include <I2C_priv.h>`
Collaboration diagram for I2C_tstrBuffer:



Data Fields

- I2C_tstrAddress m_u8Add
- pu8 m_pu8Data
- u8 m_u8Size
- u8 m_u8Idx
- u8 m_u8Status

Detailed Description

: I2C TX/RX Buffer

Type : Struct **Unit** : None

Field Documentation

pu8 m_pu8Data

Data Pointer

I2C_tstrAddress m_u8Add

Slave Address

u8 m_u8Idx

Index of Data

u8 m_u8Size

Size of Data Bytes

u8 m_u8Status

Current Status

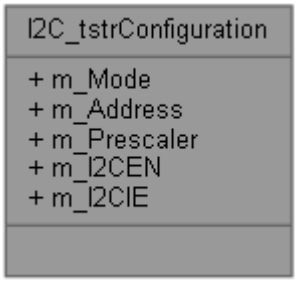
The documentation for this struct was generated from the following file:

I2C_priv.h

I2C_tstrConfiguration Struct Reference

`#include <I2C_int.h>`

Collaboration diagram for I2C_tstrConfiguration:



Data Fields

- I2C_tenuMode m_Mode
- u8 m_Address
- I2C_tenuPrescaler m_Prescaler
- LBTY_tenuFlagStatus m_I2CEN
- LBTY_tenuFlagStatus m_I2CIE

Field Documentation

u8 m_Address

LBTY_tenuFlagStatus m_I2CEN

LBTY_tenuFlagStatus m_I2CIE

I2C_tenuMode m_Mode

I2C_tenuPrescaler m_Prescaler

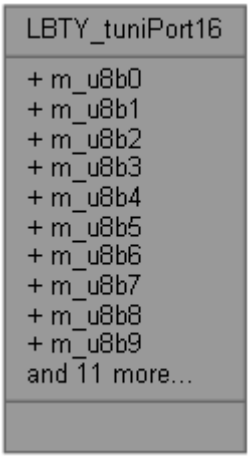
The documentation for this struct was generated from the following file:

I2C_int.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

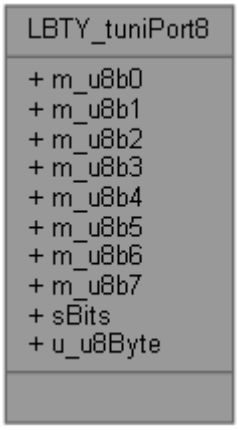
The documentation for this union was generated from the following file:

- 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

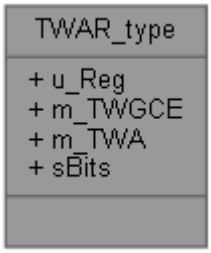
The documentation for this union was generated from the following file:

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

TWAR_type Union Reference

: Type define of Union bit field of "TWI Address Register"

`#include <I2C_priv.h>`
Collaboration diagram for TWAR_type:



Data Fields

- u8 u_Reg
- struct {
- __IO u8 m_TWGCE: 1
- __IO u8 m_TWA: 7
- } sBits

Detailed Description

: Type define of Union bit field of "TWI Address Register"

Type : Union **Unit** : None

Field Documentation

__IO u8 m_TWA

TWI Slave Address

__IO u8 m_TWGCE

TWI General Call Recognition Enable Bit

struct { ... } sBits

u8 u_Reg

Byte

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

I2C_priv.h

TWCR_type Union Reference

: Type define of Union bit field of "TWI Control Register"

`#include <I2C_priv.h>`
Collaboration diagram for TWCR_type:

Data Fields

- u8 u_Reg
- struct {
- __IO u8 m_TWIE: 1
- __IO u8: 1
- __IO u8 m_TWEN: 1
- __I u8 m_TWWC: 1
- __IO u8 m_TWSTO: 1
- __IO u8 m_TWSTA: 1
- __IO u8 m_TWEA: 1
- __IO u8 m_TWINT: 1
- } sBits

Detailed Description

: Type define of Union bit field of "TWI Control Register"

Type : Union **Unit** : None

Field Documentation

__IO u8 m_TWEA

TWI Enable Acknowledge Bit

__IO u8 m_TWEN

TWI Enable Bit

__IO u8 **m_TWIE**

TWI Interrupt Enable

__IO u8 **m_TWINT**

TWI Interrupt Flag "0 -> set" "1 -> reset"

__IO u8 **m_TWSTA**

TWI START Condition Bit

__IO u8 **m_TWSTO**

TWI STOP Condition Bit

__I u8 **m_TWWC**

TWI Write Collision Flag

struct { ... } sBits

__IO u8

Reversed

u8 **u_Reg**

Byte

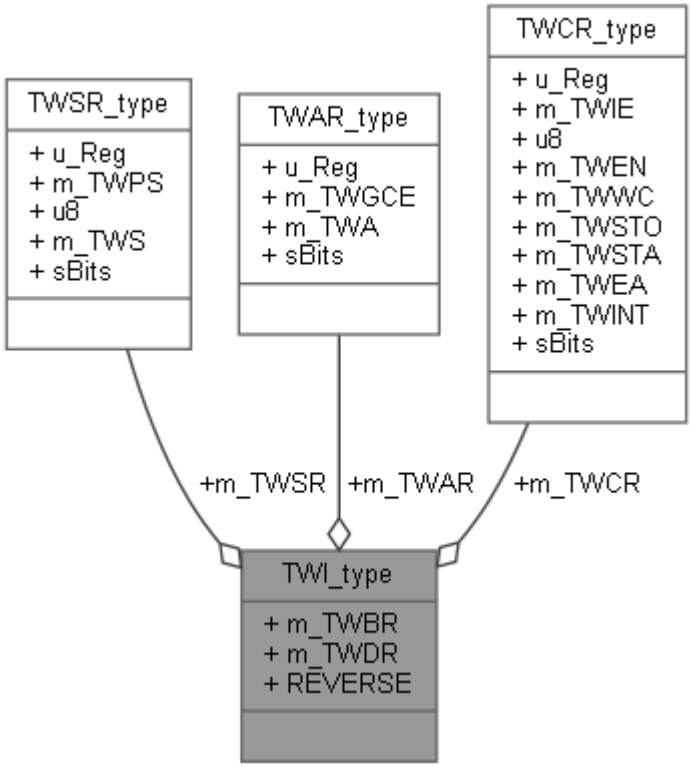
The documentation for this union was generated from the following file:

I2C_priv.h

TWI_type Struct Reference

: I2C "TWI" Registers

`#include <I2C_priv.h>`
Collaboration diagram for TWI_type:



Data Fields

- __IO u8 m_TWBR
- __IO TWSR_type m_TWSR
- __IO TWAR_type m_TWAR
- __IO u8 m_TWDR
- __I u8 REVERSE [50]
- __IO TWCR_type m_TWCR

Detailed Description

: I2C "TWI" Registers

Type : Struct **Unit** : None

Field Documentation

__IO TWAR_type m_TWAR

TWI Address Register

__IO u8 m_TWBR

TWI Bit Rate Register

__IO TWCR_type m_TWCR

TWI Control Register

__IO u8 m_TWDR

TWI Data Register

__IO TWSR_type m_TWSR

TWI Status Register

__I u8 REVERSE[50]

Reversed

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

I2C_priv.h

TWSR_type Union Reference

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

`#include <I2C_priv.h>`
Collaboration diagram for TWSR_type:

Data Fields

- u8 u_Reg
- struct {
- __IO u8 m_TWPS: 2
- __IO u8: 1
- __I u8 m_TWS: 5
- } sBits

Detailed Description

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

Type : Union **Unit** : None

Field Documentation

__IO u8 m_TWPS

TWI Prescaler Bits

__I u8 m_TWS

TWI Status

struct { ... } sBits

__IO u8

Reversed

u8 u_Reg

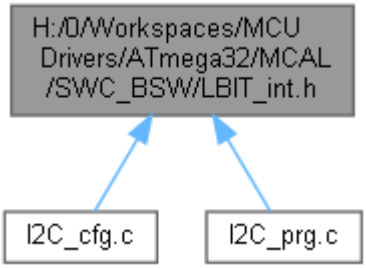
Byte

The documentation for this union was generated from the following file:
I2C_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.

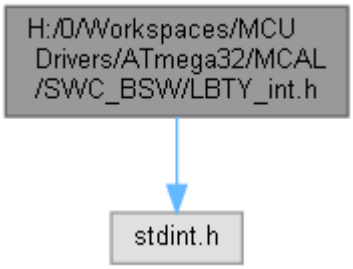
```
1 /*
**************************************************************************** */
2 /* ********************** FILE DEFINITION SECTION ************************** */
3 /* ************************************************************************** */
4 /* File Name   : LBIT_int.h                                                   */
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) ******************************/
```

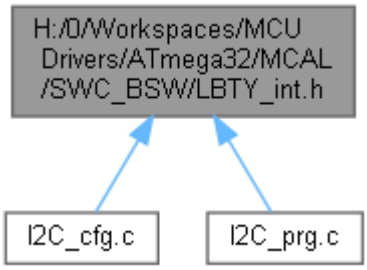
H:/0/Workspaces/MCU Drivers/ATmega32/MCAL/SWC_BSW/LBTY_int.h File Reference

```
#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.

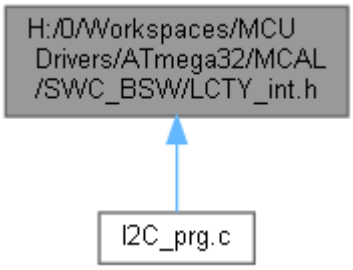
```
1 /*
********************************************************************************** */
2 /* ********************** FILE DEFINITION SECTION ************************** */
3 /* ********************************************************************************** */
4 /* File Name   : LBTY_int.h                                                       */
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

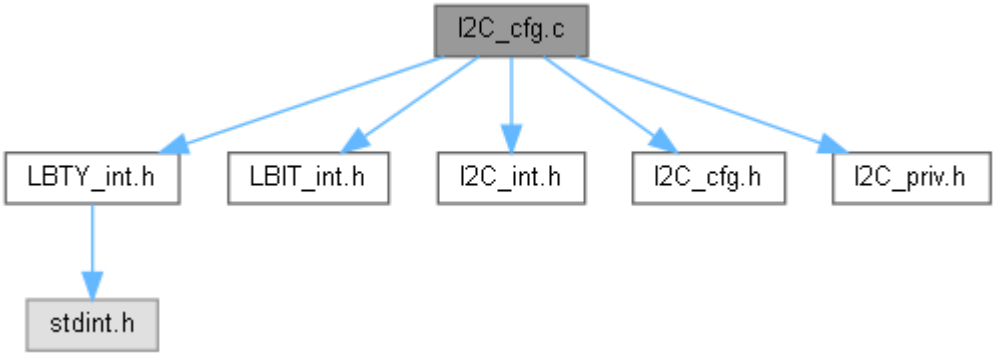
Go to the documentation of this file.

```
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) ****************************/
```

I2C_cfg.c File Reference

```
#include "LBTY_int.h"
#include "LBIT_int.h"
#include "I2C_int.h"
#include "I2C_cfg.h"
#include "I2C_priv.h"
```

Include dependency graph for I2C_cfg.c:



Functions

- void _vidStart (void)
- void _vidStop (void)
- void _vidRelease (void)
- void _vidError (void)
- void _vidCMD (void)
- void _vidReplay (void)
- void _vidWR (void)
- void _vidRD (void)
- void _vidTX (u8 u8State)
- void _vidRX (u8 u8State)
- void I2C_vidStep (void)

Variables

- volatile I2C_tstrBuffer strBuffer_GLB

Function Documentation

void _vidCMD (void)

```
71                      {
72      S_TWI->m_TWDR = strBuffer_GLB.m_u8Add.u_Reg;
73      if(I2C_MODE == I2C_Master){
74          I2C_NACK();
75      }else if(I2C_MODE == I2C_Slave){
76          I2C_ACK();
77      }
78      strBuffer_GLB.m_u8Status = I2C_ADDRESS;
79  }
```

Here is the caller graph for this function:

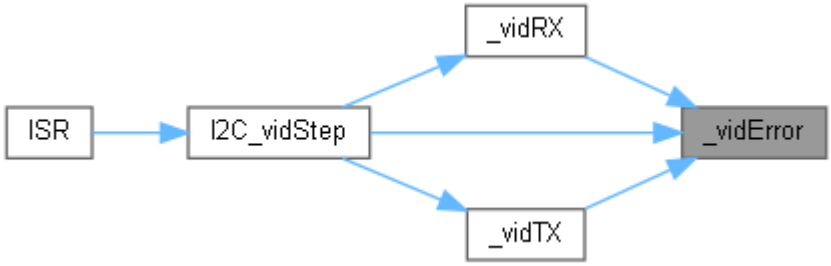
void _vidError (void)

```
65                      {
```

```
66      I2C_STOP();
67      //S_TWI->m_TWCR.sBits.m_TWEN = LBTY_RESET;
68      strBuffer_GLB.m_u8Status    = I2C_ERROR;
69      _vidRelease();
70  }
```

Here is the call graph for this function:

Here is the caller graph for this function:



void _vidRD (void)

```
100                 {
101     if(strBuffer_GLB.m_u8Idx < strBuffer_GLB.m_u8Size){
102         strBuffer_GLB.m_pu8Data[strBuffer_GLB.m_u8Idx++] = S_TWI->m_TWDR;
103         _vidReplay();
104         strBuffer_GLB.m_u8Status = I2C_DATA;
105     }else{
106         _vidStop();
107     }
108 }
```

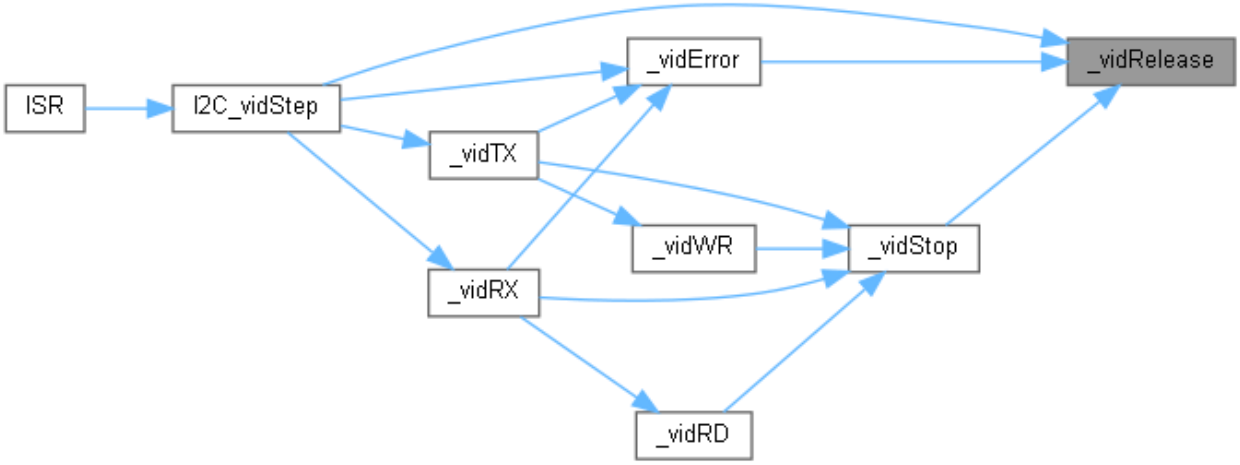
Here is the call graph for this function:

Here is the caller graph for this function:

void _vidRelease (void)

```
61                      {
62      I2C_DEF(LBTY_SET, TWEA);
63      strBuffer_GLB.m_u8Status = I2C_IDLE;
64  }
```

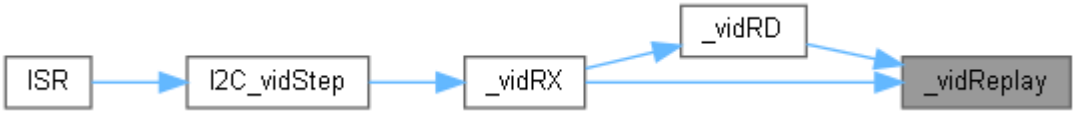
Here is the caller graph for this function:



void _vidReplay (void)

```
80                          {
81      if(strBuffer_GLB.m_u8Idx < (strBuffer_GLB.m_u8Size /*- 1*/)){
82          I2C_ACK();
83      }else{
84          I2C_NACK();
85      }
86  }
```

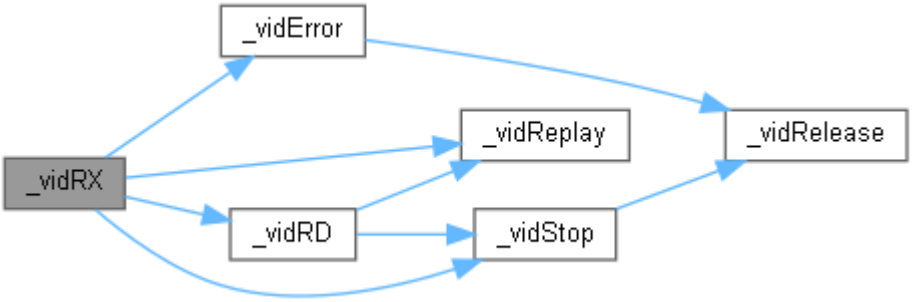
Here is the caller graph for this function:



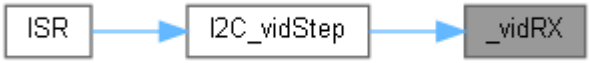
void _vidRX (u8 *u8State*)

```
127                             {
128     switch(u8State){
129         case I2C_MR_SLA_ACK:
130         case I2C_SR_SLA_ACK:
131         case I2C_SR_GCALL_ACK:
132             _vidReplay();
133             break;
134         case I2C_MR_DATA_ACK:
135         case I2C_SR_DATA_ACK:
136         case I2C_SR_GCALL_Data_ACK:
137             _vidRD();
138             break;
139         case I2C_MR_DATA_NACK:
140         case I2C_SR_DATA_NACK:
141         case I2C_SR_GCALL_Data_NACK:
142         case I2C_ST_LAST_Data:
143             _vidRD();
144             _vidStop();
145             break;
146         case I2C_MR_SLA_NACK:
147         default:
148             _vidError();
149     }
150 }
```

Here is the call graph for this function:



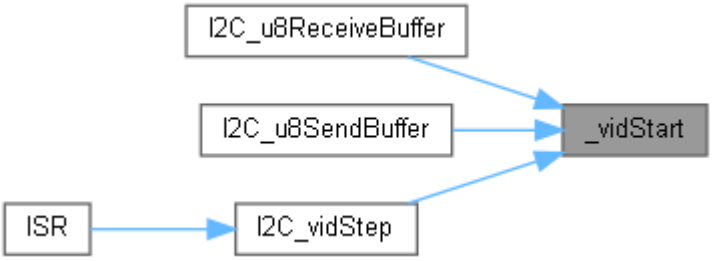
Here is the caller graph for this function:



void _vidStart (void)

```
42                          {
43      if(I2C_MODE == I2C_Master){
44          I2C_START();
45      }else if(I2C_MODE == I2C_Slave){
46          I2C_ACK();
47      }
48
49      strBuffer_GLB.m_u8Status = I2C_START;
50  }
```

Here is the caller graph for this function:

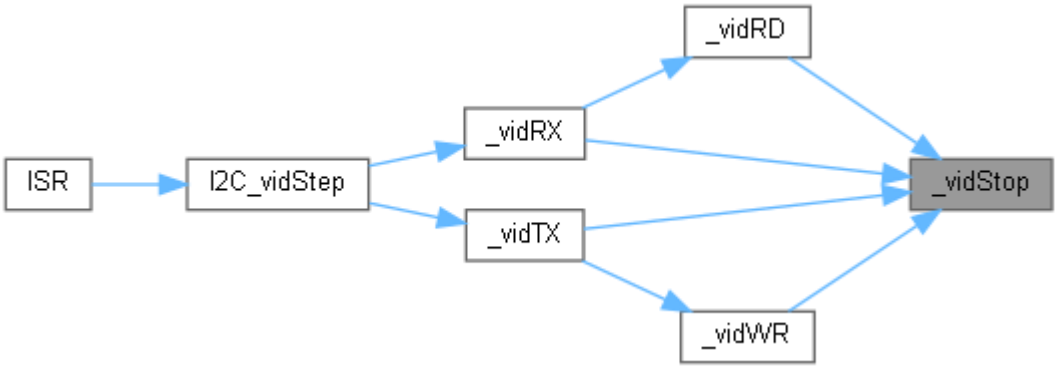


void _vidStop (void)

```
51                      {
52      I2C_STOP();
53      while(S_TWI->m_TWCR.sBits.m_TWSTO);
54      S_TWI->m_TWCR.sBits.m_TWSTO = LBTY_SET;
55      strBuffer_GLB.m_u8Status = I2C_STOP;
56
57 //   if(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK) == I2C_Stop){
58          _vidRelease();
59 //   }
60 }
```

Here is the call graph for this function:

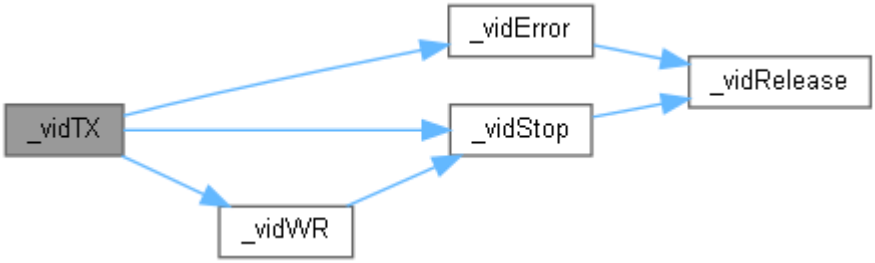
Here is the caller graph for this function:



void _vidTX (u8 *u8State*)

```
109                         {
110     switch(u8State){
111         case I2C_MT_SLA_ACK:
112         case I2C_MT_DATA_ACK:
113         case I2C_ST_SLA_ACK:
114         case I2C_ST_DATA_ACK:
115         case I2C_MT_SLA_NACK:
116             _vidWR();
117             break;
118         case I2C_MT_DATA_NACK:
119         case I2C_ST_DATA_NACK:
120         case I2C_ST_LAST_Data:
121             _vidStop();
122             break;
123         default:
124             _vidError();
125     }
126 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

void _vidWR (void)

```
87                  {
88      if(strBuffer_GLB.m_u8Idx < strBuffer_GLB.m_u8Size){
89          S_TWI->m_TWDR = strBuffer_GLB.m_pu8Data[strBuffer_GLB.m_u8Idx++];
90          if(I2C_MODE == I2C_Master){
91              I2C_NACK();
92          }else if(I2C_MODE == I2C_Slave){
93              I2C_ACK();
94          }
95          strBuffer_GLB.m_u8Status = I2C_DATA;
96      }else{
97          _vidStop();
98      }
99  }
```

Here is the call graph for this function:

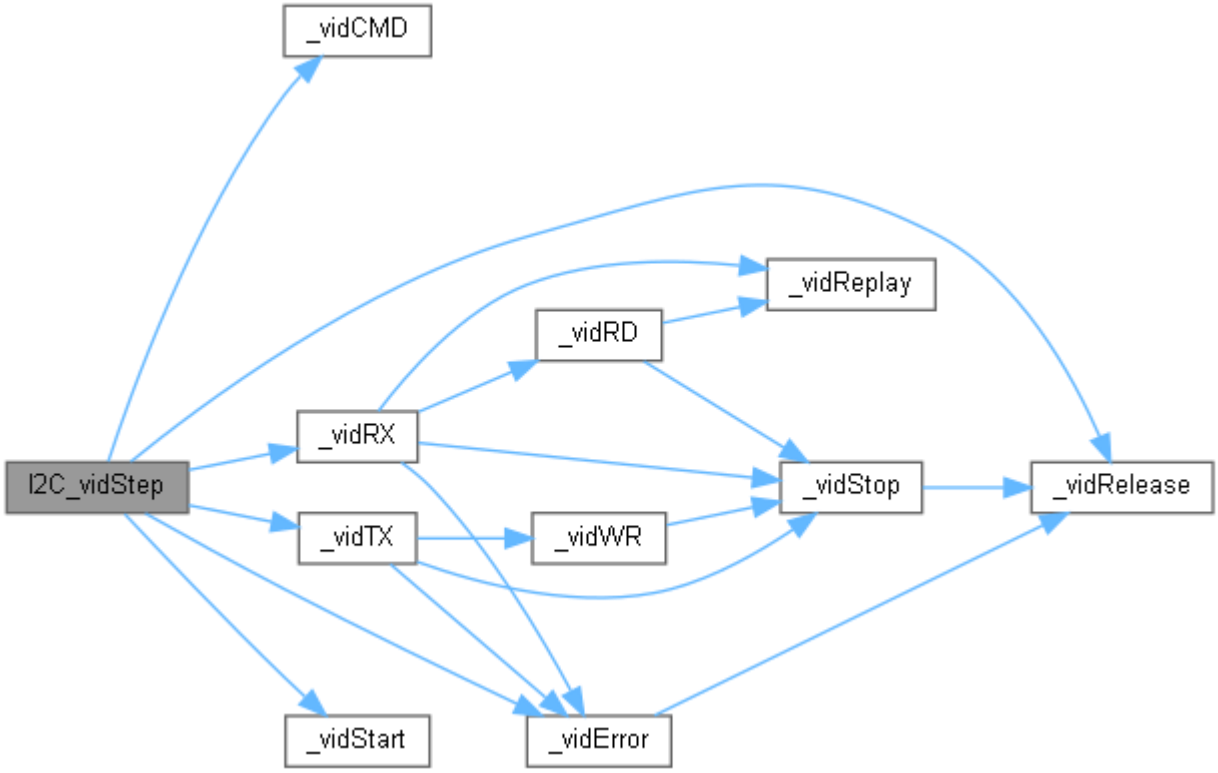
Here is the caller graph for this function:

void I2C_vidStep (void)

```
152                     {
153      volatile u8 u8State = GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK);
154      switch(u8State) {
155          case I2C_Start:                  // Start condition transmitted
156          case I2C_RepeatStart:            // Repeated start condition transmitted
157
158              _vidCMD();
159              break;
160  /* ************************************************************************ */
161          case I2C_MT_SLA_ACK:             // SLA+W transmitted, ACK received = Slave
receiver ACKed address
162          case I2C_MT_DATA_ACK:            // Data transmitted, ACK received = Slave
receiver ACKed data
163          case I2C_ST_SLA_ACK:             // SLA+R received, ACK returned =
Addressed, returned ACK
164          case I2C_ST_DATA_ACK:            // Data transmitted, ACK received
165
166          case I2C_MT_SLA_NACK:            // SLA+W transmitted, NACK received =
Slave receiver with transmitted address doesn't exists?
167          case I2C_MT_DATA_NACK:           // Data transmitted, NACK received
168          case I2C_ST_DATA_NACK:           // Data transmitted, NACK received =
Received NACK, so we are done
169
170              _vidTX(u8State);
171              break;
172  /* ************************************************************************ */
173          case I2C_MR_SLA_ACK:             // SLA+R transmitted, ACK received = Slave
transmitter ACKed address
174          case I2C_MR_DATA_ACK:            // Data received, ACK returned
175          case I2C_SR_SLA_ACK:             // SLA+W received, ACK returned =
Addressed, returned ACK
176          case I2C_SR_DATA_ACK:            // Data received, ACK returned
177          case I2C_SR_GCALL_ACK:           // General call received, ACK returned =
Addressed generally, returned ACK
```

```
178         case I2C_SR_GCALL_Data_ACK:     // General call data received, ACK
returned
179
180         case I2C_MR_SLA_NACK:           // SLA+R transmitted, NACK received =
Slave transmitter with transmitted address doesn't exists?
181         case I2C_MR_DATA_NACK:          // Data received, NACK returned
182         case I2C_SR_DATA_NACK:          // Data received, NACK returned
183         case I2C_SR_GCALL_Data_NACK:    // General call data received, NACK
returned
184
185         case I2C_ST_LAST_Data:          // Last data byte transmitted, ACK
received = Received ACK, but we are done already!
186
187             _vidRX(u8State);
188             break;
189 /* ************************************************************************** */
190         case I2C_MT_ARB_LOST:           // Arbitration lost in SLA+W or data
191 //      case I2C_MR_ARB_LOST:           // Arbitration lost in SLA+R or NACK
192         case I2C_ST_ARB_LOST:           // Arbitration lost in SLA+RW, SLA+R
received, ACK returned
193         case I2C_SR_SLA_ARB_LOST:       // Arbitration lost in SLA+RW, SLA+W
received, ACK returned
194         case I2C_SR_GCALL_ARB_LOST:     // Arbitration lost in SLA+RW, general
call received, ACK returned
195
196             _vidStart();
197             break;
198 /* ************************************************************************** */
199         case I2C_Stop:                  // Stop or repeated start condition
received while selected
200
201             _vidRelease();
202             break;
203 /* ************************************************************************** */
204         case I2C_NoInfo:                // No state information available
205         case I2C_Bus_Error:             // Bus error; Illegal START or STOP
condition
206         default:
207
208             _vidError();
209     }
210 }
```

Here is the call graph for this function:



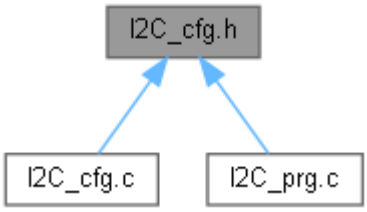
Here is the caller graph for this function:

Variable Documentation

volatile [I2C_tstrBuffer](#) strBuffer_GLB `[extern]`

I2C_cfg.h File Reference

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



Macros

- #define I2C_MODE I2C_Master
- #define I2C_SLAVE_ADDRESS 0x7A
- #define I2C_SCL_FREQ 400000u
- #define I2C_CLOCK_PRESCALER I2C_Prescaler_1
- #define I2C_INIT_STATE LBTY_SET
- #define I2C_INT_STATE LBTY_RESET
- #define I2C_SLAVE_WAIT 10

Macro Definition Documentation

#define I2C_CLOCK_PRESCALER I2C_Prescaler_1

#define I2C_INIT_STATE LBTY_SET

#define I2C_INT_STATE LBTY_RESET

#define I2C_MODE I2C_Master

#define I2C_SCL_FREQ 400000u

#define I2C_SLAVE_ADDRESS 0x7A

#define I2C_SLAVE_WAIT 10

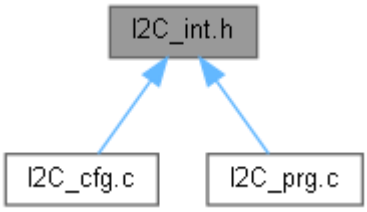
I2C_cfg.h

Go to the documentation of this file.

```
1 /*
2 /* ********************** FILE DEFINITION SECTION ************************** */
3 /* ************************************************************************** */
4 /* File Name   : I2C_cfg.h                                                    */
5 /* Author      : MAAM                                                         */
6 /* Version     : v01.2                                                        */
7 /* date        : Apr 13, 2023                                                 */
8 /* ************************************************************************** */
9 /* ************************* HEADER FILES INCLUDES ************************  */
10 /* ************************************************************************** */
11
12 #ifndef I2C_CFG_H_
13 #define I2C_CFG_H_
14
15 /* ************************************************************************** */
16 /* ********************** TYPE_DEF/STRUCT/ENUM SECTION ********************** */
17 /* ************************************************************************** */
18
19 /* ************************************************************************** */
20 /* ************************** MACRO/DEFINE SECTION ************************** */
21 /* ************************************************************************** */
22
23 #define I2C_MODE                    I2C_Master
24
25 #define I2C_SLAVE_ADDRESS           0x7A
26 #define I2C_SCL_FREQ                400000u
27 #define I2C_CLOCK_PRESCALER         I2C_Prescaler_1
28
29 #define I2C_INIT_STATE              LBTY_SET
30 #define I2C_INT_STATE               LBTY_RESET
31
32 #define I2C_SLAVE_WAIT              10  //5
33
34 /* ************************************************************************** */
35 /* ***************************** CONST SECTION ****************************** */
36 /* ************************************************************************** */
37
38 /* ************************************************************************** */
39 /* ***************************** VARIABLE SECTION *************************** */
40 /* ************************************************************************** */
41
42 /* ************************************************************************** */
43 /* ***************************** FUNCTION SECTION *************************** */
44 /* ************************************************************************** */
45
46
47 #endif /* I2C_CFG_H_ */
48 /*************************** E N D (I2C_cfg.h) ****************************/
```

I2C_int.h File Reference

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



Data Structures

struct I2C_tstrConfigurationEnumerations

- enum USART_tstrRW { I2C_WRITE = (u8)0u, I2C_READ }
- enum I2C_tenuMode { I2C_Master = (u8)0u, I2C_Slave }
- enum I2C_tenuPrescaler { I2C_Prescaler_1 = (u8)0u, I2C_Prescaler_4, I2C_Prescaler_16, I2C_Prescaler_64 }
- enum I2C_tenuStatusCode { I2C_Bus_Error = (u8)0x00, I2C_Start = (u8)0x08, I2C_RepeatStart = (u8)0x10, I2C_MT_SLA_ACK = (u8)0x18, I2C_MT_SLA_NACK = (u8)0x20, I2C_MT_DATA_ACK = (u8)0x28, I2C_MT_DATA_NACK = (u8)0x30, I2C_MT_ARB_LOST = (u8)0x38, I2C_MR_ARB_LOST = (u8)0x38, I2C_MR_SLA_ACK = (u8)0x40, I2C_MR_SLA_NACK = (u8)0x48, I2C_MR_DATA_ACK = (u8)0x50, I2C_MR_DATA_NACK = (u8)0x58, I2C_SR_SLA_ACK = (u8)0x60, I2C_SR_SLA_ARB_LOST = (u8)0x68, I2C_SR_GCALL_ACK = (u8)0x70, I2C_SR_GCALL_ARB_LOST = (u8)0x78, I2C_SR_DATA_ACK = (u8)0x80, I2C_SR_DATA_NACK = (u8)0x88, I2C_SR_GCALL_Data_ACK = (u8)0x90, I2C_SR_GCALL_Data_NACK = (u8)0x98, I2C_Stop = (u8)0xA0, I2C_ST_SLA_ACK = (u8)0xA8, I2C_ST_ARB_LOST = (u8)0xB0, I2C_ST_DATA_ACK = (u8)0xB8, I2C_ST_DATA_NACK = (u8)0xC0, I2C_ST_LAST_Data = (u8)0xC8, I2C_NoInfo = (u8)0xF8 }

Functions

- void I2C_vidSetConfig (I2C_tstrConfiguration const *const pstrConfig)
- void I2C_vidResetConfig (I2C_tstrConfiguration *const pstrConfig)
- void I2C_vidInit (void)
- void I2C_vidEnable (void)
- void I2C_vidDisable (void)
- u8 I2C_u8GetStatus (void)
- u8 I2C_u8GetINTF (void)
- LBTY_tenuErrorStatus I2C_u8SetSTART (void)
- LBTY_tenuErrorStatus I2C_u8SetRepeatSTART (void)
- LBTY_tenuErrorStatus I2C_u8SetAddress (u8 u8Address, u8 Operation)
- LBTY_tenuErrorStatus I2C_u8GetRequest (u8 *Operation)
- LBTY_tenuErrorStatus I2C_u8SetData (u8 u8Data)
- LBTY_tenuErrorStatus I2C_u8GetData (u8 *pu8Data, u8 u8ACK)
- LBTY_tenuErrorStatus I2C_u8SetSTOP (void)
- void I2C_u8SetChar (u8 u8char, u8 u8Address)
- void I2C_u8GetChar (u8 *pu8char, u8 u8Address)
- void I2C_SlaveListen (u8 *pu8char, u8 u8Address)
- LBTY_tenuErrorStatus I2C_u8SendBuffer (u8 *pu8Data, u8 u8Size, u8 u8Address)
- LBTY_tenuErrorStatus I2C_u8ReceiveBuffer (u8 *pu8Data, u8 u8Size, u8 u8Address)
- void I2C_vidEnableINT (void)
- void I2C_vidDisableINT (void)
- void I2C_vidResetINT_Flag (void)
- void I2C_vidSetCallBack_OverFlow (void(*pCallBack)(void))

Enumeration Type Documentation

enum I2C_tenuMode

Enumerator:

I2C_Master	
I2C_Slave	

```
24              {
25      I2C_Master = (u8)0u,
26      I2C_Slave
27  }I2C_tenuMode;
```

enum I2C_tenuPrescaler

Enumerator:

I2C_Prescaler_1	
I2C_Prescaler_4	
I2C_Prescaler_16	
I2C_Prescaler_64	

```
29              {
30      I2C_Prescaler_1 = (u8)0u,
31      I2C_Prescaler_4,
32      I2C_Prescaler_16,
33      I2C_Prescaler_64
34  }I2C_tenuPrescaler;
```

enum I2C_tenuStatusCode

SLA+W = SLave Address + Write bit

SLA+R = SLave Address + Read bit

Enumerator:

I2C_Bus_Error	
I2C_Start	
I2C_RepeatStart	
I2C_MT_SLA_ACK	
I2C_MT_SLA_NACK	
I2C_MT_DATA_ACK	
I2C_MT_DATA_NACK	
I2C_MT_ARB_LOST	
I2C_MR_ARB_LOST	
I2C_MR_SLA_ACK	
I2C_MR_SLA_NACK	
I2C_MR_DATA_ACK	
I2C_MR_DATA_NACK	

I2C_SR_SLA_ACK	
I2C_SR_SLA_ARB_LOST	
I2C_SR_GCALL_ACK	
I2C_SR_GCALL_ARB_LOST	
I2C_SR_DATA_ACK	
I2C_SR_DATA_NACK	
I2C_SR_GCALL_Data_ACK	
I2C_SR_GCALL_Data_NACK	
I2C_Stop	
I2C_ST_SLA_ACK	
I2C_ST_ARB_LOST	
I2C_ST_DATA_ACK	
I2C_ST_DATA_NACK	
I2C_ST_LAST_Data	
I2C_NoInfo	

```
37              {
38      I2C_Bus_Error               = (u8)0x00, // illegal start or stop condition
39      I2C_Start                   = (u8)0x08, // start condition transmitted
40      I2C_RepeatStart             = (u8)0x10, // repeated start condition
transmitted
41
42      I2C_MT_SLA_ACK              = (u8)0x18, // SLA+W transmitted, ACK received
43      I2C_MT_SLA_NACK             = (u8)0x20, // SLA+W transmitted, NACK received
44      I2C_MT_DATA_ACK             = (u8)0x28, // data transmitted, ACK received
45      I2C_MT_DATA_NACK            = (u8)0x30, // data transmitted, NACK received
46
47      I2C_MT_ARB_LOST             = (u8)0x38, // arbitration lost in SLA+W or data
48      I2C_MR_ARB_LOST             = (u8)0x38, // arbitration lost in SLA+W or data
49
50      I2C_MR_SLA_ACK              = (u8)0x40, // SLA+R transmitted, ACK received
51      I2C_MR_SLA_NACK             = (u8)0x48, // SLA+R transmitted, NACK received
52      I2C_MR_DATA_ACK             = (u8)0x50, // data received, ACK returned
53      I2C_MR_DATA_NACK            = (u8)0x58, // data received, NACK returned
54
55      I2C_SR_SLA_ACK              = (u8)0x60, // SLA+W received, ACK returned
56      I2C_SR_SLA_ARB_LOST         = (u8)0x68, // arbitration lost in SLA+RW, SLA+W
received, ACK returned
57      I2C_SR_GCALL_ACK            = (u8)0x70, // general call received, ACK returned
58      I2C_SR_GCALL_ARB_LOST       = (u8)0x78, // arbitration lost in SLA+RW, general
call received, ACK returned
59      I2C_SR_DATA_ACK             = (u8)0x80, // data received, ACK returned
60      I2C_SR_DATA_NACK            = (u8)0x88, // data received, NACK returned
61      I2C_SR_GCALL_Data_ACK       = (u8)0x90, // general call data received, ACK
returned
62      I2C_SR_GCALL_Data_NACK      = (u8)0x98, // general call data received, NACK
returned
63
64      I2C_Stop                    = (u8)0xA0, // stop or repeated start condition
received while selected
65
66      I2C_ST_SLA_ACK              = (u8)0xA8, // SLA+R received, ACK returned
67      I2C_ST_ARB_LOST             = (u8)0xB0, // arbitration lost in SLA+RW, SLA+R
received, ACK returned
68      I2C_ST_DATA_ACK             = (u8)0xB8, // data transmitted, ACK received
69      I2C_ST_DATA_NACK            = (u8)0xC0, // data transmitted, NACK received
```

```
70
71      I2C_ST_LAST_Data            = (u8)0xC8, // last data byte transmitted, ACK
received
72
73      I2C_NoInfo                  = (u8)0xF8, // no state information available
74
75 }I2C_tenuStatusCode;
```

enum USART_tstrRW

Enumerator:

I2C_WRITE	
I2C_READ	

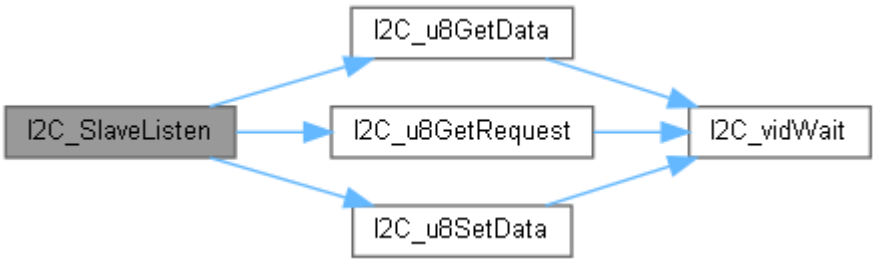
```
19              {
20      I2C_WRITE = (u8)0u,
21      I2C_READ,
22 }USART_tstrRW;
```

Function Documentation

void I2C_SlaveListen (u8 * *pu8char*, u8 *u8Address*)

```
332                                                 {
333     u8 u8Operation;
334     while(I2C_u8GetRequest(&u8Operation));
335     if(u8Operation == I2C_WRITE){
336         for(u8 i = u8Size ; i-- ; ){
337             if(I2C_u8SetData(*pu8char++))                       break;
338         }
339     }else if(u8Operation == I2C_READ){
340         for(u8 i = u8Size ; i-- ; ){
341             if(I2C_u8GetData( pu8char, i?LBTY_SET:LBTY_RESET))  break;
342         }
343     }
344 }
```

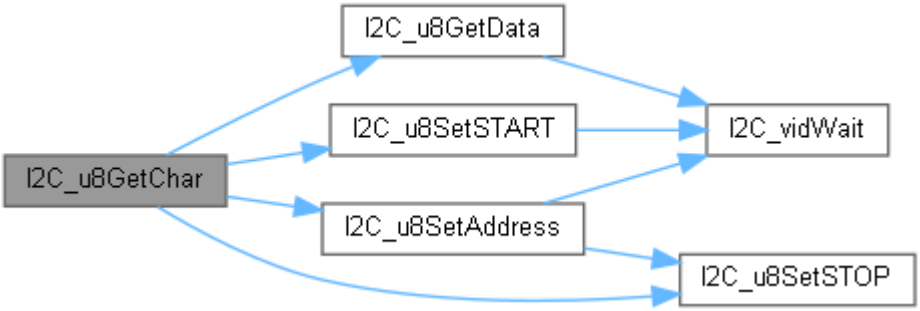
Here is the call graph for this function:



void I2C_u8GetChar (u8 * *pu8char*, u8 *u8Address*)

```
321                                                 {
322     while(S_TWI->m_TWCR.sBits.m_TWIE == LBTY_SET);
323     while(1){
324         if(I2C_u8SetSTART())                        continue;
325         if(I2C_u8SetAddress(u8Address, I2C_READ))   continue;
326         break;
327     }vidMyDelay_ms(I2C_SLAVE_WAIT);
328     I2C_u8GetData(pu8char, LBTY_SET);
329     I2C_u8SetSTOP();
330 }
```

Here is the call graph for this function:

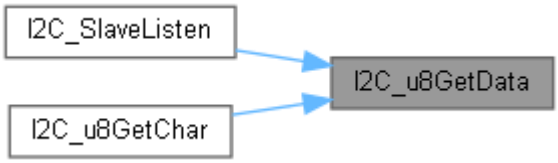


LBTY_tenuErrorStatus I2C_u8GetData (u8 * *pu8Data*, u8 *u8ACK*)

```
262                                                                 {
263     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
264
265     I2C_DEF(u8ACK, TWEA);
266     I2C_vidWait();
267
268     if(I2C_MODE == I2C_Master){
269         switch(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK)){
270             case I2C_MR_DATA_ACK:
271             case I2C_MR_DATA_NACK:
272                 *pu8Data = S_TWI->m_TWDR;
273                 break;
274             default:
275                 u8RetErrorState = LBTY_NOK;
276         }
277
278     }
279     if(I2C_MODE == I2C_Slave){
280         switch(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK)){
281             case I2C_SR_DATA_ACK:
282             case I2C_SR_DATA_NACK:
283             case I2C_SR_GCALL_Data_ACK:
284             case I2C_SR_GCALL_Data_NACK:
285                 *pu8Data = S_TWI->m_TWDR;
286                 break;
287             case I2C_Stop:
288                 S_TWI->m_TWCR.sBits.m_TWINT = LBTY_SET;     // The TWINT Flag
must be cleared by software by writing a logic one to it
289                 u8RetErrorState = LBTY_NOK;
290                 break;
291             default:
292                 u8RetErrorState = LBTY_NOK;
293         }
294     }
295
296     return u8RetErrorState;
297 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



u8 I2C_u8GetINTF (void)

```
132                         {
133     return (u8)S_TWI->m_TWCR.sBits.m_TWIE;
134 }
```

LBTY_tenuErrorStatus I2C_u8GetRequest (u8 * *Operation*)

```
191                                                         {
192     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
```

```
193
194     I2C_DEF(1u, TWEA);
195     I2C_vidWait();
196
197 //  if((S_TWI->m_TWDR>>1) == S_TWI->m_TWAR.sBits.m_TWA){
198 //      *Operation = (S_TWI->m_TWDR & 0x01);
199 //  }
200
201     switch(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK)){
202         case I2C_ST_SLA_ACK:
203         case I2C_ST_ARB_LOST:
204             *Operation = I2C_WRITE;
205             break;
206         case I2C_SR_SLA_ACK:
207         case I2C_SR_SLA_ARB_LOST:
208             *Operation = I2C_READ;
209             break;
210         case I2C_SR_GCALL_ACK:
211         case I2C_SR_GCALL_ARB_LOST:
212             *Operation = I2C_READ;
213             break;
214         default:
215             u8RetErrorState = LBTY_NOK;
216     }
217
218     return u8RetErrorState;
219 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

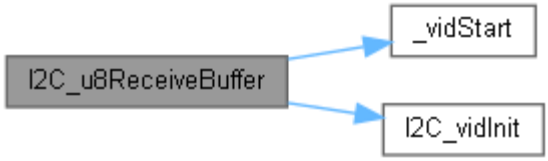
u8 I2C_u8GetStatus (void)

```
129                         {
130     return (u8)GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK);
131 }
```

LBTY_tenuErrorStatus I2C_u8ReceiveBuffer (u8 * *pu8Data*, u8 *u8Size*, u8 *u8Address*)

```
370
{
371     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
372
373     if(pu8Data == LBTY_NULL){
374         u8RetErrorState = LBTY_NULL_POINTER;
375     }else{
376         if(S_TWI->m_TWCR.sBits.m_TWIE == LBTY_RESET && strBuffer_GLB.m_u8Status
== I2C_IDLE){
377             strBuffer_GLB.m_u8Add.sBits.m_ADD = u8Address;
378             strBuffer_GLB.m_u8Add.sBits.m_OP  = I2C_READ;
379             strBuffer_GLB.m_pu8Data   = pu8Data;
380             strBuffer_GLB.m_u8Size    = u8Size;
381             strBuffer_GLB.m_u8Idx     = LBTY_u8ZERO;
382             strBuffer_GLB.m_u8Status  = I2C_IDLE;
383
384             I2C_vidInit();
385             _vidStart();
386         }else{
387             u8RetErrorState = LBTY_NOK;
388         }
389     }
390
391     return u8RetErrorState;
392 }
```

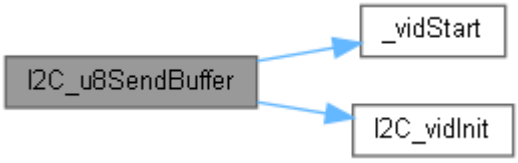
Here is the call graph for this function:



LBTY_tenuErrorStatus I2C_u8SendBuffer (u8 * *pu8Data*, u8 *u8Size*, u8 *u8Address*)

```
347                                                                                 {
348     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
349
350     if(pu8Data == LBTY_NULL){
351         u8RetErrorState = LBTY_NULL_POINTER;
352     }else{
353         if(S_TWI->m_TWCR.sBits.m_TWIE == LBTY_RESET && strBuffer_GLB.m_u8Status
== I2C_IDLE){
354             strBuffer_GLB.m_u8Add.sBits.m_ADD = u8Address;
355             strBuffer_GLB.m_u8Add.sBits.m_OP  = I2C_WRITE;
356             strBuffer_GLB.m_pu8Data   = pu8Data;
357             strBuffer_GLB.m_u8Size    = u8Size;
358             strBuffer_GLB.m_u8Idx     = LBTY_u8ZERO;
359             strBuffer_GLB.m_u8Status  = I2C_IDLE;
360
361             I2C_vidInit();
362             _vidStart();
363         }else{
364             u8RetErrorState = LBTY_NOK;
365         }
366     }
367
368     return u8RetErrorState;
369 }
```

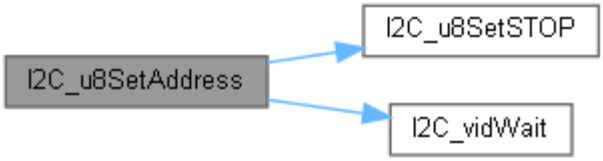
Here is the call graph for this function:



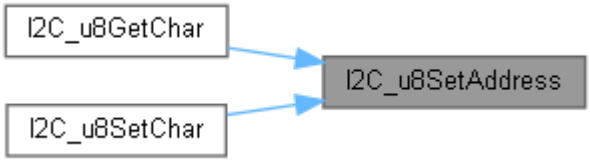
LBTY_tenuErrorStatus I2C_u8SetAddress (u8 *u8Address*, u8 *Operation*)

```
167                                                                             {
168     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
169
170     if((Operation == I2C_WRITE) || (Operation == I2C_READ)){
171         S_TWI->m_TWDR = (u8Address<<1) | (Operation & 0x01);
172         I2C_DEF(0u, 0);
173         I2C_vidWait();
174
175         switch(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK)){
176             case I2C_MT_SLA_ACK:
177             case I2C_MR_SLA_ACK:
178                 break;
179             case I2C_MT_SLA_NACK:
180             case I2C_MR_SLA_NACK:
181             default:
182                 u8RetErrorState = LBTY_NOK;
183         }
184     }else{
185         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
186     }
187
188     if(u8RetErrorState)     I2C_u8SetSTOP();
189     return u8RetErrorState;
190 }
```

Here is the call graph for this function:



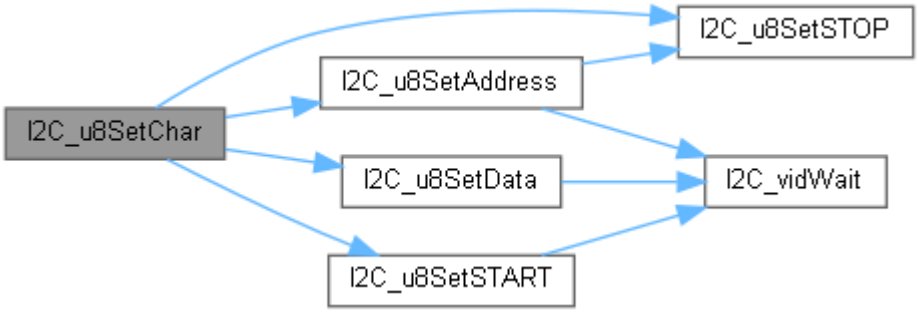
Here is the caller graph for this function:



void I2C_u8SetChar (u8 *u8char*, u8 *u8Address*)

```
311                                                 {
312     while(S_TWI->m_TWCR.sBits.m_TWIE == LBTY_SET);
313     while(1){
314         if(I2C_u8SetSTART())                            continue;
315         if(I2C_u8SetAddress(u8Address, I2C_WRITE))      continue;
316         break;
317     }vidMyDelay_ms(I2C_SLAVE_WAIT);
318     I2C_u8SetData(u8char);
319     I2C_u8SetSTOP();
320 }
```

Here is the call graph for this function:



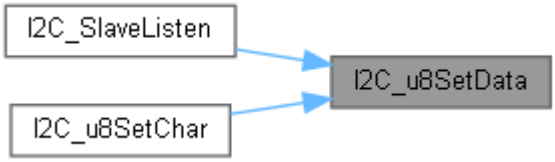
LBTY_tenuErrorStatus I2C_u8SetData (u8 *u8Data*)

```
220                                                 {
221     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
222
223     S_TWI->m_TWDR = u8Data;
224
225     if(S_TWI->m_TWCR.sBits.m_TWWC){
226         u8RetErrorState = LBTY_WRITE_ERROR;
227     }else{
228
229         if(I2C_MODE == I2C_Master){
230             I2C_DEF(0u, 0);
231             I2C_vidWait();
232
233             switch(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK)){
234                 case I2C_MT_DATA_ACK:
235                 case I2C_MT_DATA_NACK:
236                     break;
237                 default:
238                     u8RetErrorState = LBTY_NOK;
239             }
240         }
241         if(I2C_MODE == I2C_Slave){
242             I2C_DEF(1u, TWEA);
243             I2C_vidWait();
244
245             switch(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK)){
246                 case I2C_ST_DATA_ACK:
247                     break;
248                 case I2C_ST_DATA_NACK:
249                 case I2C_Stop:
```

```
250                     S_TWI->m_TWCR.sBits.m_TWINT = LBTY_SET;      // The TWINT Flag
must be cleared by software by writing a logic one to it
251                     u8RetErrorState = LBTY_NOK;
252                     break;
253                 case I2C_ST_LAST_Data:
254                 default:
255                     u8RetErrorState = LBTY_NOK;
256             }
257         }
258     }
259
260     return u8RetErrorState;
261 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



LBTY_tenuErrorStatus I2C_u8SetRepeatSTART (void)

```
156                                                 {
157     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
158
159     I2C_DEF(1u, TWSTA);
160     I2C_vidWait();
161
162     if(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK) != I2C_RepeatStart){
163         u8RetErrorState = LBTY_NOK;
164     }
165     return u8RetErrorState;
166 }
```

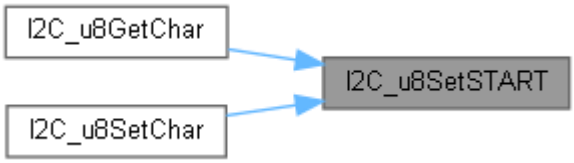
Here is the call graph for this function:

LBTY_tenuErrorStatus I2C_u8SetSTART (void)

```
145                                             {
146     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
147
148     I2C_DEF(1u, TWSTA);
149     I2C_vidWait();
150
151     if(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK) != I2C_Start){
152         u8RetErrorState = LBTY_NOK;
153     }
154     return u8RetErrorState;
155 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

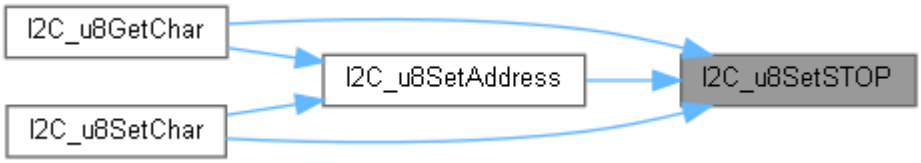


LBTY_tenuErrorStatus I2C_u8SetSTOP (void)

```
298                                             {
299     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
300
```

```
301     I2C_DEF(1u, TWSTO);
302 //  I2C_vidWait();
303     while(S_TWI->m_TWCR.sBits.m_TWSTO);
304 
305     if(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK) != I2C_Stop){
306         u8RetErrorState = LBTY_NOK;
307     }
308     return u8RetErrorState;
309 }
```

Here is the caller graph for this function:



void I2C_vidDisable (void)

```
125                             {
126     S_TWI->m_TWCR.sBits.m_TWEN  = strI2C_Config_GLB.m_I2CEN = LBTY_RESET;
127 }
```

void I2C_vidDisableINT (void)

```
397 {S_TWI->m_TWCR.sBits.m_TWIE = strI2C_Config_GLB.m_I2CIE = LBTY_RESET;}
```

void I2C_vidEnable (void)

```
122                             {
123     S_TWI->m_TWCR.sBits.m_TWEN  = strI2C_Config_GLB.m_I2CEN = LBTY_SET;
124 }
```

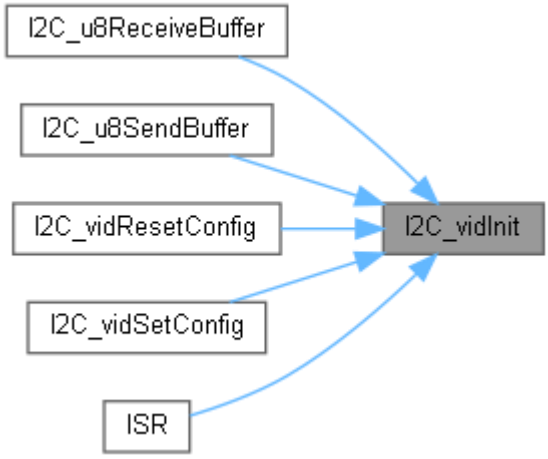
void I2C_vidEnableINT (void)

```
396 {S_TWI->m_TWCR.sBits.m_TWIE = strI2C_Config_GLB.m_I2CIE = LBTY_SET;}
```

void I2C_vidInit (void)

```
90                          {
91 
92     if(strI2C_Config_GLB.m_Mode == I2C_Master){
93         /*SCL frequency =  F_CPU / (16 + 2 * TWBR * pow(4,TWPS))*/
94         volatile u32 u32TWBR = 1u;
95         for(u8 i = strI2C_Config_GLB.m_Prescaler ; i-- ; )  u32TWBR *= 4;
96         S_TWI->m_TWBR = (u8)(((f32)F_CPU / I2C_SCL_FREQ - 16) / 2 * u32TWBR);
97 
98         S_TWI->m TWSR.sBits.m TWPS  = strI2C_Config_GLB.m_Prescaler;
99 
100         I2C_DEF(0u, 0);
101     }else if(strI2C_Config_GLB.m_Mode == I2C_Slave){
102         S_TWI->m_TWBR = LBTY_u8ZERO;
103         I2C_DEF(1u, TWEA);
104     }
105 
106     GPIO_u8PinInit(kau8I2C_PinConfig_GLB[0]);
107     GPIO_u8PinInit(kau8I2C_PinConfig_GLB[1]);
108 
109     S_TWI->m_TWAR.sBits.m_TWGCE = LBTY_RESET;
110     S_TWI->m_TWAR.sBits.m_TWA   = strI2C_Config_GLB.m_Address;
111 
112 //  S_TWI->m_TWCR.sBits.m_TWSTA = LBTY_RESET;
113 //  S_TWI->m_TWCR.sBits.m_TWSTO = LBTY_RESET;
114 //  S_TWI->m_TWCR.sBits.m_TWEA  = LBTY_RESET;
115 
116     S_TWI->m_TWCR.sBits.m_TWEN  = strI2C_Config_GLB.m_I2CEN;
117     S_TWI->m_TWCR.sBits.m_TWIE  = strI2C_Config_GLB.m_I2CIE;
118     S_TWI->m_TWCR.sBits.m_TWINT = LBTY_SET;      // The TWINT Flag must be cleared
by software by writing a logic one to it
119 
120 }
```

Here is the caller graph for this function:



void I2C_vidResetConfig (I2C_tstrConfiguration *const *pstrConfig*)

```
77                                                                          {
78      strI2C_Config_GLB.m_Mode        = I2C_MODE;
79      strI2C_Config_GLB.m_Address     = I2C_SLAVE_ADDRESS;
80      strI2C_Config_GLB.m_Prescaler   = I2C_CLOCK_PRESCALER;
81      strI2C_Config_GLB.m_I2CEN       = I2C_INIT_STATE;
82      strI2C_Config_GLB.m_I2CIE       = I2C_INT_STATE;
83
84      if(pstrConfig != LBTY_NULL){
85          *pstrConfig = strI2C_Config_GLB;
86      }
87      I2C_vidInit();
88  }
```

Here is the call graph for this function:

void I2C_vidResetINT_Flag (void)

```
399                                 {
400      S_TWI->m_TWCR.sBits.m_TWINT = LBTY_SET;// The TWINT Flag must be cleared by
software by writing a logic one to it
401  }
```

void I2C_vidSetCallBack_OverFlow (void(*)(void) *pCallBack*)

```
403                                                                 {
404      pFuncCallBack_I2C = pCallBack;
405  }
```

void I2C_vidSetConfig (I2C_tstrConfiguration const *const *pstrConfig*)

```
70                                                                      {
71      if(pstrConfig != LBTY_NULL){
72          strI2C_Config_GLB = *pstrConfig;
73      }
74      I2C_vidInit();
75  }
```

Here is the call graph for this function:

I2C_int.h

Go to the documentation of this file.

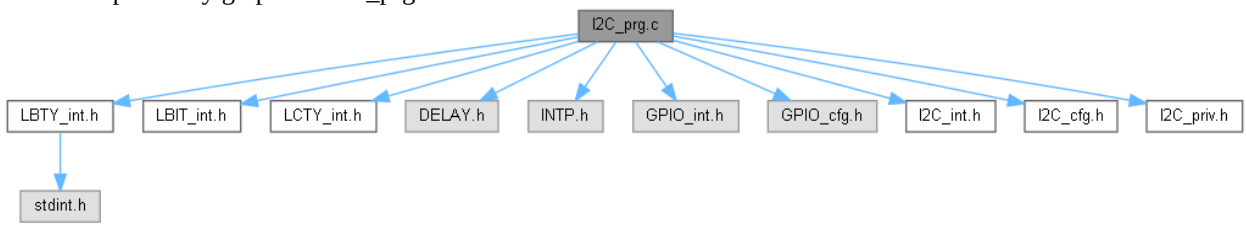
```
1 /*
2 /* ********************** FILE DEFINITION SECTION ************************** */
3 /* ************************************************************************* */
4 /* File Name    : I2C_int.h                                                  */
5 /* Author       : MAAM                                                       */
6 /* Version      : v01.2                                                      */
7 /* date         : Apr 13, 2023                                               */
8 /* ************************************************************************* */
9 /* ************************ HEADER FILES INCLUDES ************************  */
10 /* ************************************************************************* */
11
12 #ifndef I2C_INT_H_
13 #define I2C_INT_H_
14
15 /* ************************************************************************* */
16 /* ********************** TYPE_DEF/STRUCT/ENUM SECTION ********************** */
17 /* ************************************************************************* */
18
19 typedef enum{
20     I2C_WRITE = (u8)0u,
21     I2C_READ,
22 }USART_tstrRW;
23
24 typedef enum{
25     I2C_Master = (u8)0u,
26     I2C_Slave
27 }I2C_tenuMode;
28
29 typedef enum{
30     I2C_Prescaler_1 = (u8)0u,
31     I2C_Prescaler_4,
32     I2C_Prescaler_16,
33     I2C_Prescaler_64
34 }I2C_tenuPrescaler;
35
37 typedef enum{
38     I2C_Bus_Error               = (u8)0x00, // illegal start or stop condition
39     I2C_Start                   = (u8)0x08, // start condition transmitted
40     I2C_RepeatStart             = (u8)0x10, // repeated start condition transmitted
41
42     I2C_MT_SLA_ACK              = (u8)0x18, // SLA+W transmitted, ACK received
43     I2C_MT_SLA_NACK             = (u8)0x20, // SLA+W transmitted, NACK received
44     I2C_MT_DATA_ACK             = (u8)0x28, // data transmitted, ACK received
45     I2C_MT_DATA_NACK            = (u8)0x30, // data transmitted, NACK received
46
47     I2C_MT_ARB_LOST             = (u8)0x38, // arbitration lost in SLA+W or data
48     I2C_MR_ARB_LOST             = (u8)0x38, // arbitration lost in SLA+W or data
49
50     I2C_MR_SLA_ACK              = (u8)0x40, // SLA+R transmitted, ACK received
51     I2C_MR_SLA_NACK             = (u8)0x48, // SLA+R transmitted, NACK received
52     I2C_MR_DATA_ACK             = (u8)0x50, // data received, ACK returned
53     I2C_MR_DATA_NACK            = (u8)0x58, // data received, NACK returned
54
55     I2C_SR_SLA_ACK              = (u8)0x60, // SLA+W received, ACK returned
56     I2C_SR_SLA_ARB_LOST         = (u8)0x68, // arbitration lost in SLA+RW, SLA+W
received, ACK returned
57     I2C_SR_GCALL_ACK            = (u8)0x70, // general call received, ACK returned
58     I2C_SR_GCALL_ARB_LOST       = (u8)0x78, // arbitration lost in SLA+RW, general call
received, ACK returned
59     I2C_SR_DATA_ACK             = (u8)0x80, // data received, ACK returned
60     I2C_SR_DATA_NACK            = (u8)0x88, // data received, NACK returned
61     I2C_SR_GCALL_Data_ACK       = (u8)0x90, // general call data received, ACK returned
62     I2C_SR_GCALL_Data_NACK      = (u8)0x98, // general call data received, NACK
returned
63
64     I2C_Stop                    = (u8)0xA0, // stop or repeated start condition
received while selected
65
66     I2C_ST_SLA_ACK              = (u8)0xA8, // SLA+R received, ACK returned
67     I2C_ST_ARB_LOST             = (u8)0xB0, // arbitration lost in SLA+RW, SLA+R
received, ACK returned
68     I2C_ST_DATA_ACK             = (u8)0xB8, // data transmitted, ACK received
```

```
    I2C_ST_DATA_NACK            = (u8)0xC0, // data transmitted, NACK received

    I2C_ST_LAST_Data            = (u8)0xC8, // last data byte transmitted, ACK received

    I2C_NoInfo                  = (u8)0xF8, // no state information available

}I2C_tenuStatusCode;

typedef struct{
    I2C_tenuMode            m_Mode;
    u8                      m_Address;
    I2C_tenuPrescaler       m_Prescaler;

    LBTY_tenuFlagStatus     m_I2CEN;
    LBTY_tenuFlagStatus     m_I2CIE;
}I2C_tstrConfiguration;

/* ********************************************************************************* */
/* ************************** MACRO/DEFINE SECTION ************************** */
/* ********************************************************************************* */

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

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

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

extern void I2C_vidSetConfig(I2C_tstrConfiguration const* const pstrConfig);
extern void I2C_vidResetConfig(I2C_tstrConfiguration* const pstrConfig);

extern void I2C_vidInit(void);

extern void I2C_vidEnable(void);
extern void I2C_vidDisable(void);

extern u8 I2C_u8GetStatus(void);
extern u8 I2C_u8GetINTF(void);


/*******************************************************************************************
*******************************/

extern LBTY_tenuErrorStatus I2C_u8SetSTART(void);
extern LBTY_tenuErrorStatus I2C_u8SetRepeatSTART(void);
extern LBTY_tenuErrorStatus I2C_u8SetAddress(u8 u8Address, u8 Operation);
extern LBTY_tenuErrorStatus I2C_u8GetRequest(u8* Operation);
extern LBTY_tenuErrorStatus I2C_u8SetData(u8 u8Data);
extern LBTY_tenuErrorStatus I2C_u8GetData(u8* pu8Data, u8 u8ACK);
extern LBTY_tenuErrorStatus I2C_u8SetSTOP(void);

extern void I2C_u8SetChar(u8 u8char, u8 u8Address);
extern void I2C_u8GetChar(u8* pu8char, u8 u8Address);
extern void I2C_SlaveListen(u8* pu8char, u8 u8Address);

extern LBTY_tenuErrorStatus I2C_u8SendBuffer    (u8* pu8Data, u8 u8Size, u8 u8Address);
extern LBTY_tenuErrorStatus I2C_u8ReceiveBuffer(u8* pu8Data, u8 u8Size, u8 u8Address);


/*******************************************************************************************
*******************************/

extern void I2C_vidEnableINT(void);
extern void I2C_vidDisableINT(void);

extern void I2C_vidResetINT_Flag(void);

extern void I2C_vidSetCallBack_OverFlow(void (*pCallBack)(void));

#endif /* I2C_INT_H_ */
/*************************** E N D (I2C_int.h) ******************************/
```

I2C_prg.c File Reference

```
#include "LBTY_int.h"
#include "LBIT_int.h"
#include "LCTY_int.h"
#include "DELAY.h"
#include "INTP.h"
#include "GPIO_int.h"
#include "GPIO_cfg.h"
#include "I2C_int.h"
#include "I2C_cfg.h"
#include "I2C_priv.h"
```

Include dependency graph for I2C_prg.c:



Functions

- void I2C_vidSetConfig (I2C_tstrConfiguration const *const pstrConfig)
- void I2C_vidResetConfig (I2C_tstrConfiguration *const pstrConfig)
- void I2C_vidInit (void)
- void I2C_vidEnable (void)
- void I2C_vidDisable (void)
- u8 I2C_u8GetStatus (void)
- u8 I2C_u8GetINTF (void)
- void I2C_vidWait (void)
- LBTY_tenuErrorStatus I2C_u8SetSTART (void)
- LBTY_tenuErrorStatus I2C_u8SetRepeatSTART (void)
- LBTY_tenuErrorStatus I2C_u8SetAddress (u8 u8Address, u8 Operation)
- LBTY_tenuErrorStatus I2C_u8GetRequest (u8 *Operation)
- LBTY_tenuErrorStatus I2C_u8SetData (u8 u8Data)
- LBTY_tenuErrorStatus I2C_u8GetData (u8 *pu8Data, u8 u8ACK)
- LBTY_tenuErrorStatus I2C_u8SetSTOP (void)
- void I2C_u8SetChar (u8 u8char, u8 u8Address)
- void I2C_u8GetChar (u8 *pu8char, u8 u8Address)
- void I2C_SlaveListen (u8 *pu8char, u8 u8Size)
- LBTY_tenuErrorStatus I2C_u8SendBuffer (u8 *pu8Data, u8 u8Size, u8 u8Address)
- LBTY_tenuErrorStatus I2C_u8ReceiveBuffer (u8 *pu8Data, u8 u8Size, u8 u8Address)
- void I2C_vidEnableINT (void)
- void I2C_vidDisableINT (void)
- void I2C_vidResetINT_Flag (void)
- void I2C_vidSetCallBack_OverFlow (void(*pCallBack)(void))
- ISR (TWI_vect)

Variables

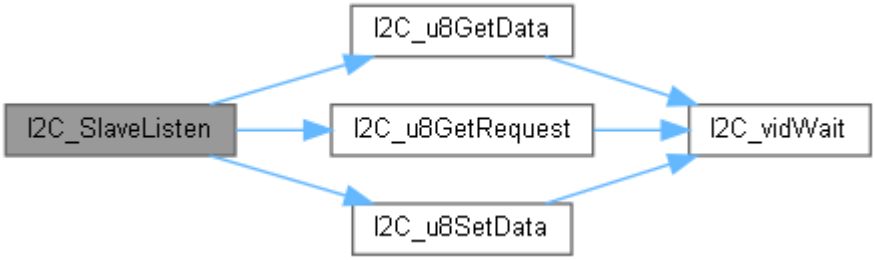
- const GPIO_tstrPinConfig kau8I2C_PinConfig_GLB []
- static volatile u8 I2C_u8Flag_GLB
- static void(* pFuncCallBack_I2C)(void) = INTP_vidCallBack
- volatile I2C_tstrBuffer strBuffer_GLB
- static volatile I2C_tstrConfiguration strI2C_Config_GLB

Function Documentation

void I2C_SlaveListen (u8 * *pu8char*, u8 *u8Size*)

```
332                                                 {
333     u8 u8Operation;
334     while(I2C_u8GetRequest(&u8Operation));
335     if(u8Operation == I2C_WRITE){
336         for(u8 i = u8Size ; i-- ; ){
337             if(I2C_u8SetData(*pu8char++))                       break;
338         }
339     }else if(u8Operation == I2C_READ){
340         for(u8 i = u8Size ; i-- ; ){
341             if(I2C_u8GetData( pu8char, i?LBTY_SET:LBTY_RESET))  break;
342         }
343     }
344 }
```

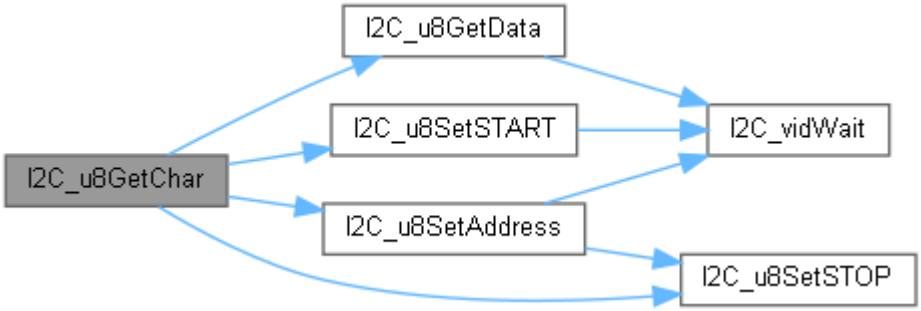
Here is the call graph for this function:



void I2C_u8GetChar (u8 * *pu8char*, u8 *u8Address*)

```
321                                                 {
322     while(S_TWI->m_TWCR.sBits.m_TWIE == LBTY_SET);
323     while(1){
324         if(I2C_u8SetSTART())                            continue;
325         if(I2C_u8SetAddress(u8Address, I2C_READ))       continue;
326         break;
327     }vidMyDelay_ms(I2C_SLAVE_WAIT);
328     I2C_u8GetData(pu8char, LBTY_SET);
329     I2C_u8SetSTOP();
330 }
```

Here is the call graph for this function:



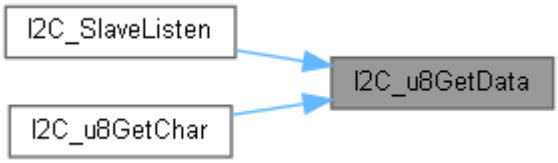
LBTY_tenuErrorStatus I2C_u8GetData (u8 * *pu8Data*, u8 *u8ACK*)

```
262                                                 {
263     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
264
265     I2C_DEF(u8ACK, TWEA);
266     I2C_vidWait();
267
268     if(I2C_MODE == I2C_Master){
269         switch(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK)){
270             case I2C_MR_DATA_ACK:
271             case I2C_MR_DATA_NACK:
272                 *pu8Data = S_TWI->m_TWDR;
273                 break;
274             default:
```

```
275                 u8RetErrorState = LBTY_NOK;
276             }
277 
278         }
279         if(I2C_MODE == I2C_Slave){
280             switch(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK)){
281                 case I2C_SR_DATA_ACK:
282                 case I2C_SR_DATA_NACK:
283                 case I2C_SR_GCALL_Data_ACK:
284                 case I2C_SR_GCALL_Data_NACK:
285                     *pu8Data = S_TWI->m_TWDR;
286                     break;
287                 case I2C_Stop:
288                     S_TWI->m_TWCR.sBits.m_TWINT = LBTY_SET;     // The TWINT Flag
must be cleared by software by writing a logic one to it
289                     u8RetErrorState = LBTY_NOK;
290                     break;
291                 default:
292                     u8RetErrorState = LBTY_NOK;
293             }
294         }
295 
296         return u8RetErrorState;
297 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



u8 I2C_u8GetINTF (void)

```
132                         {
133     return (u8)S_TWI->m_TWCR.sBits.m_TWIE;
134 }
```

LBTY_tenuErrorStatus I2C_u8GetRequest (u8 * *Operation*)

```
191                                                         {
192     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
193 
194     I2C_DEF(1u, TWEA);
195     I2C_vidWait();
196 
197 //  if((S_TWI->m_TWDR>>1) == S_TWI->m_TWAR.sBits.m_TWA){
198 //      *Operation = (S_TWI->m_TWDR & 0x01);
199 //  }
200 
201     switch(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK)){
202         case I2C_ST_SLA_ACK:
203         case I2C_ST_ARB_LOST:
204             *Operation = I2C_WRITE;
205             break;
206         case I2C_SR_SLA_ACK:
207         case I2C_SR_SLA_ARB_LOST:
208             *Operation = I2C_READ;
209             break;
210         case I2C_SR_GCALL_ACK:
211         case I2C_SR_GCALL_ARB_LOST:
212             *Operation = I2C_READ;
213             break;
214         default:
215             u8RetErrorState = LBTY_NOK;
216     }
217 
218     return u8RetErrorState;
219 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

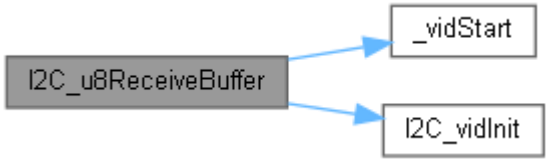
u8 I2C_u8GetStatus (void)

```
129                          {
130     return (u8)GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK);
131 }
```

LBTY_tenuErrorStatus I2C_u8ReceiveBuffer (u8 * *pu8Data*, u8 *u8Size*, u8 *u8Address*)

```
370
{
371     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
372
373     if(pu8Data == LBTY_NULL){
374         u8RetErrorState = LBTY_NULL_POINTER;
375     }else{
376         if(S_TWI->m_TWCR.sBits.m_TWIE == LBTY_RESET && strBuffer_GLB.m_u8Status
== I2C_IDLE){
377             strBuffer_GLB.m_u8Add.sBits.m_ADD = u8Address;
378             strBuffer_GLB.m_u8Add.sBits.m_OP  = I2C_READ;
379             strBuffer_GLB.m_pu8Data   = pu8Data;
380             strBuffer_GLB.m_u8Size    = u8Size;
381             strBuffer_GLB.m_u8Idx     = LBTY_u8ZERO;
382             strBuffer_GLB.m_u8Status  = I2C_IDLE;
383
384             I2C_vidInit();
385              vidStart();
386         }else{
387             u8RetErrorState = LBTY_NOK;
388         }
389     }
390
391     return u8RetErrorState;
392 }
```

Here is the call graph for this function:

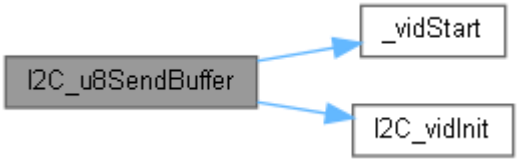


LBTY_tenuErrorStatus I2C_u8SendBuffer (u8 * *pu8Data*, u8 *u8Size*, u8 *u8Address*)

```
347                                                                          {
348     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
349
350     if(pu8Data == LBTY_NULL){
351         u8RetErrorState = LBTY_NULL_POINTER;
352     }else{
353         if(S_TWI->m_TWCR.sBits.m_TWIE == LBTY_RESET && strBuffer_GLB.m_u8Status
== I2C_IDLE){
354             strBuffer_GLB.m_u8Add.sBits.m_ADD = u8Address;
355             strBuffer_GLB.m_u8Add.sBits.m_OP  = I2C_WRITE;
356             strBuffer_GLB.m_pu8Data   = pu8Data;
357             strBuffer_GLB.m_u8Size    = u8Size;
358             strBuffer_GLB.m_u8Idx     = LBTY_u8ZERO;
359             strBuffer_GLB.m_u8Status  = I2C_IDLE;
360
361             I2C_vidInit();
362              vidStart();
363         }else{
364             u8RetErrorState = LBTY_NOK;
365         }
366     }
367
368     return u8RetErrorState;
```

```
369 }
```

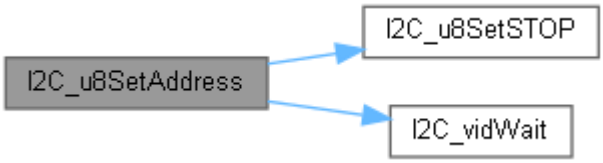
Here is the call graph for this function:



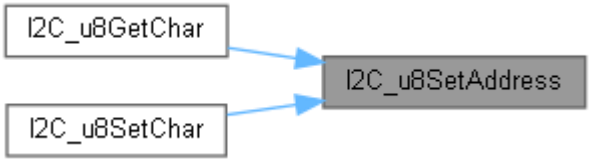
LBTY_tenuErrorStatus I2C_u8SetAddress (u8 *u8Address*, u8 *Operation*)

```
167                                                                 {
168     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
169
170     if((Operation == I2C_WRITE) || (Operation == I2C_READ)){
171         S_TWI->m_TWDR = (u8Address<<1) | (Operation & 0x01);
172         I2C_DEF(0u, 0);
173         I2C_vidWait();
174
175         switch(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK)){
176             case I2C_MT_SLA_ACK:
177             case I2C_MR_SLA_ACK:
178                 break;
179             case I2C_MT_SLA_NACK:
180             case I2C_MR_SLA_NACK:
181             default:
182                 u8RetErrorState = LBTY_NOK;
183         }
184     }else{
185         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
186     }
187
188     if(u8RetErrorState)     I2C_u8SetSTOP();
189     return u8RetErrorState;
190 }
```

Here is the call graph for this function:



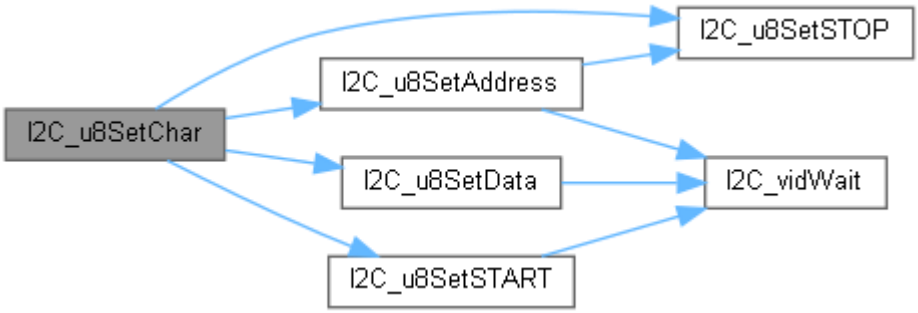
Here is the caller graph for this function:



void I2C_u8SetChar (u8 *u8char*, u8 *u8Address*)

```
311                                             {
312     while(S_TWI->m_TWCR.sBits.m_TWIE == LBTY_SET);
313     while(1){
314         if(I2C_u8SetSTART())                        continue;
315         if(I2C_u8SetAddress(u8Address, I2C_WRITE))  continue;
316         break;
317     }vidMyDelay_ms(I2C_SLAVE_WAIT);
318     I2C_u8SetData(u8char);
319     I2C_u8SetSTOP();
320 }
```

Here is the call graph for this function:

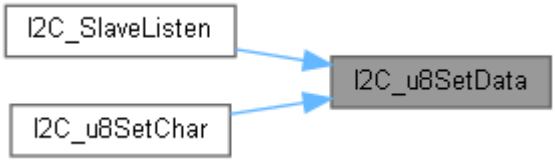


LBTY_tenuErrorStatus I2C_u8SetData (u8 *u8Data*)

```
220                                                 {
221     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
222
223     S_TWI->m_TWDR = u8Data;
224
225     if(S_TWI->m_TWCR.sBits.m_TWWC){
226         u8RetErrorState = LBTY_WRITE_ERROR;
227     }else{
228
229         if(I2C_MODE == I2C_Master){
230             I2C_DEF(0u, 0);
231             I2C_vidWait();
232
233             switch(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK)){
234                 case I2C_MT_DATA_ACK:
235                 case I2C_MT_DATA_NACK:
236                     break;
237                 default:
238                     u8RetErrorState = LBTY_NOK;
239             }
240         }
241         if(I2C_MODE == I2C_Slave){
242             I2C_DEF(1u, TWEA);
243             I2C_vidWait();
244
245             switch(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK)){
246                 case I2C_ST_DATA_ACK:
247                     break;
248                 case I2C_ST_DATA_NACK:
249                 case I2C_Stop:
250                     S_TWI->m_TWCR.sBits.m_TWINT = LBTY_SET;     // The TWINT Flag
must be cleared by software by writing a logic one to it
251                     u8RetErrorState = LBTY_NOK;
252                     break;
253                 case I2C_ST_LAST_Data:
254                 default:
255                     u8RetErrorState = LBTY_NOK;
256             }
257         }
258     }
259
260     return u8RetErrorState;
261 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



LBTY_tenuErrorStatus I2C_u8SetRepeatSTART (void)

```
156                                                 {
157     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
```

```
158
159     I2C_DEF(1u, TWSTA);
160     I2C_vidWait();
161
162     if(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK) != I2C_RepeatStart){
163         u8RetErrorState = LBTY_NOK;
164     }
165     return u8RetErrorState;
166 }
```

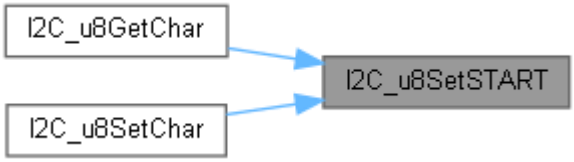
Here is the call graph for this function:

LBTY_tenuErrorStatus I2C_u8SetSTART (void)

```
145                                         {
146     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
147
148     I2C_DEF(1u, TWSTA);
149     I2C_vidWait();
150
151     if(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK) != I2C_Start){
152         u8RetErrorState = LBTY_NOK;
153     }
154     return u8RetErrorState;
155 }
```

Here is the call graph for this function:

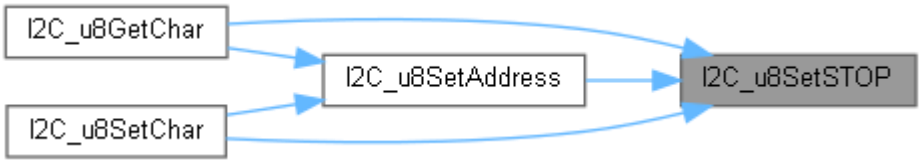
Here is the caller graph for this function:



LBTY_tenuErrorStatus I2C_u8SetSTOP (void)

```
298                                         {
299     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
300
301     I2C_DEF(1u, TWSTO);
302 //  I2C_vidWait();
303     while(S_TWI->m_TWCR.sBits.m_TWSTO);
304
305     if(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK) != I2C_Stop){
306         u8RetErrorState = LBTY_NOK;
307     }
308     return u8RetErrorState;
309 }
```

Here is the caller graph for this function:



void I2C_vidDisable (void)

```
125                             {
126     S_TWI->m_TWCR.sBits.m_TWEN  = strI2C_Config_GLB.m_I2CEN = LBTY_RESET;
127 }
```

void I2C_vidDisableINT (void)

```
397 {S_TWI->m_TWCR.sBits.m_TWIE = strI2C_Config_GLB.m_I2CIE = LBTY_RESET;}
```

void I2C_vidEnable (void)

```
122                             {
```

```
123     S_TWI->m_TWCR.sBits.m_TWEN  = strI2C_Config_GLB.m_I2CEN = LBTY_SET;
124 }
```

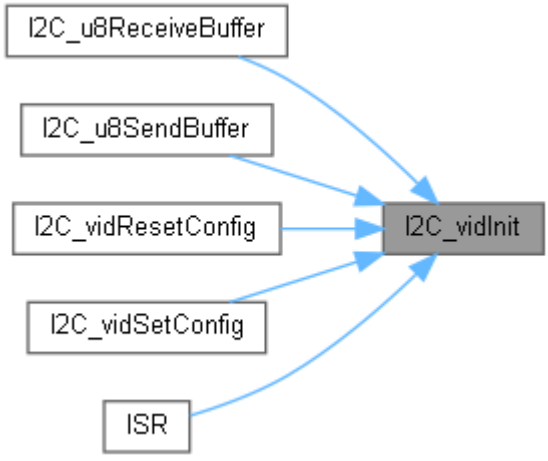
void I2C_vidEnableINT (void)

```
396 {S_TWI->m_TWCR.sBits.m_TWIE = strI2C_Config_GLB.m_I2CIE = LBTY_SET;}
```

void I2C_vidInit (void)

```
90                      {
91
92     if(strI2C_Config_GLB.m_Mode == I2C_Master){
93         /*SCL frequency =  F_CPU / (16 + 2 * TWBR * pow(4,TWPS))*/
94         volatile u32 u32TWBR = 1u;
95         for(u8 i = strI2C_Config_GLB.m_Prescaler ; i-- ; )  u32TWBR *= 4;
96         S_TWI->m_TWBR = (u8)(((f32)F_CPU / I2C_SCL_FREQ - 16) / 2 * u32TWBR);
97
98         S_TWI->m_TWSR.sBits.m_TWPS  = strI2C_Config_GLB.m_Prescaler;
99
100         I2C_DEF(0u, 0);
101     }else if(strI2C_Config_GLB.m_Mode == I2C_Slave){
102         S_TWI->m_TWBR = LBTY_u8ZERO;
103         I2C_DEF(1u, TWEA);
104     }
105
106     GPIO_u8PinInit(kau8I2C_PinConfig_GLB[0]);
107     GPIO_u8PinInit(kau8I2C_PinConfig_GLB[1]);
108
109     S_TWI->m_TWAR.sBits.m_TWGCE = LBTY_RESET;
110     S_TWI->m_TWAR.sBits.m_TWA   = strI2C_Config_GLB.m_Address;
111
112 //  S_TWI->m_TWCR.sBits.m_TWSTA = LBTY_RESET;
113 //  S_TWI->m_TWCR.sBits.m_TWSTO = LBTY_RESET;
114 //  S_TWI->m_TWCR.sBits.m_TWEA  = LBTY_RESET;
115
116     S_TWI->m_TWCR.sBits.m_TWEN  = strI2C_Config_GLB.m_I2CEN;
117     S_TWI->m_TWCR.sBits.m_TWIE  = strI2C_Config_GLB.m_I2CIE;
118     S_TWI->m_TWCR.sBits.m_TWINT = LBTY_SET;     // The TWINT Flag must be cleared
by software by writing a logic one to it
119
120 }
```

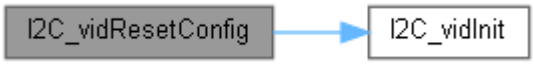
Here is the caller graph for this function:



void I2C_vidResetConfig (I2C_tstrConfiguration *const *pstrConfig*)

```
77                                                                      {
78     strI2C_Config_GLB.m_Mode        = I2C_MODE;
79     strI2C_Config_GLB.m_Address     = I2C_SLAVE_ADDRESS;
80     strI2C_Config_GLB.m_Prescaler   = I2C_CLOCK_PRESCALER;
81     strI2C_Config_GLB.m_I2CEN       = I2C_INIT_STATE;
82     strI2C_Config_GLB.m_I2CIE       = I2C_INT_STATE;
83
84     if(pstrConfig != LBTY_NULL){
85         *pstrConfig = strI2C_Config_GLB;
86     }
87     I2C_vidInit();
88 }
```

Here is the call graph for this function:



void I2C_vidResetINT_Flag (void)

```
399                                 {
400     S_TWI->m_TWCR.sBits.m_TWINT = LBTY_SET;// The TWINT Flag must be cleared by
software by writing a logic one to it
401 }
```

void I2C_vidSetCallBack_OverFlow (void(*)(void) *pCallBack*)

```
403                                                                 {
404     pFuncCallBack_I2C = pCallBack;
405 }
```

void I2C_vidSetConfig (I2C_tstrConfiguration const *const *pstrConfig*)

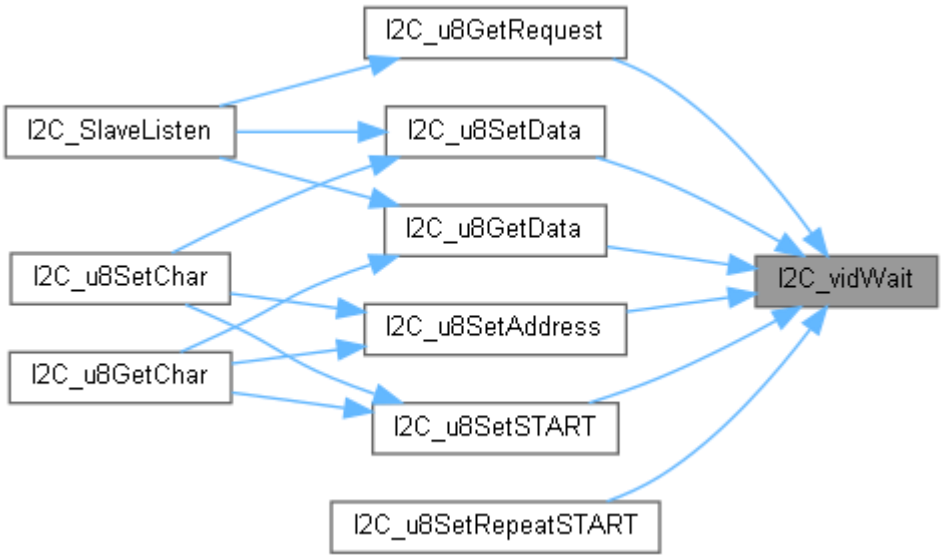
```
70                                                                      {
71     if(pstrConfig != LBTY_NULL){
72         strI2C_Config_GLB = *pstrConfig;
73     }
74     I2C_vidInit();
75 }
```

Here is the call graph for this function:

void I2C_vidWait (void)

```
138                         {
139 //  S_TWI->m_TWCR.sBits.m_TWINT = LBTY_SET;// The TWINT Flag must be cleared by
software by writing a logic one to it
140     I2C_u8Flag_GLB = LBTY_RESET;
141     while((!S_TWI->m_TWCR.sBits.m_TWINT) && (!I2C_u8Flag_GLB));
142 //  S_TWI->m_TWCR.sBits.m_TWINT = LBTY_SET;// The TWINT Flag must be cleared by
software by writing a logic one to it
143 }
```

Here is the caller graph for this function:

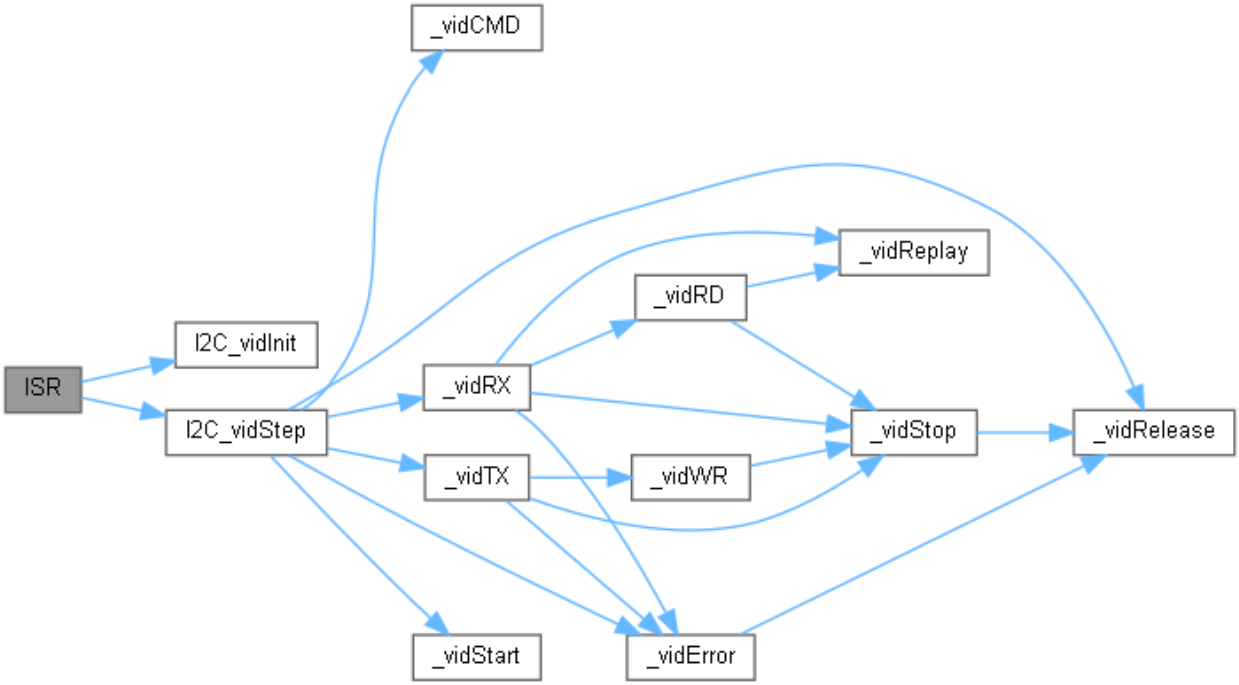


ISR (TWI_vect)

```
407             {
408     switch(strBuffer_GLB.m_u8Status){
409         case I2C_IDLE:
410             break;
411         case I2C_START:
412         case I2C_ADDRESS:
413         case I2C_DATA:
414         case I2C_STOP:
```

```
415             I2C_vidStep();
416             if(strBuffer_GLB.m_u8Status == I2C_IDLE)
417                 pFuncCallBack_I2C();
418             break;
419         case I2C_ERROR:
420             break;
421         default:
422             I2C_vidInit();
423             break;
424     }
425
426     I2C_u8Flag_GLB = LBTY_SET;
427 }
```

Here is the call graph for this function:



Variable Documentation

volatile u8 I2C_u8Flag_GLB [static]

const GPIO_tstrPinConfig kau8I2C_PinConfig_GLB[]

```
Initial value:= {
        {.m_Port = I2C_PORT, .m_Pin = I2C_SDA_PIN, .m_Dir = PIN_INPUT, .m_Res =
PULL_UP},
        {.m_Port = I2C_PORT, .m_Pin = I2C_SCL_PIN, .m_Dir = PIN_INPUT, .m_Res = PULL_UP}
}
```

void(* pFuncCallBack_I2C) (void) (void) = INTP_vidCallBack [static]

volatile I2C_tstrBuffer strBuffer_GLB

```
Initial value:= {
    .m_u8Add.u_Reg = LBTY_u8ZERO,
    .m_pu8Data     = LBTY_NULL,
    .m_u8Size      = LBTY_u8ZERO,
    .m_u8Idx       = LBTY_u8ZERO,
    .m_u8Status    = I2C_IDLE
}
```

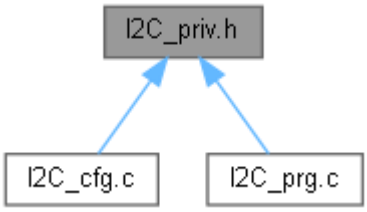
volatile I2C_tstrConfiguration strI2C_Config_GLB [static]

```
Initial value:= {
    .m_Mode          = I2C_MODE,
```

```
    .m_Address      = I2C_SLAVE_ADDRESS,
    .m_Prescaler    = I2C_CLOCK_PRESCALER,
    .m_I2CEN        = I2C_INIT_STATE,
    .m_I2CIE        = I2C_INT_STATE,
}
```

I2C_priv.h File Reference

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



Data Structures

union I2C_tstrAddress*: I2C Address*

struct I2C_tstrBuffer*: I2C TX/RX Buffer*

union TWCR_type*: Type define of Union bit field of "TWI Control Register"*

union TWSR_type*: Type define of Union bit field of "TWI Status Register"*

union TWAR_type*: Type define of Union bit field of "TWI Address Register"*

struct TWI_type*: I2C "TWI" Registers*

Macros

- #define S_TWI ((TWI_type* const)0x20U)
- #define TWBR (*(volatile u8* const)0x20U)
- #define TWSR (*(volatile u8* const)0x21U)
- #define TWAR (*(volatile u8* const)0x22U)
- #define TWDR (*(volatile u8* const)0x23U)
- #define TWCR (*(volatile u8* const)0x56U)
- #define TWIE 0
- #define TWEN 2
- #define TWWC 3
- #define TWSTO 4
- #define TWSTA 5
- #define TWEA 6
- #define TWINT 7
- #define I2C_STATUS_MASK 0xF8u
- #define I2C_PORT C
- #define I2C_SDA_PIN GPIO_I2C_SDA
- #define I2C_SCL_PIN GPIO_I2C_SCL
- #define I2C_START() (S_TWI->m_TWCR.u_Reg = (_BV(TWINT) | _BV(TWEN) | _BV(TWIE) | _BV(TWSTA)))
- #define I2C_ACK() (S_TWI->m_TWCR.u_Reg = (_BV(TWINT) | _BV(TWEN) | _BV(TWIE) | _BV(TWEA)))
- #define I2C_NACK() (S_TWI->m_TWCR.u_Reg = (_BV(TWINT) | _BV(TWEN) | _BV(TWIE)))

- #define I2C_STOP() (S_TWI->m_TWCR.u_Reg = (_BV(TWINT) | _BV(TWEN) | _BV(TWIE) | _BV(TWSTO)))
- #define I2C_DEF(value, bit) (S_TWI->m_TWCR.u_Reg = (_BV(TWINT) | _BV(TWEN) | (value<<(bit))))

Enumerations

- enum I2C_tenuStatus { I2C_IDLE = (u8)0u, I2C_START, I2C_ADDRESS, I2C_DATA, I2C_STOP, I2C_ERROR }
  : Type define of TX/RX Status

Functions

- void _vidStart (void)
- void _vidStop (void)
- void _vidRelease (void)
- void _vidError (void)
- void _vidCMD (void)
- void _vidReplay (void)
- void _vidWR (void)
- void _vidRD (void)
- void _vidTX (u8 u8State)
- void _vidRX (u8 u8State)
- void I2C_vidStep (void)

Macro Definition Documentation

#define I2C_ACK() (S_TWI->m_TWCR.u_Reg = (_BV(TWINT) | _BV(TWEN) | _BV(TWIE) | _BV(TWEA)))

#define I2C_DEF(value, bit) (S_TWI->m_TWCR.u_Reg = (_BV(TWINT) | _BV(TWEN) | (value<<(bit))))

#define I2C_NACK() (S_TWI->m_TWCR.u_Reg = (_BV(TWINT) | _BV(TWEN) | _BV(TWIE)))

#define I2C_PORT C

#define I2C_SCL_PIN GPIO_I2C_SCL

#define I2C_SDA_PIN GPIO_I2C_SDA

#define I2C_START() (S_TWI->m_TWCR.u_Reg = (_BV(TWINT) | _BV(TWEN) | _BV(TWIE) | _BV(TWSTA)))

#define I2C_STATUS_MASK 0xF8u

#define I2C_STOP() (S_TWI->m_TWCR.u_Reg = (_BV(TWINT) | _BV(TWEN) | _BV(TWIE) | _BV(TWSTO)))

#define S_TWI ((TWI_type* const)0x20U)

Two-Wire Serial Interface

#define TWAR (*(volatile u8* const)0x22U)

#define TWBR (*(volatile u8* const)0x20U)

#define TWCR (*(volatile u8* const)0x56U)

#define TWDR (*(volatile u8* const)0x23U)

#define TWEA 6

#define TWEN 2

#define TWIE 0

#define TWINT 7

#define TWSR (*(volatile u8* const)0x21U)

#define TWSTA 5

#define TWSTO 4

#define TWWC 3

Enumeration Type Documentation

enum I2C_tenuStatus

: Type define of TX/RX Status

Type : Enum **Unit** : None

Enumerator:

I2C_IDLE	
I2C_START	
I2C_ADDRESS	
I2C_DATA	
I2C_STOP	
I2C_ERROR	

```
21              {
22      I2C_IDLE = (u8)0u,
23      I2C_START,
24      I2C_ADDRESS,
25  //  I2C_RW,
26  //  I2C_ACK,
27      I2C_DATA,
28      I2C_STOP,
29      I2C_ERROR
30  }I2C_tenuStatus;
```

Function Documentation

void _vidCMD (void)

```
71                      {
72      S_TWI->m_TWDR = strBuffer_GLB.m_u8Add.u_Reg;
73      if(I2C_MODE == I2C_Master){
74          I2C_NACK();
75      }else if(I2C_MODE == I2C_Slave){
76          I2C_ACK();
77      }
78      strBuffer_GLB.m_u8Status = I2C_ADDRESS;
79  }
```

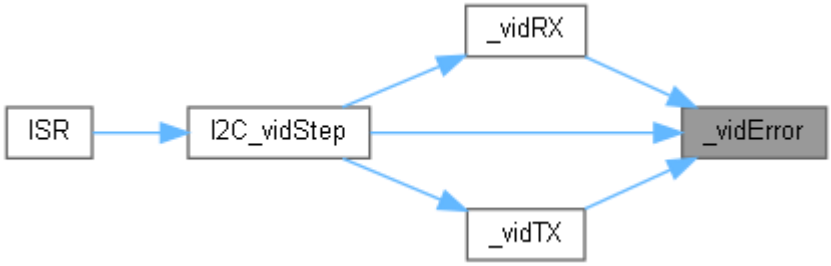
Here is the caller graph for this function:

void _vidError (void)

```
65                      {
66      I2C_STOP();
67      //S_TWI->m_TWCR.sBits.m_TWEN = LBTY_RESET;
68      strBuffer_GLB.m_u8Status    = I2C_ERROR;
69       _vidRelease();
70  }
```

Here is the call graph for this function:

Here is the caller graph for this function:



void _vidRD (void)

```
100                     {
101     if(strBuffer_GLB.m_u8Idx < strBuffer_GLB.m_u8Size){
102         strBuffer_GLB.m_pu8Data[strBuffer_GLB.m_u8Idx++] = S_TWI->m_TWDR;
103          _vidReplay();
104         strBuffer_GLB.m_u8Status = I2C_DATA;
105     }else{
106          _vidStop();
107     }
108 }
```

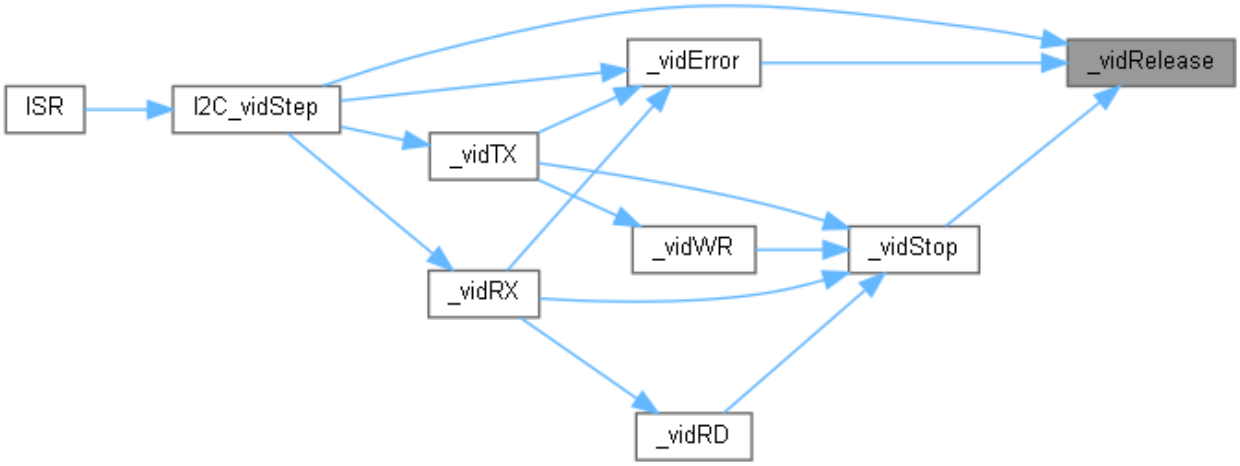
Here is the call graph for this function:

Here is the caller graph for this function:

void _vidRelease (void)

```
61                      {
62      I2C_DEF(LBTY_SET, TWEA);
63      strBuffer_GLB.m_u8Status = I2C_IDLE;
64  }
```

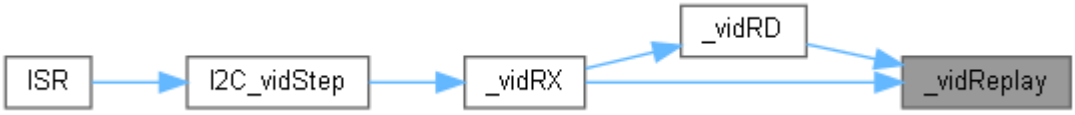
Here is the caller graph for this function:



void _vidReplay (void)

```
80                          {
81      if(strBuffer_GLB.m_u8Idx < (strBuffer_GLB.m_u8Size /*- 1*/)){
82          I2C_ACK();
83      }else{
84          I2C_NACK();
85      }
86  }
```

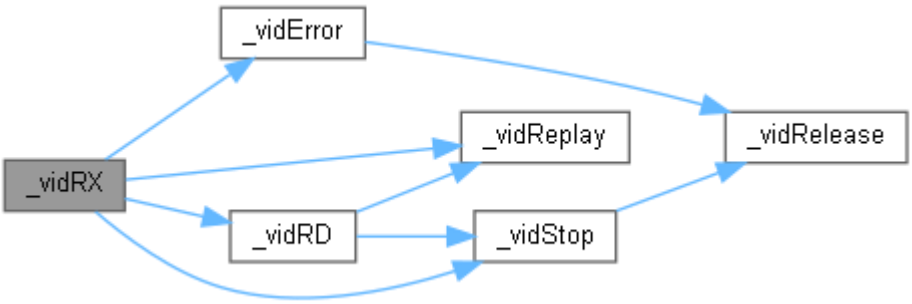
Here is the caller graph for this function:



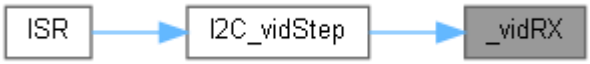
void _vidRX (u8 *u8State*)

```
127                             {
128     switch(u8State){
129         case I2C_MR_SLA_ACK:
130         case I2C_SR_SLA_ACK:
131         case I2C_SR_GCALL_ACK:
132             _vidReplay();
133             break;
134         case I2C_MR_DATA_ACK:
135         case I2C_SR_DATA_ACK:
136         case I2C_SR_GCALL_Data_ACK:
137             _vidRD();
138             break;
139         case I2C_MR_DATA_NACK:
140         case I2C_SR_DATA_NACK:
141         case I2C_SR_GCALL_Data_NACK:
142         case I2C_ST_LAST_Data:
143             _vidRD();
144             _vidStop();
145             break;
146         case I2C_MR_SLA_NACK:
147         default:
148             _vidError();
149     }
150 }
```

Here is the call graph for this function:



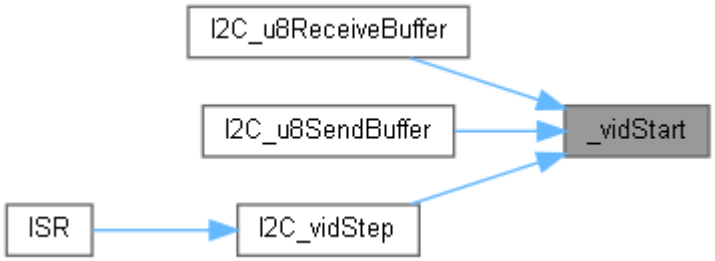
Here is the caller graph for this function:



void _vidStart (void)

```
42                          {
43      if(I2C_MODE == I2C_Master){
44          I2C_START();
45      }else if(I2C_MODE == I2C_Slave){
46          I2C_ACK();
47      }
48
49      strBuffer_GLB.m_u8Status = I2C_START;
50  }
```

Here is the caller graph for this function:

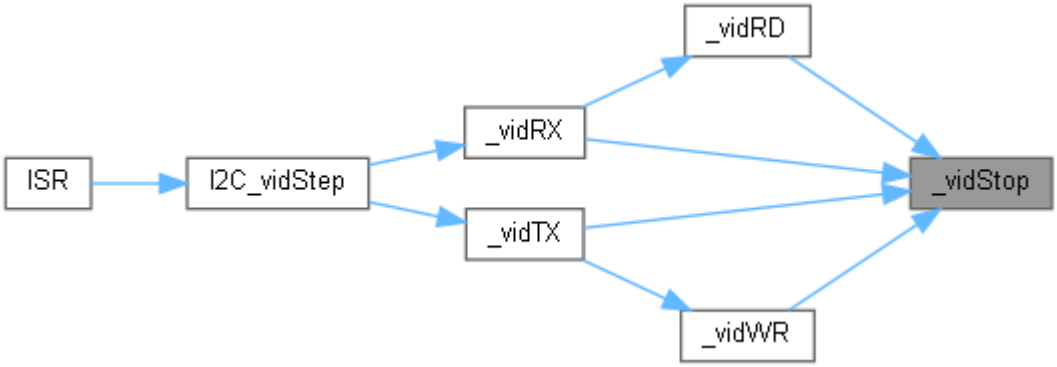


void _vidStop (void)

```
51                          {
52      I2C_STOP();
53      while(S_TWI->m_TWCR.sBits.m_TWSTO);
54      S_TWI->m_TWCR.sBits.m_TWSTO = LBTY_SET;
55      strBuffer_GLB.m_u8Status = I2C_STOP;
56
57  //  if(GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK) == I2C_Stop){
58          _vidRelease();
59  //  }
60  }
```

Here is the call graph for this function:

Here is the caller graph for this function:

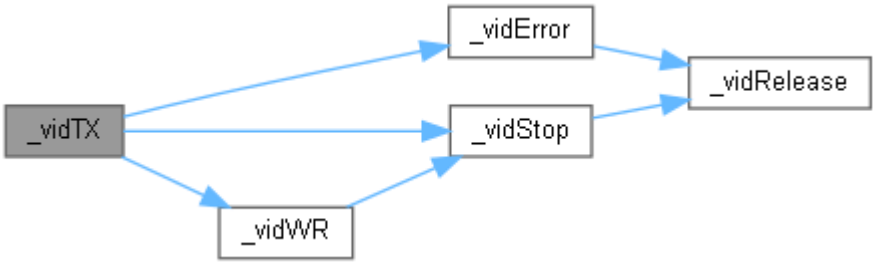


void _vidTX (u8 *u8State*)

```
109                         {
110     switch(u8State){
111         case I2C_MT_SLA_ACK:
112         case I2C_MT_DATA_ACK:
113         case I2C_ST_SLA_ACK:
114         case I2C_ST_DATA_ACK:
115         case I2C_MT_SLA_NACK:
116             _vidWR();
117             break;
118         case I2C_MT_DATA_NACK:
119         case I2C_ST_DATA_NACK:
```

```
120         case I2C_ST_LAST_Data:
121             _vidStop();
122             break;
123         default:
124             _vidError();
125     }
126 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

void _vidWR (void)

```
87                  {
88     if(strBuffer_GLB.m_u8Idx < strBuffer_GLB.m_u8Size){
89         S_TWI->m_TWDR = strBuffer_GLB.m_pu8Data[strBuffer_GLB.m_u8Idx++];
90         if(I2C_MODE == I2C_Master){
91             I2C_NACK();
92         }else if(I2C_MODE == I2C_Slave){
93             I2C_ACK();
94         }
95         strBuffer_GLB.m_u8Status = I2C_DATA;
96     }else{
97         _vidStop();
98     }
99 }
```

Here is the call graph for this function:

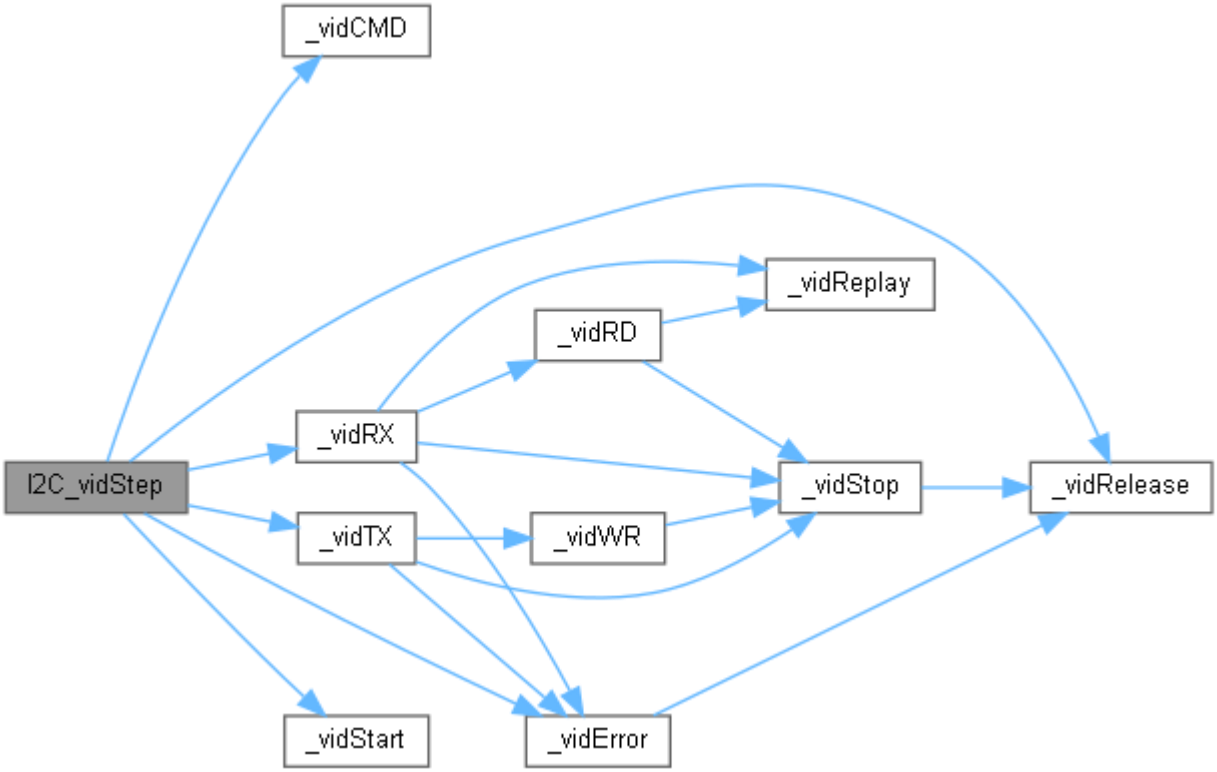
Here is the caller graph for this function:

void I2C_vidStep (void)

```
152                     {
153     volatile u8 u8State = GET_MASK(S_TWI->m_TWSR.u_Reg, I2C_STATUS_MASK);
154     switch(u8State) {
155         case I2C_Start:                 // Start condition transmitted
156         case I2C_RepeatStart:           // Repeated start condition transmitted
157
158             _vidCMD();
159             break;
160 /* ************************************************************************** */
161         case I2C_MT_SLA_ACK:            // SLA+W transmitted, ACK received = Slave
receiver ACKed address
162         case I2C_MT_DATA_ACK:           // Data transmitted, ACK received = Slave
receiver ACKed data
163         case I2C_ST_SLA_ACK:            // SLA+R received, ACK returned =
Addressed, returned ACK
164         case I2C_ST_DATA_ACK:           // Data transmitted, ACK received
165
166         case I2C_MT_SLA_NACK:           // SLA+W transmitted, NACK received =
Slave receiver with transmitted address doesn't exists?
167         case I2C_MT_DATA_NACK:          // Data transmitted, NACK received
168         case I2C_ST_DATA_NACK:          // Data transmitted, NACK received =
Received NACK, so we are done
169
170             _vidTX(u8State);
171             break;
```

```
172 /* ************************************************************************ */
173         case I2C_MR_SLA_ACK:            // SLA+R transmitted, ACK received = Slave
transmitter ACKed address
174         case I2C_MR_DATA_ACK:           // Data received, ACK returned
175         case I2C_SR_SLA_ACK:            // SLA+W received, ACK returned =
Addressed, returned ACK
176         case I2C_SR_DATA_ACK:           // Data received, ACK returned
177         case I2C_SR_GCALL_ACK:          // General call received, ACK returned =
Addressed generally, returned ACK
178         case I2C_SR_GCALL_Data_ACK:     // General call data received, ACK
returned
179
180         case I2C_MR_SLA_NACK:           // SLA+R transmitted, NACK received =
Slave transmitter with transmitted address doesn't exists?
181         case I2C_MR_DATA_NACK:          // Data received, NACK returned
182         case I2C_SR_DATA_NACK:          // Data received, NACK returned
183         case I2C_SR_GCALL_Data_NACK:    // General call data received, NACK
returned
184
185         case I2C_ST_LAST_Data:          // Last data byte transmitted, ACK
received = Received ACK, but we are done already!
186
187             vidRX(u8State);
188             break;
189 /* ************************************************************************ */
190         case I2C_MT_ARB_LOST:           // Arbitration lost in SLA+W or data
191 //      case I2C_MR_ARB_LOST:           // Arbitration lost in SLA+R or NACK
192         case I2C_ST_ARB_LOST:           // Arbitration lost in SLA+RW, SLA+R
received, ACK returned
193         case I2C_SR_SLA_ARB_LOST:       // Arbitration lost in SLA+RW, SLA+W
received, ACK returned
194         case I2C_SR_GCALL_ARB_LOST:     // Arbitration lost in SLA+RW, general
call received, ACK returned
195
196             vidStart();
197             break;
198 /* ************************************************************************ */
199         case I2C_Stop:                  // Stop or repeated start condition
received while selected
200
201             vidRelease();
202             break;
203 /* ************************************************************************ */
204         case I2C_NoInfo:                // No state information available
205         case I2C_Bus_Error:             // Bus error; Illegal START or STOP
condition
206         default:
207
208             vidError();
209     }
210 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

I2C_priv.h

Go to the documentation of this file.

```
1 /*
2 /* ********************** FILE DEFINITION SECTION ************************** */
3 /* ************************************************************************** */
4 /* File Name   : I2C_priv.h                                                   */
5 /* Author      : MAAM                                                         */
6 /* Version     : v01.2                                                        */
7 /* date        : Apr 13, 2023                                                 */
8 /* ************************************************************************** */
9 /* ************************ HEADER FILES INCLUDES ************************** */
10 /* ************************************************************************** */
11
12 #ifndef I2C_PRIV_H_
13 #define I2C_PRIV_H_
14
15 /* ************************************************************************** */
16 /* ********************** TYPE_DEF/STRUCT/ENUM SECTION ********************** */
17 /* ************************************************************************** */
18
21 typedef enum{
22     I2C_IDLE = (u8)0u,
23     I2C_START,
24     I2C_ADDRESS,
25 //  I2C_RW,
26 //  I2C_ACK,
27     I2C_DATA,
28     I2C_STOP,
29     I2C_ERROR
30 }I2C_tenuStatus;
31
34 typedef union{
35     u8 u_Reg;
36     struct {
37         __IO u8 m_OP  : 1;
38         __IO u8 m_ADD : 7;
39     }sBits;
40 }I2C_tstrAddress;
43 typedef struct{
44     I2C_tstrAddress  m_u8Add;
45     pu8 m_pu8Data;
46     u8  m_u8Size;
47     u8  m_u8Idx;
48     u8  m_u8Status;
49 }I2C_tstrBuffer;
50
51 /*****************************************************************************/
52
55 typedef union{
56     u8 u_Reg;
57     struct {
58         __IO u8 m_TWIE : 1;
59         __IO u8        : 1;
60         __IO u8 m_TWEN : 1;
61         __I  u8 m_TWWC : 1;
62         __IO u8 m_TWSTO: 1;
63         __IO u8 m_TWSTA: 1;
64         __IO u8 m_TWEA : 1;
65         __IO u8 m_TWINT: 1;
66     }sBits;
67 }TWCR_type;
68
69 /*****************************************************************************/
70
73 typedef union{
74     u8 u_Reg;
75     struct {
76         __IO u8 m_TWPS : 2;
77         __IO u8        : 1;
78         __I  u8 m_TWS  : 5;
79     }sBits;
80 }TWSR_type;
81
82 /*****************************************************************************/
```

```
83
86 typedef union{
87     u8 u_Reg;
88     struct {
89         __IO u8 m_TWGCE : 1;
90         __IO u8 m_TWA   : 7;
91     }sBits;
92 }TWAR_type;
93
94 /*****************************************************************************/
95
98 typedef struct{
99      __IO u8          m_TWBR;
100     __IO TWSR_type   m_TWSR;
101     __IO TWAR_type   m_TWAR;
102     __IO u8          m_TWDR;
103     __I  u8          REVERSE[50];
104     __IO TWCR_type   m_TWCR;
105 }TWI_type;
106
107 /* ************************************************************************** */
108 /* ************************* MACRO/DEFINE SECTION *************************** */
109 /* ************************************************************************** */
110
112 #define S_TWI           ((TWI_type* const)0x20U)
113 #define TWBR            (*(volatile u8* const)0x20U)
114 #define TWSR            (*(volatile u8* const)0x21U)
115 #define TWAR            (*(volatile u8* const)0x22U)
116 #define TWDR            (*(volatile u8* const)0x23U)
117
118 #define TWCR            (*(volatile u8* const)0x56U)
119
120 /* ************************************************************************** */
121
122 #define TWIE                        0
123 #define TWEN                        2
124 #define TWWC                        3
125 #define TWSTO                       4
126 #define TWSTA                       5
127 #define TWEA                        6
128 #define TWINT                       7
129
130 #define I2C_STATUS_MASK             0xF8u
131
132 #define I2C_PORT                    C
133 #define I2C_SDA_PIN                 GPIO_I2C_SDA
134 #define I2C_SCL_PIN                 GPIO_I2C_SCL
135
136 #define I2C_START()         ( S_TWI->m_TWCR.u_Reg = (_BV(TWINT) | _BV(TWEN) |
_BV(TWIE) | _BV(TWSTA)) )
137 #define I2C_ACK()           ( S_TWI->m_TWCR.u_Reg = (_BV(TWINT) | _BV(TWEN) |
_BV(TWIE) | _BV(TWEA) ) )
138 #define I2C_NACK()          ( S_TWI->m_TWCR.u_Reg = (_BV(TWINT) | _BV(TWEN) |
_BV(TWIE)             ) )
139 #define I2C_STOP()          ( S_TWI->m_TWCR.u_Reg = (_BV(TWINT) | _BV(TWEN) |
_BV(TWIE) | _BV(TWSTO)) )
140 #define I2C_DEF(value, bit) ( S_TWI->m_TWCR.u_Reg = (_BV(TWINT) | _BV(TWEN) |
(value<<(bit))        ) )
141
142 /* ************************************************************************** */
143 /* ************************** CONST SECTION ********************************* */
144 /* ************************************************************************** */
145
146 /* ************************************************************************** */
147 /* ************************** VARIABLE SECTION ****************************** */
148 /* ************************************************************************** */
149
150 /* ************************************************************************** */
151 /* ************************** FUNCTION SECTION ****************************** */
152 /* ************************************************************************** */
153
154 void _vidStart(void);
155 void _vidStop(void);
156 void _vidRelease(void);
157 void _vidError(void);
158 void _vidCMD(void);
159 void _vidReplay(void);
```

```
160 void  _vidWR(void);
161 void  _vidRD(void);
162
163 void  _vidTX(u8 u8State);
164 void  _vidRX(u8 u8State);
165
166 void I2C_vidStep(void);
167
168 #endif /* I2C_PRIV_H_ */
169 /*************************** E N D (I2C_priv.h) ****************************/
```

main.c File Reference