SWC_GPIO

Version v1.0
7/12/2023 2:16:00 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:

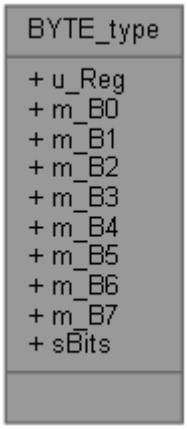
Data Structure Documentation

BYTE_type Union Reference

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

```
#include <GPIO_priv.h>
```

Collaboration diagram for BYTE_type:



Data Fields

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

Detailed Description

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

Type : Union **Unit** : None

Field Documentation

__IO u8 m_B0

Bit 0 "LSB"

__IO u8 m_B1

Bit 1

__IO u8 m_B2

Bit 2

__IO u8 m_B3

Bit 3

__IO u8 m_B4

Bit 4

__IO u8 m_B5

Bit 5

__IO u8 m_B6

Bit 6

__IO u8 m_B7

Bit 7 "MSB"

struct { ... } sBits

All Bits of the Byte

u8 u_Reg

Byte

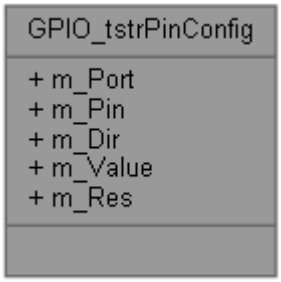
The documentation for this union was generated from the following file:

GPIO_priv.h

GPIO_tstrPinConfig Struct Reference

: type define of structure for GPIO pin Configuration

`#include <GPIO_int.h>`
Collaboration diagram for GPIO_tstrPinConfig:



Data Fields

- GPIO_tenuPortNum m_Port
- GPIO_tenuPinNum m_Pin
- GPIO_tenuDataDirection m_Dir
- GPIO_tenuDataStatus m_Value
- GPIO_tenuInputRes m_Res

Detailed Description

: type define of structure for GPIO pin Configuration

Type : struct **Unit** : None

Field Documentation

GPIO_tenuDataDirection m_Dir

Data Direction

GPIO_tenuPinNum m_Pin

Pin Number

GPIO_tenuPortNum m_Port

Port Number

GPIO_tenuInputRes m_Res

Input Pull Resistor

GPIO_tenuDataStatus m_Value

Pin State value

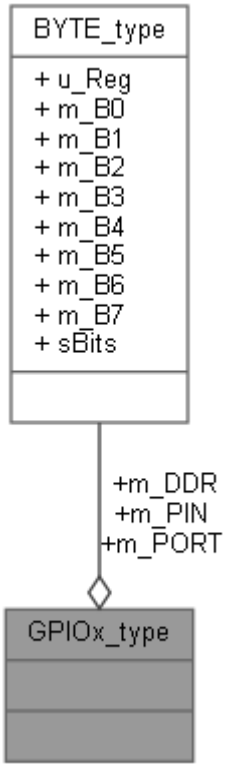
The documentation for this struct was generated from the following file:

GPIO_int.h

GPIOx_type Struct Reference

: General Purpose Input Output Registers

`#include <GPIO_priv.h>`
Collaboration diagram for GPIOx_type:



Data Fields

- __I BYTE_type m_PIN
- __IO BYTE_type m_DDR
- __IO BYTE_type m_PORT

Detailed Description

: General Purpose Input Output Registers

Type : Struct **Unit** : None

Field Documentation

__IO BYTE_type m_DDR

Data Direction Register

__I BYTE_type **m_PIN**

Pins Input Register

__IO BYTE_type **m_PORT**

Pins Output Register

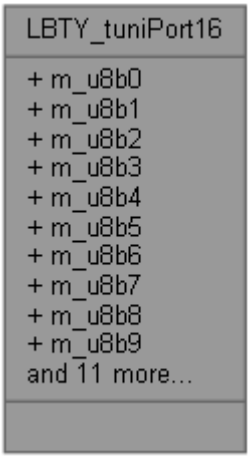
The documentation for this struct was generated from the following file:

GPIO_priv.h

LBTY_tuniPort16 Union Reference

`#include <LBTY_int.h>`

Collaboration diagram for LBTY_tuniPort16:



Data Fields

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

Field Documentation

u8 m_u8b0

u8 m_u8b1

u8 m_u8b10

u8 m_u8b11

u8 m_u8b12

u8 m_u8b13

u8 m_u8b14

u8 m_u8b15

u8 m_u8b2

u8 m_u8b3

u8 m_u8b4

u8 m_u8b5

u8 m_u8b6

u8 m_u8b7

u8 m_u8b8

u8 m_u8b9

u8 m_u8high

u8 m_u8low

struct { ... } sBits

struct { ... } sBytes

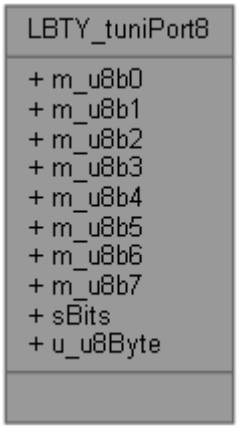
u16 u_u16Word

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

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

LBTY_tuniPort8 Union Reference

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



Data Fields

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

Detailed Description

Union Byte bit by bit

Field Documentation

u8 m_u8b0

u8 m_u8b1

u8 m_u8b2

u8 m_u8b3

u8 m_u8b4

u8 m_u8b5

u8 m_u8b6

u8 m_u8b7

struct { ... } sBits

u8 u_u8Byte

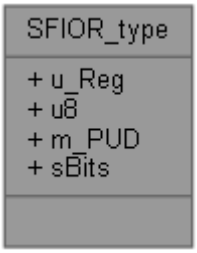
The documentation for this union was generated from the following file:

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

SFIOR_type Union Reference

: Special Function I/O Register

`#include <GPIO_priv.h>`
Collaboration diagram for SFIOR_type:



Data Fields

- u8 u_Reg
- struct {
- __IO u8: 2
- __IO u8 m_PUD: 1
- } sBits

Detailed Description

: Special Function I/O Register

Type : Union **Unit** : None

Field Documentation

__IO u8 m_PUD

Pull-up disable

struct { ... } sBits

All Bits of the Byte

__IO u8

Reversed

u8 u_Reg

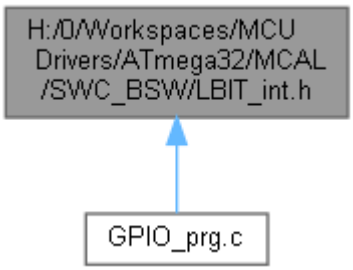
Byte

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

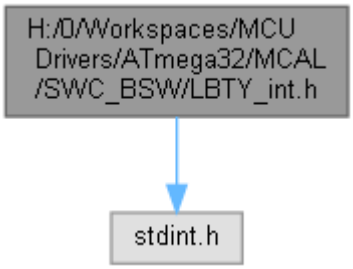
```
1 /*
******************************************************************************* */
2 /* ********************** FILE DEFINITION SECTION ************************** */
3 /* ******************************************************************************* */
4 /* File Name   : LBIT_int.h         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

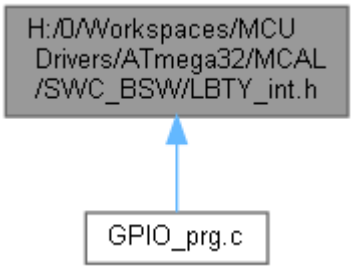
H:/0/Workspaces/MCU Drivers/ATmega32/MCAL/SWC_BSW/LBTY_int.h File Reference

```
#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.

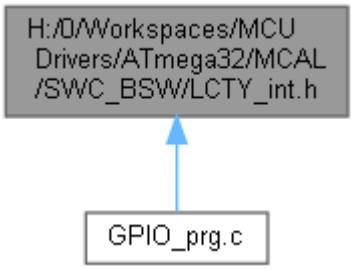
```
1 /*
2 /* ********************** FILE DEFINITION SECTION ************************** */
3 /* ************************************************************************* */
4 /* File Name   : LBTY_int.h                                                  */
5 /* Author      : MAAM                      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

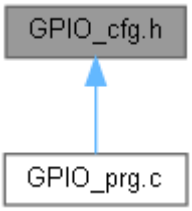
Go to the documentation of this file.

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

GPIO_cfg.c File Reference

GPIO_cfg.h File Reference

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



Enumerations

- enum GPIO_tuenuGPIO_PORTA { GPIO_ADC0 = (u8)0u, GPIO_ADC1, GPIO_ADC2, GPIO_ADC3, GPIO_ADC4, GPIO_ADC5, GPIO_ADC6, GPIO_ADC7 }
- enum GPIO_tuenuGPIO_PORTB { GPIO_TMR_EXT0_IN = (u8)0u, GPIO_TMR_EXT1_IN, GPIO_INT2, GPIO_TMR_OC0, GPIO_SPI_SS, GPIO_SPI_MOSI, GPIO_SPI_MISO, GPIO_SPI_SCK, GPIO_USART_XCK = GPIO_TMR_EXT0_IN, GPIO_AIN0 = GPIO_INT2, GPIO_AIN1 = GPIO_TMR_OC0 }
- enum GPIO_tuenuGPIO_PORTC { GPIO_I2C_SCL = (u8)0u, GPIO_I2C_SDA, GPIO_JTAG_TCK, GPIO_JTAG_TMS, GPIO_JTAG_TDO, GPIO_JTAG_TDI, GPIO_TMR_OSC1, GPIO_TMR_OSC2 }
- enum GPIO_tuenuGPIO_PORTD { GPIO_UART_RX = (u8)0u, GPIO_UART_TX, GPIO_INT0, GPIO_INT1, GPIO_TMR_OC1B, GPIO_TMR_OC1A, GPIO_TMR_ICP1, GPIO_TMR_OC2 }

Enumeration Type Documentation

enum GPIO_tuenuGPIO_PORTA

Enumerator:

GPIO_ADC0	
GPIO_ADC1	
GPIO_ADC2	
GPIO_ADC3	
GPIO_ADC4	
GPIO_ADC5	
GPIO_ADC6	
GPIO_ADC7	

```
190             {
191     GPIO_ADC0 = (u8)0u,
192     GPIO_ADC1,
193     GPIO_ADC2,
194     GPIO_ADC3,
195     GPIO_ADC4,
196     GPIO_ADC5,
197     GPIO_ADC6,
198     GPIO_ADC7
199 }GPIO_tuenuGPIO_PORTA;
```

enum GPIO_tuenuGPIO_PORTB

Enumerator:

GPIO_TMR_EXT0_IN	
GPIO_TMR_EXT	

1_IN	
GPIO_INT2	
GPIO_TMR_OC0	
GPIO_SPI_SS	
GPIO_SPI_MOSI	
GPIO_SPI_MISO	
GPIO_SPI_SCK	
GPIO_USART_XCK	
GPIO_AIN0	
GPIO_AIN1	

```
201             {
202     GPIO_TMR_EXT0_IN = (u8)0u,
203     GPIO_TMR_EXT1_IN,
204     GPIO_INT2,
205     GPIO_TMR_OC0,
206     GPIO_SPI_SS,
207     GPIO_SPI_MOSI,
208     GPIO_SPI_MISO,
209     GPIO_SPI_SCK,
210
211     GPIO_USART_XCK = GPIO_TMR_EXT0_IN,
212     GPIO_AIN0 = GPIO_INT2,
213     GPIO_AIN1 = GPIO_TMR_OC0,
214 }GPIO_tuenuGPIO_PORTB;
```

enum GPIO_tuenuGPIO_PORTC

Enumerator:

GPIO_I2C_SCL	
GPIO_I2C_SDA	
GPIO_JTAG_TCK	
GPIO_JTAG_TMS	
GPIO_JTAG_TDO	
GPIO_JTAG_TDI	
GPIO_TMR_OSC1	
GPIO_TMR_OSC2	

```
216             {
217     GPIO_I2C_SCL = (u8)0u,
218     GPIO_I2C_SDA,
219     GPIO_JTAG_TCK,
220     GPIO_JTAG_TMS,
221     GPIO_JTAG_TDO,
222     GPIO_JTAG_TDI,
223     GPIO_TMR_OSC1,
224     GPIO_TMR_OSC2
225 }GPIO_tuenuGPIO_PORTC;
```

enum GPIO_tuenuGPIO_PORTD

Enumerator:

GPIO_UART_RX	
GPIO_UART_TX	
GPIO_INT0	
GPIO_INT1	
GPIO_TMR_OC1B	

GPIO_TMR_OC1A	
GPIO_TMR_ICP1	
GPIO_TMR_OC2	

```
227              {
228     GPIO_UART_RX = (u8)0u,
229     GPIO_UART_TX,
230     GPIO_INT0,
231     GPIO_INT1,
232     GPIO_TMR_OC1B,
233     GPIO_TMR_OC1A,
234     GPIO_TMR_ICP1,
235     GPIO_TMR_OC2
236 }GPIO_tuenuGPIO_PORTD;
```

GPIO_cfg.h

```
Go to the documentation of this file.1 /*
****************************************************************************** */
2 /* ********************** FILE DEFINITION SECTION ************************** */
3 /* ****************************************************************************** */
4 /* File Name   : GPIO_cfg.h                                                   */
5 /* Author      : MAAM                                                         */
6 /* Version     : v01.2                                                        */
7 /* date        : Mar 23, 2023                                                 */
8 /* ****************************************************************************** */
9 /* ************************ HEADER FILES INCLUDES ************************  */
10 /* ****************************************************************************** */
11
12 #ifndef SWC_GPIO_GPIO_CFG_H_
13 #define SWC_GPIO_GPIO_CFG_H_
14
15 /* ****************************************************************************** */
16 /* ********************** TYPE_DEF/STRUCT/ENUM SECTION ********************** */
17 /* ****************************************************************************** */
18
19 #if defined(AMIT_KIT)
20 typedef enum {
21     AMIT_LCD0 = (u8)0u,
22     AMIT_LCD1,
23     AMIT_LCD2,
24     AMIT_LCD3,
25     AMIT_LCD4,
26     AMIT_LCD5,
27     AMIT_LCD6,
28     AMIT_LCD7
29 }GPIO_tuenuAMIT_PORTA;
30
31 typedef enum {
32     AMIT_B0 = (u8)0u,
33     AMIT_LCD_RS,
34     AMIT_LCD_RW,
35     AMIT_LCD_EN,
36     AMIT_B4,
37     AMIT_B5,
38     AMIT_B6,
39     AMIT_B7
40 }GPIO_tuenuAMIT_PORTB;
41
42 typedef enum {
43     AMIT_C0 = (u8)0u,
44     AMIT_C1,
45     AMIT_7Seg_COM0,
46     AMIT_7Seg_COM1,
47     AMIT_7Seg_A,
48     AMIT_7Seg_B,
49     AMIT_7Seg_C,
50     AMIT_7Seg_D
51 }GPIO_tuenuAMIT_PORTC;
52
53 typedef enum {
54     AMIT_PUSH0 = (u8)0u,
55     AMIT_PUSH1,
56     AMIT_PUSH2,
57     AMIT_RELAY0,
58     AMIT_RELAY1,
59     AMIT_LED0,
60     AMIT_LED1,
61     AMIT_LED2,
62
63     AMIT_BUZZER = (u8)4u
64 }GPIO_tuenuAMIT_PORTD;
65 #elif defined(ETA32_KIT)
66 typedef enum {
67     Eta32_LDR = (u8)0u,
68     Eta32_LM35,
69     Eta32_LCD_EN,
70     Eta32_LCD_RS,
71     Eta32_LED_G,
72     Eta32_LED_B,
```

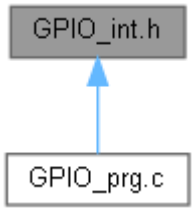
```
73      Eta32_LED_Y,
74      Eta32_RELAY1,
75
76      Eta32_VR1 = (u8)0u,
77      Eta32_VR2,
78      Eta32_7Seg_COM2,
79      Eta32_7Seg_COM3
80  }GPIO_tuenuEta32_PORTA;
81
82  typedef enum {
83      Eta32_7Seg_A = (u8)0u,
84      Eta32_7Seg_B,
85      Eta32_7Seg_C,
86
87      Eta32_7Seg_D = (u8)4,
88      Eta32_7Seg_COM1,
89      Eta32_7Seg_COM0,
90      Eta32_LED_R,
91
92      Eta32_LCD4 = (u8)0u,
93      Eta32_LCD5,
94      Eta32_LCD6,
95
96      Eta32_LCD7 = (u8)4u,
97      Eta32_ISP_MOSI,
98      Eta32_ISP_MISO,
99      Eta32_ISP_SCK,
100
101      Eta32_DC_Motor_PWM = (u8)3u,
102
103     Eta32_DC_Motor_DIR = (u8)5u,
104     Eta32_DC_Motor_ENA = (u8)6u
105
106 }GPIO_tuenuEta32_PORTB;
107
108 typedef enum {
109     Eta32_I2C_SCL = (u8)0u,
110     Eta32_I2C_SDA,
111     Eta32_JTAG_TCK,
112     Eta32_JTAG_TMS,
113     Eta32_JTAG_TDO,
114     Eta32_JTAG_TDI,
115     Eta32_BUZZER,
116     Eta32_RELAY0,
117
118     Eta32_Keypad_Row3 = (u8)2u,
119     Eta32_Keypad_Row2,
120     Eta32_Keypad_Row1,
121     Eta32_Keypad_Row0
122 }GPIO_tuenuEta32_PORTC;
123
124 typedef enum {
125     Eta32_UART_RX = (u8)0u,
126     Eta32_UART_TX,
127     Eta32_IR_Receiver,
128     Eta32_Keypad_col3,
129     Eta32_Analog_Out,
130     Eta32_Keypad_col2,
131     Eta32_Keypad_col1,
132     Eta32_Keypad_col0
133 }GPIO tuenuEta32 PORTD;
134 #elif defined(ETA32_MINI_KIT)
135 typedef enum {
136     Eta32_mini_VR1 = (u8)0u,
137     Eta32_mini_7Seg_A,
138     Eta32_mini_7Seg_B,
139     Eta32_mini_7Seg_C,
140     Eta32_mini_7Seg_D,
141     Eta32_mini_7Seg_E,
142     Eta32_mini_7Seg_F,
143     Eta32_mini_7Seg_G,
144
145     Eta32_mini_LCD_RS = (u8)1u,
146     Eta32_mini_LCD_EN,
147     Eta32_mini_LCD4,
148     Eta32_mini_LCD5,
149     Eta32_mini_LCD6,
```

```
150     Eta32_mini_LCD7,
151 }GPIO_tuenuEta32_mini_PORTA;
152 
153 typedef enum {
154     Eta32_mini_7Seg_Dot = (u8)0u,
155 
156     Eta32_mini_Keypad_Row0 = (u8)4u,
157     Eta32_mini_Keypad_Row1,
158     Eta32_mini_Keypad_Row2,
159     Eta32_mini_Keypad_Row3,
160 
161     Eta32_mini_ISP_MOSI = (u8)5,
162     Eta32_mini_ISP_MISO,
163     Eta32_mini_ISP_SCK,
164 }GPIO_tuenuEta32_mini_PORTB;
165 
166 typedef enum {
167     Eta32_mini_LED_R = (u8)0u,
168     Eta32_mini_LED_G,
169     Eta32_mini_LED_B,
170     Eta32_mini_RELAY1,
171     Eta32_mini_RELAY0,
172     Eta32_mini_BUZZER,
173     Eta32_mini_7Seg_COM0,
174     Eta32_mini_7Seg_COM1,
175 
176     Eta32_mini_I2C_SCL = (u8)0u,
177     Eta32_mini_I2C_SDA
178 }GPIO_tuenuEta32_mini_PORTC;
179 
180 typedef enum {
181     Eta32_UART_RX = (u8)0u,
182     Eta32_UART_TX,
183     Eta32_Keypad_col0,
184     Eta32_Keypad_col1,
185     Eta32_Keypad_col2,
186     Eta32_Keypad_col3,
187 }GPIO_tuenuEta32_mini_PORTD;
188 #endif
189 
190 typedef enum {
191     GPIO_ADC0 = (u8)0u,
192     GPIO_ADC1,
193     GPIO_ADC2,
194     GPIO_ADC3,
195     GPIO_ADC4,
196     GPIO_ADC5,
197     GPIO_ADC6,
198     GPIO_ADC7
199 }GPIO_tuenuGPIO_PORTA;
200 
201 typedef enum {
202     GPIO_TMR_EXT0_IN = (u8)0u,
203     GPIO_TMR_EXT1_IN,
204     GPIO_INT2,
205     GPIO_TMR_OC0,
206     GPIO_SPI_SS,
207     GPIO_SPI_MOSI,
208     GPIO_SPI_MISO,
209     GPIO_SPI_SCK,
210 
211     GPIO_USART_XCK = GPIO_TMR_EXT0_IN,
212     GPIO_AIN0 = GPIO_INT2,
213     GPIO_AIN1 = GPIO_TMR_OC0,
214 }GPIO_tuenuGPIO_PORTB;
215 
216 typedef enum {
217     GPIO_I2C_SCL = (u8)0u,
218     GPIO_I2C_SDA,
219     GPIO_JTAG_TCK,
220     GPIO_JTAG_TMS,
221     GPIO_JTAG_TDO,
222     GPIO_JTAG_TDI,
223     GPIO_TMR_OSC1,
224     GPIO_TMR_OSC2
225 }GPIO_tuenuGPIO_PORTC;
226 
```

```
227 typedef enum {
228     GPIO_UART_RX = (u8)0u,
229     GPIO_UART_TX,
230     GPIO_INT0,
231     GPIO_INT1,
232     GPIO_TMR_OC1B,
233     GPIO_TMR_OC1A,
234     GPIO_TMR_ICP1,
235     GPIO_TMR_OC2
236 }GPIO_tuenuGPIO_PORTD;
237
238 /* ********************************************************************************* */
239 /* ************************* MACRO/DEFINE SECTION ************************** */
240 /* ********************************************************************************* */
241
242 /* ********************************************************************************* */
243 /* ***************************** CONST SECTION ***************************** */
244 /* ********************************************************************************* */
245
246 /* ********************************************************************************* */
247 /* ***************************** VARIABLE SECTION ************************** */
248 /* ********************************************************************************* */
249
250 /* ********************************************************************************* */
251 /* ***************************** FUNCTION SECTION ************************** */
252 /* ********************************************************************************* */
253
254 #endif /* SWC_GPIO_GPIO_CFG_H_ */
255 /***************************** E N D (GPIO_cfg.h) ******************************/
```

GPIO_int.h File Reference

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



Data Structures

struct GPIO_tstrPinConfig*: type define of structure for GPIO pin Configuration*

Enumerations

- enum GPIO_tenuPinNum { Pin_0 = (u8)0, Pin_1, Pin_2, Pin_3, Pin_4, Pin_5, Pin_6, Pin_7, Pin_MaxNum }
- enum GPIO_tenuPortNum { A = (u8)0, B, C, D, Port_Num }
- enum GPIO_tenuDataDirection { PIN_INPUT = (u8)0u, PIN_OUTPUT = (u8)1u, PORT_INPUT = (u8)0x00U, PORT_OUTPUT = (u8)0xFFU }
- enum GPIO_tenuDataStatus { PIN_Low = (u8)0u, PIN_High = (u8)1u, PORT_Low = (u8)0x00U, PORT_High = (u8)0xFFU }
- enum GPIO_tenuInputRes { PULL_DOWN = (u8)0u, PULL_UP = (u8)1u, PULL_None }

Functions

- void GPIO_vidInit (void)
- LBTY_tenuErrorStatus GPIO_u8PinInit (GPIO_tstrPinConfig u8PinConfig)
- LBTY_tenuErrorStatus GPIO_u8SetPinDirection (GPIO_tenuPortNum u8PortNum, GPIO_tenuPinNum u8PinNum, GPIO_tenuDataDirection u8PinDir)
- LBTY_tenuErrorStatus GPIO_u8SetRangeDirection (GPIO_tenuPortNum u8PortNum, GPIO_tenuPinNum u8StartPin, GPIO_tenuPinNum u8EndPin, GPIO_tenuDataDirection u8PinDir)
- LBTY_tenuErrorStatus GPIO_u8SetMaskDirection (GPIO_tenuPortNum u8PortNum, u8 u8PortMask, GPIO_tenuDataDirection u8PortDir)
- LBTY_tenuErrorStatus GPIO_u8SetPortDirection (GPIO_tenuPortNum u8PortNum, GPIO_tenuDataDirection u8PortDir)
- LBTY_tenuErrorStatus GPIO_u8SetPinValue (GPIO_tenuPortNum u8PortNum, GPIO_tenuPinNum u8PinNum, GPIO_tenuDataStatus u8PinVlaue)
- LBTY_tenuErrorStatus GPIO_u8SetRangeValue (GPIO_tenuPortNum u8PortNum, GPIO_tenuPinNum u8StartPin, GPIO_tenuPinNum u8EndPin, GPIO_tenuDataStatus u8PinValue)
- LBTY_tenuErrorStatus GPIO_u8SetMaskValue (GPIO_tenuPortNum u8PortNum, u8 u8PortMask, GPIO_tenuDataStatus u8PortValue)
- LBTY_tenuErrorStatus GPIO_u8SetPortValue (GPIO_tenuPortNum u8PortNum, GPIO_tenuDataStatus u8PortValue)
- LBTY_tenuErrorStatus GPIO_u8GetPinValue (GPIO_tenuPortNum u8PortNum, GPIO_tenuPinNum u8PinNum, pu8 pu8Value)
- LBTY_tenuErrorStatus GPIO_u8GetRangeValue (GPIO_tenuPortNum u8PortNum, GPIO_tenuPinNum u8StartPin, GPIO_tenuPinNum u8EndPin, pu8 pu8Value)
- LBTY_tenuErrorStatus GPIO_u8GetMaskValue (GPIO_tenuPortNum u8PortNum, u8 u8PortMask, pu8 pu8Value)
- LBTY_tenuErrorStatus GPIO_u8GetPortValue (GPIO_tenuPortNum u8PortNum, pu8 pu8Value)
- LBTY_tenuErrorStatus GPIO_u8TogglePinValue (GPIO_tenuPortNum u8PortNum, GPIO_tenuPinNum u8PinNum)

- LBTY_tenuErrorStatus GPIO_u8ToggleRangeValue (GPIO_tenuPortNum u8PortNum, GPIO_tenuPinNum u8StartPin, GPIO_tenuPinNum u8EndPin)
- LBTY_tenuErrorStatus GPIO_u8ToggleMaskValue (GPIO_tenuPortNum u8PortNum, u8 u8PortMask)
- LBTY_tenuErrorStatus GPIO_u8TogglePortValue (GPIO_tenuPortNum u8PortNum)
- void GPIO_vidEnablePullUp (void)
- void GPIO_vidDisablePullUp (void)
- LBTY_tenuErrorStatus GPIO_u8SetPinPullUp (GPIO_tenuPortNum u8PortNum, GPIO_tenuPinNum u8PinNum, LBTY_tenuFlagStatus u8Pullup)

Enumeration Type Documentation

enum GPIO_tenuDataDirection

Enumerator:

PIN_INPUT	
PIN_OUTPUT	
PORT_INPUT	
PORT_OUTPUT	

```
39              {
40     PIN_INPUT = (u8)0u,
41     PIN_OUTPUT = (u8)1u,
42     PORT_INPUT = (u8)0x00U,
43     PORT_OUTPUT = (u8)0xFFU
44 }GPIO_tenuDataDirection;
```

enum GPIO_tenuDataStatus

Enumerator:

PIN_Low	
PIN_High	
PORT_Low	
PORT_High	

```
46              {
47     PIN_Low = (u8)0u,
48     PIN_High = (u8)1u,
49     PORT_Low = (u8)0x00U,
50     PORT_High = (u8)0xFFU
51 }GPIO_tenuDataStatus;
```

enum GPIO_tenuInputRes

Enumerator:

PULL_DOWN	
PULL_UP	
PULL_None	

```
53              {
54     PULL_DOWN = (u8)0u,
55     PULL_UP = (u8)1u,
56     PULL_None
57 }GPIO_tenuInputRes;
```

enum GPIO_tenuPinNum

Enumerator:

Pin_0	

Pin_1	
Pin_2	
Pin_3	
Pin_4	
Pin_5	
Pin_6	
Pin_7	
Pin_MaxNum	

```
19              {
20     Pin_0 = (u8)0,
21     Pin_1,
22     Pin_2,
23     Pin_3,
24     Pin_4,
25     Pin_5,
26     Pin_6,
27     Pin_7,
28     Pin_MaxNum
29 }GPIO_tenuPinNum;
```

enum GPIO_tenuPortNum

Enumerator:

A	
B	
C	
D	
Port_Num	

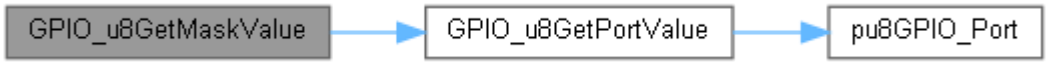
```
31              {
32     A = (u8)0,
33     B,
34     C,
35     D,
36     Port_Num
37 }GPIO_tenuPortNum;
```

Function Documentation

LBTY_tenuErrorStatus GPIO_u8GetMaskValue (GPIO_tenuPortNum *u8PortNum*, u8 *u8PortMask*, pu8 *pu8Value*)

```
375                                         {
376     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
377     if(!(u8RetErrorState = GPIO_u8GetPortValue(u8PortNum, pu8Value))){
378         *pu8Value &= u8PortMask;
379     }
380
381     return u8RetErrorState;
382 }
```

Here is the call graph for this function:



LBTY_tenuErrorStatus GPIO_u8GetPinValue (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuPinNum *u8PinNum*, pu8 *pu8Value*)

```
327                                                 {
328     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
329     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
330
331     if((GPIO == LBTY_NULL) || (pu8Value == LBTY_NULL)){
332         u8RetErrorState = LBTY_NULL_POINTER;
333     }else if((u8)u8PinNum < Pin_MaxNum){
334         *pu8Value = LBTY_u8ZERO;
335         *pu8Value = (u8)GET_BIT(GPIO->m_PIN.u_Reg, u8PinNum);
```

```
336     }else{
337         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
338     }
339
340     return u8RetErrorState;
341 }
```

Here is the call graph for this function:

LBTY_tenuErrorStatus GPIO_u8GetPortValue (GPIO_tenuPortNum *u8PortNum*, pu8 *pu8Value*)

```
390
{
391     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
392     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
393
394     if((GPIO == LBTY_NULL) || (pu8Value == LBTY_NULL)){
395         u8RetErrorState = LBTY_NULL_POINTER;
396     }else{
397         *pu8Value = (u8)GPIO->m_PIN.u_Reg;
398     }
399     return u8RetErrorState;
400 }
```

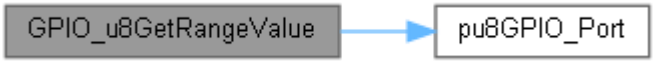
Here is the call graph for this function:

Here is the caller graph for this function:

LBTY_tenuErrorStatus GPIO_u8GetRangeValue (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuPinNum *u8StartPin*, GPIO_tenuPinNum *u8EndPin*, pu8 *pu8Value*)

```
350                                                                                 {
351     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
352     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
353
354     if((GPIO == LBTY_NULL) || (pu8Value == LBTY_NULL)){
355         u8RetErrorState = LBTY_NULL_POINTER;
356     }else if(((u8)u8StartPin < Pin_MaxNum) && ((u8)u8EndPin < Pin_MaxNum) &&
(u8EndPin > u8StartPin)){
357         *pu8Value = LBTY_u8ZERO;
358         for(u8 i = (u8EndPin - u8StartPin) ; i-- ; u8StartPin++){
359             *pu8Value |= (u8)(GPIO->m_PIN.u_Reg & (1u << u8StartPin));
360         }
361     }else{
362         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
363     }
364
365     return u8RetErrorState;
366 }
```

Here is the call graph for this function:

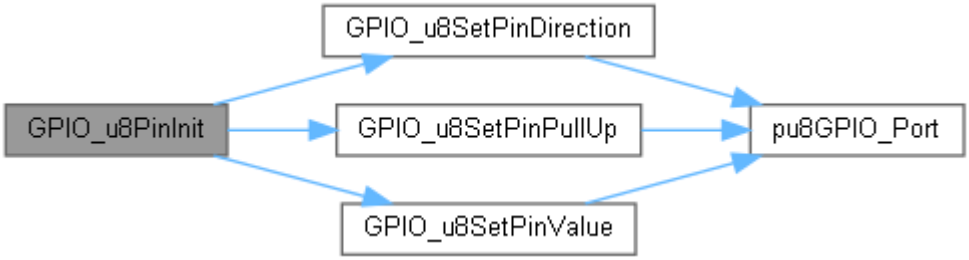


LBTY_tenuErrorStatus GPIO_u8PinInit (GPIO_tstrPinConfig *u8PinConfig*)

```
101                                                                             {
102     LBTY_tenuErrorStatus u8RetErrorState =
103             GPIO_u8SetPinDirection(u8PinConfig.m_Port, u8PinConfig.m_Pin,
u8PinConfig.m_Dir);
104     if(u8RetErrorState == LBTY_OK){
105         if(u8PinConfig.m_Dir == PIN_INPUT){
106             u8RetErrorState = GPIO_u8SetPinPullUp(u8PinConfig.m_Port,
u8PinConfig.m_Pin, u8PinConfig.m_Res);
107         }else if(u8PinConfig.m_Dir == PIN_INPUT){
108             u8RetErrorState = GPIO_u8SetPinValue(u8PinConfig.m_Port,
u8PinConfig.m_Pin, u8PinConfig.m_Value);
109         }else{}
```

```
110     }
111     return u8RetErrorState;
112 }
```

Here is the call graph for this function:



LBTY_tenuErrorStatus GPIO_u8SetMaskDirection (GPIO_tenuPortNum *u8PortNum*, u8 *u8PortMask*, GPIO_tenuDataDirection *u8PortDir*)

```
179                                                 {
180     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
181     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
182 
183     if(GPIO == LBTY_NULL){
184         u8RetErrorState = LBTY_NULL_POINTER;
185     }else if((u8)u8PortMask <= LBTY_u8MAX){
186         GPIO->m_DDR.u_Reg &= ~u8PortMask;
187         GPIO->m_DDR.u_Reg |= (u8)(u8PortMask & u8PortDir);
188     }else{
189         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
190     }
191 
192     return u8RetErrorState;
193 }
```

Here is the call graph for this function:

LBTY_tenuErrorStatus GPIO_u8SetMaskValue (GPIO_tenuPortNum *u8PortNum*, u8 *u8PortMask*, GPIO_tenuDataStatus *u8PortValue*)

```
281                                                 {
282     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
283     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
284 
285     if(GPIO == LBTY_NULL){
286         u8RetErrorState = LBTY_NULL_POINTER;
287     }else if((u8)u8PortMask <= LBTY_u8MAX){
288         GPIO->m_PORT.u_Reg &= ~u8PortMask;
289         GPIO->m_PORT.u_Reg |= (u8)(u8PortMask & u8PortValue);
290     }else{
291         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
292     }
293 
294     return u8RetErrorState;
295 }
```

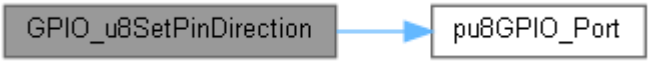
Here is the call graph for this function:

LBTY_tenuErrorStatus GPIO_u8SetPinDirection (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuPinNum *u8PinNum*, GPIO_tenuDataDirection *u8PinDir*)

```
122                                                                                 {
123     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
124     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
125 
126     if(GPIO == LBTY_NULL){
127         u8RetErrorState = LBTY_NULL_POINTER;
128     }else if((u8)u8PinNum < Pin_MaxNum){
129         if(u8PinDir == PIN_OUTPUT){
130             SET_BIT(GPIO->m_DDR.u_Reg, u8PinNum);
131         }else if(u8PinDir == PIN_INPUT){
```

```
132             CLR_BIT(GPIO->m_DDR.u_Reg, u8PinNum);
133         }else{
134             u8RetErrorState = LBTY_NOK;
135         }
136     }else{
137         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
138     }
139
140     return u8RetErrorState;
141 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

LBTY_tenuErrorStatus GPIO_u8SetPinPullUp (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuPinNum *u8PinNum*, LBTY_tenuFlagStatus *u8Pullup*)

```
511                                                                 {
512     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
513     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
514
515     if(GPIO == LBTY_NULL){
516         u8RetErrorState = LBTY_NULL_POINTER;
517     }else if((u8)u8PinNum < Pin_MaxNum){
518         if(GET_BIT(GPIO->m_DDR.u_Reg, u8PinNum) == PIN_INPUT){
519             if(u8Pullup == PULL_UP){
520                 SET_BIT(GPIO->m_PORT.u_Reg, u8PinNum);
521             }else if(u8Pullup == PULL_DOWN){
522                 CLR_BIT(GPIO->m_PORT.u_Reg, u8PinNum);
523             }else{
524                 u8RetErrorState = LBTY_NOK;
525             }
526         }else{
527             u8RetErrorState = LBTY_WRITE_ERROR;
528         }
529     }else{
530         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
531     }
532
533     return u8RetErrorState;
534 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

LBTY_tenuErrorStatus GPIO_u8SetPinValue (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuPinNum *u8PinNum*, GPIO_tenuDataStatus *u8PinVlaue*)

```
224                                                                 {
225     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
226     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
227
228     if(GPIO == LBTY_NULL){
229         u8RetErrorState = LBTY_NULL_POINTER;
230     }else if((u8)u8PinNum < Pin_MaxNum){
231         if(u8PinValue == PIN_High){
232             SET_BIT(GPIO->m_PORT.u_Reg, u8PinNum);
233         }else if(u8PinValue == PIN_Low){
234             CLR_BIT(GPIO->m_PORT.u_Reg, u8PinNum);
235         }else{
236             u8RetErrorState = LBTY_NOK;
237         }
238     }else{
239         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
240     }
```

```
241
242     return u8RetErrorState;
243 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

LBTY_tenuErrorStatus GPIO_u8SetPortDirection (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuDataDirection *u8PortDir*)

```
201                                                 {
202     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
203     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
204
205     if(GPIO == LBTY_NULL){
206         u8RetErrorState = LBTY_NULL_POINTER;
207     }else if((u8)u8PortDir <= LBTY_u8MAX){
208         GPIO->m_DDR.u_Reg = u8PortDir;
209     }else{
210         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
211     }
212
213     return u8RetErrorState;
214 }
```

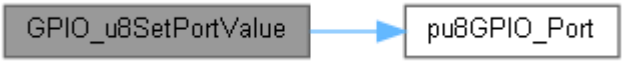
Here is the call graph for this function:

Here is the caller graph for this function:

LBTY_tenuErrorStatus GPIO_u8SetPortValue (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuDataStatus *u8PortValue*)

```
303                                                 {
304     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
305     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
306
307     if(GPIO == LBTY_NULL){
308         u8RetErrorState = LBTY_NULL_POINTER;
309     }else if((u8)u8PortValue <= LBTY_u8MAX){
310         GPIO->m_PORT.u_Reg = u8PortValue;
311     }else{
312         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
313     }
314
315     return u8RetErrorState;
316 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

LBTY_tenuErrorStatus GPIO_u8SetRangeDirection (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuPinNum *u8StartPin*, GPIO_tenuPinNum *u8EndPin*, GPIO_tenuDataDirection *u8PinDir*)

```
149
{
150     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
151     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
152
153     if(GPIO == LBTY_NULL){
```

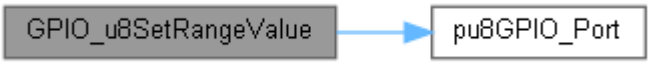
```
154         u8RetErrorState = LBTY_NULL_POINTER;
155     }else if(((u8)u8StartPin < Pin_MaxNum) && ((u8)u8EndPin < Pin_MaxNum) &&
(u8EndPin > u8StartPin)){
156         for(u8 i = (u8EndPin - u8StartPin) ; i-- ; u8StartPin++){
157             if(u8PinDir == PIN_OUTPUT){
158                 SET_BIT(GPIO->m_DDR.u_Reg, u8StartPin);
159             }else if(u8PinDir == PIN_INPUT){
160                 CLR_BIT(GPIO->m_DDR.u_Reg, u8StartPin);
161             }else{
162                 u8RetErrorState = LBTY_NOK;
163                 break;
164             }
165         }
166     }else{
167         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
168     }
169
170     return u8RetErrorState;
171 }
```

Here is the call graph for this function:

LBTY_tenuErrorStatus GPIO_u8SetRangeValue (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuPinNum *u8StartPin*, GPIO_tenuPinNum *u8EndPin*, GPIO_tenuDataStatus *u8PinValue*)

```
251
{
252     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
253     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
254
255     if(GPIO == LBTY_NULL){
256         u8RetErrorState = LBTY_NULL_POINTER;
257     }else if(((u8)u8StartPin < Pin_MaxNum) && ((u8)u8EndPin < Pin_MaxNum) &&
(u8EndPin > u8StartPin)){
258         for(u8 i = (u8EndPin - u8StartPin) ; i-- ; u8StartPin++){
259             if(u8PinValue == PIN_High){
260                 SET_BIT(GPIO->m_PORT.u_Reg, u8StartPin);
261             }else if(u8PinValue == PIN_Low){
262                 CLR_BIT(GPIO->m_PORT.u_Reg, u8StartPin);
263             }else{
264                 u8RetErrorState = LBTY_NOK;
265                 break;
266             }
267         }
268     }else{
269         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
270     }
271
272     return u8RetErrorState;
273 }
```

Here is the call graph for this function:



LBTY_tenuErrorStatus GPIO_u8ToggleMaskValue (GPIO_tenuPortNum *u8PortNum*, u8 *u8PortMask*)

```
452
{
453     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
454     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
455
456     if(GPIO == LBTY_NULL){
457         u8RetErrorState = LBTY_NULL_POINTER;
458     }else if((u8)u8PortMask <= LBTY_u8MAX){
459         TOG_MASK(GPIO->m_PORT.u_Reg, u8PortMask);
460     }else{
461         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
462     }
463
464     return u8RetErrorState;
```

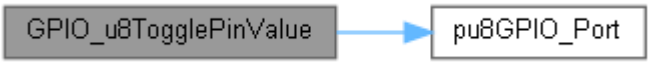
```
465 }
```

Here is the call graph for this function:

LBTY_tenuErrorStatus GPIO_u8TogglePinValue (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuPinNum *u8PinNum*)

```
409
{
410     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
411     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
412
413     if(GPIO == LBTY_NULL){
414         u8RetErrorState = LBTY_NULL_POINTER;
415     }else if((u8)u8PinNum < Pin_MaxNum){
416         TOG_BIT(GPIO->m_PORT.u_Reg, u8PinNum);
417     }else{
418         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
419     }
420
421     return u8RetErrorState;
422 }
```

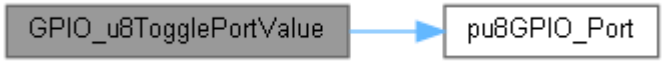
Here is the call graph for this function:



LBTY_tenuErrorStatus GPIO_u8TogglePortValue (GPIO_tenuPortNum *u8PortNum*)

```
472                                                                     {
473     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
474     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
475
476     if(GPIO == LBTY_NULL){
477         u8RetErrorState = LBTY_NULL_POINTER;
478     }else{
479         TOG_REG(GPIO->m_PORT.u_Reg);
480     }
481
482     return u8RetErrorState;
483 }
```

Here is the call graph for this function:



LBTY_tenuErrorStatus GPIO_u8ToggleRangeValue (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuPinNum *u8StartPin*, GPIO_tenuPinNum *u8EndPin*)

```
430                                                                     {
431     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
432     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
433
434     if(GPIO == LBTY_NULL){
435         u8RetErrorState = LBTY_NULL_POINTER;
436     }else if(((u8)u8StartPin < Pin_MaxNum) && ((u8)u8EndPin < Pin_MaxNum) &&
(u8EndPin > u8StartPin)){
437         for(u8 i = (u8EndPin - u8StartPin) ; i-- ; u8StartPin++){
438             TOG_BIT(GPIO->m_PORT.u_Reg, u8StartPin);
439         }
440     }else{
441         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
442     }
443
444     return u8RetErrorState;
445 }
```

Here is the call graph for this function:

void GPIO_vidDisablePullUp (void)

```
501                                     {
502     S_SFIOR->sBits.m_PUD  = LBTY_SET;       // PUD: Pull-up disable
503 }
```

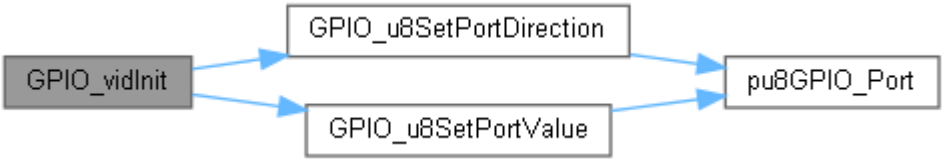
void GPIO_vidEnablePullUp (void)

```
492                                     {
493     S_SFIOR->sBits.m_PUD  = LBTY_RESET;     // PUD: Pull-up disable
494 }
```

void GPIO_vidInit (void)

```
65                          {
66
67 #if(~(GPIOA_DDR_INIT_DEF | (~GPIOA_PORT_INIT_DEF)))
68 #warning "there is some pin's in Port A direction is input so it will need to be
set to the default value!"
69 #endif
70 #if(~(GPIOB_DDR_INIT_DEF | ~GPIOB_PORT_INIT_DEF))
71 #warning "there is some pin's in Port B direction is input so it will need to be
set to the default value!"
72 #endif
73 #if(~(GPIOC_DDR_INIT_DEF | ~GPIOC_PORT_INIT_DEF))
74 #warning "there is some pin's in Port C direction is input so it will need to be
set to the default value!"
75 #endif
76 #if(~(GPIOD_DDR_INIT_DEF | ~GPIOD_PORT_INIT_DEF))
77 #warning "there is some pin's in Port D direction is input so it will need to be
set to the default value!"
78 #endif
79
80     S_SFIOR->sBits.m_PUD  = PULL_UP_DISABLE;        // PUD: Pull-up disable
81
82     GPIO_u8SetPortDirection(A, GPIOA_DDR_INIT_DEF);
83     GPIO_u8SetPortValue     (A, GPIOA_PORT_INIT_DEF);
84
85     GPIO_u8SetPortDirection(B, GPIOB_DDR_INIT_DEF);
86     GPIO_u8SetPortValue     (B, GPIOB_PORT_INIT_DEF);
87
88     GPIO_u8SetPortDirection(C, GPIOC_DDR_INIT_DEF);
89     GPIO_u8SetPortValue     (C, GPIOC_PORT_INIT_DEF);
90
91     GPIO_u8SetPortDirection(D, GPIOD_DDR_INIT_DEF);
92     GPIO_u8SetPortValue     (D, GPIOD_PORT_INIT_DEF);
93
94 }
```

Here is the call graph for this function:



GPIO_int.h

Go to the documentation of this file.

```c
/*
/* ********************************************************************************** */
/* ********************** FILE DEFINITION SECTION ************************************ */
/* ********************************************************************************** */
/* File Name    : GPIO_int.h                                                          */
/* Author       : MAAM                                                                */
/* Version      : v01.2                                                               */
/* date         : Mar 23, 2023                                                        */
/* ********************************************************************************** */
/* ************************ HEADER FILES INCLUDES ***************************  */
/* ********************************************************************************** */

#ifndef SWC_GPIO_GPIO_INT_H_
#define SWC_GPIO_GPIO_INT_H_

/* ********************************************************************************** */
/* ********************** TYPE_DEF/STRUCT/ENUM SECTION ********************** */
/* ********************************************************************************** */

typedef enum {
    Pin_0 = (u8)0,
    Pin_1,
    Pin_2,
    Pin_3,
    Pin_4,
    Pin_5,
    Pin_6,
    Pin_7,
    Pin_MaxNum
}GPIO_tenuPinNum;

typedef enum {
    A = (u8)0,
    B,
    C,
    D,
    Port_Num
}GPIO_tenuPortNum;

typedef enum{
    PIN_INPUT = (u8)0u,
    PIN_OUTPUT = (u8)1u,
    PORT_INPUT = (u8)0x00U,
    PORT_OUTPUT = (u8)0xFFU
}GPIO_tenuDataDirection;

typedef enum{
    PIN_Low = (u8)0u,
    PIN_High = (u8)1u,
    PORT_Low = (u8)0x00U,
    PORT_High = (u8)0xFFU
}GPIO_tenuDataStatus;

typedef enum{
    PULL_DOWN = (u8)0u,
    PULL_UP = (u8)1u,
    PULL_None
}GPIO_tenuInputRes;


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

typedef struct{
    GPIO_tenuPortNum        m_Port;
    GPIO_tenuPinNum         m_Pin;
    GPIO_tenuDataDirection  m_Dir;
    GPIO_tenuDataStatus     m_Value;
    GPIO_tenuInputRes       m_Res;
}GPIO_tstrPinConfig;

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

```
73 /* ********************************************************************************* */
74
75 /* ********************************************************************************* */
76 /* ***************************** CONST SECTION ************************************* */
77 /* ********************************************************************************* */
78
79 /* ********************************************************************************* */
80 /* ***************************** VARIABLE SECTION ********************************** */
81 /* ********************************************************************************* */
82
83 /* ********************************************************************************* */
84 /* ***************************** FUNCTION SECTION ********************************** */
85 /* ********************************************************************************* */
86
87 /* ********************************************************************************* */
88 /* Description : Initialize the pins direction and value                             */
89 /* Input       :    void                                                             */
90 /* Return      :    void                                                             */
91 /* ********************************************************************************* */
92 extern void GPIO_vidInit(void);
93
94 /* ********************************************************************************* */
95 /* Description :    Set the pin Initialization                                       */
96 /* Input       :    u8PinConfig                                                      */
97 /* Return      :    LBTY_tenuErrorStatus                                             */
98 /* ********************************************************************************* */
99 extern LBTY_tenuErrorStatus GPIO_u8PinInit(GPIO_tstrPinConfig u8PinConfig);
100
101
/*************************************************************************************
*****************************/
102
103 /* ********************************************************************************* */
104 /* Description :    Set the pin direction                                            */
105 /* Input       :    u8PortNum, u8PinNum, u8PinDir                                    */
106 /* Return      :    LBTY_tenuErrorStatus                                             */
107 /* ********************************************************************************* */
108 extern LBTY_tenuErrorStatus GPIO_u8SetPinDirection(GPIO_tenuPortNum u8PortNum,
109         GPIO_tenuPinNum u8PinNum, GPIO_tenuDataDirection u8PinDir);
110
111 /* ********************************************************************************* */
112 /* Description :    Set the pin direction                                            */
113 /* Input       :    u8PortNumm, u8StartPin, u8EndPin, u8PinDir                       */
114 /* Return      :    LBTY_tenuErrorStatus                                             */
115 /* ********************************************************************************* */
116 extern LBTY_tenuErrorStatus GPIO_u8SetRangeDirection(GPIO_tenuPortNum u8PortNum,
117         GPIO_tenuPinNum u8StartPin, GPIO_tenuPinNum u8EndPin, GPIO_tenuDataDirection
u8PinDir);
118
119 /* ********************************************************************************* */
120 /* Description :    Set the pin direction                                            */
121 /* Input       :    u8PortNumm, u8PortMask, u8PortDir                                */
122 /* Return      :    LBTY_tenuErrorStatus                                             */
123 /* ********************************************************************************* */
124 extern LBTY_tenuErrorStatus GPIO_u8SetMaskDirection(GPIO_tenuPortNum u8PortNum, u8
u8PortMask,
125                                                 GPIO_tenuDataDirection u8PortDir);
126
127 /* ********************************************************************************* */
128 /* Description :    Set the port direction                                           */
129 /* Input       :    u8PortNum,  u8PortDir                                            */
130 /* Return      :    LBTY_tenuErrorStatus                                             */
131 /* ********************************************************************************* */
132 extern LBTY_tenuErrorStatus GPIO_u8SetPortDirection(GPIO_tenuPortNum u8PortNum,
133                                                   GPIO_tenuDataDirection u8PortDir);
134
135
/*************************************************************************************
*****************************/
136
137 /* ********************************************************************************* */
138 /* Description :    Set the pin value                                                */
139 /* Input       :    u8PortNum, u8PinNum, u8PinValue                                  */
140 /* Return      :    LBTY_tenuErrorStatus                                             */
141 /* ********************************************************************************* */
142 extern LBTY_tenuErrorStatus GPIO_u8SetPinValue(GPIO_tenuPortNum u8PortNum,
143         GPIO_tenuPinNum u8PinNum, GPIO_tenuDataStatus u8PinVlaue);
```

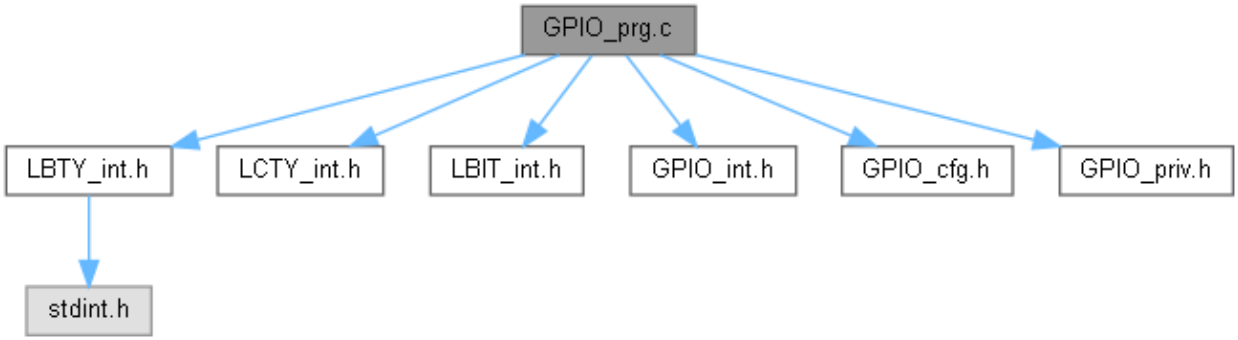
```
144
145 /* ************************************************************************** */
146 /* Description :    Set the pin value                                         */
147 /* Input       :    u8PortNumm, u8StartPin, u8EndPin, u8PinValue              */
148 /* Return      :    LBTY_tenuErrorStatus                                      */
149 /* ************************************************************************** */
150 extern LBTY_tenuErrorStatus GPIO_u8SetRangeValue(GPIO_tenuPortNum u8PortNum,
151         GPIO_tenuPinNum u8StartPin, GPIO_tenuPinNum u8EndPin, GPIO_tenuDataStatus
u8PinValue);
152
153 /* ************************************************************************** */
154 /* Description :    Set the pin value                                         */
155 /* Input       :    u8PortNumm, u8PortMask, u8PortValue                       */
156 /* Return      :    LBTY_tenuErrorStatus                                      */
157 /* ************************************************************************** */
158 extern LBTY_tenuErrorStatus GPIO_u8SetMaskValue(GPIO_tenuPortNum u8PortNum, u8
u8PortMask,
159                                                 GPIO_tenuDataStatus u8PortValue);
160
161 /* ************************************************************************** */
162 /* Description :    Set the port value                                        */
163 /* Input       :    u8PortNum, u8PortValue                                    */
164 /* Return      :    LBTY_tenuErrorStatus                                      */
165 /* ************************************************************************** */
166 extern LBTY_tenuErrorStatus GPIO_u8SetPortValue(GPIO_tenuPortNum u8PortNum,
167                                                 GPIO_tenuDataStatus u8PortValue);
168
169
/*****************************************************************************************
*******************************/
170
171 /* ************************************************************************** */
172 /* Description :    Get the pin value                                         */
173 /* Input       :    u8PortNum, u8PinNum                                       */
174 /* Input/Output:    pu8Value                                                  */
175 /* Return      :    LBTY_tenuErrorStatus                                      */
176 /* ************************************************************************** */
177 extern LBTY_tenuErrorStatus GPIO_u8GetPinValue(GPIO_tenuPortNum u8PortNum,
178         GPIO_tenuPinNum u8PinNum, pu8 pu8Value);
179
180 /* ************************************************************************** */
181 /* Description :    Get the pin value                                         */
182 /* Input       :    u8PortNumm, u8StartPin, u8EndPin                          */
183 /* Input/Output:    pu8Value                                                  */
184 /* Return      :    LBTY_tenuErrorStatus                                      */
185 /* ************************************************************************** */
186 extern LBTY_tenuErrorStatus GPIO_u8GetRangeValue(GPIO_tenuPortNum u8PortNum,
187         GPIO_tenuPinNum u8StartPin, GPIO_tenuPinNum u8EndPin, pu8 pu8Value);
188
189 /* ************************************************************************** */
190 /* Description :    Get the pin value                                         */
191 /* Input       :    u8PortNum, u8PortMask                                     */
192 /* Input/Output:    pu8Value                                                  */
193 /* Return      :    LBTY_tenuErrorStatus                                      */
194 /* ************************************************************************** */
195 extern LBTY_tenuErrorStatus GPIO_u8GetMaskValue(GPIO_tenuPortNum u8PortNum,
196                                                 u8 u8PortMask, pu8 pu8Value);
197
198 /* ************************************************************************** */
199 /* Description :    Get the port value                                        */
200 /* Input       :    u8PortNum                                                 */
201 /* Input/Output:    pu8Value                                                  */
202 /* Return      :    LBTY_tenuErrorStatus                                      */
203 /* ************************************************************************** */
204 extern LBTY_tenuErrorStatus GPIO_u8GetPortValue(GPIO_tenuPortNum u8PortNum, pu8
pu8Value);
205
206
/*****************************************************************************************
*******************************/
207
208 /* ************************************************************************** */
209 /* Description :    Toggle the pin value                                      */
210 /* Input       :    u8PortNum, u8PinNum                                       */
211 /* Return      :    LBTY_tenuErrorStatus                                      */
212 /* ************************************************************************** */
```

```
213 extern LBTY_tenuErrorStatus GPIO_u8TogglePinValue(GPIO_tenuPortNum u8PortNum,
GPIO_tenuPinNum u8PinNum);
214
215 /* ************************************************************************** */
216 /* Description :    Toggle the pin value                                      */
217 /* Input       :    u8PortNumm, u8StartPin, u8EndPin                          */
218 /* Return      :    LBTY_tenuErrorStatus                                      */
219 /* ************************************************************************** */
220 extern LBTY_tenuErrorStatus GPIO_u8ToggleRangeValue(GPIO_tenuPortNum u8PortNum,
221         GPIO_tenuPinNum u8StartPin, GPIO_tenuPinNum u8EndPin);
222
223 /* ************************************************************************** */
224 /* Description :    Toggle the pin value                                      */
225 /* Input       :    u8PortNum, u8PortMask                                     */
226 /* Return      :    LBTY_tenuErrorStatus                                      */
227 /* ************************************************************************** */
228 extern LBTY_tenuErrorStatus GPIO_u8ToggleMaskValue(GPIO_tenuPortNum u8PortNum, u8
u8PortMask);
229
230 /* ************************************************************************** */
231 /* Description :    Toggle the pin value                                      */
232 /* Input       :    u8PortNum                                                 */
233 /* Return      :    LBTY_tenuErrorStatus                                      */
234 /* ************************************************************************** */
235 extern LBTY_tenuErrorStatus GPIO_u8TogglePortValue(GPIO_tenuPortNum u8PortNum);
236
237
/*****************************************************************************************
*******************************/
238
239 /* ************************************************************************** */
240 /* Description :    Enable the pull up res                                    */
241 /* Input       :    void                                                      */
242 /* Return      :    void                                                      */
243 /* ************************************************************************** */
244 extern void GPIO_vidEnablePullUp(void);
245
246 /* ************************************************************************** */
247 /* Description :    Disable the pull up res                                   */
248 /* Input       :    void                                                      */
249 /* Return      :    void                                                      */
250 /* ************************************************************************** */
251 extern void GPIO_vidDisablePullUp(void);
252
253 /* ************************************************************************** */
254 /* Description :    Set the pin pull up res                                   */
255 /* Input       :    u8PortNum, u8PinNum, u8Pullup                             */
256 /* Return      :    LBTY_tenuErrorStatus                                      */
257 /* ************************************************************************** */
258 extern LBTY_tenuErrorStatus GPIO_u8SetPinPullUp(GPIO_tenuPortNum u8PortNum,
259         GPIO_tenuPinNum u8PinNum, LBTY_tenuFlagStatus u8Pullup);
260
261 #endif /* SWC_GPIO_GPIO_INT_H_ */
262 /************************** E N D (GPIO_int.h) ****************************/
```

GPIO_prg.c File Reference

```
#include "LBTY_int.h"
#include "LCTY_int.h"
#include "LBIT_int.h"
#include "GPIO_int.h"
#include "GPIO_cfg.h"
#include "GPIO_priv.h"
```

Include dependency graph for GPIO_prg.c:



Functions

- LCTY_INLINE GPIOx_type * pu8GPIO_Port (u8 u8PortNum)
- void GPIO_vidInit (void)
- LBTY_tenuErrorStatus GPIO_u8PinInit (GPIO_tstrPinConfig u8PinConfig)
- LBTY_tenuErrorStatus GPIO_u8SetPinDirection (GPIO_tenuPortNum u8PortNum, GPIO_tenuPinNum u8PinNum, GPIO_tenuDataDirection u8PinDir)
- LBTY_tenuErrorStatus GPIO_u8SetRangeDirection (GPIO_tenuPortNum u8PortNum, GPIO_tenuPinNum u8StartPin, GPIO_tenuPinNum u8EndPin, GPIO_tenuDataDirection u8PinDir)
- LBTY_tenuErrorStatus GPIO_u8SetMaskDirection (GPIO_tenuPortNum u8PortNum, u8 u8PortMask, GPIO_tenuDataDirection u8PortDir)
- LBTY_tenuErrorStatus GPIO_u8SetPortDirection (GPIO_tenuPortNum u8PortNum, GPIO_tenuDataDirection u8PortDir)
- LBTY_tenuErrorStatus GPIO_u8SetPinValue (GPIO_tenuPortNum u8PortNum, GPIO_tenuPinNum u8PinNum, GPIO_tenuDataStatus u8PinValue)
- LBTY_tenuErrorStatus GPIO_u8SetRangeValue (GPIO_tenuPortNum u8PortNum, GPIO_tenuPinNum u8StartPin, GPIO_tenuPinNum u8EndPin, GPIO_tenuDataStatus u8PinValue)
- LBTY_tenuErrorStatus GPIO_u8SetMaskValue (GPIO_tenuPortNum u8PortNum, u8 u8PortMask, GPIO_tenuDataStatus u8PortValue)
- LBTY_tenuErrorStatus GPIO_u8SetPortValue (GPIO_tenuPortNum u8PortNum, GPIO_tenuDataStatus u8PortValue)
- LBTY_tenuErrorStatus GPIO_u8GetPinValue (GPIO_tenuPortNum u8PortNum, GPIO_tenuPinNum u8PinNum, pu8 pu8Value)
- LBTY_tenuErrorStatus GPIO_u8GetRangeValue (GPIO_tenuPortNum u8PortNum, GPIO_tenuPinNum u8StartPin, GPIO_tenuPinNum u8EndPin, pu8 pu8Value)
- LBTY_tenuErrorStatus GPIO_u8GetMaskValue (GPIO_tenuPortNum u8PortNum, u8 u8PortMask, pu8 pu8Value)
- LBTY_tenuErrorStatus GPIO_u8GetPortValue (GPIO_tenuPortNum u8PortNum, pu8 pu8Value)
- LBTY_tenuErrorStatus GPIO_u8TogglePinValue (GPIO_tenuPortNum u8PortNum, GPIO_tenuPinNum u8PinNum)
- LBTY_tenuErrorStatus GPIO_u8ToggleRangeValue (GPIO_tenuPortNum u8PortNum, GPIO_tenuPinNum u8StartPin, GPIO_tenuPinNum u8EndPin)
- LBTY_tenuErrorStatus GPIO_u8ToggleMaskValue (GPIO_tenuPortNum u8PortNum, u8 u8PortMask)

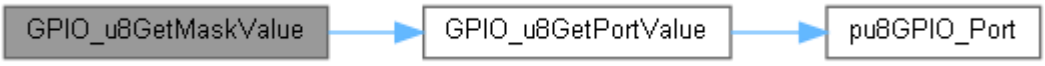
- LBTY_tenuErrorStatus GPIO_u8TogglePortValue (GPIO_tenuPortNum u8PortNum)
- void GPIO_vidEnablePullUp (void)
- void GPIO_vidDisablePullUp (void)
- LBTY_tenuErrorStatus GPIO_u8SetPinPullUp (GPIO_tenuPortNum u8PortNum, GPIO_tenuPinNum u8PinNum, LBTY_tenuFlagStatus u8Pullup)

Function Documentation

LBTY_tenuErrorStatus GPIO_u8GetMaskValue (GPIO_tenuPortNum *u8PortNum*, u8 *u8PortMask*, pu8 *pu8Value*)

```
375                                         {
376     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
377     if(!(u8RetErrorState = GPIO_u8GetPortValue(u8PortNum, pu8Value))){
378         *pu8Value &= u8PortMask;
379     }
380 
381     return u8RetErrorState;
382 }
```

Here is the call graph for this function:



LBTY_tenuErrorStatus GPIO_u8GetPinValue (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuPinNum *u8PinNum*, pu8 *pu8Value*)

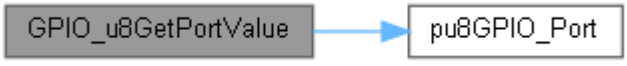
```
327                                                 {
328     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
329     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
330 
331     if((GPIO == LBTY_NULL) || (pu8Value == LBTY_NULL)){
332         u8RetErrorState = LBTY_NULL_POINTER;
333     }else if((u8)u8PinNum < Pin_MaxNum){
334         *pu8Value = LBTY_u8ZERO;
335         *pu8Value = (u8)GET_BIT(GPIO->m_PIN.u_Reg, u8PinNum);
336     }else{
337         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
338     }
339 
340     return u8RetErrorState;
341 }
```

Here is the call graph for this function:

LBTY_tenuErrorStatus GPIO_u8GetPortValue (GPIO_tenuPortNum *u8PortNum*, pu8 *pu8Value*)

```
390 
{
391     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
392     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
393 
394     if((GPIO == LBTY_NULL) || (pu8Value == LBTY_NULL)){
395         u8RetErrorState = LBTY_NULL_POINTER;
396     }else{
397         *pu8Value = (u8)GPIO->m_PIN.u_Reg;
398     }
399     return u8RetErrorState;
400 }
```

Here is the call graph for this function:

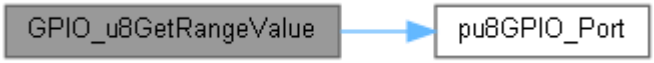


Here is the caller graph for this function:

LBTY_tenuErrorStatus GPIO_u8GetRangeValue (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuPinNum *u8StartPin*, GPIO_tenuPinNum *u8EndPin*, pu8 *pu8Value*)

```
350                                                                                 {
351     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
352     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
353
354     if((GPIO == LBTY_NULL) || (pu8Value == LBTY_NULL)){
355         u8RetErrorState = LBTY_NULL_POINTER;
356     }else if(((u8)u8StartPin < Pin_MaxNum) && ((u8)u8EndPin < Pin_MaxNum) &&
(u8EndPin > u8StartPin)){
357         *pu8Value = LBTY_u8ZERO;
358         for(u8 i = (u8EndPin - u8StartPin) ; i-- ; u8StartPin++){
359             *pu8Value |= (u8)(GPIO->m_PIN.u_Reg & (1u << u8StartPin));
360         }
361     }else{
362         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
363     }
364
365     return u8RetErrorState;
366 }
```

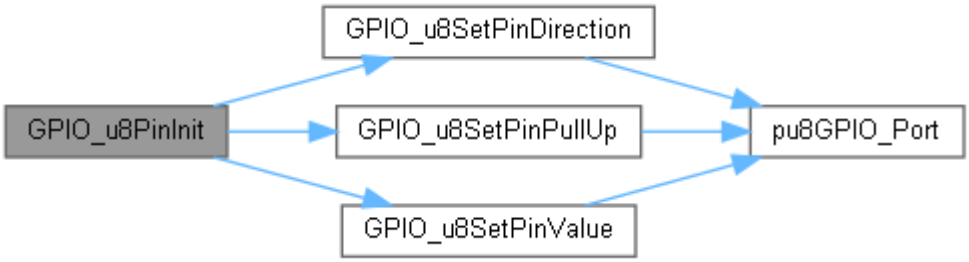
Here is the call graph for this function:



LBTY_tenuErrorStatus GPIO_u8PinInit (GPIO_tstrPinConfig *u8PinConfig*)

```
101                                                                                 {
102     LBTY_tenuErrorStatus u8RetErrorState =
103             GPIO_u8SetPinDirection(u8PinConfig.m_Port, u8PinConfig.m_Pin,
u8PinConfig.m_Dir);
104     if(u8RetErrorState == LBTY_OK){
105         if(u8PinConfig.m_Dir == PIN_INPUT){
106             u8RetErrorState = GPIO_u8SetPinPullUp(u8PinConfig.m_Port,
u8PinConfig.m_Pin, u8PinConfig.m_Res);
107         }else if(u8PinConfig.m_Dir == PIN_INPUT){
108             u8RetErrorState = GPIO_u8SetPinValue(u8PinConfig.m_Port,
u8PinConfig.m_Pin, u8PinConfig.m_Value);
109         }else{}
110     }
111     return u8RetErrorState;
112 }
```

Here is the call graph for this function:



LBTY_tenuErrorStatus GPIO_u8SetMaskDirection (GPIO_tenuPortNum *u8PortNum*, u8 *u8PortMask*, GPIO_tenuDataDirection *u8PortDir*)

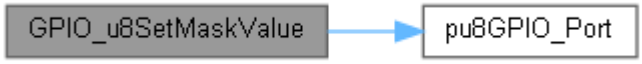
```
179                                             {
180     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
181     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
182
183     if(GPIO == LBTY_NULL){
184         u8RetErrorState = LBTY_NULL_POINTER;
185     }else if((u8)u8PortMask <= LBTY_u8MAX){
186         GPIO->m_DDR.u_Reg &= ~u8PortMask;
187         GPIO->m_DDR.u_Reg |= (u8)(u8PortMask & u8PortDir);
188     }else{
189         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
190     }
191
192     return u8RetErrorState;
193 }
```

Here is the call graph for this function:

LBTY_tenuErrorStatus GPIO_u8SetMaskValue (GPIO_tenuPortNum *u8PortNum*, u8 *u8PortMask*, GPIO_tenuDataStatus *u8PortValue*)

```
281                                                 {
282     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
283     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
284
285     if(GPIO == LBTY_NULL){
286         u8RetErrorState = LBTY_NULL_POINTER;
287     }else if((u8)u8PortMask <= LBTY_u8MAX){
288         GPIO->m_PORT.u_Reg &= ~u8PortMask;
289         GPIO->m_PORT.u_Reg |= (u8)(u8PortMask & u8PortValue);
290     }else{
291         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
292     }
293
294     return u8RetErrorState;
295 }
```

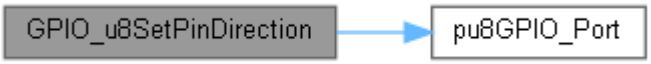
Here is the call graph for this function:



LBTY_tenuErrorStatus GPIO_u8SetPinDirection (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuPinNum *u8PinNum*, GPIO_tenuDataDirection *u8PinDir*)

```
122                                                                                 {
123     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
124     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
125
126     if(GPIO == LBTY_NULL){
127         u8RetErrorState = LBTY_NULL_POINTER;
128     }else if((u8)u8PinNum < Pin_MaxNum){
129         if(u8PinDir == PIN_OUTPUT){
130             SET_BIT(GPIO->m_DDR.u_Reg, u8PinNum);
131         }else if(u8PinDir == PIN_INPUT){
132             CLR_BIT(GPIO->m_DDR.u_Reg, u8PinNum);
133         }else{
134             u8RetErrorState = LBTY_NOK;
135         }
136     }else{
137         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
138     }
139
140     return u8RetErrorState;
141 }
```

Here is the call graph for this function:



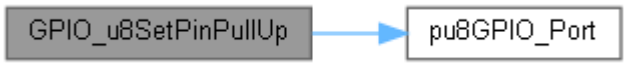
Here is the caller graph for this function:

LBTY_tenuErrorStatus GPIO_u8SetPinPullUp (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuPinNum *u8PinNum*, LBTY_tenuFlagStatus *u8Pullup*)

```
511                                                                                 {
512     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
513     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
514
515     if(GPIO == LBTY_NULL){
516         u8RetErrorState = LBTY_NULL_POINTER;
517     }else if((u8)u8PinNum < Pin_MaxNum){
518         if(GET_BIT(GPIO->m_DDR.u_Reg, u8PinNum) == PIN_INPUT){
519             if(u8Pullup == PULL_UP){
520                 SET_BIT(GPIO->m_PORT.u_Reg, u8PinNum);
521             }else if(u8Pullup == PULL_DOWN){
522                 CLR_BIT(GPIO->m_PORT.u_Reg, u8PinNum);
523             }else{
```

```
524                 u8RetErrorState = LBTY_NOK;
525             }
526         }else{
527             u8RetErrorState = LBTY_WRITE_ERROR;
528         }
529     }else{
530         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
531     }
532 
533     return u8RetErrorState;
534 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

LBTY_tenuErrorStatus GPIO_u8SetPinValue (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuPinNum *u8PinNum*, GPIO_tenuDataStatus *u8PinValue*)

```
224                                                                     {
225     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
226     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
227 
228     if(GPIO == LBTY_NULL){
229         u8RetErrorState = LBTY_NULL_POINTER;
230     }else if((u8)u8PinNum < Pin_MaxNum){
231         if(u8PinValue == PIN_High){
232             SET_BIT(GPIO->m_PORT.u_Reg, u8PinNum);
233         }else if(u8PinValue == PIN_Low){
234             CLR_BIT(GPIO->m_PORT.u_Reg, u8PinNum);
235         }else{
236             u8RetErrorState = LBTY_NOK;
237         }
238     }else{
239         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
240     }
241 
242     return u8RetErrorState;
243 }
```

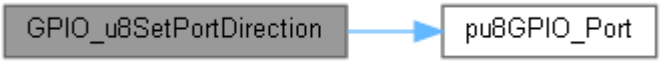
Here is the call graph for this function:

Here is the caller graph for this function:

LBTY_tenuErrorStatus GPIO_u8SetPortDirection (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuDataDirection *u8PortDir*)

```
201                                                 {
202     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
203     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
204 
205     if(GPIO == LBTY_NULL){
206         u8RetErrorState = LBTY_NULL_POINTER;
207     }else if((u8)u8PortDir <= LBTY_u8MAX){
208         GPIO->m_DDR.u_Reg = u8PortDir;
209     }else{
210         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
211     }
212 
213     return u8RetErrorState;
214 }
```

Here is the call graph for this function:

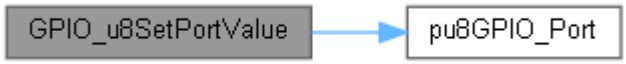


Here is the caller graph for this function:

LBTY_tenuErrorStatus GPIO_u8SetPortValue (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuDataStatus *u8PortValue*)

```
303                                                 {
304     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
305     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
306
307     if(GPIO == LBTY_NULL){
308         u8RetErrorState = LBTY_NULL_POINTER;
309     }else if((u8)u8PortValue <= LBTY_u8MAX){
310         GPIO->m_PORT.u_Reg = u8PortValue;
311     }else{
312         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
313     }
314
315     return u8RetErrorState;
316 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

LBTY_tenuErrorStatus GPIO_u8SetRangeDirection (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuPinNum *u8StartPin*, GPIO_tenuPinNum *u8EndPin*, GPIO_tenuDataDirection *u8PinDir*)

```
149
{
150     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
151     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
152
153     if(GPIO == LBTY_NULL){
154         u8RetErrorState = LBTY_NULL_POINTER;
155     }else if(((u8)u8StartPin < Pin_MaxNum) && ((u8)u8EndPin < Pin_MaxNum) &&
(u8EndPin > u8StartPin)){
156         for(u8 i = (u8EndPin - u8StartPin) ; i-- ; u8StartPin++){
157             if(u8PinDir == PIN_OUTPUT){
158                 SET_BIT(GPIO->m_DDR.u_Reg, u8StartPin);
159             }else if(u8PinDir == PIN_INPUT){
160                 CLR_BIT(GPIO->m_DDR.u_Reg, u8StartPin);
161             }else{
162                 u8RetErrorState = LBTY_NOK;
163                 break;
164             }
165         }
166     }else{
167         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
168     }
169
170     return u8RetErrorState;
171 }
```

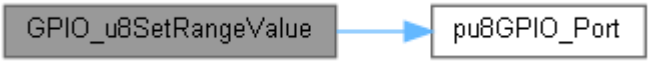
Here is the call graph for this function:

LBTY_tenuErrorStatus GPIO_u8SetRangeValue (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuPinNum *u8StartPin*, GPIO_tenuPinNum *u8EndPin*, GPIO_tenuDataStatus *u8PinValue*)

```
251
{
252     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
253     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
254
255     if(GPIO == LBTY_NULL){
256         u8RetErrorState = LBTY_NULL_POINTER;
```

```
257     }else if(((u8)u8StartPin < Pin_MaxNum) && ((u8)u8EndPin < Pin_MaxNum) &&
(u8EndPin > u8StartPin)){
258         for(u8 i = (u8EndPin - u8StartPin) ; i-- ; u8StartPin++){
259             if(u8PinValue == PIN_High){
260                 SET_BIT(GPIO->m_PORT.u_Reg, u8StartPin);
261             }else if(u8PinValue == PIN_Low){
262                 CLR_BIT(GPIO->m_PORT.u_Reg, u8StartPin);
263             }else{
264                 u8RetErrorState = LBTY_NOK;
265                 break;
266             }
267         }
268     }else{
269         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
270     }
271
272     return u8RetErrorState;
273 }
```

Here is the call graph for this function:



LBTY_tenuErrorStatus GPIO_u8ToggleMaskValue (GPIO_tenuPortNum *u8PortNum*, u8 *u8PortMask*)

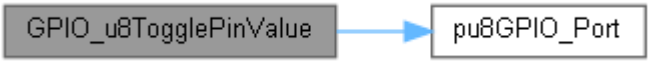
```
452
{
453     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
454     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
455
456     if(GPIO == LBTY_NULL){
457         u8RetErrorState = LBTY_NULL_POINTER;
458     }else if((u8)u8PortMask <= LBTY_u8MAX){
459         TOG_MASK(GPIO->m_PORT.u_Reg, u8PortMask);
460     }else{
461         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
462     }
463
464     return u8RetErrorState;
465 }
```

Here is the call graph for this function:

LBTY_tenuErrorStatus GPIO_u8TogglePinValue (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuPinNum *u8PinNum*)

```
409
{
410     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
411     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
412
413     if(GPIO == LBTY_NULL){
414         u8RetErrorState = LBTY_NULL_POINTER;
415     }else if((u8)u8PinNum < Pin_MaxNum){
416         TOG_BIT(GPIO->m_PORT.u_Reg, u8PinNum);
417     }else{
418         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
419     }
420
421     return u8RetErrorState;
422 }
```

Here is the call graph for this function:

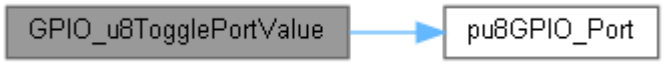


LBTY_tenuErrorStatus GPIO_u8TogglePortValue (GPIO_tenuPortNum *u8PortNum*)

```
472                                                                     {
473     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
474     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
475
```

```
476     if(GPIO == LBTY_NULL){
477         u8RetErrorState = LBTY_NULL_POINTER;
478     }else{
479         TOG_REG(GPIO->m_PORT.u_Reg);
480     }
481
482     return u8RetErrorState;
483 }
```

Here is the call graph for this function:



LBTY_tenuErrorStatus GPIO_u8ToggleRangeValue (GPIO_tenuPortNum *u8PortNum*, GPIO_tenuPinNum *u8StartPin*, GPIO_tenuPinNum *u8EndPin*)

```
430                                                                 {
431     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
432     GPIOx_type * GPIO = pu8GPIO_Port(u8PortNum);
433
434     if(GPIO == LBTY_NULL){
435         u8RetErrorState = LBTY_NULL_POINTER;
436     }else if(((u8)u8StartPin < Pin_MaxNum) && ((u8)u8EndPin < Pin_MaxNum) &&
(u8EndPin > u8StartPin)){
437         for(u8 i = (u8EndPin - u8StartPin) ; i-- ; u8StartPin++){
438             TOG_BIT(GPIO->m_PORT.u_Reg, u8StartPin);
439         }
440     }else{
441         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
442     }
443
444     return u8RetErrorState;
445 }
```

Here is the call graph for this function:

void GPIO_vidDisablePullUp (void)

```
501                                 {
502     S_SFIOR->sBits.m_PUD  = LBTY_SET;       // PUD: Pull-up disable
503 }
```

void GPIO_vidEnablePullUp (void)

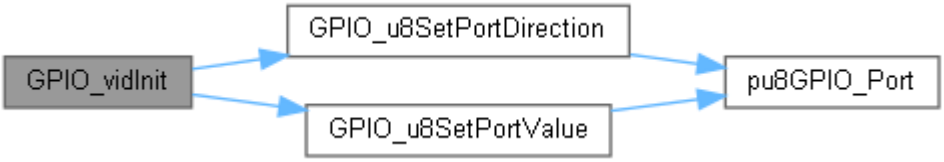
```
492                                 {
493     S_SFIOR->sBits.m_PUD  = LBTY_RESET;     // PUD: Pull-up disable
494 }
```

void GPIO_vidInit (void)

```
65                          {
66
67 #if(~(GPIOA_DDR_INIT_DEF | (~GPIOA_PORT_INIT_DEF)))
68 #warning "there is some pin's in Port A direction is input so it will need to be
set to the default value!"
69 #endif
70 #if(~(GPIOB_DDR_INIT_DEF | ~GPIOB_PORT_INIT_DEF))
71 #warning "there is some pin's in Port B direction is input so it will need to be
set to the default value!"
72 #endif
73 #if(~(GPIOC_DDR_INIT_DEF | ~GPIOC_PORT_INIT_DEF))
74 #warning "there is some pin's in Port C direction is input so it will need to be
set to the default value!"
75 #endif
76 #if(~(GPIOD_DDR_INIT_DEF | ~GPIOD_PORT_INIT_DEF))
77 #warning "there is some pin's in Port D direction is input so it will need to be
set to the default value!"
78 #endif
79
80     S_SFIOR->sBits.m_PUD  = PULL_UP_DISABLE;        // PUD: Pull-up disable
81
82     GPIO_u8SetPortDirection(A, GPIOA_DDR_INIT_DEF);
83     GPIO_u8SetPortValue    (A, GPIOA_PORT_INIT_DEF);
84
```

```
85      GPIO_u8SetPortDirection(B, GPIOB_DDR_INIT_DEF);
86      GPIO_u8SetPortValue    (B, GPIOB_PORT_INIT_DEF);
87
88      GPIO_u8SetPortDirection(C, GPIOC_DDR_INIT_DEF);
89      GPIO_u8SetPortValue    (C, GPIOC_PORT_INIT_DEF);
90
91      GPIO_u8SetPortDirection(D, GPIOD_DDR_INIT_DEF);
92      GPIO_u8SetPortValue    (D, GPIOD_PORT_INIT_DEF);
93
94  }
```

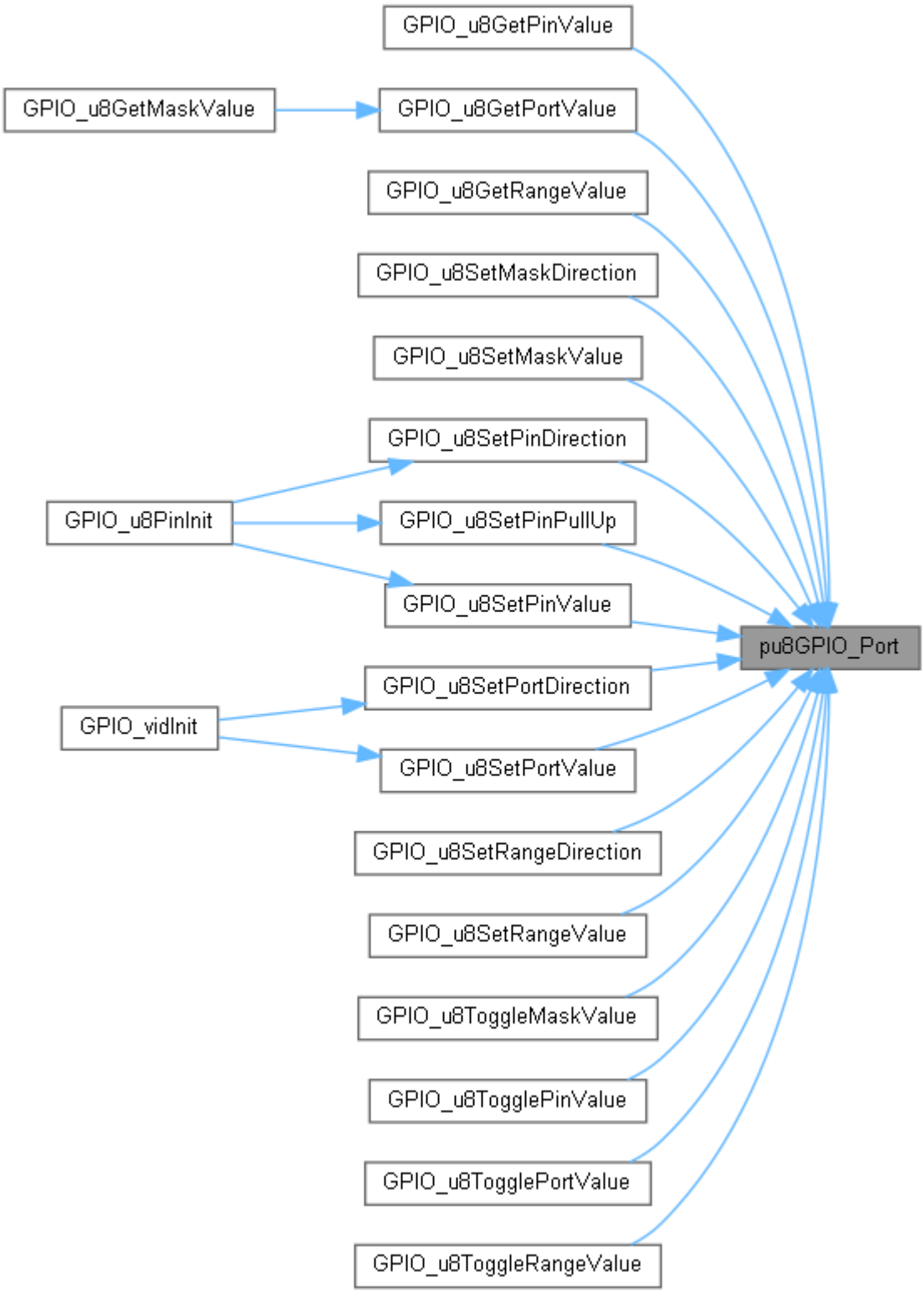
Here is the call graph for this function:



LCTY_INLINE GPIOx_type * pu8GPIO_Port (u8 *u8PortNum*)

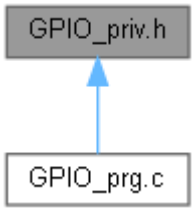
```
31                                                      {
32      GPIOx_type * GPIO = LBTY_NULL;
33      switch(u8PortNum){
34      case A:
35          GPIO = S_GPIOA;
36          break;
37      case B:
38          GPIO = S_GPIOB;
39          break;
40      case C:
41          GPIO = S_GPIOC;
42          break;
43      case D:
44          GPIO = S_GPIOD;
45          break;
46      default:
47          GPIO = (GPIOx_type *)LBTY_NULL;
48      }
49      return GPIO;
50  }
```

Here is the caller graph for this function:


GPIO_u8GetPinValue
GPIO_u8GetMaskValue
GPIO_u8GetPortValue
GPIO_u8GetRangeValue
GPIO_u8SetMaskDirection
GPIO_u8SetMaskValue
GPIO_u8SetPinDirection
GPIO_u8PinInit
GPIO_u8SetPinPullUp
GPIO_u8SetPinValue
pu8GPIO_Port
GPIO_u8SetPortDirection
GPIO_vidInit
GPIO_u8SetPortValue
GPIO_u8SetRangeDirection
GPIO_u8SetRangeValue
GPIO_u8ToggleMaskValue
GPIO_u8TogglePinValue
GPIO_u8TogglePortValue
GPIO_u8ToggleRangeValue

GPIO_priv.h File Reference

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



Data Structures

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

struct GPIOx_type: *General Purpose Input Output Registers*

union SFIOR_type: *Special Function I/O Register*

Macros

- #define PULL_UP_DISABLE LBTY_RESET
- #define GPIOA_DDR_INIT_DEF PORT_OUTPUT
- #define GPIOA_PORT_INIT_DEF PORT_Low
- #define GPIOB_DDR_INIT_DEF PORT_OUTPUT
- #define GPIOB_PORT_INIT_DEF PORT_Low
- #define GPIOC_DDR_INIT_DEF PORT_OUTPUT
- #define GPIOC_PORT_INIT_DEF PORT_Low
- #define GPIOD_DDR_INIT_DEF PORT_OUTPUT
- #define GPIOD_PORT_INIT_DEF PORT_Low
- #define S_GPIOD ((GPIOx_type* const)0x30U)
- #define PIND (*(volatile u8* const)0x30U)
- #define DDRD (*(volatile u8* const)0x31U)
- #define PORTD (*(volatile u8* const)0x32U)
- #define S_GPIOC ((GPIOx_type* const)0x33U)
- #define PINC (*(volatile u8* const)0x33U)
- #define DDRC (*(volatile u8* const)0x34U)
- #define PORTC (*(volatile u8* const)0x35U)
- #define S_GPIOB ((GPIOx_type* const)0x36U)
- #define PINB (*(volatile u8* const)0x36U)
- #define DDRB (*(volatile u8* const)0x37U)
- #define PORTB (*(volatile u8* const)0x38U)
- #define S_GPIOA ((GPIOx_type* const)0x39U)
- #define PINA (*(volatile u8* const)0x39U)
- #define DDRA (*(volatile u8* const)0x3AU)
- #define PORTA (*(volatile u8* const)0x3BU)
- #define S_SFIOR ((SFIOR_type* const)0x50U)
- #define SFIOR (*(volatile u8* const)0x50U)

Macro Definition Documentation

#define DDRA (*(volatile u8* const)0x3AU)

#define DDRB (*(volatile u8* const)0x37U)

#define DDRC (*(volatile u8* const)0x34U)

#define DDRD (*(volatile u8* const)0x31U)

#define GPIOA_DDR_INIT_DEF PORT_OUTPUT

#define GPIOA_PORT_INIT_DEF PORT_Low

#define GPIOB_DDR_INIT_DEF PORT_OUTPUT

#define GPIOB_PORT_INIT_DEF PORT_Low

#define GPIOC_DDR_INIT_DEF PORT_OUTPUT

#define GPIOC_PORT_INIT_DEF PORT_Low

#define GPIOD_DDR_INIT_DEF PORT_OUTPUT

#define GPIOD_PORT_INIT_DEF PORT_Low

#define PINA (*(volatile u8* const)0x39U)

#define PINB (*(volatile u8* const)0x36U)

#define PINC (*(volatile u8* const)0x33U)

#define PIND (*(volatile u8* const)0x30U)

#define PORTA (*(volatile u8* const)0x3BU)

#define PORTB (*(volatile u8* const)0x38U)

#define PORTC (*(volatile u8* const)0x35U)

#define PORTD (*(volatile u8* const)0x32U)

#define PULL_UP_DISABLE LBTY_RESET

#define S_GPIOA ((GPIOx_type* const)0x39U)

GPIOA

#define S_GPIOB ((GPIOx_type* const)0x36U)

GPIOB

#define S_GPIOC ((GPIOx_type* const)0x33U)

GPIOC

#define S_GPIOD ((GPIOx_type* const)0x30U)

GPIOD

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

Special Function I/O Register

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

GPIO_priv.h

```
Go to the documentation of this file.1 /*
****************************************************************************** */
2 /* ********************** FILE DEFINITION SECTION ************************** */
3 /* ****************************************************************************** */
4 /* File Name    : GPIO_priv.h                                                  */
5 /* Author       : MAAM                                                         */
6 /* Version      : v01.2                                                        */
7 /* date         : Mar 24, 2023                                                 */
8 /* ****************************************************************************** */
9 /* ************************* HEADER FILES INCLUDES ***************************  */
10 /* ****************************************************************************** */
11
12 #ifndef GPIO_PRIV_H_
13 #define GPIO_PRIV_H_
14
15 /* ****************************************************************************** */
16 /* ********************** TYPE_DEF/STRUCT/ENUM SECTION ********************** */
17 /* ****************************************************************************** */
18
21 typedef union{
22     u8 u_Reg;
23     struct {
24         __IO u8 m_B0 : 1;
25         __IO u8 m_B1 : 1;
26         __IO u8 m_B2 : 1;
27         __IO u8 m_B3 : 1;
28         __IO u8 m_B4 : 1;
29         __IO u8 m_B5 : 1;
30         __IO u8 m_B6 : 1;
31         __IO u8 m_B7 : 1;
32     }sBits;
33 }BYTE_type;
34
35 /*****************************************************************************/
36
39 typedef struct{
40     __I  BYTE_type m_PIN;
41     __IO BYTE_type m_DDR;
42     __IO BYTE_type m_PORT;
43 }GPIOx_type;
44
45 /*****************************************************************************/
46
49 typedef union{
50     u8 u_Reg;
51     struct {
52         __IO u8         : 2;
53         __IO u8 m_PUD   : 1;
54         __IO u8         : 5;
55     }sBits;
56 }SFIOR_type;
57
58 /* ****************************************************************************** */
59 /* ************************** MACRO/DEFINE SECTION ***************************** */
60 /* ****************************************************************************** */
61
62 #define PULL_UP_DISABLE         LBTY_RESET
63
64 #define GPIOA_DDR_INIT_DEF      PORT_OUTPUT
65 #define GPIOA_PORT_INIT_DEF     PORT_Low
66
67 #define GPIOB_DDR_INIT_DEF      PORT_OUTPUT
68 #define GPIOB_PORT_INIT_DEF     PORT_Low
69
70 #define GPIOC_DDR_INIT_DEF      PORT_OUTPUT
71 #define GPIOC_PORT_INIT_DEF     PORT_Low
72
73 #define GPIOD_DDR_INIT_DEF      PORT_OUTPUT
74 #define GPIOD_PORT_INIT_DEF     PORT_Low
75
76 /* ****************************************************************************** */
77
79 #define S_GPIOD         ((GPIOx_type* const)0x30U)
```

```
80 #define PIND            (*(volatile u8* const)0x30U)
81 #define DDRD            (*(volatile u8* const)0x31U)
82 #define PORTD           (*(volatile u8* const)0x32U)
83
85 #define S_GPIOC         ((GPIOx_type* const)0x33U)
86 #define PINC            (*(volatile u8* const)0x33U)
87 #define DDRC            (*(volatile u8* const)0x34U)
88 #define PORTC           (*(volatile u8* const)0x35U)
89
91 #define S_GPIOB         ((GPIOx_type* const)0x36U)
92 #define PINB            (*(volatile u8* const)0x36U)
93 #define DDRB            (*(volatile u8* const)0x37U)
94 #define PORTB           (*(volatile u8* const)0x38U)
95
97 #define S_GPIOA         ((GPIOx_type* const)0x39U)
98 #define PINA            (*(volatile u8* const)0x39U)
99 #define DDRA            (*(volatile u8* const)0x3AU)
100 #define PORTA            (*(volatile u8* const)0x3BU)
101
103 #define S_SFIOR          ((SFIOR_type* const)0x50U)
104 #define SFIOR            (*(volatile u8* const)0x50U)
105
106 /* ********************************************************************************* */
107 /* ***************************** CONST SECTION ***************************** */
108 /* ********************************************************************************* */
109
110 /* ********************************************************************************* */
111 /* ***************************** VARIABLE SECTION ***************************** */
112 /* ********************************************************************************* */
113
114 /* ********************************************************************************* */
115 /* ***************************** FUNCTION SECTION ***************************** */
116 /* ********************************************************************************* */
117
118
119 #endif /* GPIO_PRIV_H_ */
120 /***************************** E N D (GPIO_priv.h) *****************************/
```

main.c File Reference