SWC_LCD

Version v1.0
7/15/2023 4:06:00 PM

Table of Contents

Data Structure Index

Data Structures

Here are the data structures with brief descriptions:

File Index

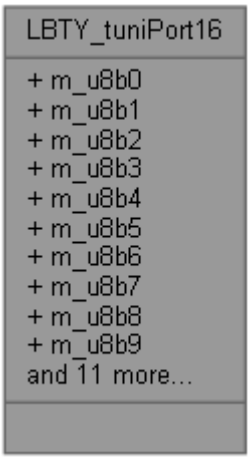
File List

Here is a list of all files with brief descriptions:

Data Structure Documentation

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

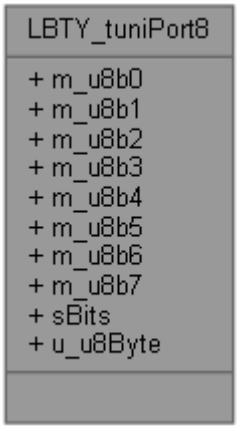
The documentation for this union was generated from the following file:

- 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:

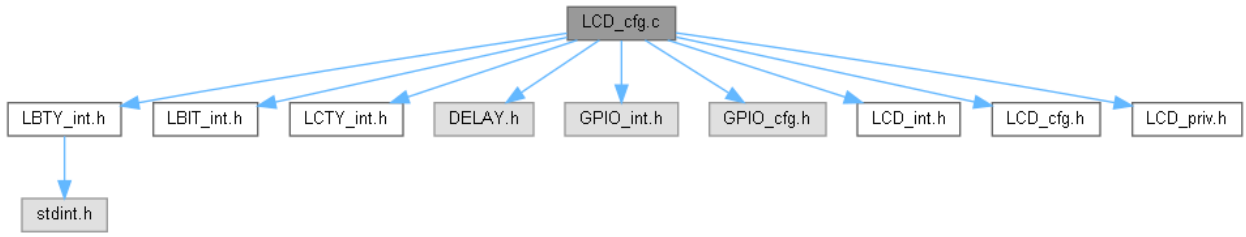
- H:/0/Workspaces/MCU Drivers/ATmega32/MCAL/SWC_BSW/LBTY_int.h

File Documentation

LCD_cfg.c File Reference

```
#include "LBTY_int.h"
#include "LBIT_int.h"
#include "LCTY_int.h"
#include "DELAY.h"
#include "GPIO_int.h"
#include "GPIO_cfg.h"
#include "LCD_int.h"
#include "LCD_cfg.h"
#include "LCD_priv.h"
```

Include dependency graph for LCD_cfg.c:



Functions

- LBTY_tenuErrorStatus LCD_u8FunctionSet (void)
- void LCD_vidInitPins (void)
- void LCD_vidDirection (u8 u8PinDir)
- void LCD_vidTriger (void)
- LBTY_tenuErrorStatus LCD_u8Write (u8 u8Byte)
- LBTY_tenuErrorStatus LCD_u8Read (u8 *pu8Byte)
- LBTY_tenuErrorStatus LCD_u8CMD_W (u8 u8CMD)
- LBTY_tenuErrorStatus LCD_u8CMD_R (u8 *pu8CMD)
- LBTY_tenuErrorStatus LCD_u8CHAR_W (u8 u8Char)
- LBTY_tenuErrorStatus LCD_u8CHAR_R (u8 *pu8Char)
- LBTY_tenuErrorStatus LCD_u8Set_CGRAM_Address (u8 u8Address)
- LBTY_tenuErrorStatus LCD_u8Set_DDRAM_Address (u8 u8Address)
- LBTY_tenuErrorStatus LCD_u8Get_DDRAM_Address (u8 *pu8Address)
- u8 LCD_u8GetBusyFlag (void)

Variables

- const u8 ETA32 [][LCD_CGRAM_LOCATIONS_NUM]

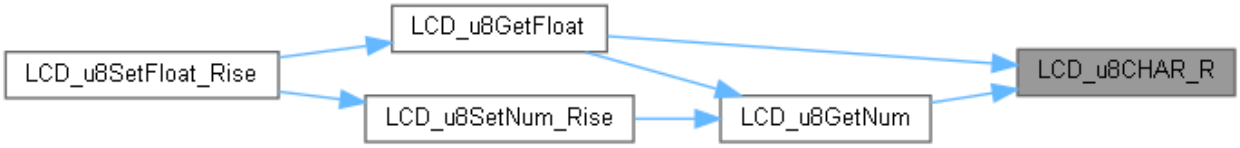
Function Documentation

LBTY_tenuErrorStatus LCD_u8CHAR_R (u8 * *pu8Char*)

```
249                                                 {
250     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
251
252     u8RetErrorState = GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_RS, LCD_RS_DATA);
253     u8RetErrorState = LCD_u8Read(pu8Char);
254
255     return u8RetErrorState;
256 }
```

Here is the call graph for this function:

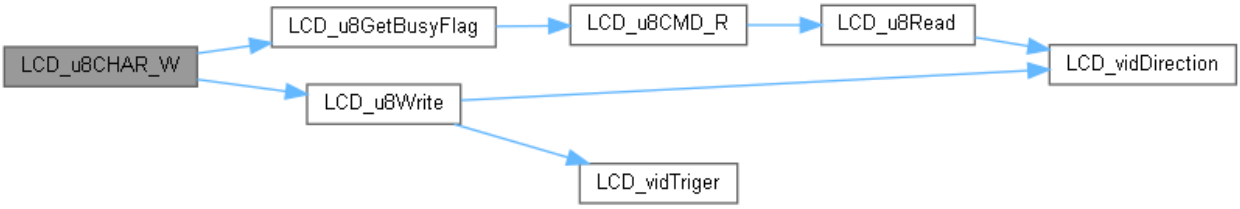
Here is the caller graph for this function:



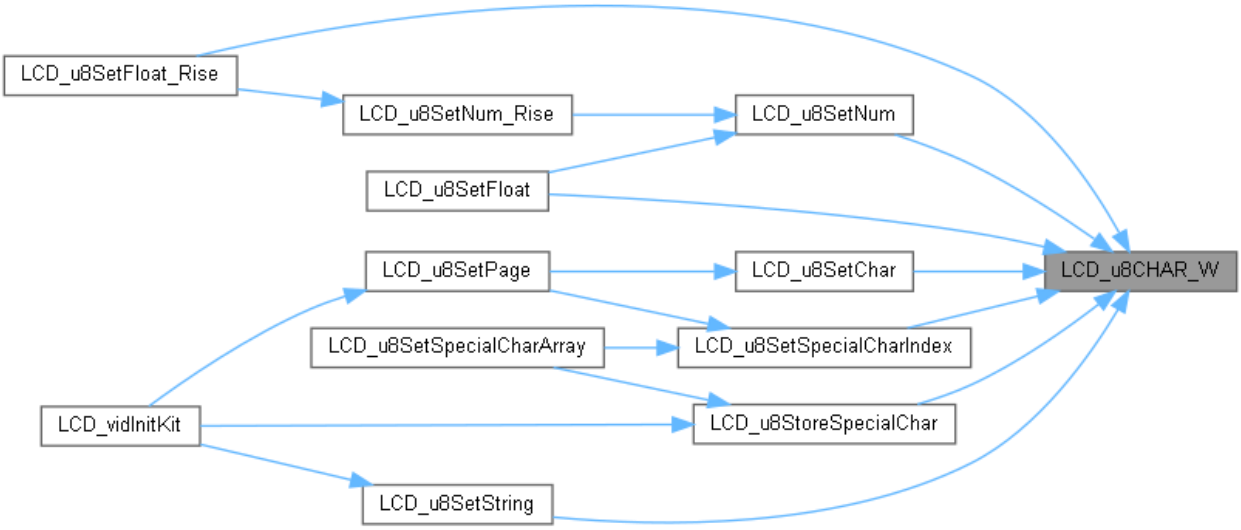
LBTY_tenuErrorStatus LCD_u8CHAR_W (u8 *u8Char*)

```
237                                                 {
238     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
239
240 #ifdef LCD_RW
241     while(LCD_u8GetBusyFlag())       vidMyDelay_ms(LCD_DELAY_CMD);
242 #endif
243     u8RetErrorState = GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_RS, LCD_RS_DATA);
244     u8RetErrorState = LCD_u8Write(u8Char);
245
246     return u8RetErrorState;
247 }
```

Here is the call graph for this function:



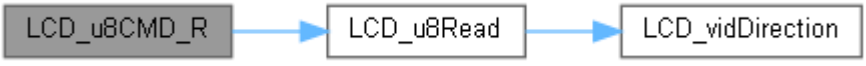
Here is the caller graph for this function:



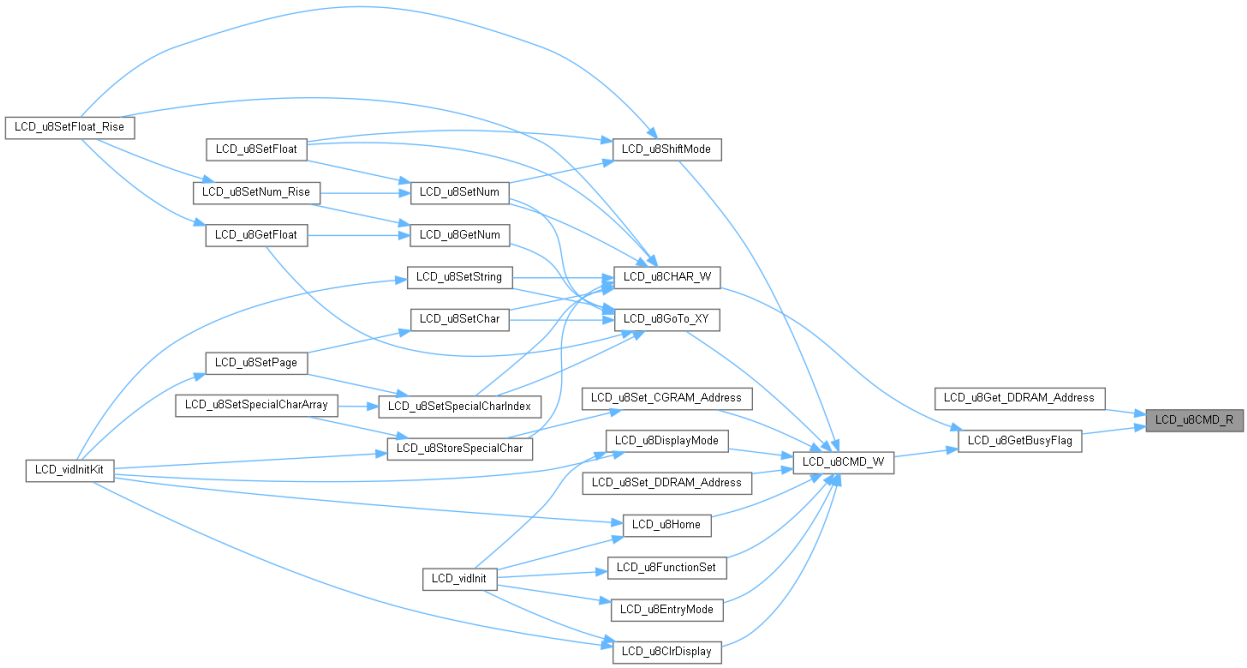
LBTY_tenuErrorStatus LCD_u8CMD_R (u8 * *pu8CMD*)

```
228                                                 {
229     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
230
231     u8RetErrorState = GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_RS, LCD_RS_CMD);
232     u8RetErrorState = LCD_u8Read(pu8CMD);
233
234     return u8RetErrorState;
235 }
```

Here is the call graph for this function:



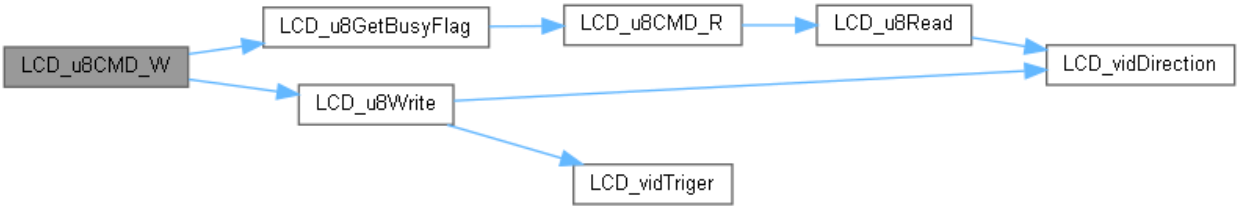
Here is the caller graph for this function:



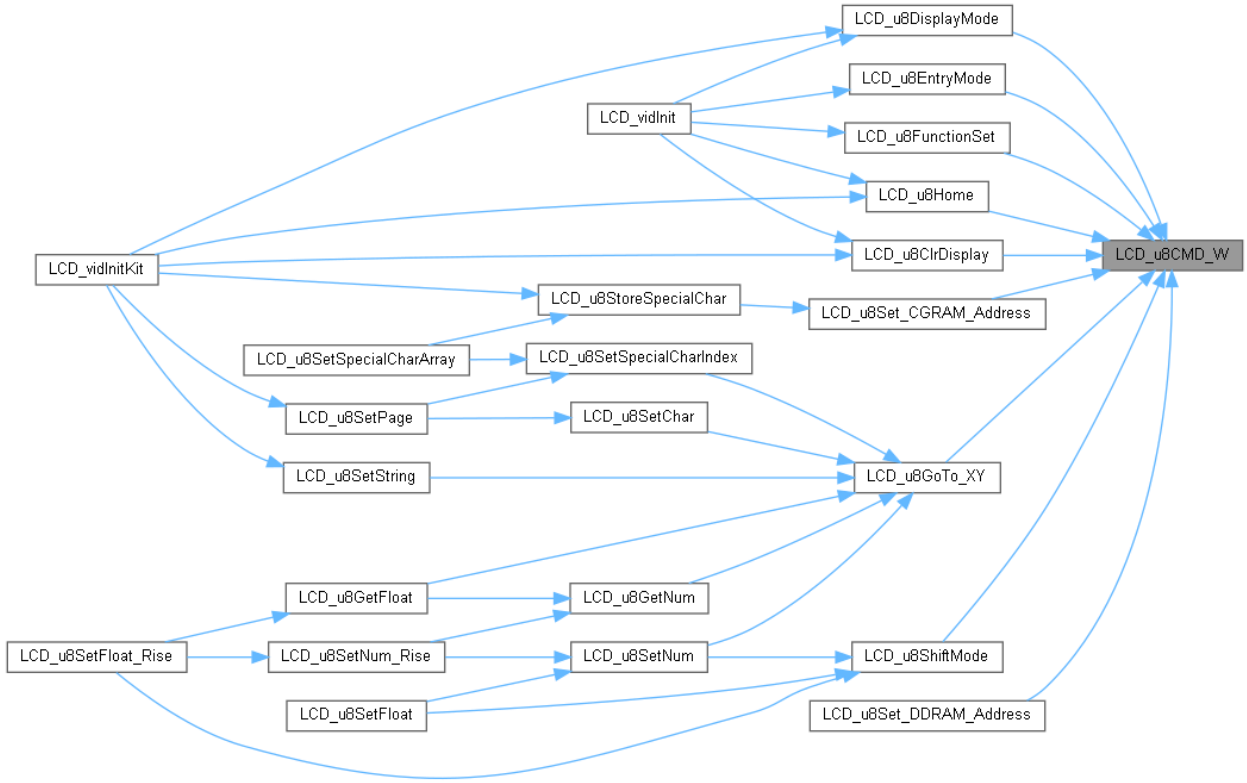
LBTY_tenuErrorStatus LCD_u8CMD_W (u8 *u8CMD*)

```
217                                                 {
218     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
219 #ifdef LCD_RW
220     while(LCD_u8GetBusyFlag())      vidMyDelay_ms(LCD_DELAY_CMD);
221 #endif
222     u8RetErrorState = GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_RS, LCD_RS_CMD);
223     u8RetErrorState = LCD_u8Write(u8CMD);
224 
225     return u8RetErrorState;
226 }
```

Here is the call graph for this function:



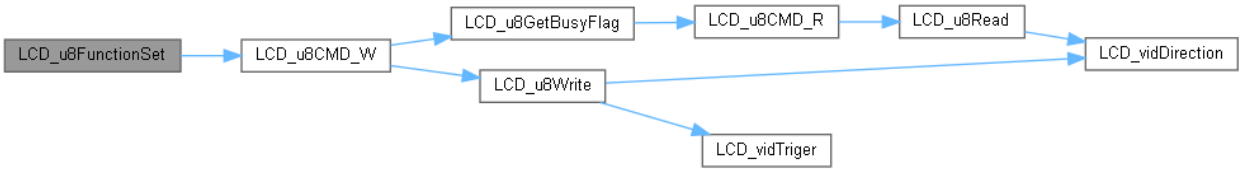
Here is the caller graph for this function:



LBTY_tenuErrorStatus LCD_u8FunctionSet (void)

```
52                                              {
53     LBTY_tenuErrorStatus u8RetErrorState =  LCD_u8CMD_W(LCD_CURSOR_HOME);
54
55     if(u8RetErrorState == LBTY_OK){
56 #ifdef LCD_10DOT
57 #if (LCD_ROW_NUM > LCD_ROW_NUM_1)
58 #if (LCD_FUNCTION_SET == LCD_FUNCTION_SET_8Bits)
59         u8RetErrorState = LCD_u8CMD_W(LCD_CONFIG_2LINE_8BIT_10ROW);
60 #elif  (LCD_FUNCTION_SET == LCD_FUNCTION_SET_4Bits)
61         u8RetErrorState = LCD_u8CMD_W(LCD_CONFIG_2LINE_4BIT_10ROW);
62 #endif
63 #else
64 #if (LCD_FUNCTION_SET == LCD_FUNCTION_SET_8Bits)
65         u8RetErrorState = LCD_u8CMD_W(LCD_CONFIG_1LINE_8BIT_10ROW);
66 #elif  (LCD_FUNCTION_SET == LCD_FUNCTION_SET_4Bits)
67         u8RetErrorState = LCD_u8CMD_W(LCD_CONFIG_1LINE_4BIT_10ROW);
68 #endif
69 #endif
70
71 #else
72
73 #if (LCD_ROW_NUM > LCD_ROW_NUM_1)
74 #if (LCD_FUNCTION_SET == LCD_FUNCTION_SET_8Bits)
75         u8RetErrorState = LCD_u8CMD_W(LCD_CONFIG_2LINE_8BIT_5ROW);
76 #elif  (LCD_FUNCTION_SET == LCD_FUNCTION_SET_4Bits)
77         u8RetErrorState = LCD_u8CMD_W(LCD_CONFIG_2LINE_4BIT_5ROW);
78 #endif
79 #else
80 #if (LCD_FUNCTION_SET == LCD_FUNCTION_SET_8Bits)
81         u8RetErrorState = LCD_u8CMD_W(LCD_CONFIG_1LINE_8BIT_5ROW);
82 #elif  (LCD_FUNCTION_SET == LCD_FUNCTION_SET_4Bits)
83         u8RetErrorState = LCD_u8CMD_W(LCD_CONFIG_1LINE_4BIT_5ROW);
84 #endif
85 #endif
86
87 #endif
88     }
89     return u8RetErrorState;
90 }
```

Here is the call graph for this function:

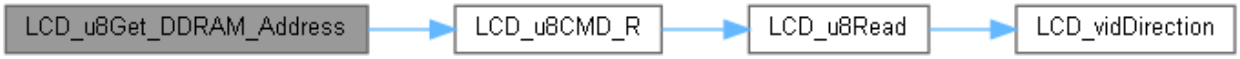


Here is the caller graph for this function:

LBTY_tenuErrorStatus LCD_u8Get_DDRAM_Address (u8 * *pu8Address*)

```
266                                                     {
267     LBTY_tenuErrorStatus u8RetErrorState = LCD_u8CMD_R(pu8Address);
268     *pu8Address = GET_MASK(*pu8Address, LCD_DDRAM_ADDRESS_MASK);
269     return u8RetErrorState;
270 }
```

Here is the call graph for this function:

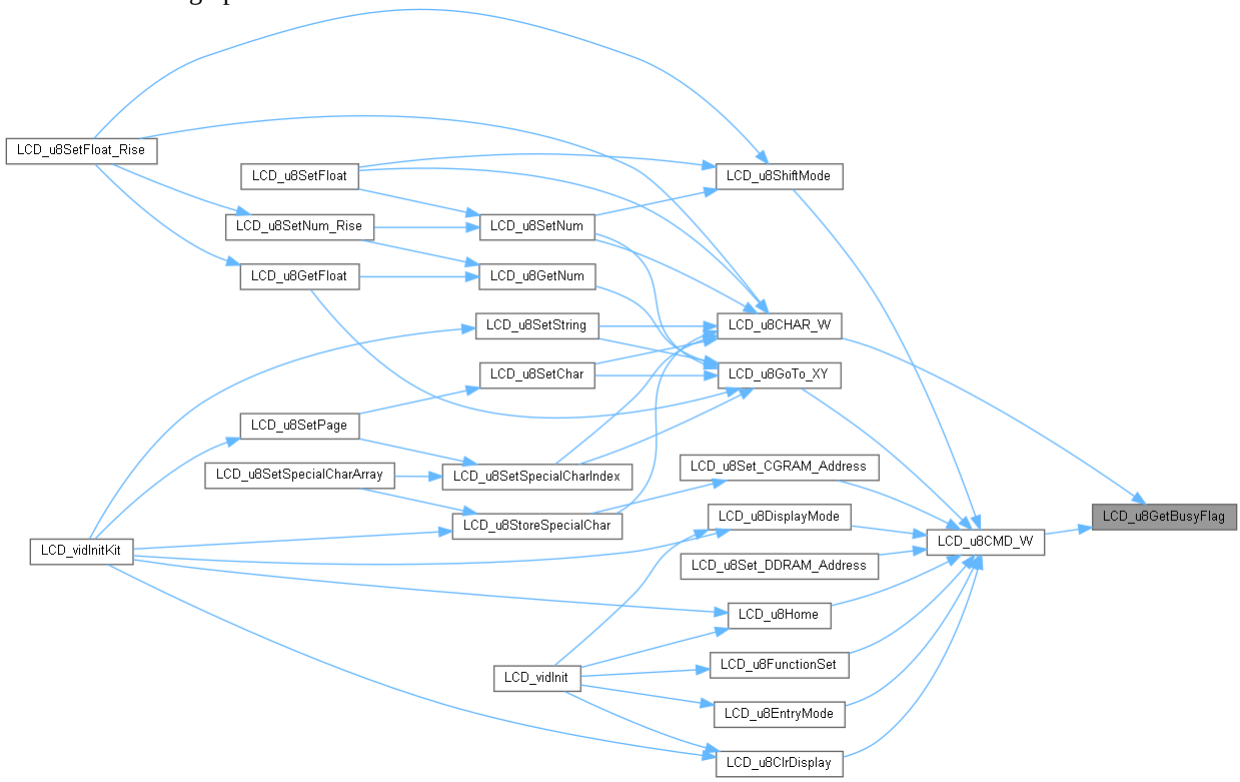


u8 LCD_u8GetBusyFlag (void)

```
272                             {
273     u8 u8RetValue = LBTY_RESET;
274     LCD_u8CMD_R(&u8RetValue);
275     return GET_BIT(u8RetValue, BUSY_FLAG_BIT);
276 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



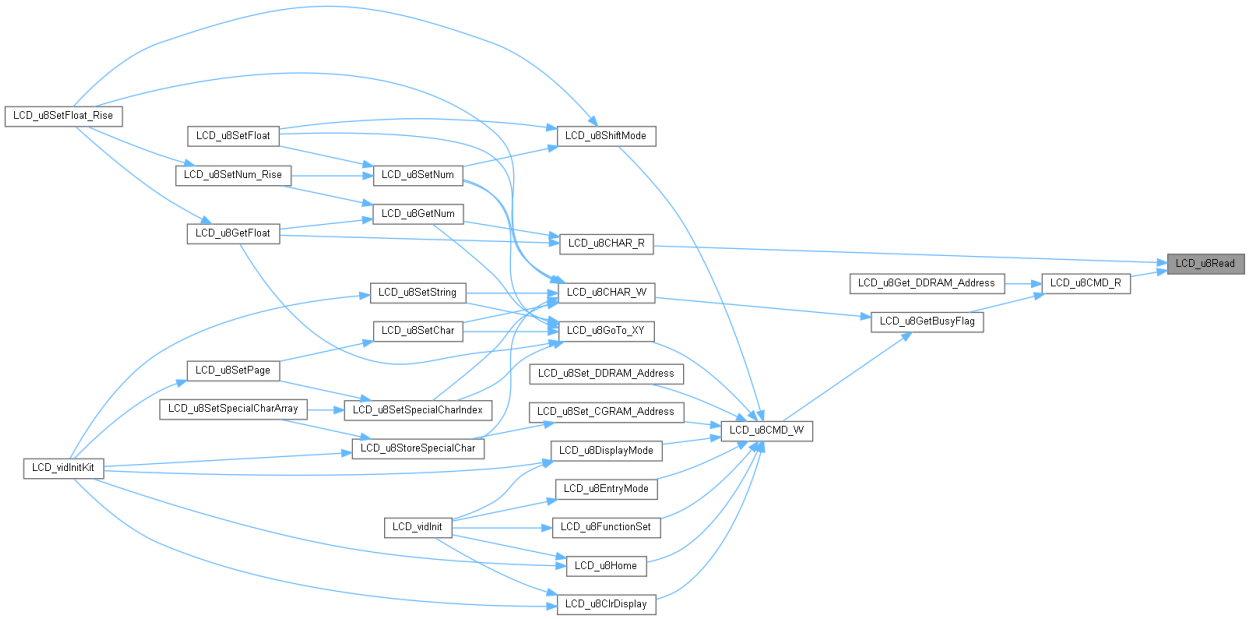
LBTY_tenuErrorStatus LCD_u8Read (u8 * *pu8Byte*)

```
159                                             {
160     LBTY_tenuErrorStatus u8RetErrorState = LBTY_NOK;
161
162 #ifdef LCD_RW
163     LBTY_tuniPort8* pu8LcdByte = (LBTY_tuniPort8*)pu8Byte;
```

```
164     u8 u8ReadValue;
165
166     u8RetErrorState = GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_RW, LCD_RW_READ);
167
168     LCD_vidDirection(PIN_INPUT);
169
170 #if (LCD_FUNCTION_SET == LCD_FUNCTION_SET_4Bits)
171     GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_EN, PIN_High);
172     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D4, &u8ReadValue);
173     pu8LcdByte->sBits.m_u8b4 = u8ReadValue;
174     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D5, &u8ReadValue);
175     pu8LcdByte->sBits.m_u8b5 = u8ReadValue;
176     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D6, &u8ReadValue);
177     pu8LcdByte->sBits.m_u8b6 = u8ReadValue;
178     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D7, &u8ReadValue);
179     pu8LcdByte->sBits.m_u8b7 = u8ReadValue;
180     GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_EN, PIN_Low);
181     vidMyDelay_ms(LCD_DELAY_CMD);
182     GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_EN, PIN_High);
183     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D4, &u8ReadValue);
184     pu8LcdByte->sBits.m_u8b0 = u8ReadValue;
185     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D5, &u8ReadValue);
186     pu8LcdByte->sBits.m_u8b1 = u8ReadValue;
187     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D6, &u8ReadValue);
188     pu8LcdByte->sBits.m_u8b2 = u8ReadValue;
189     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D7, &u8ReadValue);
190     pu8LcdByte->sBits.m_u8b3 = u8ReadValue;
191     GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_EN, PIN_Low);
192 #elif (LCD_FUNCTION_SET == LCD_FUNCTION_SET_8Bits)
193     GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_EN, PIN_High);
194     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D0, &u8ReadValue);
195     pu8LcdByte->sBits.m_u8b0 = u8ReadValue;
196     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D1, &u8ReadValue);
197     pu8LcdByte->sBits.m_u8b1 = u8ReadValue;
198     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D2, &u8ReadValue);
199     pu8LcdByte->sBits.m_u8b2 = u8ReadValue;
200     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D3, &u8ReadValue);
201     pu8LcdByte->sBits.m_u8b3 = u8ReadValue;
202     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D4, &u8ReadValue);
203     pu8LcdByte->sBits.m_u8b4 = u8ReadValue;
204     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D5, &u8ReadValue);
205     pu8LcdByte->sBits.m_u8b5 = u8ReadValue;
206     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D6, &u8ReadValue);
207     pu8LcdByte->sBits.m_u8b6 = u8ReadValue;
208     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D7, &u8ReadValue);
209     pu8LcdByte->sBits.m_u8b7 = u8ReadValue;
210     GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_EN, PIN_Low);
211 #endif
212
213 #endif
214     return u8RetErrorState;
215 }
```

Here is the call graph for this function:

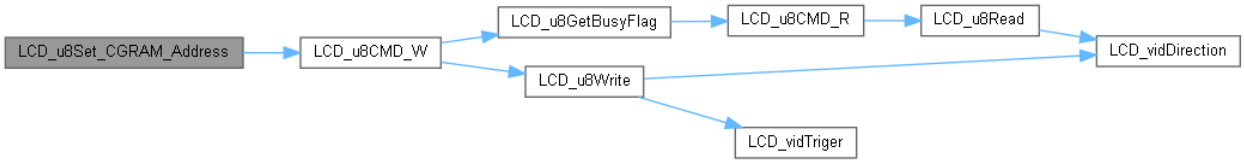
Here is the caller graph for this function:



LBTY_tenuErrorStatus LCD_u8Set_CGRAM_Address (u8 *u8Address*)

```
258                                                     {
259     return LCD_u8CMD_W(LCD_SEND_CGRAM_ADDRESS | GET_MASK(u8Address,
LCD_CGRAM_ADDRESS_MASK));
260 }
```

Here is the call graph for this function:

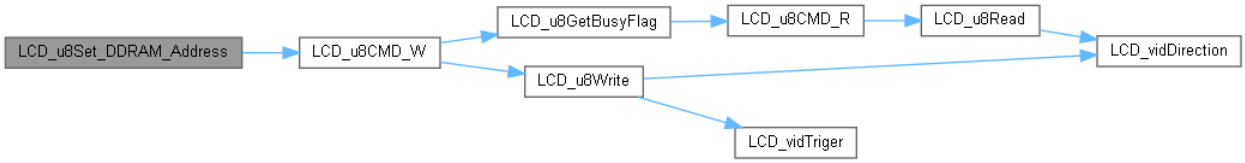


Here is the caller graph for this function:

LBTY_tenuErrorStatus LCD_u8Set_DDRAM_Address (u8 *u8Address*)

```
262                                                     {
263     return LCD_u8CMD_W(LCD_SEND_DDRAM_ADDRESS | GET_MASK(u8Address,
LCD_DDRAM_ADDRESS_MASK));
264 }
```

Here is the call graph for this function:

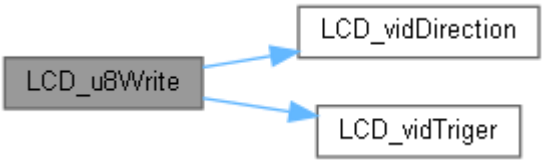


LBTY_tenuErrorStatus LCD_u8Write (u8 *u8Byte*)

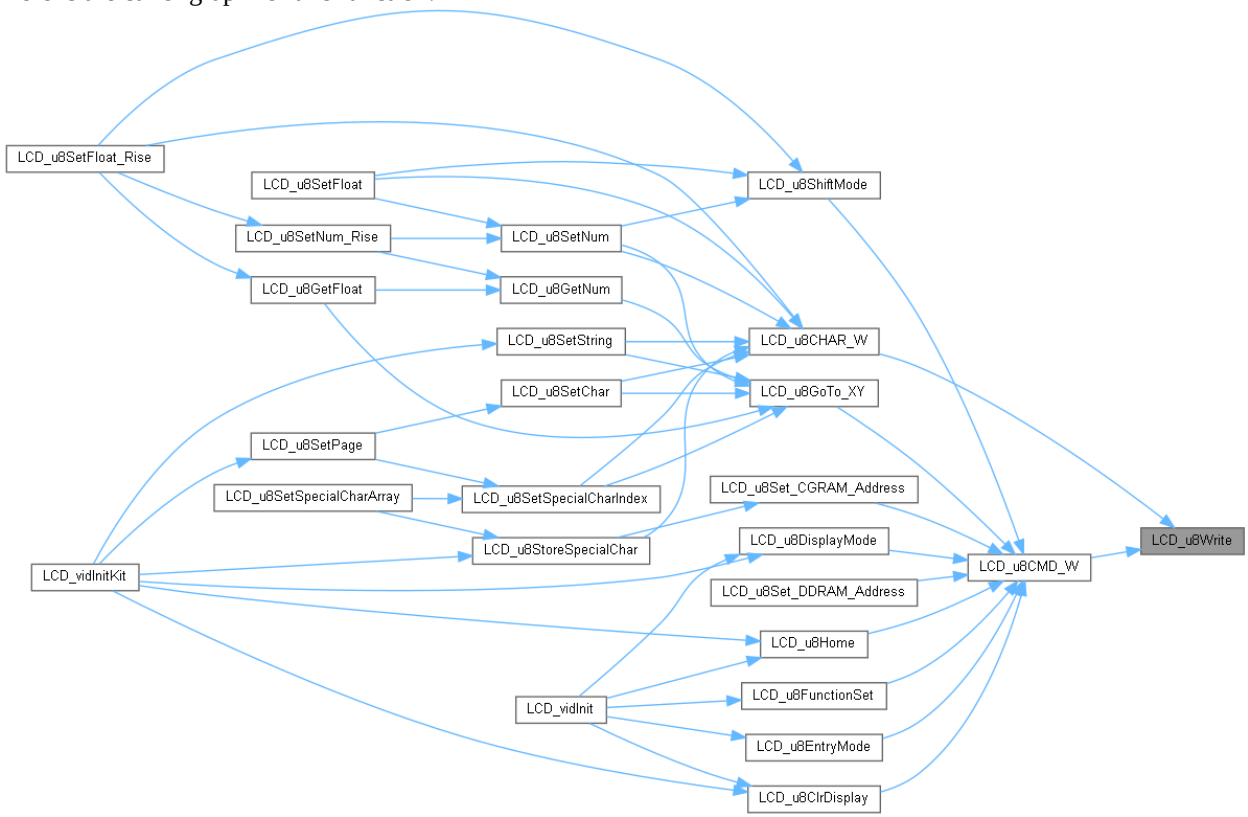
```
124                                             {
125     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
126     LBTY_tuniPort8 u8LcdByte = (LBTY_tuniPort8)u8Byte;
127
128 #ifdef LCD_RW
129     u8RetErrorState = GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_RW,
LCD_RW_WRITE);
130 #endif
131
132     LCD_vidDirection(PIN_OUTPUT);
```

```
133
134 #if (LCD_FUNCTION_SET == LCD_FUNCTION_SET_4Bits)
135     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D4, u8LcdByte.sBits.m_u8b4);
136     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D5, u8LcdByte.sBits.m_u8b5);
137     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D6, u8LcdByte.sBits.m_u8b6);
138     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D7, u8LcdByte.sBits.m_u8b7);
139     LCD_vidTriger();
140     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D4, u8LcdByte.sBits.m_u8b0);
141     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D5, u8LcdByte.sBits.m_u8b1);
142     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D6, u8LcdByte.sBits.m_u8b2);
143     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D7, u8LcdByte.sBits.m_u8b3);
144     LCD_vidTriger();
145 #elif (LCD_FUNCTION_SET == LCD_FUNCTION_SET_8Bits)
146     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D0, u8LcdByte.sBits.m_u8b0);
147     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D1, u8LcdByte.sBits.m_u8b1);
148     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D2, u8LcdByte.sBits.m_u8b2);
149     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D3, u8LcdByte.sBits.m_u8b3);
150     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D4, u8LcdByte.sBits.m_u8b4);
151     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D5, u8LcdByte.sBits.m_u8b5);
152     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D6, u8LcdByte.sBits.m_u8b6);
153     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D7, u8LcdByte.sBits.m_u8b7);
154     LCD_vidTriger();
155 #endif
156     return u8RetErrorState;
157 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

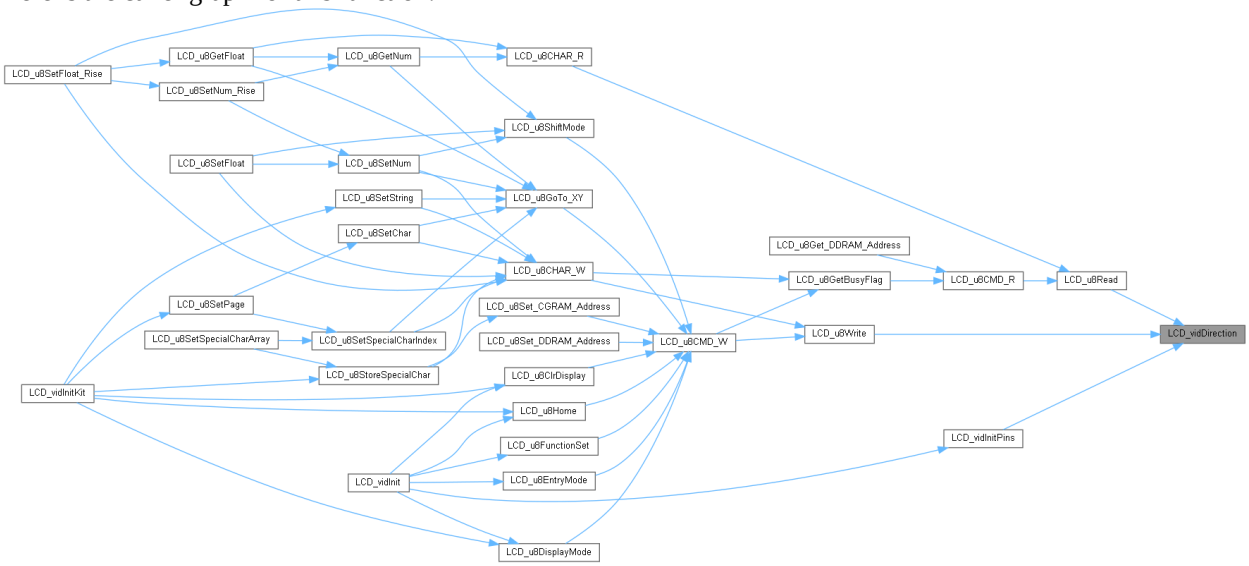


void LCD_vidDirection (u8 *u8PinDir*)

```
104                                 {
105 #if (LCD_FUNCTION_SET == LCD_FUNCTION_SET_8Bits)
106     GPIO_u8SetPinDirection(LCD_DATA_PORT, LCD_D0, u8PinDir);
107     GPIO_u8SetPinDirection(LCD_DATA_PORT, LCD_D1, u8PinDir);
108     GPIO_u8SetPinDirection(LCD_DATA_PORT, LCD_D2, u8PinDir);
```

```
109     GPIO_u8SetPinDirection(LCD_DATA_PORT, LCD_D3, u8PinDir);
110 #endif
111     GPIO_u8SetPinDirection(LCD_DATA_PORT, LCD_D4, u8PinDir);
112     GPIO_u8SetPinDirection(LCD_DATA_PORT, LCD_D5, u8PinDir);
113     GPIO_u8SetPinDirection(LCD_DATA_PORT, LCD_D6, u8PinDir);
114     GPIO_u8SetPinDirection(LCD_DATA_PORT, LCD_D7, u8PinDir);
115 }
```

Here is the caller graph for this function:



void LCD_vidInitPins (void)

```
92                              {
93     vidMyDelay_ms(LCD_DELAY_POWER_ON);       // Delay Power On
94
95     GPIO_u8SetPinDirection(LCD_CONTROL_PORT, LCD_EN, PIN_OUTPUT);
96     GPIO_u8SetPinDirection(LCD_CONTROL_PORT, LCD_RS, PIN_OUTPUT);
97 #ifdef LCD_RW
98     GPIO_u8SetPinDirection(LCD_CONTROL_PORT, LCD_RW, PIN_OUTPUT);
99 #endif
100
101     LCD_vidDirection(PIN_OUTPUT);
102 }
```

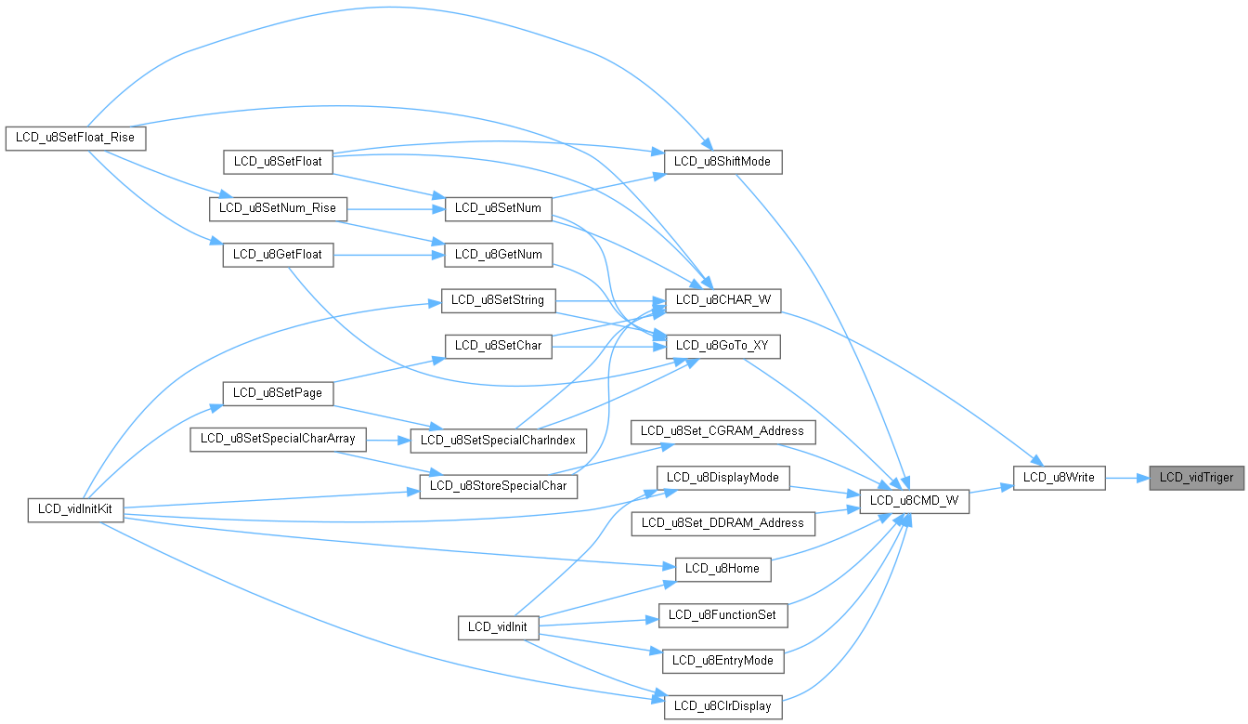
Here is the call graph for this function:

Here is the caller graph for this function:

void LCD_vidTriger (void)

```
117                             {
118     GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_EN, PIN_High);
119     vidMyDelay_ms(LCD_DELAY_CMD);
120     GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_EN, PIN_Low);
121     vidMyDelay_ms(LCD_DELAY_CMD);
122 }
```

Here is the caller graph for this function:



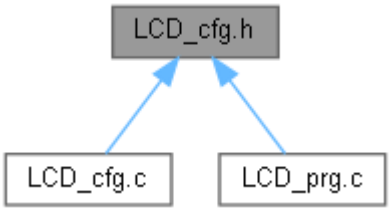
Variable Documentation

const u8 ETA32[][LCD_CGRAM_LOCATIONS_NUM]

```
Initial value:= {
    { 0x1C, 0x1C, 0x0F, 0x0F, 0x0C, 0x0C, 0x00, 0x00},
    { 0x1C, 0x1C, 0x03, 0x03, 0x03, 0x03, 0x06, 0x06},
    { 0x06, 0x0C, 0x0C, 0x0C, 0x18, 0x18, 0x1A, 0x0C}
}
```

LCD_cfg.h File Reference

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



Macros

- #define LCD_DELAY_CMD 1u
- #define LCD_DELAY_POWER_ON 40u
- #define LCD_DELAY_WAIT 50u
- #define LCD_DELAY_PAGE 5000u
- #define LCD_FUNCTION_SET LCD_FUNCTION_SET_8Bits
- #define LCD_COL_NUM LCD_COL_NUM_32
- #define LCD_ROW_NUM LCD_ROW_NUM_2
- #define LCD_DATA_PORT A
- #define LCD_D0 AMIT_LCD0
- #define LCD_D1 AMIT_LCD1
- #define LCD_D2 AMIT_LCD2
- #define LCD_D3 AMIT_LCD3
- #define LCD_D4 AMIT_LCD4
- #define LCD_D5 AMIT_LCD5
- #define LCD_D6 AMIT_LCD6
- #define LCD_D7 AMIT_LCD7
- #define LCD_CONTROL_PORT B
- #define LCD_RS AMIT_LCD_RS
- #define LCD_RW AMIT_LCD_RW
- #define LCD_EN AMIT_LCD_EN

Macro Definition Documentation

#define LCD_COL_NUM LCD_COL_NUM_32

#define LCD_CONTROL_PORT B

#define LCD_D0 AMIT_LCD0

#define LCD_D1 AMIT_LCD1

#define LCD_D2 AMIT_LCD2

#define LCD_D3 AMIT_LCD3

#define LCD_D4 AMIT_LCD4

#define LCD_D5 AMIT_LCD5

#define LCD_D6 AMIT_LCD6

#define LCD_D7 AMIT_LCD7

#define LCD_DATA_PORT A

#define LCD_DELAY_CMD 1u

#define LCD_DELAY_PAGE 5000u

#define LCD_DELAY_POWER_ON 40u

#define LCD_DELAY_WAIT 50u

#define LCD_EN AMIT_LCD_EN

#define LCD_FUNCTION_SET LCD_FUNCTION_SET_8Bits

#define LCD_ROW_NUM LCD_ROW_NUM_2

#define LCD_RS AMIT_LCD_RS

#define LCD_RW AMIT_LCD_RW

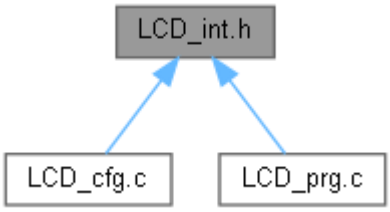
LCD_cfg.h

```
Go to the documentation of this file.1 /*
****************************************************************************** */
2 /* ********************** FILE DEFINITION SECTION ************************** */
3 /* ****************************************************************************** */
4 /* File Name   : LCD_cfg.h                                                    */
5 /* Author      : MAAM                                                         */
6 /* Version     : v01.2                                                        */
7 /* date        : Mar 31, 2023                                                 */
8 /* ****************************************************************************** */
9 /* ************************ HEADER FILES INCLUDES ************************  */
10 /* ****************************************************************************** */
11
12 #ifndef LCD_CFG_H_
13 #define LCD_CFG_H_
14
15 /* ****************************************************************************** */
16 /* ********************** TYPE_DEF/STRUCT/ENUM SECTION ********************** */
17 /* ****************************************************************************** */
18
19 /* ****************************************************************************** */
20 /* ************************** MACRO/DEFINE SECTION ************************** */
21 /* ****************************************************************************** */
22
23 #define LCD_DELAY_CMD       1u
24 #define LCD_DELAY_POWER_ON  40u
25 #define LCD_DELAY_WAIT      50u
26 #define LCD_DELAY_PAGE      5000u
27
28 #if defined(AMIT_KIT)
29
30 #define LCD_FUNCTION_SET    LCD_FUNCTION_SET_4Bits
31
32 #define LCD_COL_NUM         LCD_COL_NUM_16
33 #define LCD_ROW_NUM         LCD_ROW_NUM_2
34
35 #define LCD_DATA_PORT       A
36 #define LCD_D4              AMIT_LCD4
37 #define LCD_D5              AMIT_LCD5
38 #define LCD_D6              AMIT_LCD6
39 #define LCD_D7              AMIT_LCD7
40
41 #define LCD_CONTROL_PORT    B
42 #define LCD_RS              AMIT_LCD_RS
43 #define LCD_RW              AMIT_LCD_RW
44 #define LCD_EN              AMIT_LCD_EN
45
46 #elif defined(ETA32_KIT)
47
48 #define LCD_FUNCTION_SET    LCD_FUNCTION_SET_4Bits
49
50 #define LCD_COL_NUM         LCD_COL_NUM_20
51 #define LCD_ROW_NUM         LCD_ROW_NUM_4
52
53 #define LCD_DATA_PORT       B
54 #define LCD_D4              Eta32_LCD4
55 #define LCD_D5              Eta32_LCD5
56 #define LCD_D6              Eta32_LCD6
57 #define LCD_D7              Eta32_LCD7
58
59 #define LCD_CONTROL_PORT    A
60 #define LCD_RS              Eta32_LCD_RS
61 #define LCD_EN              Eta32_LCD_EN
62
63 #elif defined(ETA32_MINI_KIT)
64
65 #define LCD_FUNCTION_SET    LCD_FUNCTION_SET_4Bits
66
67 #define LCD_COL_NUM         LCD_COL_NUM_16
68 #define LCD_ROW_NUM         LCD_ROW_NUM_2
69
70 #define LCD_DATA_PORT       A
71 #define LCD_D4              Eta32_mini_LCD4
72 #define LCD_D5              Eta32_mini_LCD5
```

```
73 #define LCD_D6              Eta32_mini_LCD6
74 #define LCD_D7              Eta32_mini_LCD7
75
76 #define LCD_CONTROL_PORT    A
77 #define LCD_RS              Eta32_mini_LCD_RS
78 #define LCD_EN              Eta32_mini_LCD_EN
79
80 #else
81
82 #define LCD_FUNCTION_SET    LCD_FUNCTION_SET_8Bits
83
84 #define LCD_COL_NUM         LCD_COL_NUM_32
85 #define LCD_ROW_NUM         LCD_ROW_NUM_2
86
87 #define LCD_DATA_PORT       A
88 #define LCD_D0              AMIT_LCD0
89 #define LCD_D1              AMIT_LCD1
90 #define LCD_D2              AMIT_LCD2
91 #define LCD_D3              AMIT_LCD3
92 #define LCD_D4              AMIT_LCD4
93 #define LCD_D5              AMIT_LCD5
94 #define LCD_D6              AMIT_LCD6
95 #define LCD_D7              AMIT_LCD7
96
97 #define LCD_CONTROL_PORT    B
98 #define LCD_RS              AMIT_LCD_RS
99 #define LCD_RW              AMIT_LCD_RW
100 #define LCD_EN               AMIT_LCD_EN
101
102 #endif
103
104 /* ********************************************************************************** */
105 /* ****************************** CONST SECTION ****************************** */
106 /* ********************************************************************************** */
107
108 /* ********************************************************************************** */
109 /* ****************************** VARIABLE SECTION ****************************** */
110 /* ********************************************************************************** */
111
112 /* ********************************************************************************** */
113 /* ****************************** FUNCTION SECTION ****************************** */
114 /* ********************************************************************************** */
115
116
117 #endif /* LCD_CFG_H_ */
118 /***************************** E N D (LCD_cfg.h) *****************************/
```

LCD_int.h File Reference

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



Macros

- #define LCD_FUNCTION_SET_8Bits 0u
- #define LCD_FUNCTION_SET_4Bits 1u
- #define LCD_COL_NUM_8 8u
- #define LCD_COL_NUM_16 16u
- #define LCD_COL_NUM_20 20u
- #define LCD_COL_NUM_32 32u
- #define LCD_ROW_NUM_1 1u
- #define LCD_ROW_NUM_2 2u
- #define LCD_ROW_NUM_4 4u
- #define LCD_FLOAT_MUL 100
- #define LCD_vidGetPrintf(str, ...) sprintf(str, __VA_ARGS__);

Enumerations

- enum LCD_tenuFunctionSet { LCD_Function_Set_8Bits = (u8)0u, LCD_Function_Set_4Bits }
- enum LCD_tenuEntryMode { LCD_Entry_Dec = (u8)0u, LCD_Entry_Dec_Shift, LCD_Entry_Inc, LCD_Entry_Inc_Shift }
- enum LCD_tenuDisplayCursorControl { LCD_Display_OFF = (u8)0u, LCD_Cursor_OFF, LCD_Cursor_OFF_Blink, LCD_Cursor_UnderLine, LCD_Cursor_UnderLine_Blinking }
- enum LCD_tenuDisplayCursorShift { LCD_Cursor_Shift_Left = (u8)0u, LCD_Cursor_Shift_Right, LCD_Display_Shift_Left, LCD_Display_Shift_Right }
- enum LCD_tenuLinePosition { LCD_Line_1 = (u8)0u, LCD_Line_2, LCD_Line_3, LCD_Line_4 }

Functions

- void LCD_vidInit (void)
- void LCD_vidInitKit (void)
- LBTY_tenuErrorStatus LCD_u8SetPage (const u8 pu8String1[], const u8 pu8String2[])
- LBTY_tenuErrorStatus LCD_u8SetString (const u8 pu8String[], u8 u8Row, u8 u8Col)
- LBTY_tenuErrorStatus LCD_u8SetChar (u8 u8Char, u8 u8Row, u8 u8Col)
- LBTY_tenuErrorStatus LCD_u8GetNum (s32 *ps32Num, u8 u8Col, u8 u8Row)
- LBTY_tenuErrorStatus LCD_u8SetNum (s32 s32Num, u8 u8Col, u8 u8Row)
- LBTY_tenuErrorStatus LCD_u8GetFloat (f32 *pf32Num, u8 u8Col, u8 u8Row)
- LBTY_tenuErrorStatus LCD_u8SetFloat (f32 f32Num, u8 u8Col, u8 u8Row)
- LBTY_tenuErrorStatus LCD_u8SetNum_Rise (s32 s32Num, u8 u8Col, u8 u8Row)
- LBTY_tenuErrorStatus LCD_u8SetFloat_Rise (f32 f32Num, u8 u8Col, u8 u8Row)
- LBTY_tenuErrorStatus LCD_u8GoTo_XY (u8 u8Col, u8 u8Row)
- LBTY_tenuErrorStatus LCD_u8ClrDisplay (void)
- LBTY_tenuErrorStatus LCD_u8Home (void)
- LBTY_tenuErrorStatus LCD_u8EntryMode (LCD_tenuEntryMode u8Mode)
- LBTY_tenuErrorStatus LCD_u8DisplayMode (LCD_tenuDisplayCursorControl u8Mode)
- LBTY_tenuErrorStatus LCD_u8ShiftMode (LCD_tenuDisplayCursorShift u8Shift)
- LBTY_tenuErrorStatus LCD_u8StoreSpecialChar (u8 *pu8char, u8 u8Index)

- LBTY_tenuErrorStatus LCD_u8SetSpecialCharIndex (u8 u8Index, u8 u8Col, u8 u8Row)
- LBTY_tenuErrorStatus LCD_u8SetSpecialCharArray (u8 *pu8char, u8 u8Col, u8 u8Row)

Macro Definition Documentation

#define LCD_COL_NUM_16 16u

#define LCD_COL_NUM_20 20u

#define LCD_COL_NUM_32 32u

#define LCD_COL_NUM_8 8u

#define LCD_FLOAT_MUL 100

#define LCD_FUNCTION_SET_4Bits 1u

#define LCD_FUNCTION_SET_8Bits 0u

#define LCD_ROW_NUM_1 1u

#define LCD_ROW_NUM_2 2u

#define LCD_ROW_NUM_4 4u

#define LCD_vidGetPrintf(str, ...) sprintf(str, __VA_ARGS__);

Enumeration Type Documentation

enum LCD_tenuDisplayCursorControl

Enumerator:

LCD_Display_OFF	
LCD_Cursor_OFF	
LCD_Cursor_OFF_Blink	
LCD_Cursor_UnderLine	
LCD_Cursor_UnderLine_Blinking	

```
31              {
32      LCD_Display_OFF = (u8)0u,
33      LCD_Cursor_OFF,
34      LCD_Cursor_OFF_Blink,
35      LCD_Cursor_UnderLine,
36      LCD_Cursor_UnderLine_Blinking
37  }LCD_tenuDisplayCursorControl;
```

enum LCD_tenuDisplayCursorShift

Enumerator:

LCD_Cursor_Shift_Left	
LCD_Cursor_Shift_Right	
LCD_Display_Shift_Left	
LCD_Display_Shift_Right	

```
39              {
40      LCD_Cursor_Shift_Left = (u8)0u,
41      LCD_Cursor_Shift_Right,
42      LCD_Display_Shift_Left,
43      LCD_Display_Shift_Right
44  }LCD_tenuDisplayCursorShift;
```

enum LCD_tenuEntryMode

Enumerator:

LCD_Entry_Dec	
LCD_Entry_Dec_Shift	
LCD_Entry_Inc	
LCD_Entry_Inc_Shift	

```
24              {
25      LCD_Entry_Dec = (u8)0u,
26      LCD_Entry_Dec_Shift,
27      LCD_Entry_Inc,
28      LCD_Entry_Inc_Shift
29  }LCD_tenuEntryMode;
```

enum LCD_tenuFunctionSet

Enumerator:

LCD_Function_Set_8Bits	
LCD_Function_Set_4Bits	

```
19              {
20      LCD_Function_Set_8Bits = (u8)0u,
21      LCD_Function_Set_4Bits
22  }LCD_tenuFunctionSet;
```

enum LCD_tenuLinePosition

Enumerator:

LCD_Line_1	
LCD_Line_2	
LCD_Line_3	
LCD_Line_4	

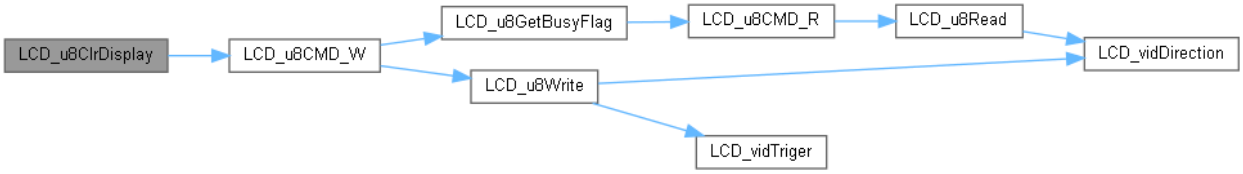
```
46              {
47      LCD_Line_1 = (u8)0u,
48      LCD_Line_2,
49      LCD_Line_3,
50      LCD_Line_4
51  }LCD_tenuLinePosition;
```

Function Documentation

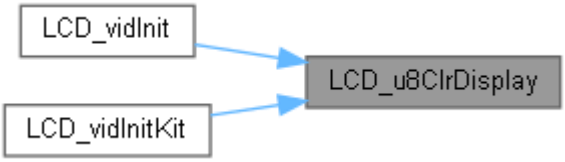
LBTY_tenuErrorStatus LCD_u8ClrDisplay (void)

```
535                                                 {
536     return LCD_u8CMD_W(LCD_CLEAR_DISPLAY);
537 }
```

Here is the call graph for this function:



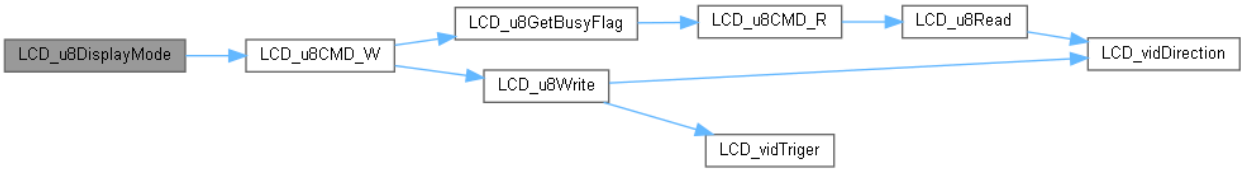
Here is the caller graph for this function:



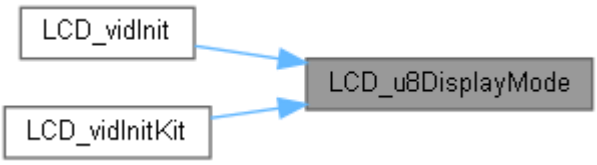
LBTY_tenuErrorStatus LCD_u8DisplayMode (LCD_tenuDisplayCursorControl *u8Mode*)

```
580                                                                                 {
581     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
582
583     switch(u8Mode){
584     case LCD_Display_OFF:
585         u8RetErrorState = LCD_u8CMD_W(LCD_DISPLAY_OFF_CURSOR_OFF);
586         break;
587     case LCD_Cursor_OFF:
588         u8RetErrorState = LCD_u8CMD_W(LCD_DISPLAY_ON_CURSOR_OFF);
589         break;
590     case LCD_Cursor_OFF_Blink:
591         u8RetErrorState = LCD_u8CMD_W(LCD_DISPLAY_ON_CURSOR_OFF_BLINK);
592         break;
593     case LCD_Cursor_UnderLine:
594         u8RetErrorState = LCD_u8CMD_W(LCD_DISPLAY_ON_CURSOR_UNDERLINE);
595         break;
596     case LCD_Cursor_UnderLine_Blinking:
597         u8RetErrorState = LCD_u8CMD_W(LCD_DISPLAY_ON_CURSOR_BLINK);
598         break;
599     default:
600         u8RetErrorState = LBTY_NOK;
601     }
602     return u8RetErrorState;
603 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

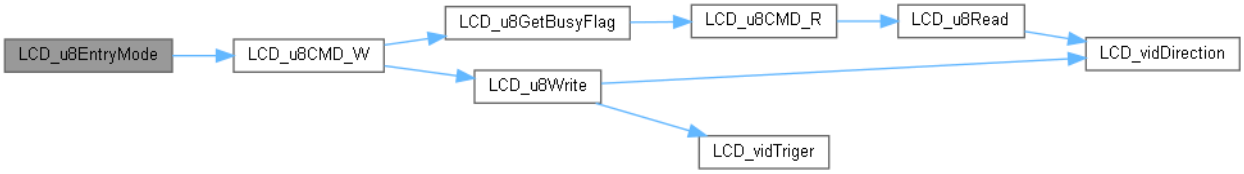


LBTY_tenuErrorStatus LCD_u8EntryMode (LCD_tenuEntryMode *u8Mode*)

```
553                                                 {
554     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
555
556     switch(u8Mode){
```

```
557     case LCD_Entry_Dec:
558         u8RetErrorState = LCD_u8CMD_W(LCD_Entry_DEC);
559         break;
560     case LCD_Entry_Dec_Shift:
561         u8RetErrorState = LCD_u8CMD_W(LCD_Entry_DEC_SHIFT);
562         break;
563     case LCD_Entry_Inc:
564         u8RetErrorState = LCD_u8CMD_W(LCD_Entry_INC);
565         break;
566     case LCD_Entry_Inc_Shift:
567         u8RetErrorState = LCD_u8CMD_W(LCD_Entry_INC_SHIFT);
568         break;
569     default:
570         u8RetErrorState = LBTY_NOK;
571     }
572     return u8RetErrorState;
573 }
```

Here is the call graph for this function:

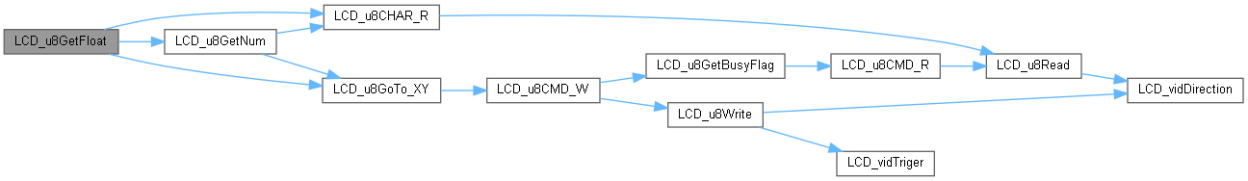


Here is the caller graph for this function:

LBTY_tenuErrorStatus LCD_u8GetFloat (f32 * *pf32Num*, u8 *u8Col*, u8 *u8Row*)

```
277                                                                                 {
278     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
279     u8 u8Log = 1u;
280     u32 u32Factor = 1u;
281     u8 u8Char = LBTY_u8ZERO;
282     s32 s32NumL = LBTY_u32ZERO;
283     s32 s32NumR = LBTY_u32ZERO;
284 
285     u8RetErrorState = LCD_u8GetNum(&s32NumL, u8Col, u8Row);
286     if(u8RetErrorState == LBTY_OK){
287 
288         for(u32 i = (u32)((s32NumL >= (s32)LBTY_u32ZERO) ? s32NumL : s32NumL * -1.0)
; i/=10 ; u8Log++);
289         u8Col += u8Log + ((s32NumL >= (s32)LBTY_u32ZERO) ? 1u : 2u);
290 
291         while(!u8RetErrorState){
292             if(LCD_u8GoTo_XY(u8Col++, u8Row)){
293                 u8RetErrorState = LBTY_NULL_POINTER;
294             }else{
295                 u8RetErrorState = LCD_u8CHAR_R(&u8Char);
296                 if(u8Char>='0' && u8Char<='9'){
297                     s32NumR = (s32NumR * 10) + (u8Char - '0');
298                     u32Factor *= 10;
299                     continue;
300                 }else if(u8Char == '.'){
301                 }
302                 break;
303             }
304         }
305     }
306     if(s32NumL >= (s32)LBTY_u32ZERO){
307         *pf32Num = s32NumL + (f32)s32NumR / u32Factor;
308     }else{
309         *pf32Num = s32NumL - (f32)s32NumR / u32Factor;
310     }
311 
312     return u8RetErrorState;
313 }
```

Here is the call graph for this function:

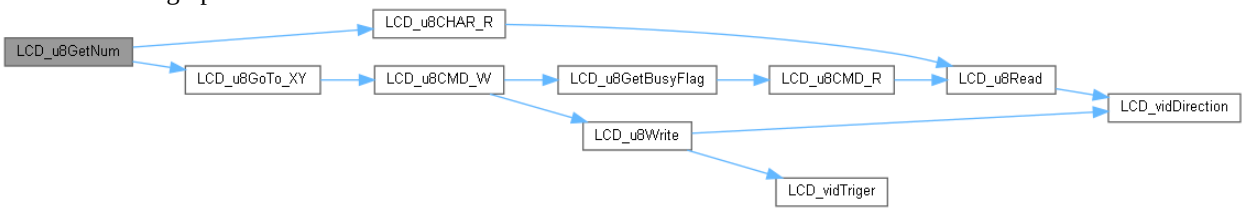


Here is the caller graph for this function:

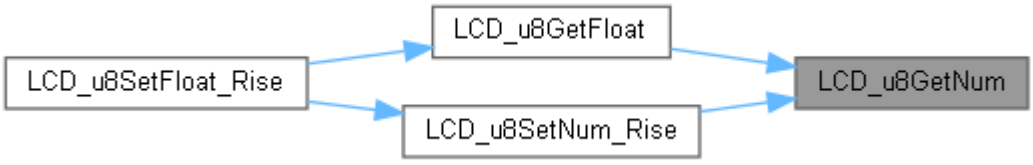
LBTY_tenuErrorStatus LCD_u8GetNum (s32 * *ps32Num*, u8 *u8Col*, u8 *u8Row*)

```
214                                                                 {
215     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
216     u8 u8Char = LBTY_u8ZERO;
217     u8 u8SignChar = LBTY_RESET;
218     u8 u8FirstCharFlag = LBTY_SET;
219     u32 u32NumRead = LBTY_u32ZERO;
220 
221     while(!u8RetErrorState){
222         if(LCD_u8GoTo_XY(u8Col++, u8Row)){
223             u8RetErrorState = LBTY_NULL_POINTER;
224         }else{
225             u8RetErrorState = LCD_u8CHAR_R(&u8Char);
226             if(u8FirstCharFlag && (u8Char=='-' || u8Char=='+')){
227                 if(u8Char=='-')      {   u8SignChar = LBTY_SET;  }
228                 else if(u8Char=='+'){   u8SignChar = LBTY_RESET;}
229                 u8FirstCharFlag = LBTY_RESET;
230                 continue;
231             }else if(u8Char>='0' && u8Char<='9'){
232                 u32NumRead = (u32NumRead * 10) + (u8Char - '0');
233                 continue;
234             }else if(u8Char == '.'){
235             }
236             break;
237         }
238     }
239     *ps32Num = (u8SignChar? (s32)u32NumRead * -1 : (s32)u32NumRead);
240     return u8RetErrorState;
241 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

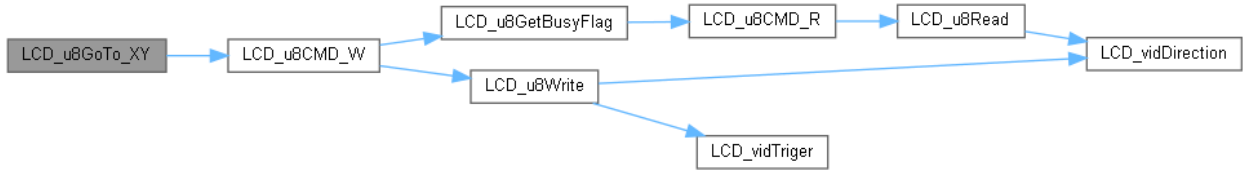


LBTY_tenuErrorStatus LCD_u8GoTo_XY (u8 *u8Col*, u8 *u8Row*)

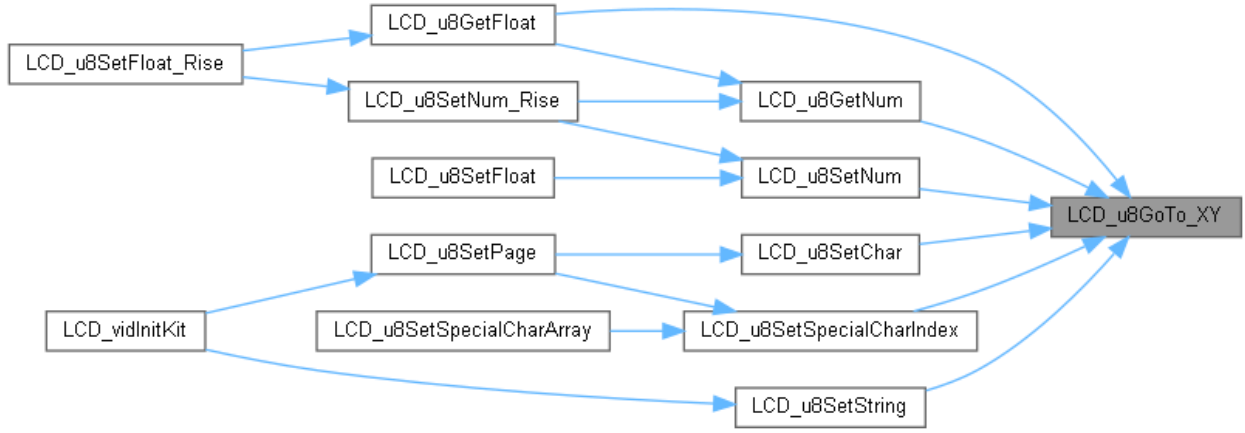
```
500                                                                 {
501     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
502     if((u8Col < LCD_COL_NUM)){
503         switch(u8Row){
504         case LCD_Line_1:    LCD_u8CMD_W(LCD_FIRST_LINE_POSITION_0  + u8Col);
break;
505 #if (LCD_ROW_NUM >= LCD_ROW_NUM_2)
506         case LCD_Line_2:    LCD_u8CMD_W(LCD_SECOND_LINE_POSITION_0 + u8Col);
break;
507 #endif
508 #if (LCD_ROW_NUM >= LCD_ROW_NUM_4)
509 #if (LCD_COL_NUM == LCD_COL_NUM_16)
```

```
510         case LCD_Line_3:    LCD_u8CMD_W(LCD_THIRD_LINE_POSITION_0  + u8Col);
break;
511         case LCD_Line_4:    LCD_u8CMD_W(LCD_FOURTH_LINE_POSITION_0 + u8Col);
break;
512 #endif
513 #if (LCD_COL_NUM == LCD_COL_NUM_20)
514         case LCD_Line_3:    LCD_u8CMD_W(LCD_THIRD_LINE_POSITION_0_20  + u8Col);
break;
515         case LCD_Line_4:    LCD_u8CMD_W(LCD_FOURTH_LINE_POSITION_0_20 + u8Col);
break;
516 #endif
517 #if (LCD_COL_NUM == LCD_COL_NUM_32)
518         case LCD_Line_3:    LCD_u8CMD_W(LCD_THIRD_LINE_POSITION_0_32  + u8Col);
break;
519         case LCD_Line_4:    LCD_u8CMD_W(LCD_FOURTH_LINE_POSITION_0_32 + u8Col);
break;
520 #endif
521 #endif
522         default:            u8RetErrorState = LBTY_NULL_POINTER;;
break;
523         }
524     }else{
525         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
526     }
527     return u8RetErrorState;
528 }
```

Here is the call graph for this function:



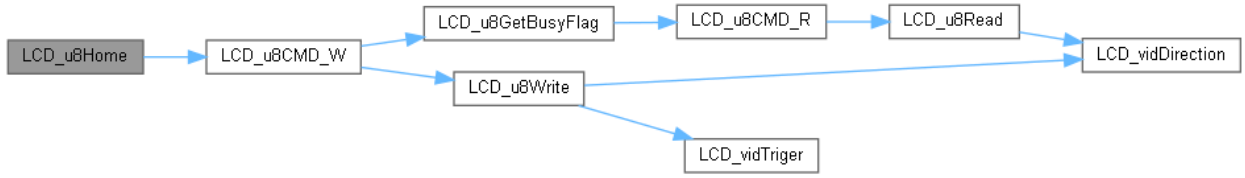
Here is the caller graph for this function:



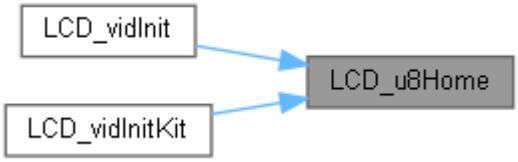
LBTY_tenuErrorStatus LCD_u8Home (void)

```
544                                     {
545     return LCD_u8CMD_W(LCD_CURSOR_HOME);
546 }
```

Here is the call graph for this function:



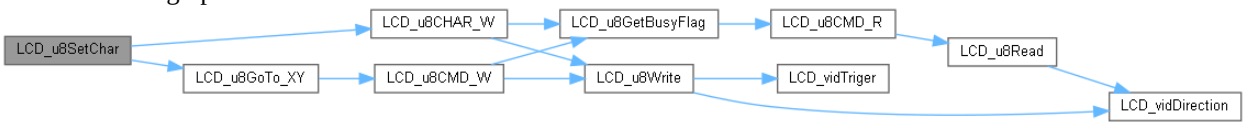
Here is the caller graph for this function:



LBTY_tenuErrorStatus LCD_u8SetChar (u8 *u8Char*, u8 *u8Row*, u8 *u8Col*)

```
198                                                                 {
199     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
200     if(LCD_u8GoTo_XY(u8Col, u8Row)){
201         u8RetErrorState = LBTY_NOK;
202     }else{
203         u8RetErrorState = LCD_u8CHAR_W(u8Char);
204     }
205     return u8RetErrorState;
206 }
```

Here is the call graph for this function:

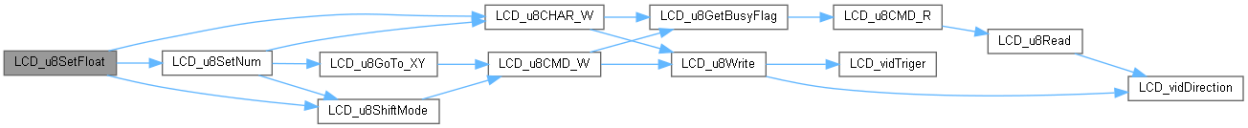


Here is the caller graph for this function:

LBTY_tenuErrorStatus LCD_u8SetFloat (f32 *f32Num*, u8 *u8Col*, u8 *u8Row*)

```
320                                                                 {
321     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
322
323     LCD_u8SetNum((s32)f32Num, u8Col, u8Row);
324     LCD_u8CHAR_W('.');
325
326     if(f32Num < 0.0){
327         f32Num *= -1.0;
328         u8Col++;
329     }
330
331     u32 u32Lcd_R = (u32)((u32)(f32Num * LCD_FLOAT_MUL) % LCD_FLOAT_MUL);
332
333     for(u32 u32Factor = LCD_FLOAT_MUL; u32Factor/=10 ; ){
334         LCD_u8CHAR_W(((u32Lcd_R/u32Factor)%10u) + '0');
335     }
336     LCD_u8CHAR_W(' ');
337     LCD_u8CHAR_W(' ');
338     LCD_u8ShiftMode(LCD_Cursor_Shift_Left);
339     LCD_u8ShiftMode(LCD_Cursor_Shift_Left);
340
341     return u8RetErrorState;
342 }
```

Here is the call graph for this function:



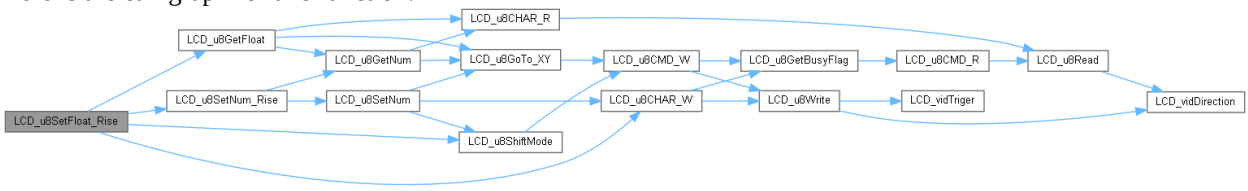
LBTY_tenuErrorStatus LCD_u8SetFloat_Rise (f32 *f32Num*, u8 *u8Col*, u8 *u8Row*)

TODO: Redesign this function with float work

```
420                                                                 {
421     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
422     u32 u32NumNew = LBTY_u32ZERO;
423     u32 u32NumRead = LBTY_u32ZERO;
424
425     f32 f32LcdRead = 0.0f;
426     u8RetErrorState = LCD_u8GetFloat(&f32LcdRead, u8Col, u8Row);
427
428     if(u8RetErrorState == LBTY_OK){
429
432         u8RetErrorState = LCD_u8SetNum_Rise((s32)f32Num, u8Col, u8Row);
433
434         if(u8RetErrorState){
435             LCD_u8CHAR_W('.');
436
437             if(f32Num >= (f32)LBTY_u32ZERO){
```

```
438                 u32NumNew = ((u32)( f32Num * LCD_FLOAT_MUL) % (u32)LCD_FLOAT_MUL);
439             }else{
440                 u32NumNew = ((u32)(-f32Num * LCD_FLOAT_MUL) %
(u32)LCD_FLOAT_MUL);
441             }
442             if(f32LcdRead >= (f32)LBTY_u32ZERO){
443                 u32NumRead = ((u32)( f32LcdRead * LCD_FLOAT_MUL) %
(u32)LCD_FLOAT_MUL);
444             }else{
445                 u32NumRead = ((u32)(-f32LcdRead * LCD_FLOAT_MUL) %
(u32)LCD_FLOAT_MUL);
446             }
447
448             u32 u32Step = LCD_FLOAT_MUL;
449             if(!u8RetErrorState && u32NumRead != u32NumNew){
450                 if(u32NumRead < u32NumNew){
451                     while(u32Step){
452                         if(u32NumNew >= (u32NumRead + u32Step)){
453                             u32NumRead += u32Step;
454                             u8RetErrorState = LBTY_NOK;
455                             break;
456                         }
457                         u32Step /= 10;
458                     }
459                 }else if(u32NumRead > u32NumNew){
460                     while(u32Step){
461                         if(u32NumNew <= (u32NumRead - u32Step)){
462                             while(u32Step){
463                                 if(u32NumRead >= u32Step){
464                                     u32NumRead -= u32Step;
465                                     u8RetErrorState = LBTY_NOK;
466                                     break;
467                                 }
468                                 u32Step /= 10;
469                             }
470                             break;
471                         }
472                         u32Step /= 10;
473                     }
474                 }else{
475                     u8RetErrorState = LBTY_OK;
476                 }
477             }
478             if(u8RetErrorState){
479                 for(u32 u32Factor = LCD_FLOAT_MUL; u32Factor/=10 ; ){
480                     LCD_u8CHAR_W(((u32NumRead/u32Factor)%10u) + '0');
481                 }
482                 LCD_u8CHAR_W(' ');
483                 LCD_u8CHAR_W(' ');
484                 LCD_u8ShiftMode(LCD_Cursor_Shift_Left);
485                 LCD_u8ShiftMode(LCD_Cursor_Shift_Left);
486             }
487         }
488     }else{
489         u8RetErrorState = LBTY_OK;
490     }
491
492     return u8RetErrorState;
493 }
```

Here is the call graph for this function:

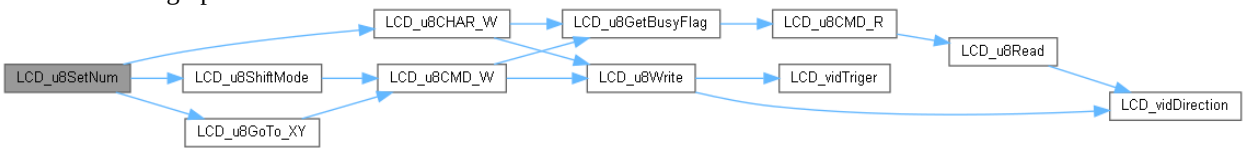


LBTY_tenuErrorStatus LCD_u8SetNum (s32 *s32Num*, u8 *u8Col*, u8 *u8Row*)

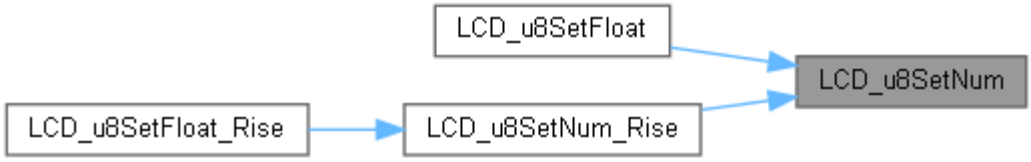
```
248                                                                 {
249     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
250     u8 u8Log = 1u;
251     u32 u32Factor = 1u;
252     u32 u32NumLoc = (u32)((s32Num >= (s32)LBTY_u32ZERO) ? s32Num : s32Num * -1);
253
```

```
254     if(LCD_u8GoTo_XY(u8Col, u8Row)){
255         u8RetErrorState = LBTY_NOK;
256     }else{
257         for(u32 i = u32NumLoc ; i/=10 ; u8Log++, u32Factor*=10);
258         if(s32Num<0){
259             u8RetErrorState = LCD_u8CHAR_W('-');
260         }
261         while(u8Log--){
262             u8RetErrorState = LCD_u8CHAR_W((u8)((u32NumLoc/u32Factor)%10u) +
'0');
263             u32Factor/=10;
264         }
265         LCD_u8CHAR_W(' ');
266         LCD_u8ShiftMode(LCD_Cursor_Shift_Left);
267     }
268     return u8RetErrorState;
269 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

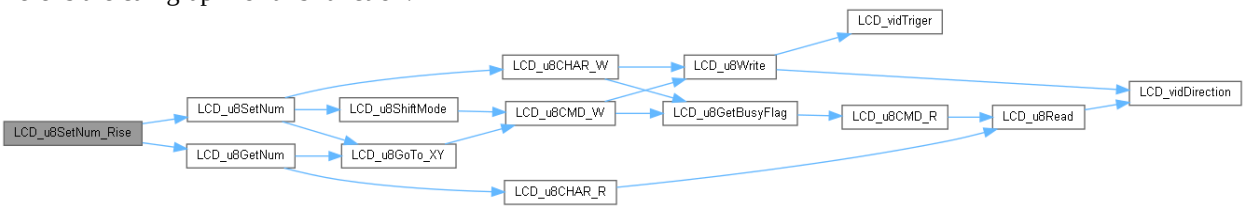


LBTY_tenuErrorStatus LCD_u8SetNum_Rise (s32 *s32Num*, u8 *u8Col*, u8 *u8Row*)

```
349                                                                     {
350     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
351     u8 u8SignChar = LBTY_RESET;
352     u32 u32NumNew = LBTY_u32ZERO;
353     u32 u32NumRead = LBTY_u32ZERO;
354 
355     s32 s32LcdNum = (s32)LBTY_u32ZERO;
356     u8RetErrorState = LCD_u8GetNum(&s32LcdNum, u8Col, u8Row);
357 
358     if(u8RetErrorState == LBTY_OK){
359         if(s32Num >= (s32)LBTY_u32ZERO){
360             u32NumNew = (u32)s32Num;
361             u8SignChar = LBTY_RESET;
362             if((s32LcdNum < (s32)LBTY_u32ZERO)){
363                 u32NumRead = (u32)(s32LcdNum * -1);
364             }else{
365                 u32NumRead = (u32)s32LcdNum;
366             }
367         }else{
368             u32NumNew = (u32)(s32Num * -1);
369             u8SignChar = LBTY_SET;
370             if((s32LcdNum < (s32)LBTY_u32ZERO)){
371                 u32NumRead = (u32)(s32LcdNum * -1);
372             }else{
373                 u32NumRead = (u32)s32LcdNum;
374                 s32LcdNum *= -1;
375             }
376         }
377 
378         u32 u32Step = LCD_FLOAT_MUL;
379         if(u32NumRead != u32NumNew){
380             if(u32NumRead < u32NumNew){
381                 while(u32Step){
382                     if(u32NumNew >= (u32NumRead + u32Step)){
383                         u32NumRead += u32Step;
384                         break;
385                     }
386                     u32Step /= 10;
387                 }
388             }else if(u32NumRead > u32NumNew){
389                 while(u32Step){
```

```
390                     if(u32NumNew <= (u32NumRead - u32Step)){
391                         while(u32Step){
392                             if(u32NumRead >= u32Step){
393                                 u32NumRead -= u32Step;
394                                 break;
395                             }
396                             u32Step /= 10;
397                         }
398                         break;
399                     }
400                     u32Step /= 10;
401                 }
402             }
403             s32LcdNum = u8SignChar ? (s32)u32NumRead * -1 : (s32)u32NumRead;
404             LCD_u8SetNum(s32LcdNum, u8Col, u8Row);
405             u8RetErrorState = LBTY_NOK;
406         }
407 
408     }else{
409         u8RetErrorState = LBTY_OK;
410     }
411 
412     return u8RetErrorState;
413 }
```

Here is the call graph for this function:



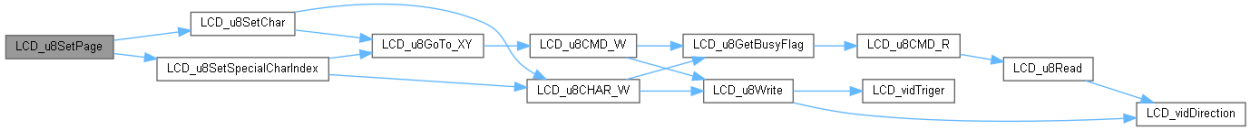
Here is the caller graph for this function:

LBTY_tenuErrorStatus LCD_u8SetPage (const u8 *pu8String1*[], const u8 *pu8String2*[])

```
129 
{
130     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
131     for(u8 i = 0, j = 0 ; i<LCD_COL_NUM && *pu8String1 && *pu8String2 ; i++){
132         if(*pu8String1 == '@'){
133             if((u8RetErrorState = LCD_u8SetSpecialCharIndex(j++, i, 0))){
134                 break;
135             }
136         }else{
137             if((u8RetErrorState = LCD_u8SetChar(*pu8String1, i, 0))){
138                 break;
139             }
140         }
141         if(*pu8String2 == '@'){
142             if((u8RetErrorState = LCD_u8SetSpecialCharIndex(j++, i, 1))){
143                 break;
144             }
145         }else{
146             if((u8RetErrorState = LCD_u8SetChar(*pu8String2, i, 1))){
147                 break;
148             }
149         }
150 #if (LCD_ROW_NUM >= LCD_ROW_NUM_4)
151         if(*(pu8String1 + LCD_COL_NUM) == '@'){
152             if((u8RetErrorState = LCD_u8SetSpecialCharIndex(j++, i, 2))){
153                 break;
154             }
155         }else{
156             if((u8RetErrorState = LCD_u8SetChar(*(pu8String1 + LCD_COL_NUM), i,
2))){
157                 break;
158             }
159         }
160         if(*(pu8String2 + LCD_COL_NUM) == '@'){
```

```
161             if((u8RetErrorState = LCD_u8SetSpecialCharIndex(j++, i, 3))){
162                 break;
163             }
164         }else{
165             if((u8RetErrorState = LCD_u8SetChar(*(pu8String2 + LCD_COL_NUM), i,
3))){
166                 break;
167             }
168         }
169 #endif
170         vidMyDelay_ms(LCD_DELAY_WAIT);
171         pu8String1++;
172         pu8String2++;
173     }
174     vidMyDelay_ms(LCD_DELAY_PAGE);
175     return u8RetErrorState;
176 }
```

Here is the call graph for this function:

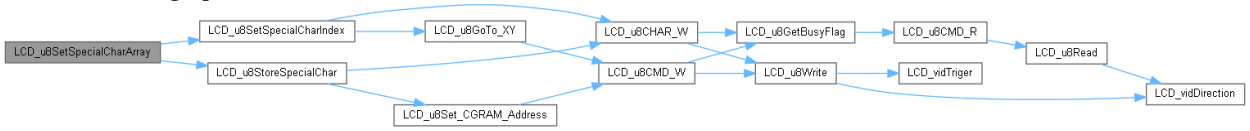


Here is the caller graph for this function:

LBTY_tenuErrorStatus LCD_u8SetSpecialCharArray (u8 * *pu8char*, u8 *u8Col*, u8 *u8Row*)

```
679
{
680     static u8 u8Index = LBTY_u8ZERO;
681     LBTY_tenuErrorStatus u8RetErrorState =
682             LCD_u8StoreSpecialChar(pu8char,u8Index);
683
684     if(u8RetErrorState ==  LBTY_OK){
685         u8RetErrorState = LCD_u8SetSpecialCharIndex(u8Index++, u8Col, u8Row);
686         u8Index = u8Index % LCD_CGRAM_SECTIONS_NUM;
687     }
688     return u8RetErrorState;
689 }
```

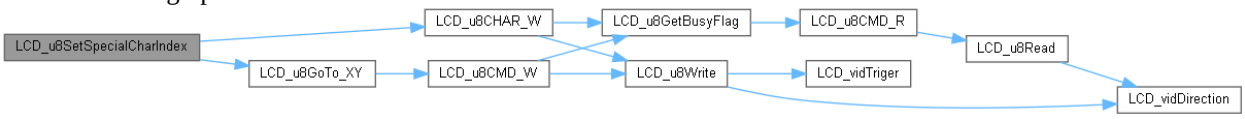
Here is the call graph for this function:



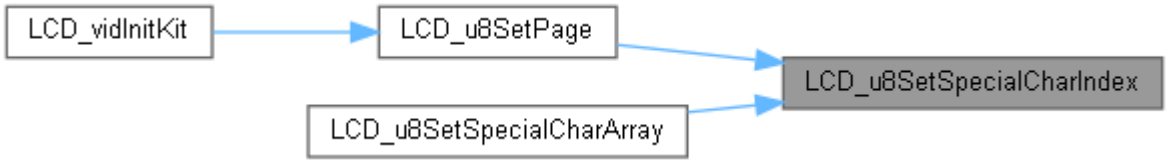
LBTY_tenuErrorStatus LCD_u8SetSpecialCharIndex (u8 *u8Index*, u8 *u8Col*, u8 *u8Row*)

```
658
{
659     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
660
661     if(LCD_u8GoTo_XY(u8Col, u8Row)){
662         u8RetErrorState = LBTY_NOK;
663     }else{
664         if(u8Index < LCD_CGRAM_SECTIONS_NUM){
665             u8RetErrorState = LCD_u8CHAR_W(u8Index);
666         }else{
667             u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
668         }
669     }
670
671     return u8RetErrorState;
672 }
```

Here is the call graph for this function:



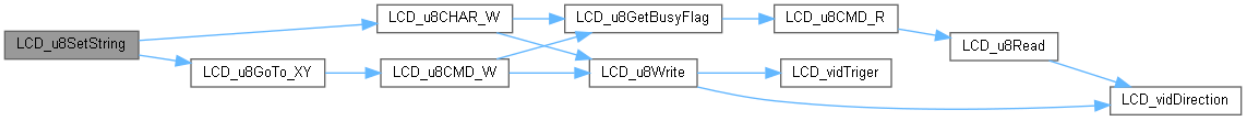
Here is the caller graph for this function:



LBTY_tenuErrorStatus LCD_u8SetString (const u8 *pu8String*[], u8 *u8Row*, u8 *u8Col*)

```
184
{
185     LBTY_tenuErrorStatus u8RetErrorState = LCD_u8GoTo_XY(u8Col, u8Row);
186     while(*pu8String){
187         if(u8RetErrorState)         break;
188         u8RetErrorState = LCD_u8CHAR_W(*(pu8String++));
189     }
190     return u8RetErrorState;
191 }
```

Here is the call graph for this function:

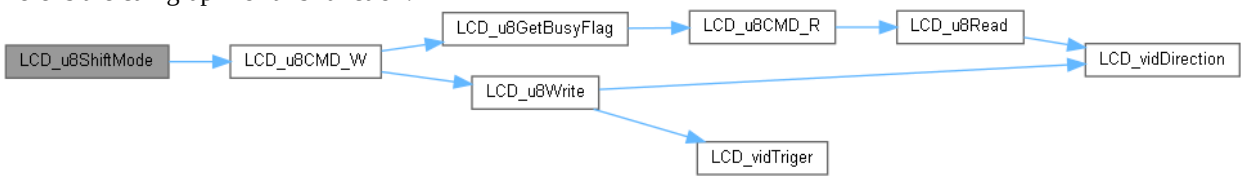


Here is the caller graph for this function:

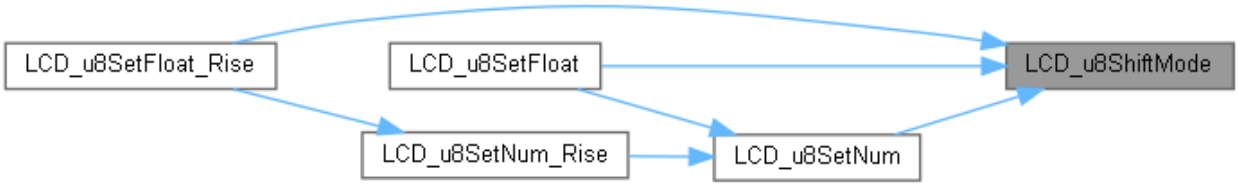
LBTY_tenuErrorStatus LCD_u8ShiftMode (LCD_tenuDisplayCursorShift *u8Shift*)

```
610                                                                 {
611     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
612
613     switch(u8Shift){
614     case LCD_Cursor_Shift_Left:
615         u8RetErrorState = LCD_u8CMD_W(LCD_CURSOR_SHIFT_LEFT);
616         break;
617     case LCD_Cursor_Shift_Right:
618         u8RetErrorState = LCD_u8CMD_W(LCD_CURSOR_SHIFT_RIGHT);
619         break;
620     case LCD_Display_Shift_Left:
621         u8RetErrorState = LCD_u8CMD_W(LCD_DISPLAY_SHIFT_LEFT);
622         break;
623     case LCD_Display_Shift_Right:
624         u8RetErrorState = LCD_u8CMD_W(LCD_DISPLAY_SHIFT_RIGHT);
625         break;
626     default:
627         u8RetErrorState = LBTY_NOK;
628     }
629     return u8RetErrorState;
630 }
```

Here is the call graph for this function:



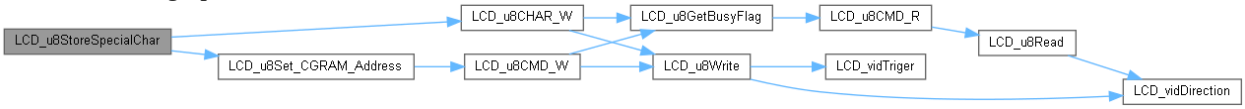
Here is the caller graph for this function:



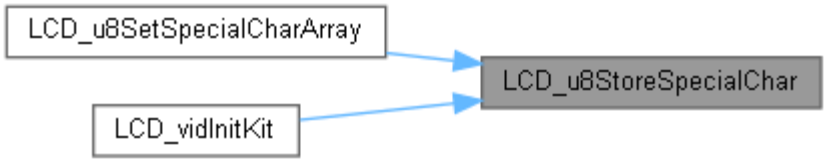
LBTY_tenuErrorStatus LCD_u8StoreSpecialChar (u8 * *pu8char*, u8 *u8Index*)

```
638                                                                     {
639     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
640
641     if(u8Index < LCD_CGRAM_SECTIONS_NUM){
642         u8RetErrorState = LCD_u8Set_CGRAM_Address(u8Index *
LCD_CGRAM_LOCATIONS_NUM);
643         for(u8 i = 0 ; i<LCD_CGRAM_LOCATIONS_NUM  && !u8RetErrorState ; i++){
644             u8RetErrorState = LCD_u8CHAR_W(pu8char[i]);
645         }
646     }else{
647         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
648     }
649
650     return u8RetErrorState;
651 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

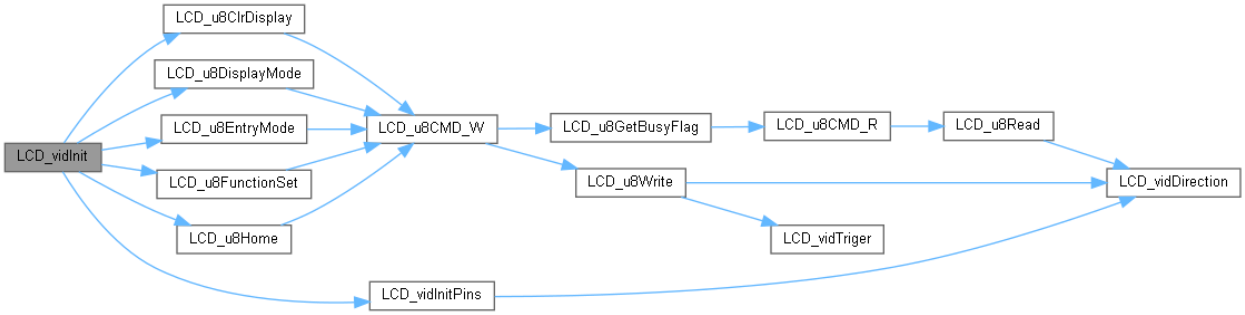


void LCD_vidInit (void)

TODO: sprintf function

```
52                          {
53     LCD_vidInitPins();
54     LCD_u8Home();
55     LCD_u8FunctionSet();
56     LCD_u8ClrDisplay();
57     LCD_u8EntryMode(LCD_Entry_Inc);
58     LCD_u8DisplayMode(LCD_Cursor_OFF);
59 }
```

Here is the call graph for this function:

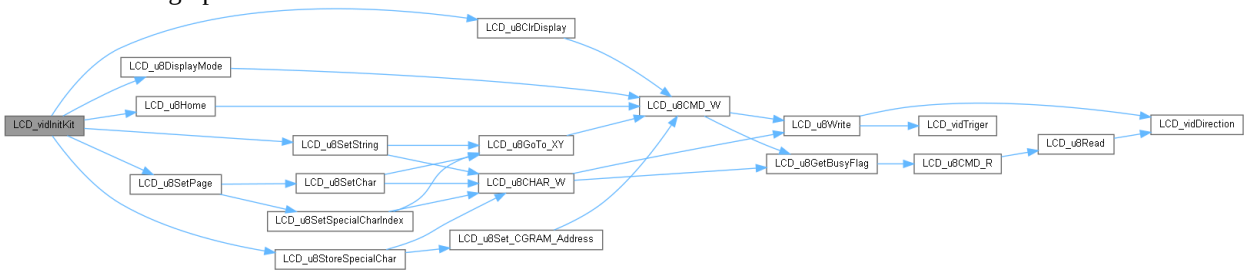


void LCD_vidInitKit (void)

```
61                              {
62
63     LCD_u8Home();
64     LCD_u8ClrDisplay();
65     LCD_u8DisplayMode(LCD_Cursor_OFF);
66
67 #if defined(AMIT_KIT)
68
69     LCD_u8SetString((u8*)"Welcome To", 3, 0);
70     LCD_u8SetString((u8*)"AMIT"      , 6, 1);
71     vidMyDelay_ms(LCD_DELAY_PAGE);
72     LCD_u8ClrDisplay();
73     LCD_u8SetString((u8*)"Choose a"  , 4, 0);
74     LCD_u8SetString((u8*)"Peripheral", 3, 1);
75     vidMyDelay_ms(LCD_DELAY_PAGE);
76     LCD_u8ClrDisplay();
```

```
77
78 #elif defined(ETA32_KIT)
79
80     for(u8 i = 0 ; i<3u ; i++){
81         LCD_u8StoreSpecialChar((u8*)ETA32[i], i);
82     }
83
84     LCD_u8SetPage((u8*)"====================Educational Products"
85             , (u8*)"FARES PCB Co. for   ====================");
86
87     LCD_u8SetPage((u8*)"====================       @32 Kit      "
88             , (u8*)"      @@            ====================");
89
90     LCD_u8SetPage((u8*)"====================   Eta32 Test Code  "
91             , (u8*)"    ATMEL AVR Kit   ====================");
92
93 #elif defined(ETA32_MINI_KIT)
94
95     for(u8 i = 0 ; i<3u ; i++){
96         LCD_u8StoreSpecialChar((u8*)ETA32[i], i);
97     }
98
99     LCD_u8SetPage((u8*)" FARES PCB Co.  "
100             , (u8*)"================");
101
102     LCD_u8SetPage((u8*)"    @@          "
103             , (u8*)"     @32 mini   ");
104
105     LCD_u8SetPage((u8*)" ATMEL AVR Kit  "
106             , (u8*)" Eta32 mini Kit ");
107
108     LCD_u8SetString((u8*)"Eta32mini Kit",    1, 0);
109     LCD_u8SetString((u8*)"SW -- is pressed", 0, 1);
110     vidMyDelay_ms(LCD_DELAY_PAGE);
111
112 #else
113
114     LCD_u8SetString((u8*)"=====Welcome====", 0, 0);
115     LCD_u8SetString((u8*)"FARES PCB Co.",    1, 1);
116     vidMyDelay_ms(LCD_DELAY_PAGE);
117
118     LCD_u8ClrDisplay();
119
120 #endif
121
122 }
```

Here is the call graph for this function:



LCD_int.h

Go to the documentation of this file.

```c
/*
 ******************************************************************************* */
/* ********************** FILE DEFINITION SECTION ************************** */
/* ******************************************************************************* */
/* File Name    : LCD_int.h                                                      */
/* Author       : MAAM                                                           */
/* Version      : v01.2                                                          */
/* date         : Mar 31, 2023                                                   */
/* ******************************************************************************* */
/* ************************ HEADER FILES INCLUDES ************************  */
/* ******************************************************************************* */

#ifndef LCD_INT_H_
#define LCD_INT_H_

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

typedef enum{
    LCD_Function_Set_8Bits = (u8)0u,
    LCD_Function_Set_4Bits
}LCD_tenuFunctionSet;

typedef enum{
    LCD_Entry_Dec = (u8)0u,
    LCD_Entry_Dec_Shift,
    LCD_Entry_Inc,
    LCD_Entry_Inc_Shift
}LCD_tenuEntryMode;

typedef enum{
    LCD_Display_OFF = (u8)0u,
    LCD_Cursor_OFF,
    LCD_Cursor_OFF_Blink,
    LCD_Cursor_UnderLine,
    LCD_Cursor_UnderLine_Blinking
}LCD_tenuDisplayCursorControl;

typedef enum{
    LCD_Cursor_Shift_Left = (u8)0u,
    LCD_Cursor_Shift_Right,
    LCD_Display_Shift_Left,
    LCD_Display_Shift_Right
}LCD_tenuDisplayCursorShift;

typedef enum{
    LCD_Line_1 = (u8)0u,
    LCD_Line_2,
    LCD_Line_3,
    LCD_Line_4
}LCD_tenuLinePosition;

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

#define LCD_FUNCTION_SET_8Bits          0u
#define LCD_FUNCTION_SET_4Bits          1u

#define LCD_COL_NUM_8                   8u
#define LCD_COL_NUM_16                  16u
#define LCD_COL_NUM_20                  20u
#define LCD_COL_NUM_32                  32u

#define LCD_ROW_NUM_1                   1u
#define LCD_ROW_NUM_2                   2u
#define LCD_ROW_NUM_4                   4u

#define LCD_FLOAT_MUL                   100

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

```

```
73 #define LCD_vidGetPrintf(str, ...)      sprintf(str, __VA_ARGS__);
74
75 /* ********************************************************************************* */
76 /* ***************************** CONST SECTION ************************************* */
77 /* ********************************************************************************* */
78
79 /* ********************************************************************************* */
80 /* ***************************** VARIABLE SECTION ********************************** */
81 /* ********************************************************************************* */
82
83 /* ********************************************************************************* */
84 /* ***************************** FUNCTION SECTION ********************************** */
85 /* ********************************************************************************* */
86
87 /* ********************************************************************************* */
88 /* Description :    Initialize the LCD configuration and pins                        */
89 /* Input       :    void                                                             */
90 /* Return      :    void                                                             */
91 /* ********************************************************************************* */
92 extern void LCD_vidInit(void);
93 extern void LCD_vidInitKit(void);
94
95 /* ********************************************************************************* */
96 /* Description :    Set Page of LCD Column by Column                                 */
97 /* Input/Output:    pu8String1, pu8String2                                           */
98 /* Return      :    LBTY_tenuErrorStatus                                             */
99 /* ********************************************************************************* */
100 extern LBTY_tenuErrorStatus LCD_u8SetPage(const u8 pu8String1[],const u8
pu8String2[]);
101
102 /* ********************************************************************************* */
103 /* Description :    Set String at X Column and Y Row                                 */
104 /* Input       :    u8Col, u8Row                                                     */
105 /* Input/Output:    pu8String                                                        */
106 /* Return      :    LBTY_tenuErrorStatus                                             */
107 /* ********************************************************************************* */
108 extern LBTY_tenuErrorStatus LCD_u8SetString(const u8 pu8String[], u8 u8Row, u8 u8Col);
109
110 /* ********************************************************************************* */
111 /* Description :    Set Char at X Column and Y Row                                   */
112 /* Input       :    u8Char, u8Col, u8Row                                             */
113 /* Return      :    LBTY_tenuErrorStatus                                             */
114 /* ********************************************************************************* */
115 extern LBTY_tenuErrorStatus LCD_u8SetChar(u8 u8Char, u8 u8Row, u8 u8Col);
116
117 /* ********************************************************************************* */
118 /* Description :    Get Num at X Column and Y Row                                    */
119 /* Input       :    u8Col, u8Row                                                     */
120 /* Input/Output:    ps32Num                                                          */
121 /* Return      :    LBTY_tenuErrorStatus                                             */
122 /* ********************************************************************************* */
123 extern LBTY_tenuErrorStatus LCD_u8GetNum(s32* ps32Num, u8 u8Col, u8 u8Row);
124
125 /* ********************************************************************************* */
126 /* Description :    Set Num at X Column and Y Row                                    */
127 /* Input       :    s32Num, u8Col, u8Row                                             */
128 /* Return      :    LBTY_tenuErrorStatus                                             */
129 /* ********************************************************************************* */
130 extern LBTY_tenuErrorStatus LCD_u8SetNum(s32 s32Num, u8 u8Col, u8 u8Row);
131
132 /* ********************************************************************************* */
133 /* Description :    Get Real float Num at X Column and Y Row                         */
134 /* Input       :    u8Col, u8Row                                                     */
135 /* Input/Output:    pf32Num                                                          */
136 /* Return      :    LBTY_tenuErrorStatus                                             */
137 /* ********************************************************************************* */
138 extern LBTY_tenuErrorStatus LCD_u8GetFloat(f32* pf32Num, u8 u8Col, u8 u8Row);
139
140 /* ********************************************************************************* */
141 /* Description :    Set Real float Num at X Column and Y Row                         */
142 /* Input       :    f32Num, u8Col, u8Row                                             */
143 /* Return      :    LBTY tenuErrorStatus                                             */
144 /* ********************************************************************************* */
145 extern LBTY_tenuErrorStatus LCD_u8SetFloat(f32 f32Num, u8 u8Col, u8 u8Row);
146
147 /* ********************************************************************************* */
148 /* Description :    Set Num Rise at X Column and Y Row                               */
```

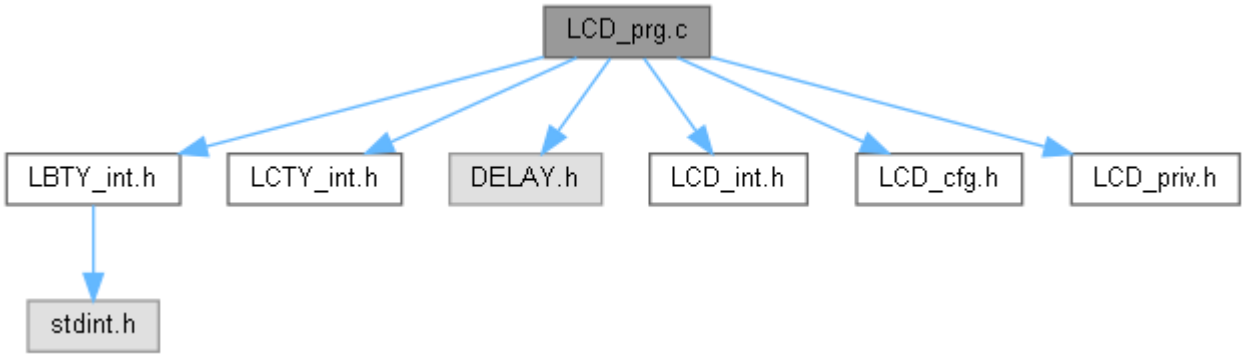
```
149 /* Input        :    s32Num, u8Col, u8Row                                    */
150 /* Return       :    LBTY_tenuErrorStatus                                    */
151 /* ************************************************************************ */
152 extern LBTY_tenuErrorStatus LCD_u8SetNum_Rise(s32 s32Num, u8 u8Col, u8 u8Row);
153
154 /* ************************************************************************ */
155 /* Description :    Set Real float Num Rise at X Column and Y Row           */
156 /* Input        :    f32Num, u8Col, u8Row                                    */
157 /* Return       :    LBTY_tenuErrorStatus                                    */
158 /* ************************************************************************ */
159 extern LBTY_tenuErrorStatus LCD_u8SetFloat_Rise(f32 f32Num, u8 u8Col, u8 u8Row);
160
161 /* ************************************************************************ */
162 /* Description :    Jump Cursor to X Column and Y Row                       */
163 /* Input        :    u8Col, u8Row                                            */
164 /* Return       :    LBTY_tenuErrorStatus                                    */
165 /* ************************************************************************ */
166 extern LBTY_tenuErrorStatus LCD_u8GoTo_XY(u8 u8Col, u8 u8Row);
167
168 /* ************************************************************************ */
169 /* Description :    Clear Display (also clear DDRAM content)                */
170 /* Input        :    void                                                    */
171 /* Return       :    LBTY_tenuErrorStatus                                    */
172 /* ************************************************************************ */
173 extern LBTY_tenuErrorStatus LCD_u8ClrDisplay(void);
174
175 /* ************************************************************************ */
176 /* Description :    Back the cursor to the home (0, 0)                      */
177 /* Input        :    void                                                    */
178 /* Return       :    LBTY_tenuErrorStatus                                    */
179 /* ************************************************************************ */
180 extern LBTY_tenuErrorStatus LCD_u8Home(void);
181
182 /* ************************************************************************ */
183 /* Description :    Set Entry Mode                                          */
184 /* Input        :    u8Mode                                                  */
185 /* Return       :    LBTY_tenuErrorStatus                                    */
186 /* ************************************************************************ */
187 extern LBTY_tenuErrorStatus LCD_u8EntryMode(LCD_tenuEntryMode u8Mode);
188
189 /* ************************************************************************ */
190 /* Description :    Set Display and Cursor Mode
191 /* Input        :    u8Mode                                                  */
192 /* Return       :    LBTY_tenuErrorStatus                                    */
193 /* ************************************************************************ */
194 extern LBTY_tenuErrorStatus LCD_u8DisplayMode(LCD_tenuDisplayCursorControl u8Mode);
195
196 /* ************************************************************************ */
197 /* Description :    Set Shifting Mode                                       */
198 /* Input        :    u8Mode                                                  */
199 /* Return       :    LBTY_tenuErrorStatus                                    */
200 /* ************************************************************************ */
201 extern LBTY_tenuErrorStatus LCD_u8ShiftMode(LCD_tenuDisplayCursorShift u8Shift);
202
203 /* ************************************************************************ */
204 /* Description :    Store Special Char                                      */
205 /* Input        :    u8Index                                                 */
206 /* Input/Output:    pu8char                                                 */
207 /* Return       :    LBTY_tenuErrorStatus                                    */
208 /* ************************************************************************ */
209 extern LBTY_tenuErrorStatus LCD_u8StoreSpecialChar(u8* pu8char, u8 u8Index);
210
211 /* ************************************************************************ */
212 /* Description :    Set Special Char with index                             */
213 /* Input        :    u8Index, u8Col, u8Row                                   */
214 /* Return       :    LBTY_tenuErrorStatus                                    */
215 /* ************************************************************************ */
216 extern LBTY_tenuErrorStatus LCD_u8SetSpecialCharIndex(u8 u8Index, u8 u8Col, u8 u8Row);
217
218 /* ************************************************************************ */
219 /* Description :    Set Special Char with Array                             */
220 /* Input        :    u8Col, u8Row                                            */
221 /* Input/Output:    pu8char                                                 */
222 /* Return       :    LBTY_tenuErrorStatus                                    */
223 /* ************************************************************************ */
```

```
224 extern LBTY tenuErrorStatus LCD_u8SetSpecialCharArray(u8* pu8char, u8 u8Col, u8
u8Row);
225
226 #endif /* LCD_INT_H_ */
227 /************************** E N D (LCD_int.h) *****************************/
```

LCD_prg.c File Reference

```
#include "LBTY_int.h"
#include "LCTY_int.h"
#include "DELAY.h"
#include "LCD_int.h"
#include "LCD_cfg.h"
#include "LCD_priv.h"
```

Include dependency graph for LCD_prg.c:



Functions

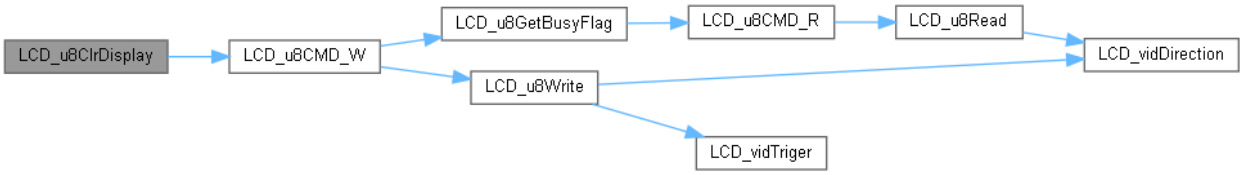
- void LCD_vidInit (void)
- void LCD_vidInitKit (void)
- LBTY_tenuErrorStatus LCD_u8SetPage (const u8 pu8String1[], const u8 pu8String2[])
- LBTY_tenuErrorStatus LCD_u8SetString (const u8 pu8String[], u8 u8Col, u8 u8Row)
- LBTY_tenuErrorStatus LCD_u8SetChar (u8 u8Char, u8 u8Col, u8 u8Row)
- LBTY_tenuErrorStatus LCD_u8GetNum (s32 *ps32Num, u8 u8Col, u8 u8Row)
- LBTY_tenuErrorStatus LCD_u8SetNum (s32 s32Num, u8 u8Col, u8 u8Row)
- LBTY_tenuErrorStatus LCD_u8GetFloat (f32 *pf32Num, u8 u8Col, u8 u8Row)
- LBTY_tenuErrorStatus LCD_u8SetFloat (f32 f32Num, u8 u8Col, u8 u8Row)
- LBTY_tenuErrorStatus LCD_u8SetNum_Rise (s32 s32Num, u8 u8Col, u8 u8Row)
- LBTY_tenuErrorStatus LCD_u8SetFloat_Rise (f32 f32Num, u8 u8Col, u8 u8Row)
- LBTY_tenuErrorStatus LCD_u8GoTo_XY (u8 u8Col, u8 u8Row)
- LBTY_tenuErrorStatus LCD_u8ClrDisplay (void)
- LBTY_tenuErrorStatus LCD_u8Home (void)
- LBTY_tenuErrorStatus LCD_u8EntryMode (LCD_tenuEntryMode u8Mode)
- LBTY_tenuErrorStatus LCD_u8DisplayMode (LCD_tenuDisplayCursorControl u8Mode)
- LBTY_tenuErrorStatus LCD_u8ShiftMode (LCD_tenuDisplayCursorShift u8Shift)
- LBTY_tenuErrorStatus LCD_u8StoreSpecialChar (u8 *pu8char, u8 u8Index)
- LBTY_tenuErrorStatus LCD_u8SetSpecialCharIndex (u8 u8Index, u8 u8Col, u8 u8Row)
- LBTY_tenuErrorStatus LCD_u8SetSpecialCharArray (u8 *pu8char, u8 u8Col, u8 u8Row)

Function Documentation

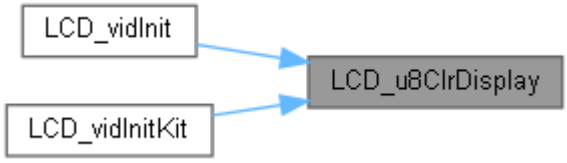
LBTY_tenuErrorStatus LCD_u8ClrDisplay (void)

```
535                                                 {
536     return LCD_u8CMD_W(LCD_CLEAR_DISPLAY);
537 }
```

Here is the call graph for this function:



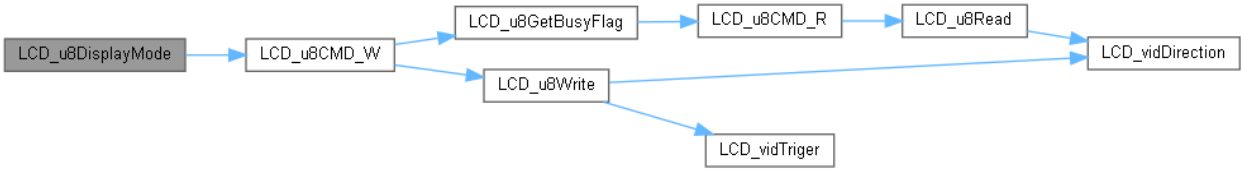
Here is the caller graph for this function:



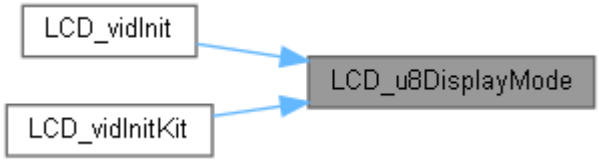
LBTY_tenuErrorStatus LCD_u8DisplayMode (LCD_tenuDisplayCursorControl *u8Mode*)

```
580                                                                    {
581     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
582
583     switch(u8Mode){
584     case LCD_Display_OFF:
585         u8RetErrorState = LCD_u8CMD_W(LCD_DISPLAY_OFF_CURSOR_OFF);
586         break;
587     case LCD_Cursor_OFF:
588         u8RetErrorState = LCD_u8CMD_W(LCD_DISPLAY_ON_CURSOR_OFF);
589         break;
590     case LCD_Cursor_OFF_Blink:
591         u8RetErrorState = LCD_u8CMD_W(LCD_DISPLAY_ON_CURSOR_OFF_BLINK);
592         break;
593     case LCD_Cursor_UnderLine:
594         u8RetErrorState = LCD_u8CMD_W(LCD_DISPLAY_ON_CURSOR_UNDERLINE);
595         break;
596     case LCD_Cursor_UnderLine_Blinking:
597         u8RetErrorState = LCD_u8CMD_W(LCD_DISPLAY_ON_CURSOR_BLINK);
598         break;
599     default:
600         u8RetErrorState = LBTY_NOK;
601     }
602     return u8RetErrorState;
603 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

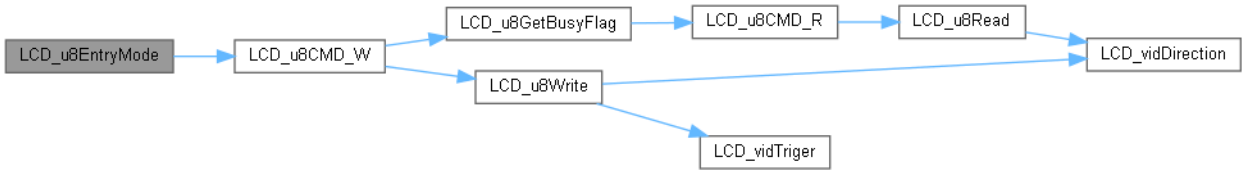


LBTY_tenuErrorStatus LCD_u8EntryMode (LCD_tenuEntryMode *u8Mode*)

```
553                                                                    {
554     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
555
556     switch(u8Mode){
557     case LCD_Entry_Dec:
558         u8RetErrorState = LCD_u8CMD_W(LCD_Entry_DEC);
559         break;
560     case LCD_Entry_Dec_Shift:
561         u8RetErrorState = LCD_u8CMD_W(LCD_Entry_DEC_SHIFT);
562         break;
563     case LCD_Entry_Inc:
564         u8RetErrorState = LCD_u8CMD_W(LCD_Entry_INC);
565         break;
```

```
566     case LCD_Entry_Inc_Shift:
567         u8RetErrorState = LCD_u8CMD_W(LCD_Entry_INC_SHIFT);
568         break;
569     default:
570         u8RetErrorState = LBTY_NOK;
571     }
572     return u8RetErrorState;
573 }
```

Here is the call graph for this function:

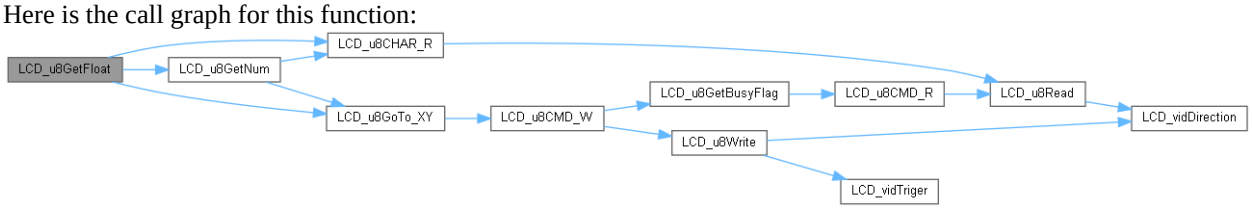


Here is the caller graph for this function:

LBTY_tenuErrorStatus LCD_u8GetFloat (f32 * *pf32Num*, u8 *u8Col*, u8 *u8Row*)

```
277                                                                     {
278     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
279     u8 u8Log = 1u;
280     u32 u32Factor = 1u;
281     u8 u8Char = LBTY_u8ZERO;
282     s32 s32NumL = LBTY_u32ZERO;
283     s32 s32NumR = LBTY_u32ZERO;
284 
285     u8RetErrorState = LCD_u8GetNum(&s32NumL, u8Col, u8Row);
286     if(u8RetErrorState == LBTY_OK){
287 
288         for(u32 i = (u32)((s32NumL >= (s32)LBTY_u32ZERO) ? s32NumL : s32NumL * -1.0)
; i/=10 ; u8Log++);
289         u8Col += u8Log + ((s32NumL >= (s32)LBTY_u32ZERO) ? 1u : 2u);
290 
291         while(!u8RetErrorState){
292             if(LCD_u8GoTo_XY(u8Col++, u8Row)){
293                 u8RetErrorState = LBTY_NULL_POINTER;
294             }else{
295                 u8RetErrorState = LCD_u8CHAR_R(&u8Char);
296                 if(u8Char>='0' && u8Char<='9'){
297                     s32NumR = (s32NumR * 10) + (u8Char - '0');
298                     u32Factor *= 10;
299                     continue;
300                 }else if(u8Char == '.'){
301                 }
302                 break;
303             }
304         }
305     }
306     if(s32NumL >= (s32)LBTY_u32ZERO){
307         *pf32Num = s32NumL + (f32)s32NumR / u32Factor;
308     }else{
309         *pf32Num = s32NumL - (f32)s32NumR / u32Factor;
310     }
311 
312     return u8RetErrorState;
313 }
```

Here is the call graph for this function:

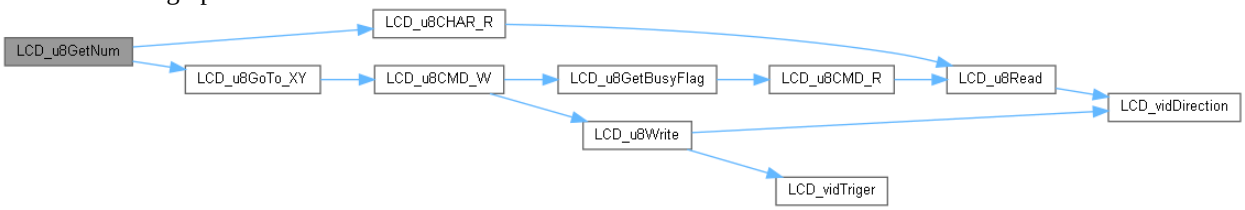


Here is the caller graph for this function:

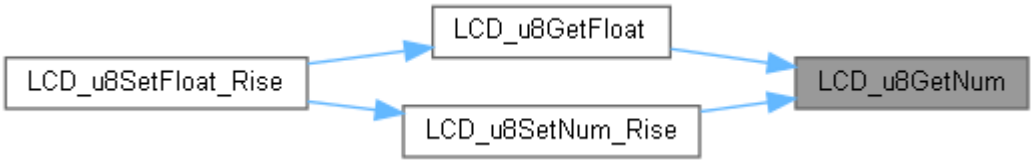
LBTY_tenuErrorStatus LCD_u8GetNum (s32 * *ps32Num*, u8 *u8Col*, u8 *u8Row*)

```
214                                                                    {
215     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
216     u8 u8Char = LBTY_u8ZERO;
217     u8 u8SignChar = LBTY_RESET;
218     u8 u8FirstCharFlag = LBTY_SET;
219     u32 u32NumRead = LBTY_u32ZERO;
220
221     while(!u8RetErrorState){
222         if(LCD_u8GoTo_XY(u8Col++, u8Row)){
223             u8RetErrorState = LBTY_NULL_POINTER;
224         }else{
225             u8RetErrorState = LCD_u8CHAR_R(&u8Char);
226             if(u8FirstCharFlag && (u8Char=='-' || u8Char=='+')){
227                 if(u8Char=='-')     {   u8SignChar = LBTY_SET;  }
228                 else if(u8Char=='+'){   u8SignChar = LBTY_RESET;}
229                 u8FirstCharFlag = LBTY_RESET;
230                 continue;
231             }else if(u8Char>='0' && u8Char<='9'){
232                 u32NumRead = (u32NumRead * 10) + (u8Char - '0');
233                 continue;
234             }else if(u8Char == '.'){
235             }
236             break;
237         }
238     }
239     *ps32Num = (u8SignChar? (s32)u32NumRead * -1 : (s32)u32NumRead);
240     return u8RetErrorState;
241 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

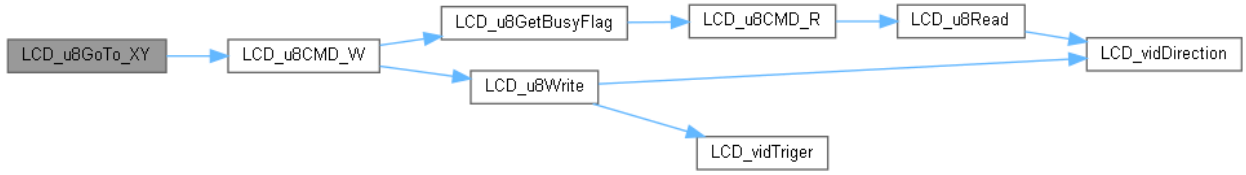


LBTY_tenuErrorStatus LCD_u8GoTo_XY (u8 *u8Col*, u8 *u8Row*)

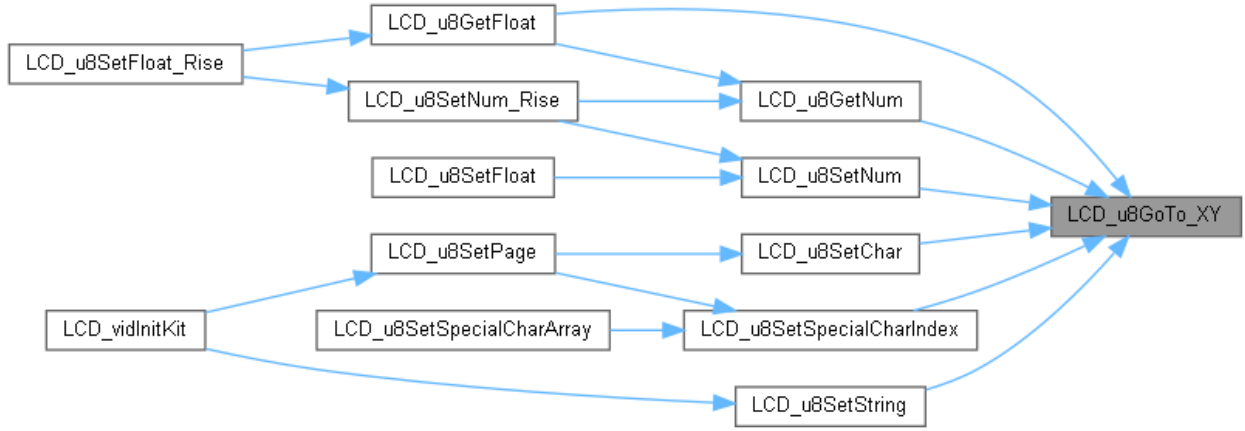
```
500                                                                    {
501     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
502     if((u8Col < LCD_COL_NUM)){
503         switch(u8Row){
504         case LCD_Line_1:    LCD_u8CMD_W(LCD_FIRST_LINE_POSITION_0  + u8Col);
break;
505 #if (LCD_ROW_NUM >= LCD_ROW_NUM_2)
506         case LCD_Line_2:    LCD_u8CMD_W(LCD_SECOND_LINE_POSITION_0 + u8Col);
break;
507 #endif
508 #if (LCD_ROW_NUM >= LCD_ROW_NUM_4)
509 #if (LCD_COL_NUM == LCD_COL_NUM_16)
510         case LCD_Line_3:    LCD_u8CMD_W(LCD_THIRD_LINE_POSITION_0  + u8Col);
break;
511         case LCD_Line_4:    LCD_u8CMD_W(LCD_FOURTH_LINE_POSITION_0 + u8Col);
break;
512 #endif
513 #if (LCD_COL_NUM == LCD_COL_NUM_20)
514         case LCD_Line_3:    LCD_u8CMD_W(LCD_THIRD_LINE_POSITION_0_20  + u8Col);
break;
515         case LCD_Line_4:    LCD_u8CMD_W(LCD_FOURTH_LINE_POSITION_0_20 + u8Col);
break;
516 #endif
517 #if (LCD_COL_NUM == LCD_COL_NUM_32)
518         case LCD_Line_3:    LCD_u8CMD_W(LCD_THIRD_LINE_POSITION_0_32  + u8Col);
break;
```

```
519         case LCD_Line_4:    LCD_u8CMD_W(LCD_FOURTH_LINE_POSITION_0_32 + u8Col);
break;
520 #endif
521 #endif
522         default:            u8RetErrorState = LBTY_NULL_POINTER;;
break;
523         }
524     }else{
525         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
526     }
527     return u8RetErrorState;
528 }
```

Here is the call graph for this function:



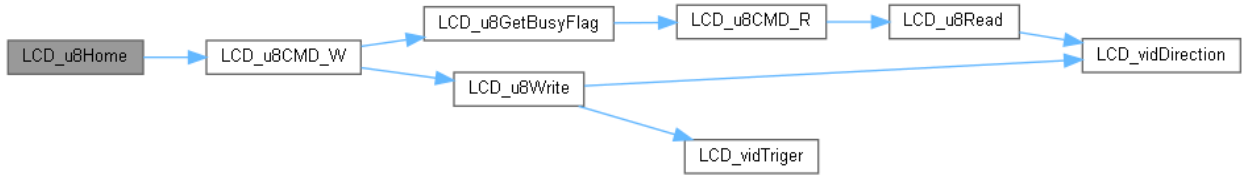
Here is the caller graph for this function:



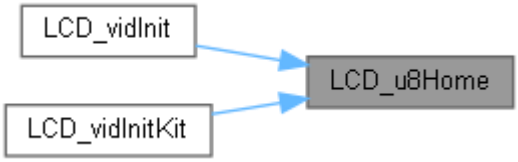
LBTY_tenuErrorStatus LCD_u8Home (void)

```
544                                 {
545     return LCD_u8CMD_W(LCD_CURSOR_HOME);
546 }
```

Here is the call graph for this function:



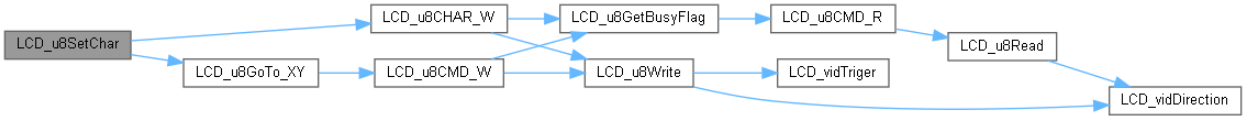
Here is the caller graph for this function:



LBTY_tenuErrorStatus LCD_u8SetChar (u8 *u8Char*, u8 *u8Col*, u8 *u8Row*)

```
198                                                                 {
199     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
200     if(LCD_u8GoTo_XY(u8Col, u8Row)){
201         u8RetErrorState = LBTY_NOK;
202     }else{
203         u8RetErrorState = LCD_u8CHAR_W(u8Char);
204     }
205     return u8RetErrorState;
206 }
```

Here is the call graph for this function:

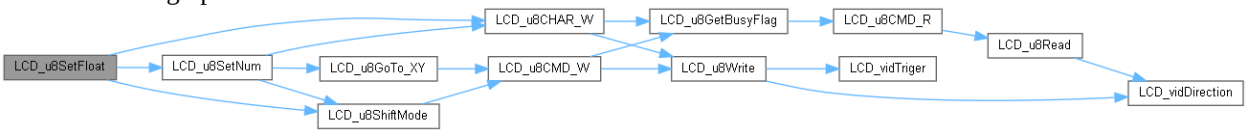


Here is the caller graph for this function:

LBTY_tenuErrorStatus LCD_u8SetFloat (f32 *f32Num*, u8 *u8Col*, u8 *u8Row*)

```
320                                                                 {
321     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
322
323     LCD_u8SetNum((s32)f32Num, u8Col, u8Row);
324     LCD_u8CHAR_W('.');
325
326     if(f32Num < 0.0){
327         f32Num *= -1.0;
328         u8Col++;
329     }
330
331     u32 u32Lcd_R = (u32)((u32)(f32Num * LCD_FLOAT_MUL) % LCD_FLOAT_MUL);
332
333     for(u32 u32Factor = LCD_FLOAT_MUL; u32Factor/=10 ; ){
334         LCD_u8CHAR_W(((u32Lcd_R/u32Factor)%10u) + '0');
335     }
336     LCD_u8CHAR_W(' ');
337     LCD_u8CHAR_W(' ');
338     LCD_u8ShiftMode(LCD_Cursor_Shift_Left);
339     LCD_u8ShiftMode(LCD_Cursor_Shift_Left);
340
341     return u8RetErrorState;
342 }
```

Here is the call graph for this function:



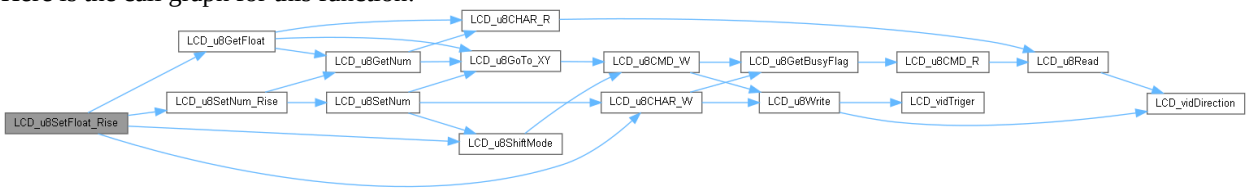
LBTY_tenuErrorStatus LCD_u8SetFloat_Rise (f32 *f32Num*, u8 *u8Col*, u8 *u8Row*)

TODO: Redesign this function with float work

```
420                                                                 {
421     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
422     u32 u32NumNew = LBTY_u32ZERO;
423     u32 u32NumRead = LBTY_u32ZERO;
424
425     f32 f32LcdRead = 0.0f;
426     u8RetErrorState = LCD_u8GetFloat(&f32LcdRead, u8Col, u8Row);
427
428     if(u8RetErrorState == LBTY_OK){
429
432         u8RetErrorState = LCD_u8SetNum_Rise((s32)f32Num, u8Col, u8Row);
433
434         if(u8RetErrorState){
435             LCD_u8CHAR_W('.');
436
437             if(f32Num >= (f32)LBTY_u32ZERO){
438                 u32NumNew = ((u32)( f32Num * LCD_FLOAT_MUL) % (u32)LCD_FLOAT_MUL);
439             }else{
440                 u32NumNew = ((u32)(-f32Num * LCD_FLOAT_MUL) %
(u32)LCD_FLOAT_MUL);
441             }
442             if(f32LcdRead >= (f32)LBTY_u32ZERO){
443                 u32NumRead = ((u32)( f32LcdRead * LCD_FLOAT_MUL) %
(u32)LCD_FLOAT_MUL);
444             }else{
445                 u32NumRead = ((u32)(-f32LcdRead * LCD_FLOAT_MUL) %
(u32)LCD_FLOAT_MUL);
446             }
```

```
447
448             u32 u32Step = LCD_FLOAT_MUL;
449             if(!u8RetErrorState && u32NumRead != u32NumNew){
450                 if(u32NumRead < u32NumNew){
451                     while(u32Step){
452                         if(u32NumNew >= (u32NumRead + u32Step)){
453                             u32NumRead += u32Step;
454                             u8RetErrorState = LBTY_NOK;
455                             break;
456                         }
457                         u32Step /= 10;
458                     }
459                 }else if(u32NumRead > u32NumNew){
460                     while(u32Step){
461                         if(u32NumNew <= (u32NumRead - u32Step)){
462                             while(u32Step){
463                                 if(u32NumRead >= u32Step){
464                                     u32NumRead -= u32Step;
465                                     u8RetErrorState = LBTY_NOK;
466                                     break;
467                                 }
468                                 u32Step /= 10;
469                             }
470                             break;
471                         }
472                         u32Step /= 10;
473                     }
474                 }else{
475                     u8RetErrorState = LBTY_OK;
476                 }
477             }
478             if(u8RetErrorState){
479                 for(u32 u32Factor = LCD_FLOAT_MUL; u32Factor/=10 ; ){
480                     LCD_u8CHAR_W(((u32NumRead/u32Factor)%10u) + '0');
481                 }
482                 LCD_u8CHAR_W(' ');
483                 LCD_u8CHAR_W(' ');
484                 LCD_u8ShiftMode(LCD_Cursor_Shift_Left);
485                 LCD_u8ShiftMode(LCD_Cursor_Shift_Left);
486             }
487         }
488     }else{
489         u8RetErrorState = LBTY_OK;
490     }
491
492     return u8RetErrorState;
493 }
```

Here is the call graph for this function:

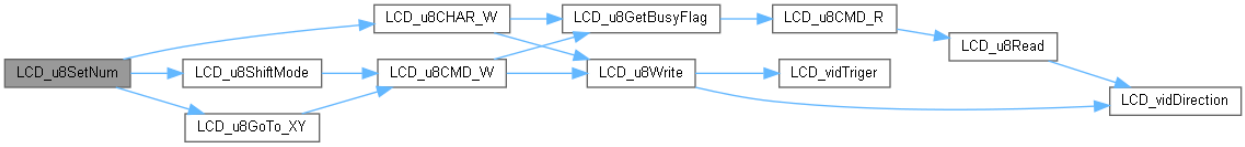


LBTY_tenuErrorStatus LCD_u8SetNum (s32 *s32Num*, u8 *u8Col*, u8 *u8Row*)

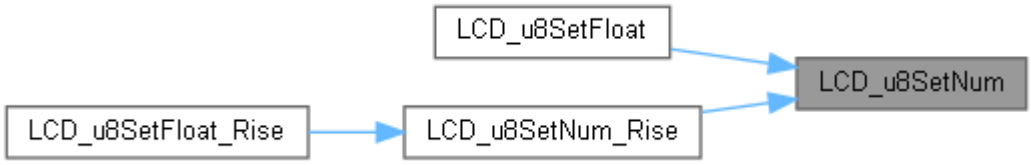
```
248                                                                 {
249     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
250     u8 u8Log = 1u;
251     u32 u32Factor = 1u;
252     u32 u32NumLoc = (u32)((s32Num >= (s32)LBTY_u32ZERO) ? s32Num : s32Num * -1);
253
254     if(LCD_u8GoTo_XY(u8Col, u8Row)){
255         u8RetErrorState = LBTY_NOK;
256     }else{
257         for(u32 i = u32NumLoc ; i/=10 ; u8Log++, u32Factor*=10);
258         if(s32Num<0){
259             u8RetErrorState = LCD_u8CHAR_W('-');
260         }
261         while(u8Log--){
262             u8RetErrorState = LCD_u8CHAR_W((u8)((u32NumLoc/u32Factor)%10u) +
'0');
263             u32Factor/=10;
264         }
```

```
265         LCD_u8CHAR_W(' ');
266         LCD_u8ShiftMode(LCD_Cursor_Shift_Left);
267     }
268     return u8RetErrorState;
269 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

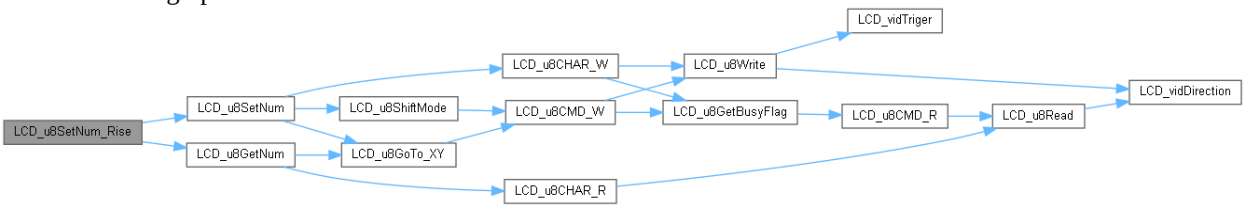


LBTY_tenuErrorStatus LCD_u8SetNum_Rise (s32 *s32Num*, u8 *u8Col*, u8 *u8Row*)

```
349                                                                 {
350     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
351     u8 u8SignChar = LBTY_RESET;
352     u32 u32NumNew = LBTY_u32ZERO;
353     u32 u32NumRead = LBTY_u32ZERO;
354 
355     s32 s32LcdNum = (s32)LBTY_u32ZERO;
356     u8RetErrorState = LCD_u8GetNum(&s32LcdNum, u8Col, u8Row);
357 
358     if(u8RetErrorState == LBTY_OK){
359         if(s32Num >= (s32)LBTY_u32ZERO){
360             u32NumNew = (u32)s32Num;
361             u8SignChar = LBTY_RESET;
362             if((s32LcdNum < (s32)LBTY_u32ZERO)){
363                 u32NumRead = (u32)(s32LcdNum * -1);
364             }else{
365                 u32NumRead = (u32)s32LcdNum;
366             }
367         }else{
368             u32NumNew = (u32)(s32Num * -1);
369             u8SignChar = LBTY_SET;
370             if((s32LcdNum < (s32)LBTY_u32ZERO)){
371                 u32NumRead = (u32)(s32LcdNum * -1);
372             }else{
373                 u32NumRead = (u32)s32LcdNum;
374                 s32LcdNum *= -1;
375             }
376         }
377 
378         u32 u32Step = LCD_FLOAT_MUL;
379         if(u32NumRead != u32NumNew){
380             if(u32NumRead < u32NumNew){
381                 while(u32Step){
382                     if(u32NumNew >= (u32NumRead + u32Step)){
383                         u32NumRead += u32Step;
384                         break;
385                     }
386                     u32Step /= 10;
387                 }
388             }else if(u32NumRead > u32NumNew){
389                 while(u32Step){
390                     if(u32NumNew <= (u32NumRead - u32Step)){
391                         while(u32Step){
392                             if(u32NumRead >= u32Step){
393                                 u32NumRead -= u32Step;
394                                 break;
395                             }
396                             u32Step /= 10;
397                         }
398                         break;
399                     }
400                     u32Step /= 10;
401                 }
```

```
402             }
403             s32LcdNum = u8SignChar ? (s32)u32NumRead * -1 : (s32)u32NumRead;
404             LCD_u8SetNum(s32LcdNum, u8Col, u8Row);
405             u8RetErrorState = LBTY_NOK;
406         }
407
408     }else{
409         u8RetErrorState = LBTY_OK;
410     }
411
412     return u8RetErrorState;
413 }
```

Here is the call graph for this function:



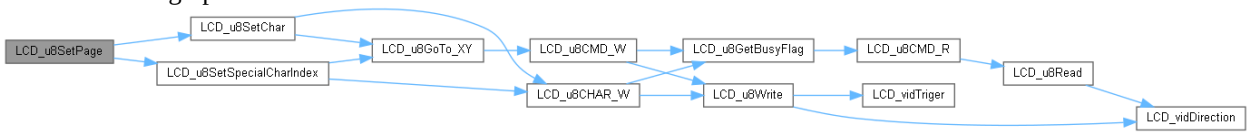
Here is the caller graph for this function:

LBTY_tenuErrorStatus LCD_u8SetPage (const u8 *pu8String1*[], const u8 *pu8String2*[])

```
129
130 {
131     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
132     for(u8 i = 0, j = 0 ; i<LCD_COL_NUM && *pu8String1 && *pu8String2 ; i++){
133         if(*pu8String1 == '@'){
134             if((u8RetErrorState = LCD_u8SetSpecialCharIndex(j++, i, 0))){
135                 break;
136             }
137         }else{
138             if((u8RetErrorState = LCD_u8SetChar(*pu8String1, i, 0))){
139                 break;
140             }
141         }
142         if(*pu8String2 == '@'){
143             if((u8RetErrorState = LCD_u8SetSpecialCharIndex(j++, i, 1))){
144                 break;
145             }
146         }else{
147             if((u8RetErrorState = LCD_u8SetChar(*pu8String2, i, 1))){
148                 break;
149             }
150 #if (LCD_ROW_NUM >= LCD_ROW_NUM_4)
151         if(*(pu8String1 + LCD_COL_NUM) == '@'){
152             if((u8RetErrorState = LCD_u8SetSpecialCharIndex(j++, i, 2))){
153                 break;
154             }
155         }else{
156             if((u8RetErrorState = LCD_u8SetChar(*(pu8String1 + LCD_COL_NUM), i,
2))){
157                 break;
158             }
159         }
160         if(*(pu8String2 + LCD_COL_NUM) == '@'){
161             if((u8RetErrorState = LCD_u8SetSpecialCharIndex(j++, i, 3))){
162                 break;
163             }
164         }else{
165             if((u8RetErrorState = LCD_u8SetChar(*(pu8String2 + LCD_COL_NUM), i,
3))){
166                 break;
167             }
168         }
169 #endif
170         vidMyDelay_ms(LCD_DELAY_WAIT);
171         pu8String1++;
```

```
172         pu8String2++;
173     }
174     vidMyDelay_ms(LCD_DELAY_PAGE);
175     return u8RetErrorState;
176 }
```

Here is the call graph for this function:

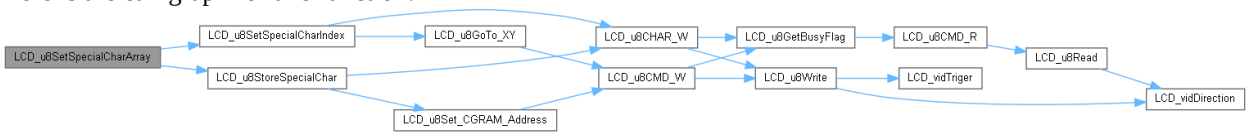


Here is the caller graph for this function:

LBTY_tenuErrorStatus LCD_u8SetSpecialCharArray (u8 * *pu8char*, u8 *u8Col*, u8 *u8Row*)

```
679
{
680     static u8 u8Index = LBTY_u8ZERO;
681     LBTY_tenuErrorStatus u8RetErrorState =
682             LCD_u8StoreSpecialChar(pu8char,u8Index);
683
684     if(u8RetErrorState ==  LBTY_OK){
685         u8RetErrorState = LCD_u8SetSpecialCharIndex(u8Index++, u8Col, u8Row);
686         u8Index = u8Index % LCD_CGRAM_SECTIONS_NUM;
687     }
688     return u8RetErrorState;
689 }
```

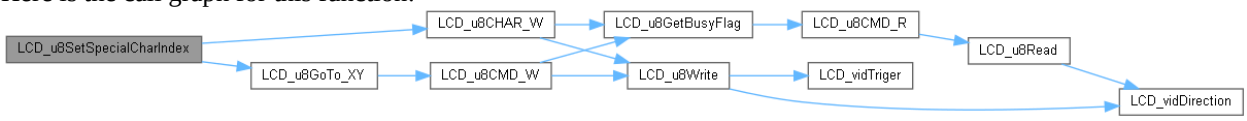
Here is the call graph for this function:



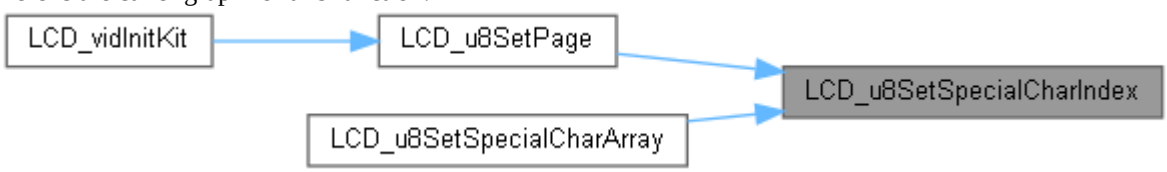
LBTY_tenuErrorStatus LCD_u8SetSpecialCharIndex (u8 *u8Index*, u8 *u8Col*, u8 *u8Row*)

```
658
{
659     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
660
661     if(LCD_u8GoTo_XY(u8Col, u8Row)){
662         u8RetErrorState = LBTY_NOK;
663     }else{
664         if(u8Index < LCD_CGRAM_SECTIONS_NUM){
665             u8RetErrorState = LCD_u8CHAR_W(u8Index);
666         }else{
667             u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
668         }
669     }
670
671     return u8RetErrorState;
672 }
```

Here is the call graph for this function:



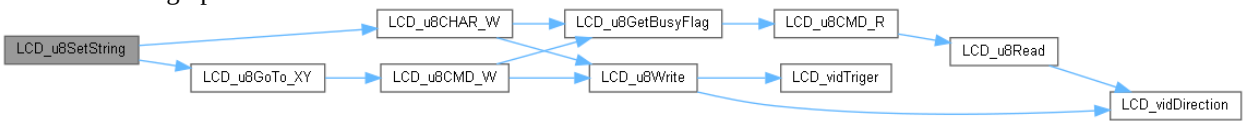
Here is the caller graph for this function:



LBTY_tenuErrorStatus LCD_u8SetString (const u8 *pu8String*[], u8 *u8Col*, u8 *u8Row*)

```
184
{
185     LBTY_tenuErrorStatus u8RetErrorState = LCD_u8GoTo_XY(u8Col, u8Row);
186     while(*pu8String){
187         if(u8RetErrorState)         break;
188         u8RetErrorState = LCD_u8CHAR_W(*(pu8String++));
189     }
190     return u8RetErrorState;
191 }
```

Here is the call graph for this function:

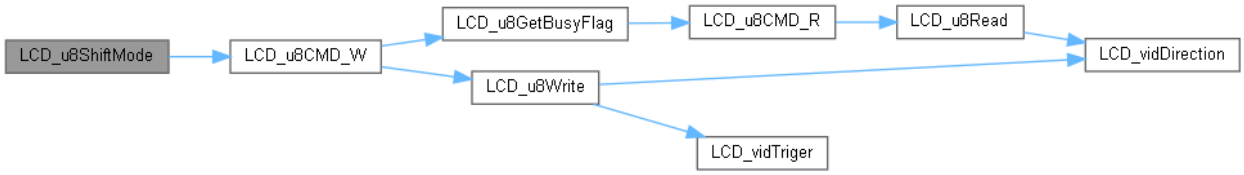


Here is the caller graph for this function:

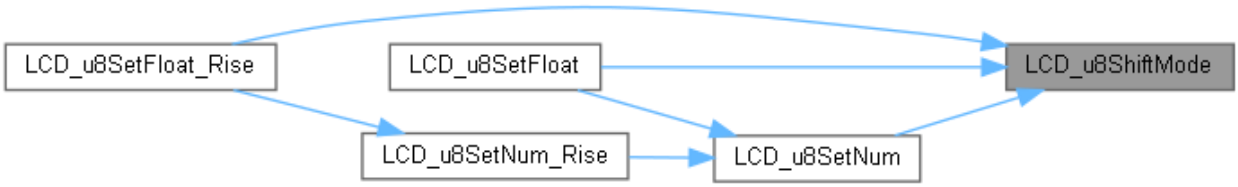
LBTY_tenuErrorStatus LCD_u8ShiftMode (LCD_tenuDisplayCursorShift *u8Shift*)

```
610                                                                     {
611     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
612
613     switch(u8Shift){
614     case LCD_Cursor_Shift_Left:
615         u8RetErrorState = LCD_u8CMD_W(LCD_CURSOR_SHIFT_LEFT);
616         break;
617     case LCD_Cursor_Shift_Right:
618         u8RetErrorState = LCD_u8CMD_W(LCD_CURSOR_SHIFT_RIGHT);
619         break;
620     case LCD_Display_Shift_Left:
621         u8RetErrorState = LCD_u8CMD_W(LCD_DISPLAY_SHIFT_LEFT);
622         break;
623     case LCD_Display_Shift_Right:
624         u8RetErrorState = LCD_u8CMD_W(LCD_DISPLAY_SHIFT_RIGHT);
625         break;
626     default:
627         u8RetErrorState = LBTY_NOK;
628     }
629     return u8RetErrorState;
630 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

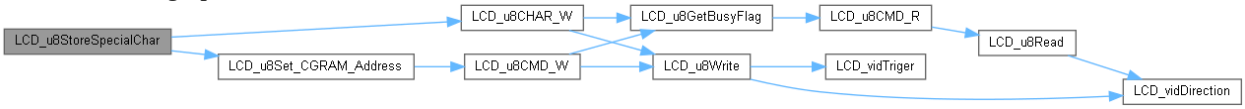


LBTY_tenuErrorStatus LCD_u8StoreSpecialChar (u8 * *pu8char*, u8 *u8Index*)

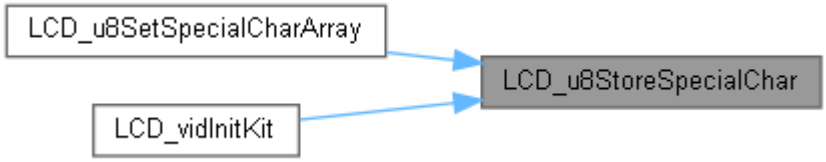
```
638                                                                     {
639     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
640
641     if(u8Index < LCD_CGRAM_SECTIONS_NUM){
642         u8RetErrorState = LCD_u8Set_CGRAM_Address(u8Index *
LCD_CGRAM_LOCATIONS_NUM);
643         for(u8 i = 0 ; i<LCD_CGRAM_LOCATIONS_NUM  && !u8RetErrorState ; i++){
644             u8RetErrorState = LCD_u8CHAR_W(pu8char[i]);
645         }
```

```
646     }else{
647         u8RetErrorState = LBTY_INDEX_OUT_OF_RANGE;
648     }
649 
650     return u8RetErrorState;
651 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

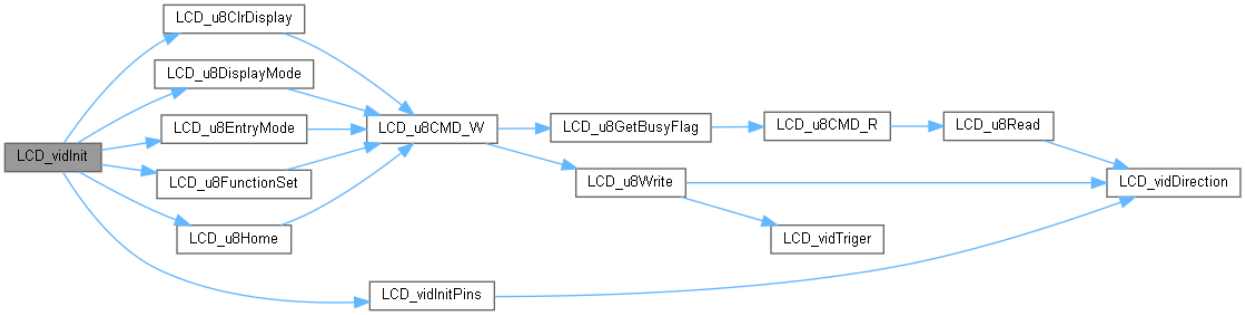


void LCD_vidInit (void)

TODO: sprintf function

```
52                          {
53     LCD_vidInitPins();
54     LCD_u8Home();
55     LCD_u8FunctionSet();
56     LCD_u8ClrDisplay();
57     LCD_u8EntryMode(LCD_Entry_Inc);
58     LCD_u8DisplayMode(LCD_Cursor_OFF);
59 }
```

Here is the call graph for this function:

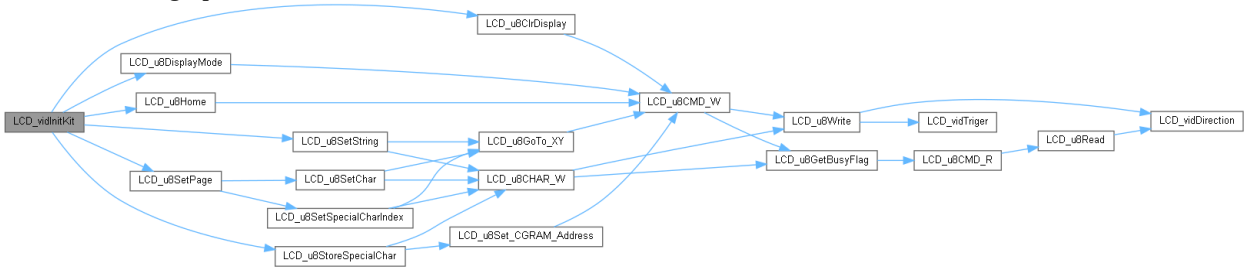


void LCD_vidInitKit (void)

```
61                              {
62 
63     LCD_u8Home();
64     LCD_u8ClrDisplay();
65     LCD_u8DisplayMode(LCD_Cursor_OFF);
66 
67 #if defined(AMIT_KIT)
68 
69     LCD_u8SetString((u8*)"Welcome To", 3, 0);
70     LCD_u8SetString((u8*)"AMIT"      , 6, 1);
71     vidMyDelay_ms(LCD_DELAY_PAGE);
72     LCD_u8ClrDisplay();
73     LCD_u8SetString((u8*)"Choose a"  , 4, 0);
74     LCD_u8SetString((u8*)"Peripheral", 3, 1);
75     vidMyDelay_ms(LCD_DELAY_PAGE);
76     LCD_u8ClrDisplay();
77 
78 #elif defined(ETA32_KIT)
79 
80     for(u8 i = 0 ; i<3u ; i++){
81         LCD_u8StoreSpecialChar((u8*)ETA32[i], i);
82     }
83 
84     LCD_u8SetPage((u8*)"====================Educational Products"
85             , (u8*)"FARES PCB Co. for   ====================");
86 
```

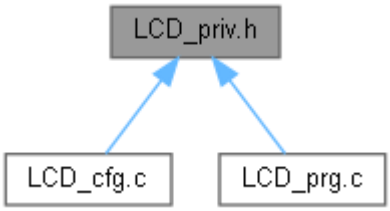
```
87      LCD_u8SetPage((u8*)"====================       @32 Kit      "
88              , (u8*)"      @@            ====================");
89
90      LCD_u8SetPage((u8*)"====================   Eta32 Test Code  "
91              , (u8*)"    ATMEL AVR Kit   ====================");
92
93  #elif defined(ETA32_MINI_KIT)
94
95      for(u8 i = 0 ; i<3u ; i++){
96          LCD_u8StoreSpecialChar((u8*)ETA32[i], i);
97      }
98
99      LCD_u8SetPage((u8*)" FARES PCB Co.  "
100             , (u8*)"================");
101
102     LCD_u8SetPage((u8*)"    @@          "
103             , (u8*)"     @32 mini    ");
104
105     LCD_u8SetPage((u8*)" ATMEL AVR Kit  "
106             , (u8*)" Eta32 mini Kit ");
107
108     LCD_u8SetString((u8*)"Eta32mini Kit",    1, 0);
109     LCD_u8SetString((u8*)"SW -- is pressed", 0, 1);
110     vidMyDelay_ms(LCD_DELAY_PAGE);
111
112 #else
113
114     LCD_u8SetString((u8*)"=====Welcome====", 0, 0);
115     LCD_u8SetString((u8*)"FARES PCB Co.",    1, 1);
116     vidMyDelay_ms(LCD_DELAY_PAGE);
117
118     LCD_u8ClrDisplay();
119
120 #endif
121
122 }
```

Here is the call graph for this function:



LCD_priv.h File Reference

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



Macros

- #define LCD_CLEAR_DISPLAY 0x01u
- #define LCD_CURSOR_HOME 0x02u
- #define LCD_Entry_DEC 0x04u
- #define LCD_Entry_DEC_SHIFT 0x05u
- #define LCD_Entry_INC 0x06u
- #define LCD_Entry_INC_SHIFT 0x07u
- #define LCD_DISPLAY_OFF_CURSOR_OFF 0x08u
- #define LCD_DISPLAY_ON_CURSOR_OFF 0x0Cu
- #define LCD_DISPLAY_ON_CURSOR_OFF_BLINK 0x0Du
- #define LCD_DISPLAY_ON_CURSOR_UNDERLINE 0x0Eu
- #define LCD_DISPLAY_ON_CURSOR_BLINK 0x0Fu
- #define LCD_CURSOR_SHIFT_LEFT 0x10u
- #define LCD_CURSOR_SHIFT_RIGHT 0x14u
- #define LCD_DISPLAY_SHIFT_LEFT 0x18u
- #define LCD_DISPLAY_SHIFT_RIGHT 0x1Cu
- #define LCD_CONFIG_1LINE_4BIT_5ROW 0x20u
- #define LCD_CONFIG_1LINE_4BIT_10ROW 0x24u
- #define LCD_CONFIG_2LINE_4BIT_5ROW 0x28u
- #define LCD_CONFIG_2LINE_4BIT_10ROW 0x2Cu
- #define LCD_CONFIG_1LINE_8BIT_5ROW 0x30u
- #define LCD_CONFIG_1LINE_8BIT_10ROW 0x34u
- #define LCD_CONFIG_2LINE_8BIT_5ROW 0x38u
- #define LCD_FIRST_LINE_POSITION_0 0x80u
- #define LCD_SECOND_LINE_POSITION_0 0xC0u
- #define LCD_THIRD_LINE_POSITION_0 0x90u
- #define LCD_THIRD_LINE_POSITION_0_20 0x94u
- #define LCD_THIRD_LINE_POSITION_0_32 0xA0u
- #define LCD_FOURTH_LINE_POSITION_0 0xD0u
- #define LCD_FOURTH_LINE_POSITION_0_20 0xD4u
- #define LCD_FOURTH_LINE_POSITION_0_32 0xE0u
- #define LCD_RS_CMD 0u
- #define LCD_RS_DATA 1u
- #define LCD_RW_WRITE 0u
- #define LCD_RW_READ 1u
- #define BUSY_FLAG_BIT 7u
- #define LCD_WRITE_INSTRUCTION_CMD 0u
- #define LCD_READ_INSTRUCTION_CMD 1u
- #define LCD_WRITE_DATA_CMD 2u
- #define LCD_READ_DATA_CMD 3u
- #define LCD_SEND_CGRAM_ADDRESS 0x40u
- #define LCD_SEND_DDRAM_ADDRESS 0x80u
- #define LCD_CGRAM_ADDRESS_MASK 0x3Fu
- #define LCD_DDRAM_ADDRESS_MASK 0x7Fu

- #define LCD_CGRAM_SECTIONS_NUM 8u
- #define LCD_CGRAM_LOCATIONS_NUM 8u

Functions

- LBTY_tenuErrorStatus LCD_u8FunctionSet (void)
- void LCD_vidInitPins (void)
- void LCD_vidDirection (u8 u8PinDir)
- void LCD_vidTriger (void)
- LBTY_tenuErrorStatus LCD_u8Write (u8 u8Byte)
- LBTY_tenuErrorStatus LCD_u8Read (u8 *pu8Byte)
- LBTY_tenuErrorStatus LCD_u8CMD_W (u8 u8CMD)
- LBTY_tenuErrorStatus LCD_u8CMD_R (u8 *pu8CMD)
- LBTY_tenuErrorStatus LCD_u8CHAR_W (u8 u8Char)
- LBTY_tenuErrorStatus LCD_u8CHAR_R (u8 *pu8Char)
- LBTY_tenuErrorStatus LCD_u8Set_CGRAM_Address (u8 u8Address)
- LBTY_tenuErrorStatus LCD_u8Set_DDRAM_Address (u8 u8Address)
- LBTY_tenuErrorStatus LCD_u8Get_DDRAM_Address (u8 *pu8Address)
- u8 LCD_u8GetBusyFlag (void)

Variables

- const u8 ETA32 [][LCD_CGRAM_LOCATIONS_NUM]

Macro Definition Documentation

#define BUSY_FLAG_BIT 7u

#define LCD_CGRAM_ADDRESS_MASK 0x3Fu

...

#define LCD_CGRAM_LOCATIONS_NUM 8u

#define LCD_CGRAM_SECTIONS_NUM 8u

#define LCD_CLEAR_DISPLAY 0x01u

Clear and Return Home

#define LCD_CONFIG_1LINE_4BIT_10ROW 0x24u

#define LCD_CONFIG_1LINE_4BIT_5ROW 0x20u

... Function Set DB7-DB4 --> 4-Bits

#define LCD_CONFIG_1LINE_8BIT_10ROW 0x34u

#define LCD_CONFIG_1LINE_8BIT_5ROW 0x30u

... DB7-DB0 --> 8-Bits

#define LCD_CONFIG_2LINE_4BIT_10ROW 0x2Cu

#define LCD_CONFIG_2LINE_4BIT_5ROW 0x28u

#define LCD_CONFIG_2LINE_8BIT_5ROW 0x38u

#define LCD_CURSOR_HOME 0x02u

#define LCD_CURSOR_SHIFT_LEFT 0x10u

Display & Cursor Shifting

#define LCD_CURSOR_SHIFT_RIGHT 0x14u

...

#define LCD_DDRAM_ADDRESS_MASK 0x7Fu

#define LCD_DISPLAY_OFF_CURSOR_OFF 0x08u

Display & Cursor Control

#define LCD_DISPLAY_ON_CURSOR_BLINK 0x0Fu

#define LCD_DISPLAY_ON_CURSOR_OFF 0x0Cu

...

#define LCD_DISPLAY_ON_CURSOR_OFF_BLINK 0x0Du

#define LCD_DISPLAY_ON_CURSOR_UNDERLINE 0x0Eu

#define LCD_DISPLAY_SHIFT_LEFT 0x18u

...

#define LCD_DISPLAY_SHIFT_RIGHT 0x1Cu

...

#define LCD_Entry_DEC 0x04u

Entry Mode Set

#define LCD_Entry_DEC_SHIFT 0x05u

#define LCD_Entry_INC 0x06u

#define LCD_Entry_INC_SHIFT 0x07u

#define LCD_FIRST_LINE_POSITION_0 0x80u

...

#define LCD_FOURTH_LINE_POSITION_0 0xD0u

...

#define LCD_FOURTH_LINE_POSITION_0_20 0xD4u

#define LCD_FOURTH_LINE_POSITION_0_32 0xE0u

#define LCD_READ_DATA_CMD 3u

#define LCD_READ_INSTRUCTION_CMD 1u

#define LCD_RS_CMD 0u

#define LCD_RS_DATA 1u

#define LCD_RW_READ 1u

#define LCD_RW_WRITE 0u

#define LCD_SECOND_LINE_POSITION_0 0xC0u

...

#define LCD_SEND_CGRAM_ADDRESS 0x40u

#define LCD_SEND_DDRAM_ADDRESS 0x80u

...

#define LCD_THIRD_LINE_POSITION_0 0x90u

...

#define LCD_THIRD_LINE_POSITION_0_20 0x94u

#define LCD_THIRD_LINE_POSITION_0_32 0xA0u

#define LCD_WRITE_DATA_CMD 2u

#define LCD_WRITE_INSTRUCTION_CMD 0u

1-RS - 0-RW bits

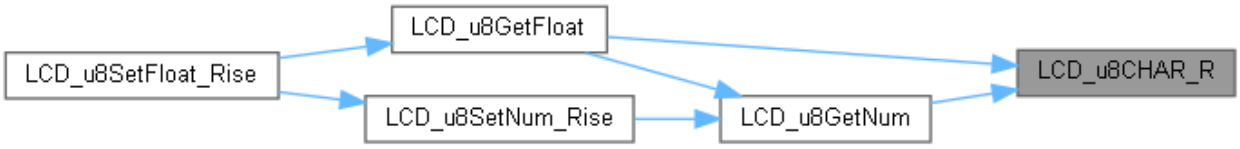
Function Documentation

LBTY_tenuErrorStatus LCD_u8CHAR_R (u8 * *pu8Char*)

```
249                                                 {
250     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
251
252     u8RetErrorState = GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_RS, LCD_RS_DATA);
253     u8RetErrorState = LCD_u8Read(pu8Char);
254
255     return u8RetErrorState;
256 }
```

Here is the call graph for this function:

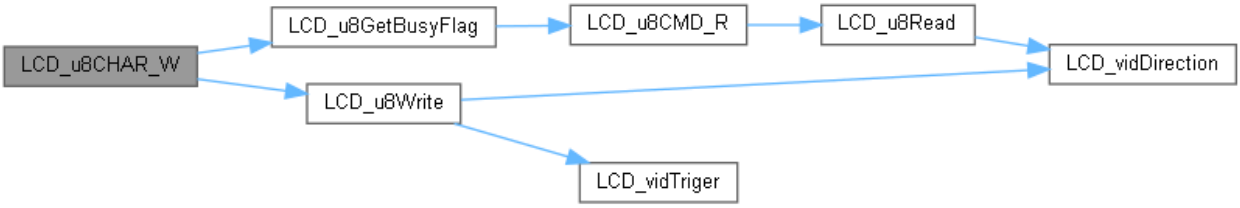
Here is the caller graph for this function:



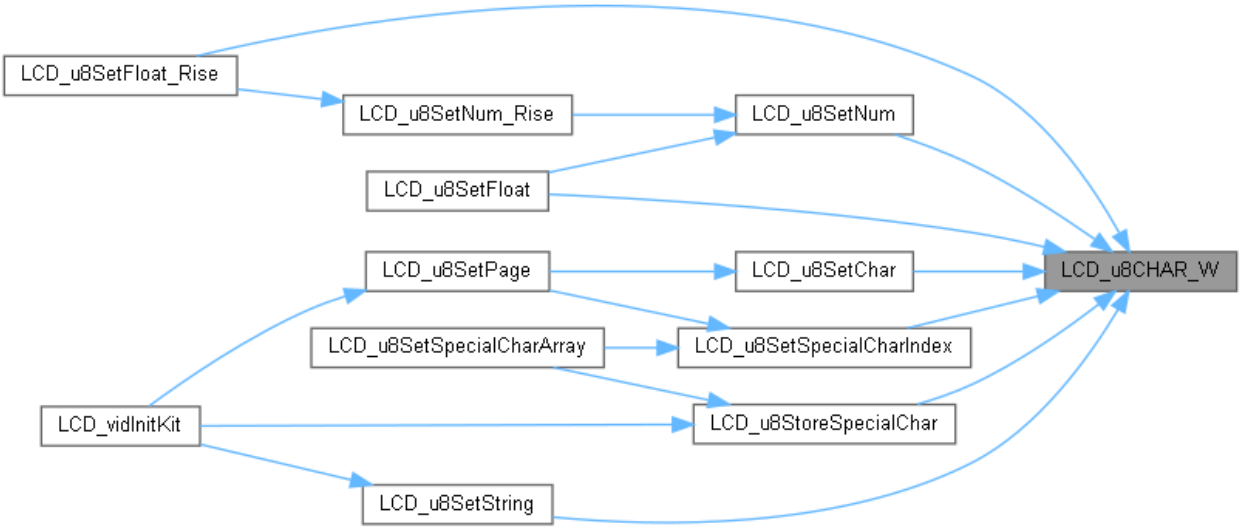
LBTY_tenuErrorStatus LCD_u8CHAR_W (u8 *u8Char*)

```
237                                                 {
238     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
239
240 #ifdef LCD_RW
241     while(LCD_u8GetBusyFlag())      vidMyDelay_ms(LCD_DELAY_CMD);
242 #endif
243     u8RetErrorState = GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_RS, LCD_RS_DATA);
244     u8RetErrorState = LCD_u8Write(u8Char);
245
246     return u8RetErrorState;
247 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

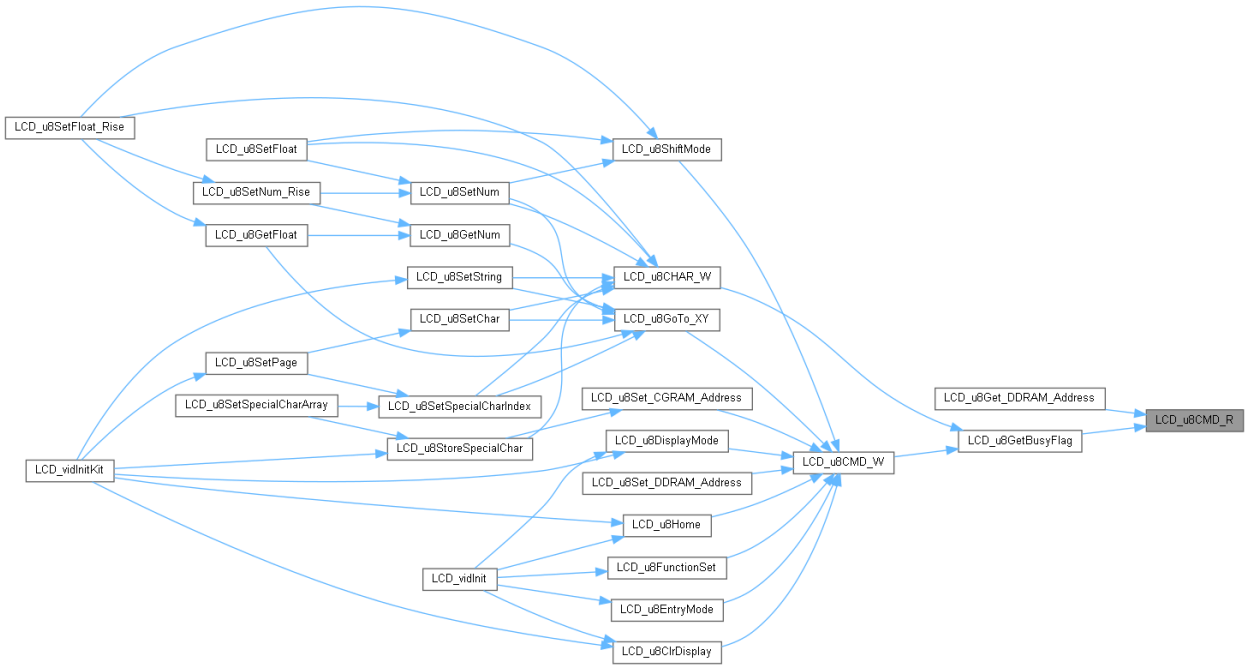


LBTY_tenuErrorStatus LCD_u8CMD_R (u8 * *pu8CMD*)

```
228                                                 {
229     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
230
231     u8RetErrorState = GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_RS, LCD_RS_CMD);
232     u8RetErrorState = LCD_u8Read(pu8CMD);
233
234     return u8RetErrorState;
235 }
```

Here is the call graph for this function:

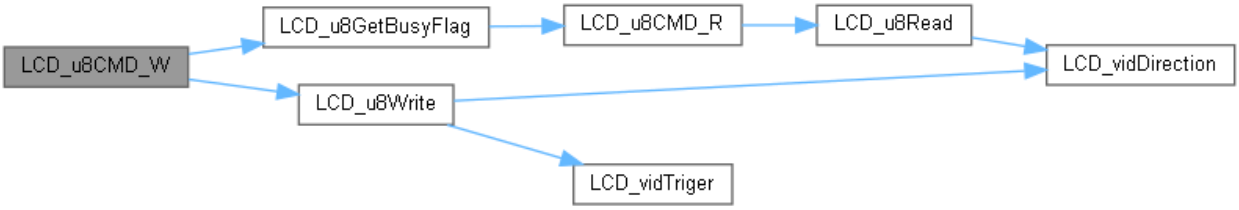
Here is the caller graph for this function:



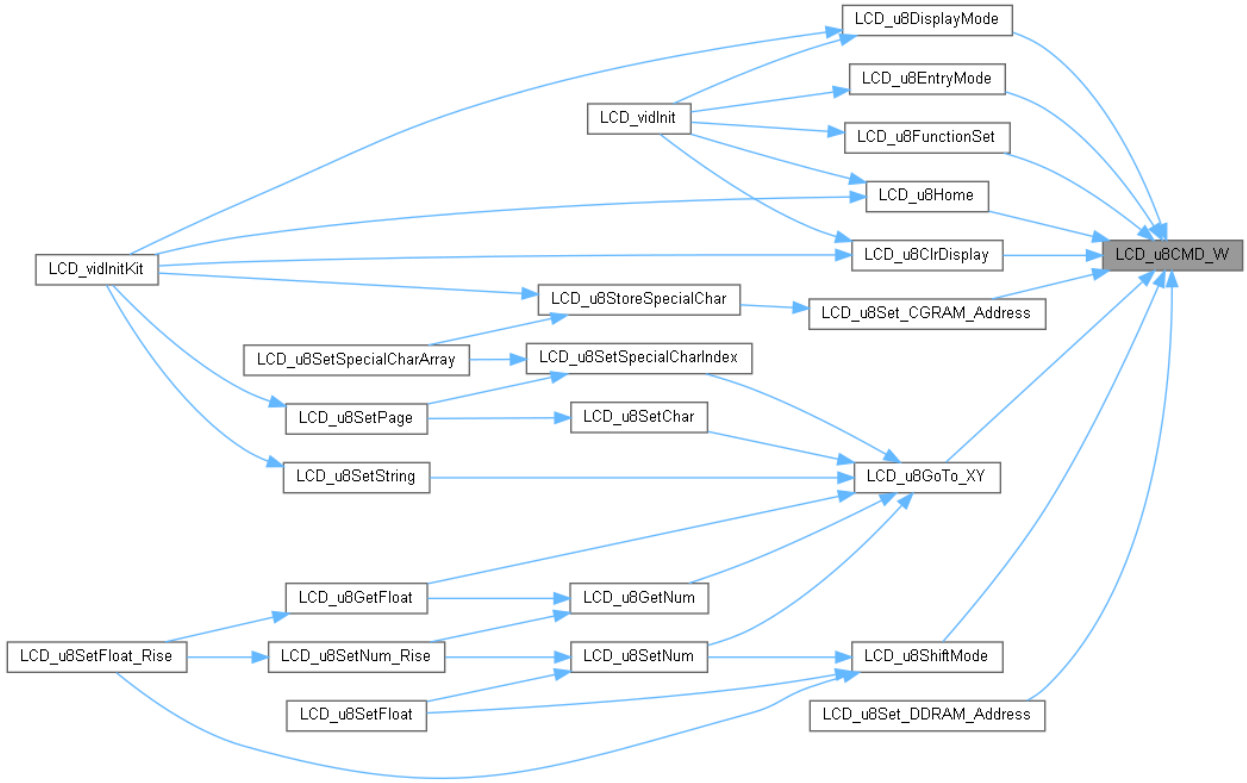
LBTY_tenuErrorStatus LCD_u8CMD_W (u8 *u8CMD*)

```
217                                                 {
218     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
219 #ifdef LCD_RW
220     while(LCD_u8GetBusyFlag())      vidMyDelay_ms(LCD_DELAY_CMD);
221 #endif
222     u8RetErrorState = GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_RS, LCD_RS_CMD);
223     u8RetErrorState = LCD_u8Write(u8CMD);
224 
225     return u8RetErrorState;
226 }
```

Here is the call graph for this function:



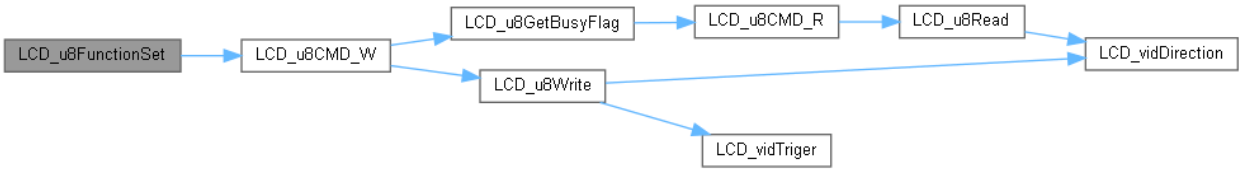
Here is the caller graph for this function:



LBTY_tenuErrorStatus LCD_u8FunctionSet (void)

```
52                                              {
53     LBTY_tenuErrorStatus u8RetErrorState =  LCD_u8CMD_W(LCD_CURSOR_HOME);
54
55     if(u8RetErrorState == LBTY_OK){
56 #ifdef LCD_10DOT
57 #if (LCD_ROW_NUM > LCD_ROW_NUM_1)
58 #if (LCD_FUNCTION_SET == LCD_FUNCTION_SET_8Bits)
59         u8RetErrorState = LCD_u8CMD_W(LCD_CONFIG_2LINE_8BIT_10ROW);
60 #elif  (LCD_FUNCTION_SET == LCD_FUNCTION_SET_4Bits)
61         u8RetErrorState = LCD_u8CMD_W(LCD_CONFIG_2LINE_4BIT_10ROW);
62 #endif
63 #else
64 #if (LCD_FUNCTION_SET == LCD_FUNCTION_SET_8Bits)
65         u8RetErrorState = LCD_u8CMD_W(LCD_CONFIG_1LINE_8BIT_10ROW);
66 #elif  (LCD_FUNCTION_SET == LCD_FUNCTION_SET_4Bits)
67         u8RetErrorState = LCD_u8CMD_W(LCD_CONFIG_1LINE_4BIT_10ROW);
68 #endif
69 #endif
70
71 #else
72
73 #if (LCD_ROW_NUM > LCD_ROW_NUM_1)
74 #if (LCD_FUNCTION_SET == LCD_FUNCTION_SET_8Bits)
75         u8RetErrorState = LCD_u8CMD_W(LCD_CONFIG_2LINE_8BIT_5ROW);
76 #elif  (LCD_FUNCTION_SET == LCD_FUNCTION_SET_4Bits)
77         u8RetErrorState = LCD_u8CMD_W(LCD_CONFIG_2LINE_4BIT_5ROW);
78 #endif
79 #else
80 #if (LCD_FUNCTION_SET == LCD_FUNCTION_SET_8Bits)
81         u8RetErrorState = LCD_u8CMD_W(LCD_CONFIG_1LINE_8BIT_5ROW);
82 #elif  (LCD_FUNCTION_SET == LCD_FUNCTION_SET_4Bits)
83         u8RetErrorState = LCD_u8CMD_W(LCD_CONFIG_1LINE_4BIT_5ROW);
84 #endif
85 #endif
86
87 #endif
88     }
89     return u8RetErrorState;
90 }
```

Here is the call graph for this function:

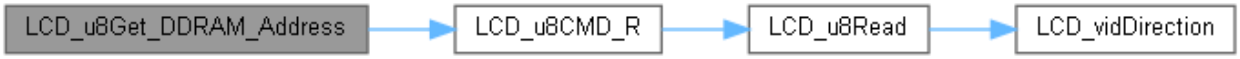


Here is the caller graph for this function:

LBTY_tenuErrorStatus LCD_u8Get_DDRAM_Address (u8 * *pu8Address*)

```
266                                                     {
267     LBTY_tenuErrorStatus u8RetErrorState = LCD_u8CMD_R(pu8Address);
268     *pu8Address = GET_MASK(*pu8Address, LCD_DDRAM_ADDRESS_MASK);
269     return u8RetErrorState;
270 }
```

Here is the call graph for this function:

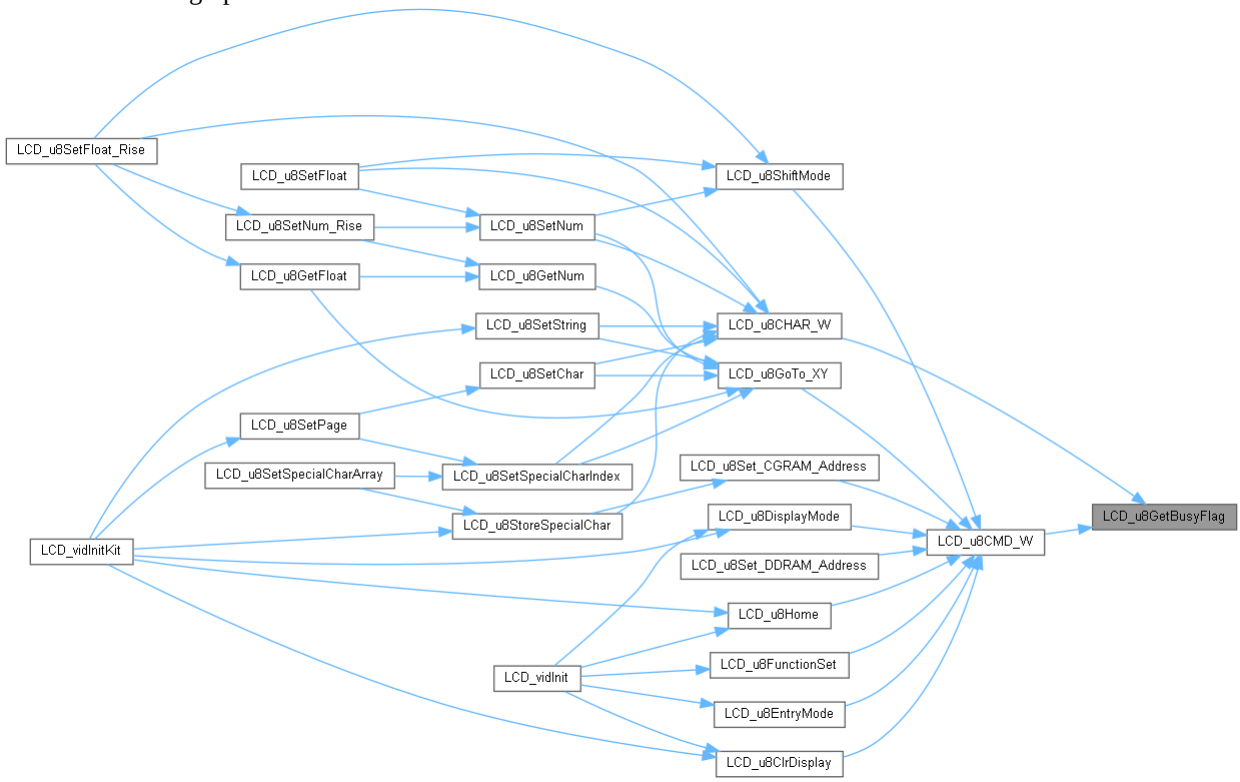


u8 LCD_u8GetBusyFlag (void)

```
272                             {
273     u8 u8RetValue = LBTY_RESET;
274     LCD_u8CMD_R(&u8RetValue);
275     return GET_BIT(u8RetValue, BUSY_FLAG_BIT);
276 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



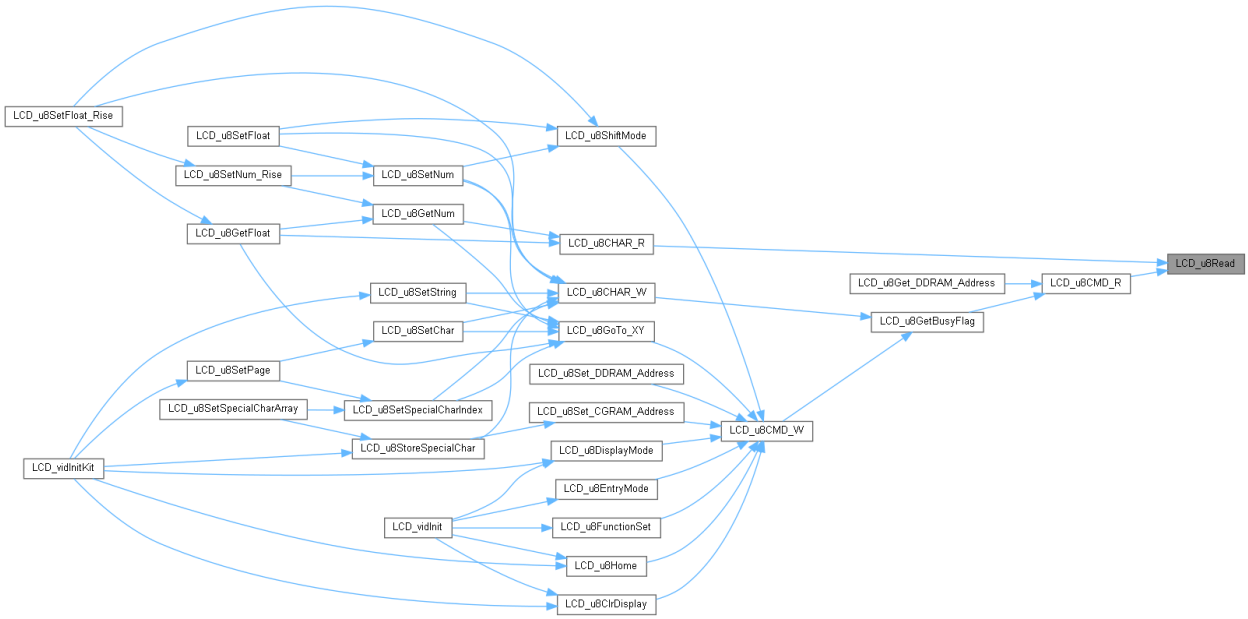
LBTY_tenuErrorStatus LCD_u8Read (u8 * *pu8Byte*)

```
159                                             {
160     LBTY_tenuErrorStatus u8RetErrorState = LBTY_NOK;
161
162 #ifdef LCD_RW
163     LBTY_tuniPort8* pu8LcdByte = (LBTY_tuniPort8*)pu8Byte;
```

```
164     u8 u8ReadValue;
165
166     u8RetErrorState = GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_RW, LCD_RW_READ);
167
168     LCD_vidDirection(PIN_INPUT);
169
170 #if (LCD_FUNCTION_SET == LCD_FUNCTION_SET_4Bits)
171     GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_EN, PIN_High);
172     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D4, &u8ReadValue);
173     pu8LcdByte->sBits.m_u8b4 = u8ReadValue;
174     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D5, &u8ReadValue);
175     pu8LcdByte->sBits.m_u8b5 = u8ReadValue;
176     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D6, &u8ReadValue);
177     pu8LcdByte->sBits.m_u8b6 = u8ReadValue;
178     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D7, &u8ReadValue);
179     pu8LcdByte->sBits.m_u8b7 = u8ReadValue;
180     GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_EN, PIN_Low);
181     vidMyDelay_ms(LCD_DELAY_CMD);
182     GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_EN, PIN_High);
183     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D4, &u8ReadValue);
184     pu8LcdByte->sBits.m_u8b0 = u8ReadValue;
185     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D5, &u8ReadValue);
186     pu8LcdByte->sBits.m_u8b1 = u8ReadValue;
187     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D6, &u8ReadValue);
188     pu8LcdByte->sBits.m_u8b2 = u8ReadValue;
189     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D7, &u8ReadValue);
190     pu8LcdByte->sBits.m_u8b3 = u8ReadValue;
191     GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_EN, PIN_Low);
192 #elif (LCD_FUNCTION_SET == LCD_FUNCTION_SET_8Bits)
193     GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_EN, PIN_High);
194     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D0, &u8ReadValue);
195     pu8LcdByte->sBits.m_u8b0 = u8ReadValue;
196     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D1, &u8ReadValue);
197     pu8LcdByte->sBits.m_u8b1 = u8ReadValue;
198     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D2, &u8ReadValue);
199     pu8LcdByte->sBits.m_u8b2 = u8ReadValue;
200     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D3, &u8ReadValue);
201     pu8LcdByte->sBits.m_u8b3 = u8ReadValue;
202     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D4, &u8ReadValue);
203     pu8LcdByte->sBits.m_u8b4 = u8ReadValue;
204     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D5, &u8ReadValue);
205     pu8LcdByte->sBits.m_u8b5 = u8ReadValue;
206     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D6, &u8ReadValue);
207     pu8LcdByte->sBits.m_u8b6 = u8ReadValue;
208     GPIO_u8GetPinValue(LCD_DATA_PORT, LCD_D7, &u8ReadValue);
209     pu8LcdByte->sBits.m_u8b7 = u8ReadValue;
210     GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_EN, PIN_Low);
211 #endif
212
213 #endif
214     return u8RetErrorState;
215 }
```

Here is the call graph for this function:

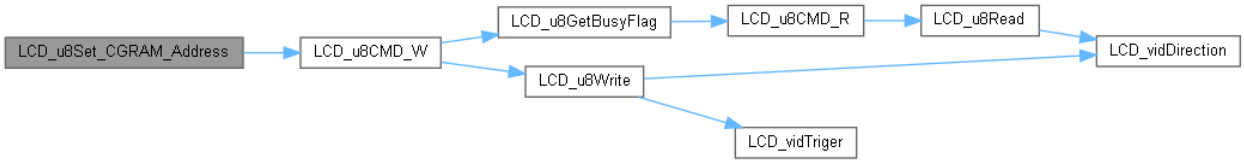
Here is the caller graph for this function:



LBTY_tenuErrorStatus LCD_u8Set_CGRAM_Address (u8 *u8Address*)

```
258                                                     {
259     return LCD_u8CMD_W(LCD_SEND_CGRAM_ADDRESS | GET_MASK(u8Address,
LCD_CGRAM_ADDRESS_MASK));
260 }
```

Here is the call graph for this function:

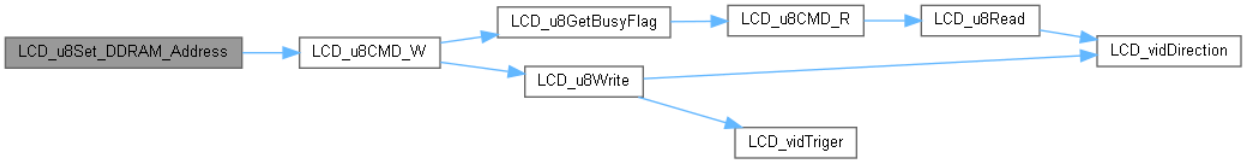


Here is the caller graph for this function:

LBTY_tenuErrorStatus LCD_u8Set_DDRAM_Address (u8 *u8Address*)

```
262                                                     {
263     return LCD_u8CMD_W(LCD_SEND_DDRAM_ADDRESS | GET_MASK(u8Address,
LCD_DDRAM_ADDRESS_MASK));
264 }
```

Here is the call graph for this function:

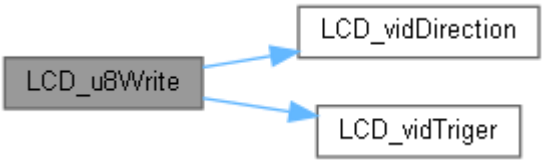


LBTY_tenuErrorStatus LCD_u8Write (u8 *u8Byte*)

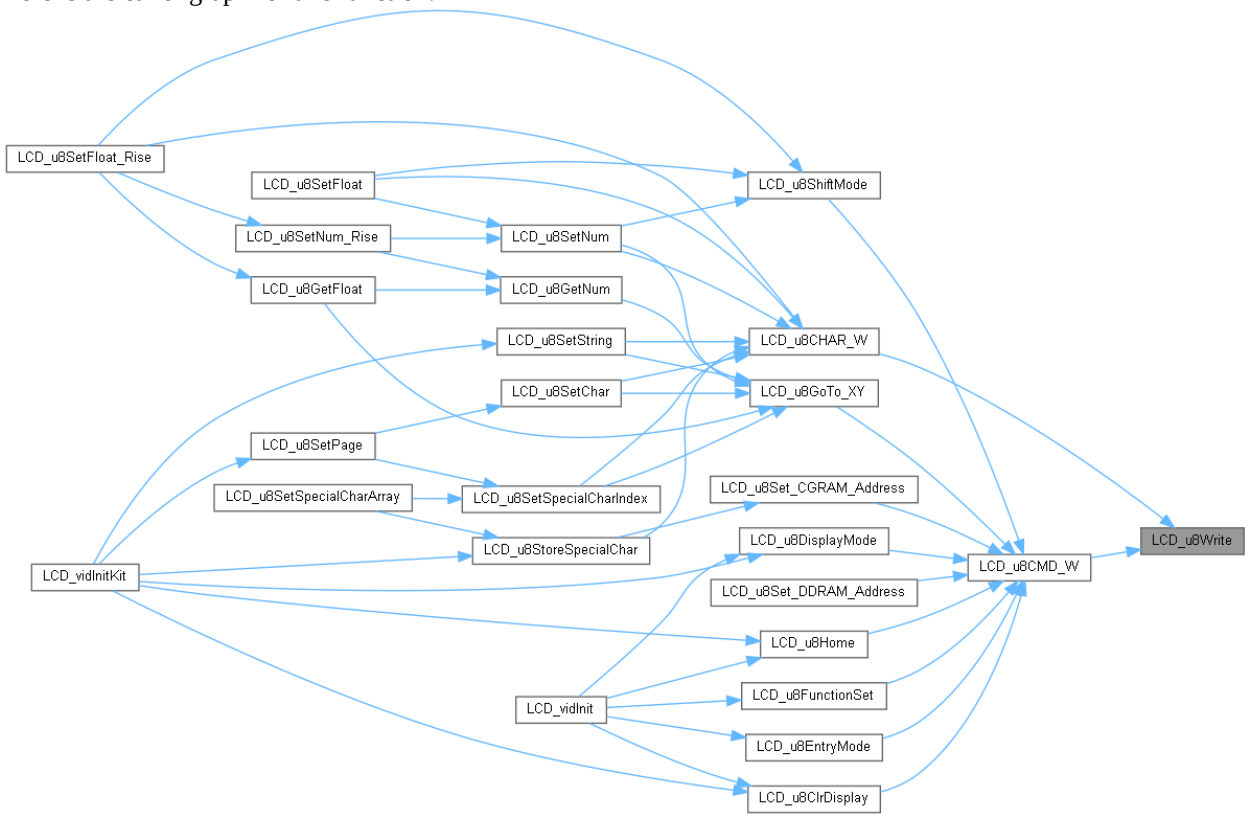
```
124                                             {
125     LBTY_tenuErrorStatus u8RetErrorState = LBTY_OK;
126     LBTY_tuniPort8 u8LcdByte = (LBTY_tuniPort8)u8Byte;
127
128 #ifdef LCD_RW
129     u8RetErrorState = GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_RW,
LCD_RW_WRITE);
130 #endif
131
132     LCD_vidDirection(PIN_OUTPUT);
```

```
133
134 #if (LCD_FUNCTION_SET == LCD_FUNCTION_SET_4Bits)
135     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D4, u8LcdByte.sBits.m_u8b4);
136     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D5, u8LcdByte.sBits.m_u8b5);
137     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D6, u8LcdByte.sBits.m_u8b6);
138     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D7, u8LcdByte.sBits.m_u8b7);
139     LCD_vidTriger();
140     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D4, u8LcdByte.sBits.m_u8b0);
141     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D5, u8LcdByte.sBits.m_u8b1);
142     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D6, u8LcdByte.sBits.m_u8b2);
143     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D7, u8LcdByte.sBits.m_u8b3);
144     LCD_vidTriger();
145 #elif (LCD_FUNCTION_SET == LCD_FUNCTION_SET_8Bits)
146     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D0, u8LcdByte.sBits.m_u8b0);
147     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D1, u8LcdByte.sBits.m_u8b1);
148     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D2, u8LcdByte.sBits.m_u8b2);
149     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D3, u8LcdByte.sBits.m_u8b3);
150     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D4, u8LcdByte.sBits.m_u8b4);
151     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D5, u8LcdByte.sBits.m_u8b5);
152     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D6, u8LcdByte.sBits.m_u8b6);
153     GPIO_u8SetPinValue(LCD_DATA_PORT, LCD_D7, u8LcdByte.sBits.m_u8b7);
154     LCD_vidTriger();
155 #endif
156     return u8RetErrorState;
157 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

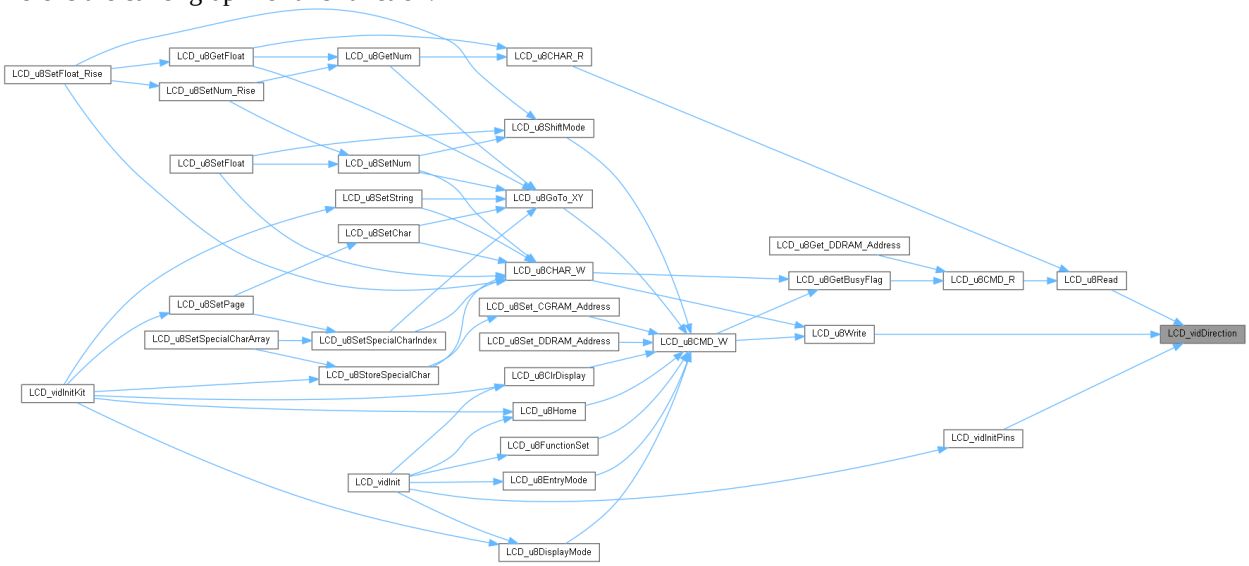


void LCD_vidDirection (u8 *u8PinDir*)

```
104                                 {
105 #if (LCD_FUNCTION_SET == LCD_FUNCTION_SET_8Bits)
106     GPIO_u8SetPinDirection(LCD_DATA_PORT, LCD_D0, u8PinDir);
107     GPIO_u8SetPinDirection(LCD_DATA_PORT, LCD_D1, u8PinDir);
108     GPIO_u8SetPinDirection(LCD_DATA_PORT, LCD_D2, u8PinDir);
```

```
109     GPIO_u8SetPinDirection(LCD_DATA_PORT, LCD_D3, u8PinDir);
110 #endif
111     GPIO_u8SetPinDirection(LCD_DATA_PORT, LCD_D4, u8PinDir);
112     GPIO_u8SetPinDirection(LCD_DATA_PORT, LCD_D5, u8PinDir);
113     GPIO_u8SetPinDirection(LCD_DATA_PORT, LCD_D6, u8PinDir);
114     GPIO_u8SetPinDirection(LCD_DATA_PORT, LCD_D7, u8PinDir);
115 }
```

Here is the caller graph for this function:



void LCD_vidInitPins (void)

```
92                              {
93     vidMyDelay_ms(LCD_DELAY_POWER_ON);       // Delay Power On
94
95     GPIO_u8SetPinDirection(LCD_CONTROL_PORT, LCD_EN, PIN_OUTPUT);
96     GPIO_u8SetPinDirection(LCD_CONTROL_PORT, LCD_RS, PIN_OUTPUT);
97 #ifdef LCD_RW
98     GPIO_u8SetPinDirection(LCD_CONTROL_PORT, LCD_RW, PIN_OUTPUT);
99 #endif
100
101     LCD_vidDirection(PIN_OUTPUT);
102 }
```

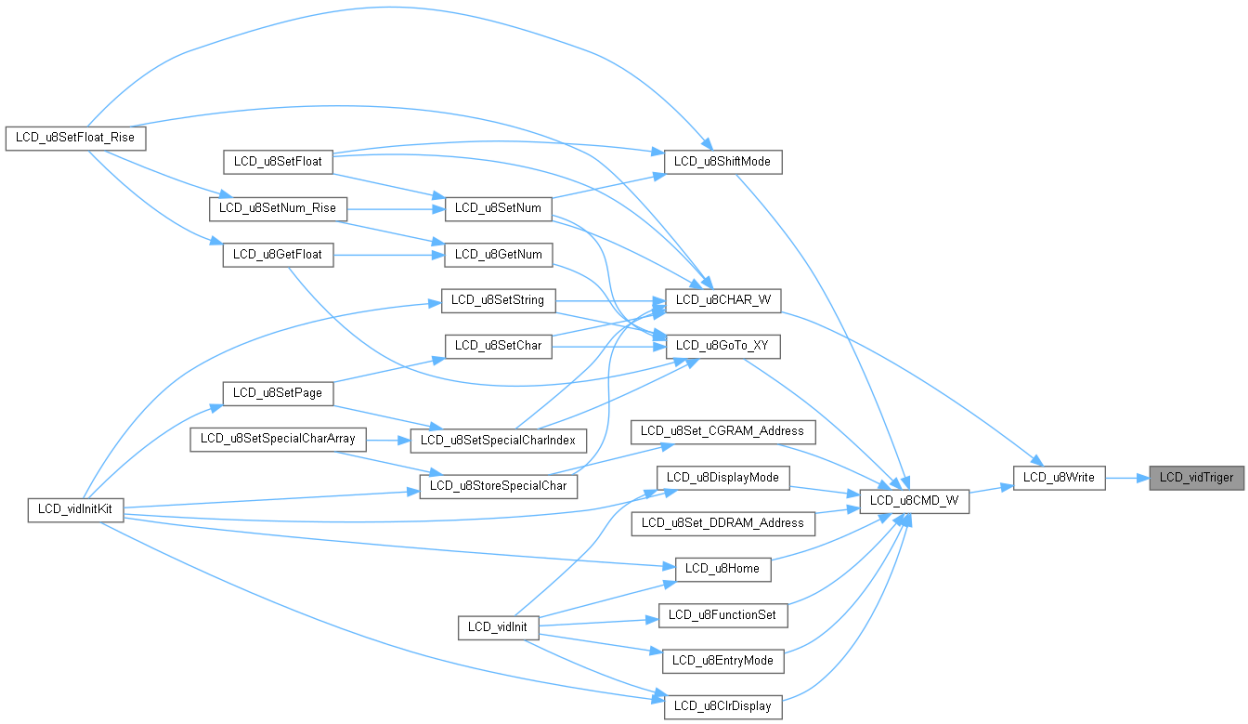
Here is the call graph for this function:

Here is the caller graph for this function:

void LCD_vidTriger (void)

```
117                             {
118     GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_EN, PIN_High);
119     vidMyDelay_ms(LCD_DELAY_CMD);
120     GPIO_u8SetPinValue(LCD_CONTROL_PORT, LCD_EN, PIN_Low);
121     vidMyDelay_ms(LCD_DELAY_CMD);
122 }
```

Here is the caller graph for this function:



Variable Documentation

const u8 ETA32[][LCD_CGRAM_LOCATIONS_NUM] `[extern]`

LCD_priv.h

Go to the documentation of this file.

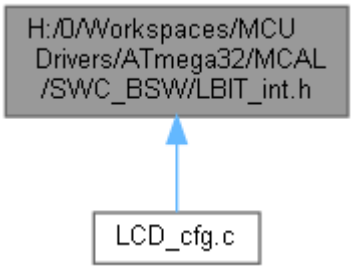
```
1 /*
2 /* ********************** FILE DEFINITION SECTION ************************** */
3 /* ************************************************************************* */
4 /* File Name    : LCD_priv.h                                                 */
5 /* Author       : MAAM                                                       */
6 /* Version      : v01.2                                                      */
7 /* date         : Mar 31, 2023                                               */
8 /* ************************************************************************* */
9 /* ************************ HEADER FILES INCLUDES ************************  */
10 /* ************************************************************************* */
11
12 #ifndef LCD_PRIV_H_
13 #define LCD_PRIV_H_
14
15 /* ************************************************************************* */
16 /* ********************** TYPE_DEF/STRUCT/ENUM SECTION ********************** */
17 /* ************************************************************************* */
18
19 /* ************************************************************************* */
20 /* ************************** MACRO/DEFINE SECTION ************************** */
21 /* ************************************************************************* */
22
24 #define LCD_CLEAR_DISPLAY                0x01u   // Clear Display (also clear DDRAM
content)
25 #define LCD_CURSOR_HOME                  0x02u    // Cursor Home
26
28 #define LCD_Entry_DEC                    0x04u   // Entry Decrement Cursor
29 #define LCD_Entry_DEC_SHIFT              0x05u   // Entry Decrement Cursor with Display
Shift
30 #define LCD_Entry_INC                    0x06u   // Entry Increment Cursor
31 #define LCD_Entry_INC_SHIFT              0x07u   // Entry Increment Cursor with Display
Shift
32
34 #define LCD_DISPLAY_OFF_CURSOR_OFF       0x08u   // Display off Cursor off(clearing
display without clearing DDRAM content)                                    // Not Used
36 #define LCD_DISPLAY_ON_CURSOR_OFF        0x0Cu   // Display on Cursor off
37 #define LCD_DISPLAY_ON_CURSOR_OFF_BLINK  0x0Du   // Display on Cursor off    Blinking
38 #define LCD_DISPLAY_ON_CURSOR_UNDERLINE  0x0Eu   // Display on Cursor on     Under Line
39 #define LCD_DISPLAY_ON_CURSOR_BLINK      0x0Fu   // Display on Cursor on     blinking
40
42 #define LCD_CURSOR_SHIFT_LEFT            0x10u   // Move cursor left by one character
44 #define LCD_CURSOR_SHIFT_RIGHT           0x14u   // Move cursor right by one character
46 #define LCD_DISPLAY_SHIFT_LEFT           0x18u   // Shift entire display left
48 #define LCD_DISPLAY_SHIFT_RIGHT          0x1Cu   // Shift entire display right
51
/*****************************************************************************
*****************************/
52 // MxN
53 // N = 1, 2, 4
54 // M = 8, 16, 20
55
58 #define LCD_CONFIG_1LINE_4BIT_5ROW       0x20u   // Function Set: 4-bit, 1 Line, 5x7
Dots
59 #define LCD_CONFIG_1LINE_4BIT_10ROW      0x24u   // Function Set: 4-bit, 1 Line, 5x10
Dots
60 #define LCD_CONFIG_2LINE_4BIT_5ROW       0x28u   // Function Set: 4-bit, 2 Line, 5x7
Dots
61 #define LCD_CONFIG_2LINE_4BIT_10ROW      0x2Cu   // Function Set: 4-bit, 2 Line, 5x10
Dots
64 #define LCD_CONFIG_1LINE_8BIT_5ROW       0x30u   // Function Set: 8-bit, 1 Line, 5x7
Dots
65 #define LCD_CONFIG_1LINE_8BIT_10ROW      0x34u   // Function Set: 8-bit, 1 Line, 5x10
Dots
66 #define LCD_CONFIG_2LINE_8BIT_5ROW       0x38u   // Function Set: 8-bit, 2 Line, 5x7
Dots
67 //#define LCD_CONFIG_2LINE_8BIT_10ROW       0x3Cu   // Function Set: 8-bit, 2 Line,
5x10 Dots (Can't Used)
70 // LCD_XXXX_LINE_POSITION_0 + x = Jump Cursor to XXXX line position x
71 #define LCD_FIRST_LINE_POSITION_0        0x80u   // Force cursor to beginning of first
line
73 #define LCD_SECOND_LINE_POSITION_0       0xC0u   // Force cursor to beginning of second
line
```

```
75 #define LCD_THIRD_LINE_POSITION_0       0x90u   // Force cursor to beginning of third
line
76 #define LCD_THIRD_LINE_POSITION_0_20    0x94u   // Force cursor to beginning of third
line
77 #define LCD_THIRD_LINE_POSITION_0_32    0xA0u   // Force cursor to beginning of third
line
79 #define LCD_FOURTH_LINE_POSITION_0      0xD0u   // Force cursor to beginning of fourth
line
80 #define LCD_FOURTH_LINE_POSITION_0_20   0xD4u   // Force cursor to beginning of fourth
line
81 #define LCD_FOURTH_LINE_POSITION_0_32   0xE0u   // Force cursor to beginning of fourth
line
82
83
/*******************************************************************************
*****************************/
84 // LCD_REGISTER_SELECT_PIN
85 #define LCD_RS_CMD                      0u
86 #define LCD_RS_DATA                     1u
87
88 //LCD_READ_WRITE_PIN
89 #define LCD_RW_WRITE                    0u
90 #define LCD_RW_READ                     1u
91
92 #define BUSY_FLAG_BIT                   7u
93                 // 40usec
95 #define LCD_WRITE_INSTRUCTION_CMD       0u
96 #define LCD_READ_INSTRUCTION_CMD        1u
97 #define LCD_WRITE_DATA_CMD              2u
98 #define LCD_READ_DATA_CMD               3u
99
100 #define LCD_SEND_CGRAM_ADDRESS          0x40u   // Character Generator RAM
102 #define LCD_SEND_DDRAM_ADDRESS          0x80u   // Display Data RAM
105 #define LCD_CGRAM_ADDRESS_MASK          0x3Fu
106 #define LCD_DDRAM_ADDRESS_MASK          0x7Fu
107
108 #define LCD_CGRAM_SECTIONS_NUM          8u
109 #define LCD_CGRAM_LOCATIONS_NUM         8u
110
111 /* ************************************************************************** */
112 /* ***************************** CONST SECTION ****************************** */
113 /* ************************************************************************** */
114
115 extern const u8 ETA32[][LCD_CGRAM_LOCATIONS_NUM];
116
117 /* ************************************************************************** */
118 /* ***************************** VARIABLE SECTION *************************** */
119 /* ************************************************************************** */
120
121 /* ************************************************************************** */
122 /* ***************************** FUNCTION SECTION *************************** */
123 /* ************************************************************************** */
124
125 extern LBTY_tenuErrorStatus LCD_u8FunctionSet(void);
126 extern void LCD_vidInitPins(void);
127
128 extern void LCD_vidDirection(u8 u8PinDir);
129 extern void LCD_vidTriger(void);
130
131 extern LBTY_tenuErrorStatus LCD_u8Write(u8 u8Byte);
132 extern LBTY_tenuErrorStatus LCD_u8Read(u8* pu8Byte);
133
134 extern LBTY_tenuErrorStatus LCD_u8CMD_W(u8 u8CMD);
135 extern LBTY_tenuErrorStatus LCD_u8CMD_R(u8* pu8CMD);
136
137 extern LBTY_tenuErrorStatus LCD_u8CHAR_W(u8 u8Char);
138 extern LBTY_tenuErrorStatus LCD_u8CHAR_R(u8* pu8Char);
139
140 extern LBTY_tenuErrorStatus LCD_u8Set_CGRAM_Address(u8 u8Address);
141 extern LBTY_tenuErrorStatus LCD_u8Set_DDRAM_Address(u8 u8Address);
142
143 extern LBTY_tenuErrorStatus LCD_u8Get_DDRAM_Address(u8* pu8Address);
144 extern u8 LCD_u8GetBusyFlag(void);
145
146 #endif /* LCD_PRIV_H_ */
147 /*************************** E N D (LCD_priv.h) ****************************/
```

main.c File Reference

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.

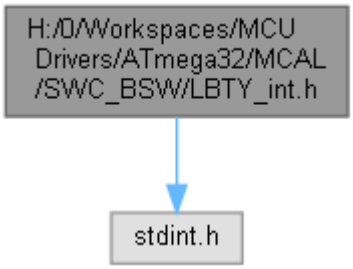
```
1 /*
****************************************************************************** */
2 /* ********************** FILE DEFINITION SECTION ************************** */
3 /* ****************************************************************************** */
4 /* File Name   : LBIT_int.h           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

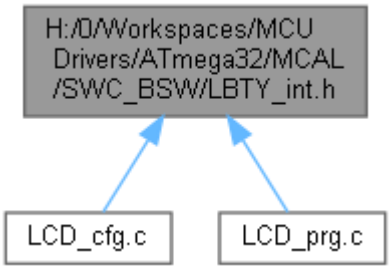
H:/0/Workspaces/MCU Drivers/ATmega32/MCAL/SWC_BSW/LBTY_int.h File Reference

```
#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.

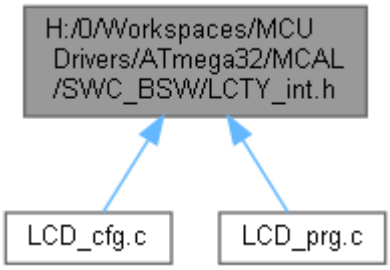
```
1 /*
2 /* ********************** FILE DEFINITION SECTION ************************** */
3 /* ************************************************************************* */
4 /* File Name   : LBTY_int.h                                                  */
5 /* Author      : MAAM                      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