

uHeartMonitor

Generated by Doxygen 1.9.8

<b>1 Topic Index</b>	<b>1</b>
1.1 Topics	1
<b>2 Data Structure Index</b>	<b>2</b>
2.1 Data Structures	2
<b>3 File Index</b>	<b>2</b>
3.1 File List	2
<b>4 Topic Documentation</b>	<b>4</b>
4.1 Application Software	4
4.1.1 Detailed Description	5
4.1.2 Data Acquisition (DAQ)	5
4.1.3 Liquid Crystal Display (LCD)	12
4.1.4 QRS Detector	36
4.1.5 Main	43
4.2 Common	50
4.2.1 Detailed Description	51
4.2.2 Function Documentation	51
4.2.3 FIFO Buffers	52
4.2.4 NewAssert	59
4.3 Middleware	59
4.3.1 Detailed Description	60
4.3.2 Debug	60
4.3.3 ILI9341	63
4.3.4 LED	74
4.4 Device Drivers	79
4.4.1 Detailed Description	80
4.4.2 Analog-to-Digital Conversion (ADC)	80
4.4.3 General-Purpose Input/Output (GPIO)	81
4.4.4 Phase-Locked Loop (PLL)	90
4.4.5 Serial Peripheral Interface (SPI)	91
4.4.6 Timer	94
4.4.7 Universal Asynchronous Receiver/Transmitter (UART)	104
4.4.8 Interrupt Service Routines	108
<b>5 Data Structure Documentation</b>	<b>112</b>
5.1 Fifo_t Struct Reference	112
5.2 GpioPort_t Struct Reference	113
5.3 Led_t Struct Reference	113
5.4 Timer_t Struct Reference	113
5.5 Uart_t Struct Reference	114
<b>6 File Documentation</b>	<b>114</b>

6.1 DAQ.c File Reference . . . . .	114
6.2 DAQ.h File Reference . . . . .	114
6.3 DAQ_lookup.c File Reference . . . . .	115
6.4 Font.c File Reference . . . . .	115
6.5 LCD.c File Reference . . . . .	116
6.6 LCD.h File Reference . . . . .	116
6.7 QRS.c File Reference . . . . .	117
6.8 QRS.h File Reference . . . . .	118
6.9 Fifo.c File Reference . . . . .	118
6.10 Fifo.h File Reference . . . . .	119
6.11 NewAssert.c File Reference . . . . .	120
6.12 NewAssert.h File Reference . . . . .	120
6.13 ADC.c File Reference . . . . .	121
6.14 ADC.h File Reference . . . . .	121
6.15 GPIO.c File Reference . . . . .	121
6.16 GPIO.h File Reference . . . . .	122
6.17 ISR.c File Reference . . . . .	123
6.18 ISR.h File Reference . . . . .	123
6.19 PLL.c File Reference . . . . .	124
6.20 PLL.h File Reference . . . . .	125
6.21 SPI.c File Reference . . . . .	125
6.22 SPI.h File Reference . . . . .	125
6.23 Timer.c File Reference . . . . .	126
6.24 Timer.h File Reference . . . . .	127
6.25 UART.c File Reference . . . . .	127
6.26 UART.h File Reference . . . . .	128
6.27 main.c File Reference . . . . .	129
6.28 Debug.h File Reference . . . . .	129
6.29 ILI9341.c File Reference . . . . .	130
6.30 ILI9341.h File Reference . . . . .	131
6.31 Led.c File Reference . . . . .	132
6.32 Led.h File Reference . . . . .	132
<b>Index</b>	<b>135</b>

## 1 Topic Index

### 1.1 Topics

Here is a list of all topics with brief descriptions:

#### Application Software

Data Acquisition (DAQ)	5
Liquid Crystal Display (LCD)	12
QRS Detector	36
Main	43
Common	50
FIFO Buffers	52
NewAssert	59
Middleware	59
Debug	60
ILI9341	63
LED	74
Device Drivers	79
Analog-to-Digital Conversion (ADC)	80
General-Purpose Input/Output (GPIO)	81
Phase-Locked Loop (PLL)	90
Serial Peripheral Interface (SPI)	91
Timer	94
Universal Asynchronous Receiver/Transmitter (UART)	104
Interrupt Service Routines	108

## 2 Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">Fifo_t</a>	112
<a href="#">GpioPort_t</a>	113
<a href="#">Led_t</a>	113
<a href="#">Timer_t</a>	113
<a href="#">Uart_t</a>	114

## 3 File Index

### 3.1 File List

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

<a href="#">DAQ.c</a>	Source code for DAQ module	114
<a href="#">DAQ.h</a>	Application software for handling data acquisition (DAQ) functions	114
<a href="#">DAQ_lookup.c</a>	Source code for DAQ module's lookup table	115
<a href="#">Font.c</a>	Contains bitmaps for a selection of ASCII characters	115
<a href="#">LCD.c</a>	Source code for LCD module	116
<a href="#">LCD.h</a>	Header file for LCD module	116
<a href="#">QRS.c</a>	Source code for QRS detection module	117
<a href="#">QRS.h</a>	Header file for QRS detection module	118
<a href="#">Fifo.c</a>	Source code for FIFO buffer module	118
<a href="#">Fifo.h</a>	Header file for FIFO buffer implementation	119
<a href="#">NewAssert.c</a>	Source code for custom <code>assert</code> implementation	120
<a href="#">NewAssert.h</a>	Header file for custom <code>assert</code> implementation	120
<a href="#">ADC.c</a>	Source code for analog-to-digital conversion (ADC) module	121
<a href="#">ADC.h</a>	Header file for analog-to-digital conversion (ADC) module	121
<a href="#">GPIO.c</a>	Source code for GPIO module	121
<a href="#">GPIO.h</a>	Header file for general-purpose input/output (GPIO) device driver	122
<a href="#">ISR.c</a>	Source code for interrupt service routine (ISR) configuration module	123
<a href="#">ISR.h</a>	Header file for interrupt service routine (ISR) configuration module	123
<a href="#">PLL.c</a>	Implementation details for phase-lock-loop (PLL) functions	124
<a href="#">PLL.h</a>	Driver module for activating the phase-locked-loop (PLL)	125
<a href="#">SPI.c</a>	Source code for serial peripheral interface (SPI) module	125

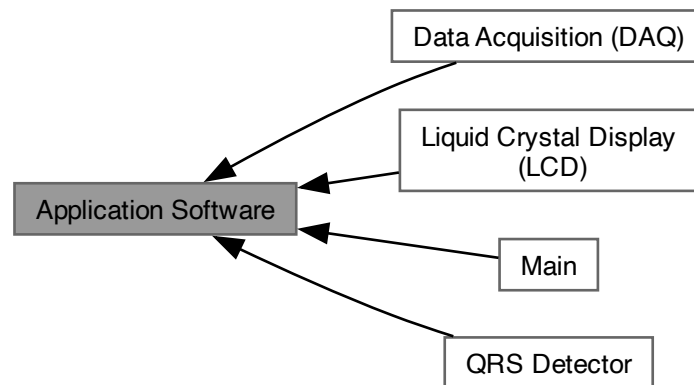
<a href="#">SPI.h</a>	Header file for serial peripheral interface (SPI) module	125
<a href="#">Timer.c</a>	Source code for Timer module	126
<a href="#">Timer.h</a>	Device driver for general-purpose timer modules	127
<a href="#">UART.c</a>	Source code for UART module	127
<a href="#">UART.h</a>	Driver module for serial communication via UART0 and UART 1	128
<a href="#">main.c</a>	Main program file	129
<a href="#">Debug.h</a>	Functions to output debugging information to a serial port via UART	129
<a href="#">ILI9341.c</a>	Source code for ILI9341 module	130
<a href="#">ILI9341.h</a>	Driver module for interfacing with an ILI9341 LCD driver	131
<a href="#">Led.c</a>	Source code for LED module	132
<a href="#">Led.h</a>	Interface for LED module	132

## 4 Topic Documentation

### 4.1 Application Software

Application-specific software modules.

Collaboration diagram for Application Software:



## Modules

- [Data Acquisition \(DAQ\)](#)  
*Module for managing data acquisition (DAQ) functions.*
- [Liquid Crystal Display \(LCD\)](#)  
*Module for displaying graphs on an LCD via the [ILI9341](#) module.*
- [QRS Detector](#)  
*Module for analyzing ECG data to determine heart rate.*
- [Main](#)  
*Main program file.*

### 4.1.1 Detailed Description

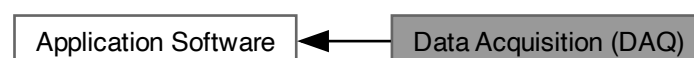
Application-specific software modules.

These modules contain functions built specifically for this project's purposes.

### 4.1.2 Data Acquisition (DAQ)

Module for managing data acquisition (DAQ) functions.

Collaboration diagram for Data Acquisition (DAQ):



## Files

- file [DAQ.c](#)  
*Source code for DAQ module.*
- file [DAQ.h](#)  
*Application software for handling data acquisition (DAQ) functions.*
- file [DAQ\\_lookup.c](#)  
*Source code for DAQ module's lookup table.*

## Macros

- `#define SAMPLING_PERIOD_MS 5`  
*sampling period in ms ( $T_s = \frac{1}{f_s}$ )*
- `#define DAQ_LOOKUP_MAX ((float32_t) 5.5f)`  
*maximum lookup table value*
- `#define DAQ_LOOKUP_MIN ((float32_t) (-5.5f))`  
*minimum lookup table value*

## Variables

- static const float32\_t **DAQ\_LOOKUP\_TABLE** [4096]  
*Lookup table for converting ADC data from unsigned 12-bit integer values to 32-bit floating point values.*

## Digital Filters

- enum {  
    **NUM\_STAGES\_NOTCH** = 6 , **NUM\_COEFFS\_NOTCH** = NUM\_STAGES\_NOTCH \* 5 , **STATE\_BUFF\_SIZE\_NOTCH** = NUM\_STAGES\_NOTCH \* 4 , **NUM\_STAGES\_BANDPASS** = 4 ,  
    **NUM\_COEFFS\_DAQ\_BANDPASS** = NUM\_STAGES\_BANDPASS \* 5 , **STATE\_BUFF\_SIZE\_BANDPASS** = NUM\_STAGES\_BANDPASS \* 4 }
- typedef arm\_biquad\_casd\_df1\_inst\_f32 **Filter\_t**
- static const float32\_t [COEFFS\\_NOTCH](#) [NUM\_COEFFS\_NOTCH]
- static const float32\_t [COEFFS\\_BANDPASS](#) [NUM\_COEFFS\_DAQ\_BANDPASS]
- static float32\_t **stateBuffer\_Notch** [STATE\_BUFF\_SIZE\_NOTCH]
- static const Filter\_t **notchFiltStruct** = { NUM\_STAGES\_NOTCH, stateBuffer\_Notch, COEFFS\_NOTCH }
- static const Filter\_t \*const **notchFilter** = &notchFiltStruct
- static float32\_t **stateBuffer\_Bandpass** [STATE\_BUFF\_SIZE\_BANDPASS]
- static const Filter\_t [bandpassFiltStruct](#)
- static const Filter\_t \*const **bandpassFilter** = &bandpassFiltStruct

## Initialization

- void [DAQ\\_Init](#) (void)  
*Initialize the data acquisition (DAQ) module.*



### Reading Input Data

- uint16\_t `DAQ_readSample` (void)  
*Read a sample from the ADC.*
- void `DAQ_acknowledgeInterrupt` (void)  
*Acknowledge the ADC interrupt.*
- float32\_t `DAQ_convertToMilliVolts` (uint16\_t sample)  
*Convert a 12-bit ADC sample to a floating-point voltage value via LUT.*

### Digital Filtering Functions

- float32\_t `DAQ_NotchFilter` (volatile float32\_t xn)  
*Apply a 60 [Hz] notch filter to an input sample.*
- float32\_t `DAQ_BandpassFilter` (volatile float32\_t xn)  
*Apply a 0.5-40 [Hz] bandpass filter to an input sample.*

#### 4.1.2.1 Detailed Description

Module for managing data acquisition (DAQ) functions.

#### 4.1.2.2 Function Documentation

##### DAQ\_Init()

```
void DAQ_Init (
    void )
```

Initialize the data acquisition (DAQ) module.

##### Postcondition

The analog-to-digital converter (ADC) is initialized and configured for timer-triggered sample capture.

The timer is initialized in PERIODIC mode and triggers the ADC every  $5ms$  (i.e. sampling frequency  $f_s = 200Hz$ ).

The DAQ module has access to its lookup table (LUT).

##### DAQ\_readSample()

```
uint16_t DAQ_readSample (
    void )
```

Read a sample from the ADC.

##### Precondition

Initialize the DAQ module.

This should be used in an interrupt handler and/or at a consistent rate (i.e. the sampling frequency).

**Parameters**

out	<i>sample</i>	12-bit sample in range [0x000, 0xFFF]
-----	---------------	---------------------------------------

**Postcondition**

The sample can now be converted to millivolts.

**See also**

[DAQ\\_convertToMilliVolts\(\)](#)

**DAQ\_acknowledgeInterrupt()**

```
void DAQ_acknowledgeInterrupt (
    void )
```

Acknowledge the ADC interrupt.

**Precondition**

This should be used within an interrupt handler.

**DAQ\_NotchFilter()**

```
float32_t DAQ_NotchFilter (
    volatile float32_t xn )
```

Apply a 60 [Hz] notch filter to an input sample.

**Precondition**

Read a sample from the ADC and convert it to millivolts.

**Parameters**

in	<i>xn</i>	Raw input sample
out	<i>yn</i>	Filtered output sample

**Postcondition**

$y[n]$  is ready for analysis and/or further processing.

**See also**

[DAQ\\_BandpassFilter\(\)](#)

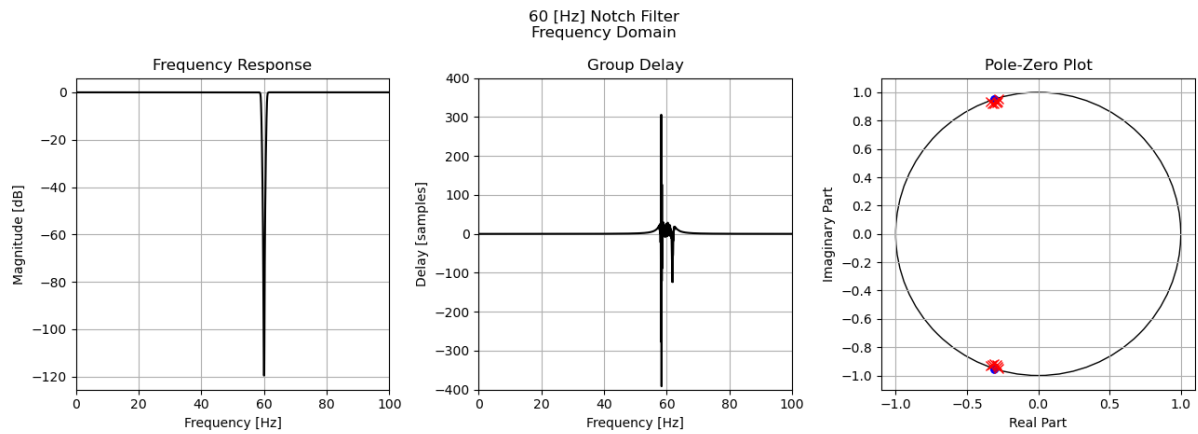


Figure 1 Frequency domain parameters for the notch filter.

### DAQ\_BandpassFilter()

```
float32_t DAQ_BandpassFilter (
    volatile float32_t xn )
```

Apply a 0.5-40 [Hz] bandpass filter to an input sample.

#### Precondition

Read a sample from the ADC and convert it to millivolts.

#### Parameters

in	$xn$	Input sample
out	$yn$	Filtered output sample

#### Postcondition

$y[n]$  is ready for analysis and/or further processing.

See also

[DAQ\\_NotchFilter\(\)](#)

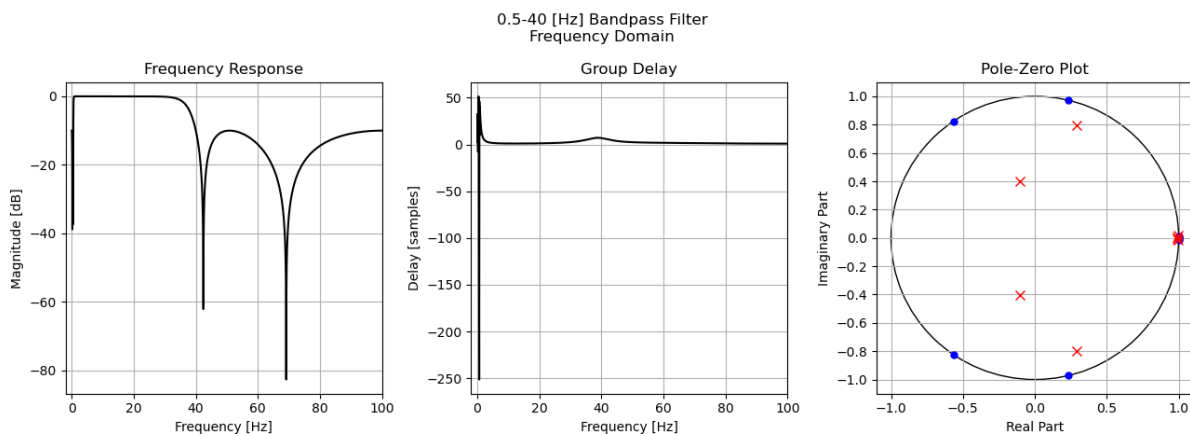


Figure 2 Frequency domain parameters for the bandpass filter.

### DAQ\_convertToMilliVolts()

```
float32_t DAQ_convertToMilliVolts (
    uint16_t sample )
```

Convert a 12-bit ADC sample to a floating-point voltage value via LUT.

#### Precondition

Read a sample from the ADC.

#### Parameters

in	<i>sample</i>	12-bit sample in range $[0x000, 0xFFF]$
out	<i>xn</i>	Voltage value in range $[-5.5, 5.5][mV]$

#### Postcondition

The sample  $x[n]$  is ready for filtering.

See also

[DAQ\\_readSample\(\)](#)

#### Note

Defined in [DAQ\\_lookup.c](#) rather than [DAQ.c](#).

### 4.1.2.3 Variable Documentation

#### COEFFS\_NOTCH

```
const float32_t COEFFS_NOTCH[NUM_COEFFS_NOTCH] [static]
```

##### Initial value:

```
= {
    0.8856732845306396f, 0.5476464033126831f, 0.8856732845306396f,
    -0.5850160717964172f, -0.9409302473068237f,

    1.0f, 0.6183391213417053f, 1.0f,
    -0.615153431892395f, -0.9412328004837036f,

    1.0f, 0.6183391213417053f, 1.0f,
    -0.5631667971611023f, -0.9562366008758545f,

    1.0f, 0.6183391213417053f, 1.0f,
    -0.6460562348365784f, -0.9568508863449097f,

    1.0f, 0.6183391213417053f, 1.0f,
    -0.5554963946342468f, -0.9837208390235901f,

    1.0f, 0.6183391213417053f, 1.0f,
    -0.6700929999351501f, -0.9840363264083862f,
}
```

#### COEFFS\_BANDPASS

```
const float32_t COEFFS_BANDPASS[NUM_COEFFS_DAO_BANDPASS] [static]
```

##### Initial value:

```
= {
    0.3240305185317993f, 0.3665695786476135f, 0.3240305185317993f,
    -0.20968256890773773f, -0.1729172021150589f,

    1.0f, -0.4715292155742645f, 1.0f,
    0.5868059992790222f, -0.7193671464920044f,

    1.0f, -1.9999638795852661f, 1.0f,
    1.9863483905792236f, -0.986438512802124f,

    1.0f, -1.9997893571853638f, 1.0f,
    1.994096040725708f, -0.9943605065345764f,
}
```

#### bandpassFiltStruct

```
const Filter_t bandpassFiltStruct [static]
```

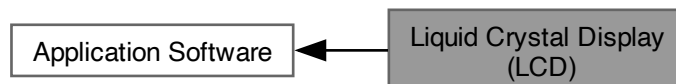
##### Initial value:

```
= { NUM_STAGES_BANDPASS, stateBuffer_Bandpass,
                                         COEFFS_BANDPASS }
```

### 4.1.3 Liquid Crystal Display (LCD)

Module for displaying graphs on an LCD via the [ILI9341](#) module.

Collaboration diagram for Liquid Crystal Display (LCD):



#### Files

- file [Font.c](#)  
*Contains bitmaps for a selection of ASCII characters.*
- file [LCD.c](#)  
*Source code for LCD module.*
- file [LCD.h](#)  
*Header file for LCD module.*

#### Macros

- `#define CONVERT_INT_TO_ASCII(X) ((unsigned char) (X + 0x30))`

#### Functions

- static void [LCD\\_plotSample](#) (uint16\_t x, uint16\_t y, uint8\_t color)  
*Plot a sample at coordinates (x, y).*

#### Variables

- const uint8\_t \*const **FONT\_ARRAY** [128]
- struct {
  - uint16\_t **x1**  
*starting x-value in range [0, x2]*
  - uint16\_t **x2**  
*ending x-value in range [0, NUM\_ROWS)*
  - uint16\_t **y1**  
*starting y-value in range [0, y2]*
  - uint16\_t **y2**  
*ending x-value in range [0, NUM\_COLS)*
  - uint16\_t **lineNum**  
*line number for text; in range [0, NUM\_LINES)*
  - uint16\_t **colNum**  
*column number for text; in range [0, NUM\_COLS)*
  - uint8\_t **color**
  - bool **isInit**  
*if true, LCD has been initialized*
- const uint8\_t \*const **FONT\_ARRAY** [128]

### Initialization & Configuration

- enum **LCD\_PLOT\_INFO** { **LCD\_X\_MAX** = ILI9341\_NUM\_ROWS - 1 , **LCD\_Y\_MAX** = ILI9341\_NUM\_COLS - 1 }
- enum **LCD\_COLORS** {  
**LCD\_BLACK** = 0x00 ^ 0x07 , **LCD\_RED** = 0x04 ^ 0x07 , **LCD\_GREEN** = 0x02 ^ 0x07 , **LCD\_BLUE** = 0x01 ^ 0x07 ,  
**LCD\_YELLOW** = 0x06 ^ 0x07 , **LCD\_CYAN** = 0x03 ^ 0x07 , **LCD\_PURPLE** = 0x05 ^ 0x07 , **LCD\_WHITE** = 0x07 ^ 0x07 }
- void **LCD\_Init** (void)  
*Initialize the LCD.*
- void **LCD\_setOutputMode** (bool isOn)  
*Toggle display output ON or OFF (OFF by default).*
- void **LCD\_setX** (uint16\_t x1, uint16\_t x2)  
*Set new x-coordinates to be written to.  $0 \leq x1 \leq x2 \leq X_{MAX}$ .*
- void **LCD\_setY** (uint16\_t y1, uint16\_t y2)  
*Set new y-coordinates to be written to.  $0 \leq y1 \leq y2 \leq Y_{MAX}$ .*
- void **LCD\_setColor** (uint8\_t color)  
*Set the color value.*

### Writing

- enum **LCD\_WRITING\_INFO** { **HEIGHT\_CHAR** = 8 , **LEN\_CHAR** = 5 , **NUM\_LINES** = 30 , **NUM\_COLS** = 64 }
- void **LCD\_setCursor** (uint16\_t lineNum, uint16\_t colNum)  
*Set the cursor to line x, column y.*
- void **LCD\_writeChar** (unsigned char inputChar)
- void **LCD\_writeStr** (void \*asciiString)
- void **LCD\_writeInt** (int32\_t num)
- void **LCD\_writeFloat** (float num)

### ASCII Characters (Punctuation)

- static const uint8\_t **FONT\_SPACE** [8]
- static const uint8\_t **FONT\_PERIOD** [8]
- static const uint8\_t **FONT\_COLON** [8]

### ASCII Characters (Numbers)

- static const uint8\_t **FONT\_0** [8]
- static const uint8\_t **FONT\_1** [8]
- static const uint8\_t **FONT\_2** [8]
- static const uint8\_t **FONT\_3** [8]
- static const uint8\_t **FONT\_4** [8]
- static const uint8\_t **FONT\_5** [8]
- static const uint8\_t **FONT\_6** [8]
- static const uint8\_t **FONT\_7** [8]
- static const uint8\_t **FONT\_8** [8]
- static const uint8\_t **FONT\_9** [8]

**ASCII Characters (Uppercase Letters)**

- static const uint8\_t [FONT\\_UPPER\\_A](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_B](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_C](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_D](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_E](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_F](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_G](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_H](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_I](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_J](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_K](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_L](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_M](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_N](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_O](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_P](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_Q](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_R](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_S](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_T](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_U](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_V](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_W](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_X](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_Y](#) [8]
- static const uint8\_t [FONT\\_UPPER\\_Z](#) [8]

**ASCII Characters (Lowercase Letters)**

- static const uint8\_t [FONT\\_LOWER\\_A](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_B](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_C](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_D](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_E](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_F](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_G](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_H](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_I](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_J](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_K](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_L](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_M](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_N](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_O](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_P](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_Q](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_R](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_S](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_T](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_U](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_V](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_W](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_X](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_Y](#) [8]
- static const uint8\_t [FONT\\_LOWER\\_Z](#) [8]



## Helper Functions

- static void `LCD_drawLine` (uint16\_t center, uint16\_t lineWidth, bool is\_horizontal)  
*Helper function for drawing straight lines.*
- static void `LCD_updateCursor` (void)  
*Update the cursor for after writing text on the display.*

## Drawing

- void `LCD_Draw` (void)  
*Draw on the LCD.*
- void `LCD_Fill` (void)  
*Fill the display with a single color.*
- void `LCD_drawHoriLine` (uint16\_t yCenter, uint16\_t lineWidth)  
*Draw a horizontal line across the entire display.*
- void `LCD_drawVertLine` (uint16\_t xCenter, uint16\_t lineWidth)  
*Draw a vertical line across the entire display.*
- void `LCD_drawRectangle` (uint16\_t x1, uint16\_t dx, uint16\_t y1, uint16\_t dy)  
*Draw a rectangle of size dx x dy onto the display. The bottom-left corner will be located at (x1, y1).*

### 4.1.3.1 Detailed Description

Module for displaying graphs on an LCD via the [ILI9341](#) module.

### 4.1.3.2 Function Documentation

#### LCD\_drawLine()

```
static void LCD_drawLine (
    uint16_t center,
    uint16_t lineWidth,
    bool is_horizontal ) [static]
```

Helper function for drawing straight lines.

#### Parameters

<i>center</i>	Row or column that the line is centered on. <code>center</code> is increased or decreased if the line to be written would have gone out of bounds.
<i>lineWidth</i>	Width of the line. Should be a positive, odd number.
<i>is_row</i>	<code>true</code> for horizontal line, <code>false</code> for vertical line

#### LCD\_Init()

```
void LCD_Init (
    void )
```

Initialize the LCD.

**Postcondition**

The display will be ready to accept commands, but output will be off.

**LCD\_setOutputMode()**

```
void LCD_setOutputMode (
    bool isOn )
```

Toggle display output ON or OFF (OFF by default).

**Parameters**

in	<i>isOn</i>	true to turn display output ON, false to turn OFF
----	-------------	---

**Postcondition**

When OFF, the display is cleared. When ON, the IC writes pixel data from its memory to the display.

**LCD\_setX()**

```
void LCD_setX (
    uint16_t x1,
    uint16_t x2 )
```

Set new x-coordinates to be written to.  $0 \leq x1 \leq x2 \leq X_{MAX}$ .

**Parameters**

in	<i>x1</i>	left-most x-coordinate
in	<i>x2</i>	right-most x-coordinate

**See also**

[LCD\\_setY\(\)](#)

**LCD\_setY()**

```
void LCD_setY (
    uint16_t y1,
    uint16_t y2 )
```

Set new y-coordinates to be written to.  $0 \leq y1 \leq y2 \leq Y_{MAX}$ .

**Parameters**

in	<i>y1</i>	lowest y-coordinate
in	<i>y2</i>	highest y-coordinate

See also

[LCD\\_setX\(\)](#)

### LCD\_setColor()

```
void LCD_setColor (
    uint8_t color )
```

Set the color value.

#### Parameters

in	<i>color</i>	Color to use.
----	--------------	---------------

#### Postcondition

Outgoing pixel data will use the selected color.

### LCD\_Draw()

```
void LCD_Draw (
    void )
```

Draw on the LCD.

#### Precondition

Set the drawable area and the color to use for that area.

#### Postcondition

The selected areas of the display will be drawn onto with the selected color.

See also

[LCD\\_setX\(\)](#), [LCD\\_setY\(\)](#), [LCD\\_setColor\(\)](#)

References [ILI9341\\_writeMemCmd\(\)](#), and [ILI9341\\_writePixel\(\)](#).

### LCD\_Fill()

```
void LCD_Fill (
    void )
```

Fill the display with a single color.

#### Precondition

Select the desired color to fill the display with.

See also

[LCD\\_setColor\(\)](#)

**LCD\_drawHoriLine()**

```
void LCD_drawHoriLine (
    uint16_t yCenter,
    uint16_t lineWidth )
```

Draw a horizontal line across the entire display.

**Precondition**

Select the desired color to use for the line.

**Parameters**

in	<i>yCenter</i>	y-coordinate to center the line on
in	<i>lineWidth</i>	width of the line; should be a positive, odd number

**See also**

[LCD\\_drawVertLine](#), [LCD\\_drawRectangle\(\)](#)

**LCD\_drawVertLine()**

```
void LCD_drawVertLine (
    uint16_t xCenter,
    uint16_t lineWidth )
```

Draw a vertical line across the entire display.

**Precondition**

Select the desired color to use for the line.

**Parameters**

in	<i>xCenter</i>	x-coordinate to center the line on
in	<i>lineWidth</i>	width of the line; should be a positive, odd number

**See also**

[LCD\\_drawHoriLine](#), [LCD\\_drawRectangle\(\)](#)

**LCD\_drawRectangle()**

```
void LCD_drawRectangle (
    uint16_t x1,
    uint16_t dx,
    uint16_t y1,
    uint16_t dy )
```

Draw a rectangle of size dx x dy onto the display. The bottom-left corner will be located at (x1, y1).

**Precondition**

Select the desired color to use for the rectangle.

**Parameters**

in	<i>x1</i>	lowest (left-most) x-coordinate
in	<i>dx</i>	length (horizontal distance) of the rectangle
in	<i>y1</i>	lowest (bottom-most) y-coordinate
in	<i>dy</i>	height (vertical distance) of the rectangle

**See also**

[LCD\\_Draw\(\)](#), [LCD\\_Fill\(\)](#), [LCD\\_drawHoriLine\(\)](#), [LCD\\_drawVertLine\(\)](#)

**LCD\_plotSample()**

```
static void LCD_plotSample (
    uint16_t x,
    uint16_t y,
    uint8_t color ) [static]
```

Plot a sample at coordinates (*x*, *y*).

**Parameters**

in	<i>x</i>	x-coordinate (i.e. sample number) in range [0, X_MAX]
in	<i>y</i>	y-coordinate (i.e. amplitude) in range [0, Y_MAX]
in	<i>color</i>	Color to use

**See also**

[LCD\\_setX\(\)](#), [LCD\\_setY\(\)](#), [LCD\\_setColor\(\)](#), [LCD\\_Draw\(\)](#)

**LCD\_setCursor()**

```
void LCD_setCursor (
    uint16_t lineNum,
    uint16_t colNum )
```

Set the cursor to line *x*, column *y*.

**Parameters**

in	<i>lineNum</i>	Line number to place characters. Should be in range [0, 30).
in	<i>colNum</i>	Column number to place characters. Should be in range [0, 64).

#### 4.1.3.3 Variable Documentation

##### FONT\_SPACE

```
const uint8_t FONT_SPACE[8]  [static]
```

###### Initial value:

```
= {  
    0x00,  
    0x00,  
    0x00,  
    0x00,  
    0x00,  
    0x00,  
    0x00,  
    0x00  
}
```

##### FONT\_PERIOD

```
const uint8_t FONT_PERIOD[8]  [static]
```

###### Initial value:

```
= {  
    0x00,  
    0x00,  
    0x00,  
    0x00,  
    0x00,  
    0x00,  
    0x04,  
    0x04  
}
```

##### FONT\_COLON

```
const uint8_t FONT_COLON[8]  [static]
```

###### Initial value:

```
= {  
    0x00,  
    0x04,  
    0x00,  
    0x00,  
    0x00,  
    0x04,  
    0x00,  
    0x00  
}
```

##### FONT\_0

```
const uint8_t FONT_0[8]  [static]
```

###### Initial value:

```
= {  
    0x0E,  
    0x11,  
    0x13,  
    0x15,  
    0x19,  
    0x11,  
    0x11,  
    0x0E  
}
```

**FONT\_1**

```
const uint8_t FONT_1[8]  [static]
```

**Initial value:**

```
= {  
    0x06,  
    0x0E,  
    0x16,  
    0x06,  
    0x06,  
    0x06,  
    0x06,  
    0x1F  
}
```

**FONT\_2**

```
const uint8_t FONT_2[8]  [static]
```

**Initial value:**

```
= {  
    0x0E,  
    0x11,  
    0x01,  
    0x06,  
    0x08,  
    0x10,  
    0x11,  
    0x1F  
}
```

**FONT\_3**

```
const uint8_t FONT_3[8]  [static]
```

**Initial value:**

```
= {  
    0x0E,  
    0x11,  
    0x01,  
    0x06,  
    0x01,  
    0x11,  
    0x11,  
    0x0E  
}
```

**FONT\_4**

```
const uint8_t FONT_4[8]  [static]
```

**Initial value:**

```
= {  
    0x02,  
    0x06,  
    0x0A,  
    0x12,  
    0x1F,  
    0x02,  
    0x02,  
    0x02  
}
```

**FONT\_5**

```
const uint8_t FONT_5[8]  [static]
```

**Initial value:**

```
= {  
    0x1F,  
    0x10,  
    0x10,  
    0x1E,  
    0x01,  
    0x11,  
    0x11,  
    0x0E  
}
```

**FONT\_6**

```
const uint8_t FONT_6[8]  [static]
```

**Initial value:**

```
= {  
    0x0E,  
    0x11,  
    0x10,  
    0x1E,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x0E  
}
```

**FONT\_7**

```
const uint8_t FONT_7[8]  [static]
```

**Initial value:**

```
= {  
    0x1F,  
    0x11,  
    0x01,  
    0x02,  
    0x04,  
    0x04,  
    0x04,  
    0x04  
}
```

**FONT\_8**

```
const uint8_t FONT_8[8]  [static]
```

**Initial value:**

```
= {  
    0x0E,  
    0x11,  
    0x11,  
    0x0E,  
    0x11,  
    0x11,  
    0x11,  
    0x0E  
}
```



**FONT\_9**

```
const uint8_t FONT_9[8]  [static]
```

**Initial value:**

```
= {  
    0x0E,  
    0x11,  
    0x11,  
    0x0F,  
    0x01,  
    0x01,  
    0x11,  
    0x0E  
}
```

**FONT\_UPPER\_A**

```
const uint8_t FONT_UPPER_A[8]  [static]
```

**Initial value:**

```
= {  
    0x0E,  
    0x11,  
    0x11,  
    0x1F,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x11  
}
```

**FONT\_UPPER\_B**

```
const uint8_t FONT_UPPER_B[8]  [static]
```

**Initial value:**

```
= {  
    0x1E,  
    0x11,  
    0x11,  
    0x1E,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x1E  
}
```

**FONT\_UPPER\_C**

```
const uint8_t FONT_UPPER_C[8]  [static]
```

**Initial value:**

```
= {  
    0x0E,  
    0x11,  
    0x10,  
    0x10,  
    0x10,  
    0x11,  
    0x0E,  
    0x0E  
}
```

## FONT\_UPPER\_D

```
const uint8_t FONT_UPPER_D[8]  [static]
```

### Initial value:

```
= {  
    0x1E,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x1E  
}
```

## FONT\_UPPER\_E

```
const uint8_t FONT_UPPER_E[8]  [static]
```

### Initial value:

```
= {  
    0x1F,  
    0x10,  
    0x10,  
    0x1E,  
    0x10,  
    0x10,  
    0x10,  
    0x10,  
    0x1F  
}
```

## FONT\_UPPER\_F

```
const uint8_t FONT_UPPER_F[8]  [static]
```

### Initial value:

```
= {  
    0x1F,  
    0x10,  
    0x10,  
    0x1E,  
    0x10,  
    0x10,  
    0x10,  
    0x10,  
    0x10  
}
```

## FONT\_UPPER\_G

```
const uint8_t FONT_UPPER_G[8]  [static]
```

### Initial value:

```
= {  
    0x0E,  
    0x11,  
    0x10,  
    0x10,  
    0x17,  
    0x11,  
    0x11,  
    0x0E  
}
```

**FONT\_UPPER\_H**

```
const uint8_t FONT_UPPER_H[8]  [static]
```

**Initial value:**

```
= {  
    0x11,  
    0x11,  
    0x11,  
    0x1F,  
    0x1F,  
    0x11,  
    0x11,  
    0x11  
}
```

**FONT\_UPPER\_I**

```
const uint8_t FONT_UPPER_I[8]  [static]
```

**Initial value:**

```
= {  
    0x1F,  
    0x0A,  
    0x0A,  
    0x0A,  
    0x0A,  
    0x0A,  
    0x0A,  
    0x0A,  
    0x1F  
}
```

**FONT\_UPPER\_J**

```
const uint8_t FONT_UPPER_J[8]  [static]
```

**Initial value:**

```
= {  
    0x0E,  
    0x05,  
    0x05,  
    0x05,  
    0x05,  
    0x15,  
    0x15,  
    0x0E  
}
```

**FONT\_UPPER\_K**

```
const uint8_t FONT_UPPER_K[8]  [static]
```

**Initial value:**

```
= {  
    0x12,  
    0x14,  
    0x18,  
    0x1C,  
    0x1C,  
    0x14,  
    0x12,  
    0x11  
}
```

**FONT\_UPPER\_L**

```
const uint8_t FONT_UPPER_L[8]  [static]
```

**Initial value:**

```
= {  
    0x10,  
    0x10,  
    0x10,  
    0x10,  
    0x10,  
    0x10,  
    0x10,  
    0x1F,  
    0x1F  
}
```

**FONT\_UPPER\_M**

```
const uint8_t FONT_UPPER_M[8]  [static]
```

**Initial value:**

```
= {  
    0x11,  
    0x1B,  
    0x1B,  
    0x15,  
    0x15,  
    0x11,  
    0x11,  
    0x11  
}
```

**FONT\_UPPER\_N**

```
const uint8_t FONT_UPPER_N[8]  [static]
```

**Initial value:**

```
= {  
    0x11,  
    0x19,  
    0x19,  
    0x1D,  
    0x15,  
    0x13,  
    0x11,  
    0x11  
}
```

**FONT\_UPPER\_O**

```
const uint8_t FONT_UPPER_O[8]  [static]
```

**Initial value:**

```
= {  
    0x0E,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x0E  
}
```

**FONT\_UPPER\_P**

```
const uint8_t FONT_UPPER_P[8]  [static]
```

**Initial value:**

```
= {  
    0x1E,  
    0x11,  
    0x11,  
    0x1E,  
    0x10,  
    0x10,  
    0x10,  
    0x10  
}
```

**FONT\_UPPER\_Q**

```
const uint8_t FONT_UPPER_Q[8]  [static]
```

**Initial value:**

```
= {  
    0x0E,  
    0x11,  
    0x11,  
    0x11,  
    0x15,  
    0x19,  
    0x16,  
    0x0D  
}
```

**FONT\_UPPER\_R**

```
const uint8_t FONT_UPPER_R[8]  [static]
```

**Initial value:**

```
= {  
    0x1E,  
    0x11,  
    0x11,  
    0x1F,  
    0x18,  
    0x14,  
    0x12,  
    0x11  
}
```

**FONT\_UPPER\_S**

```
const uint8_t FONT_UPPER_S[8]  [static]
```

**Initial value:**

```
= {  
    0x0E,  
    0x11,  
    0x11,  
    0x0E,  
    0x01,  
    0x01,  
    0x11,  
    0x0E  
}
```

## FONT\_UPPER\_T

```
const uint8_t FONT_UPPER_T[8]  [static]
```

### Initial value:

```
= {  
    0x1F,  
    0x04,  
    0x04,  
    0x04,  
    0x04,  
    0x04,  
    0x04,  
    0x04  
}
```

## FONT\_UPPER\_U

```
const uint8_t FONT_UPPER_U[8]  [static]
```

### Initial value:

```
= {  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x0E  
}
```

## FONT\_UPPER\_V

```
const uint8_t FONT_UPPER_V[8]  [static]
```

### Initial value:

```
= {  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x0A,  
    0x0A,  
    0x04  
}
```

## FONT\_UPPER\_W

```
const uint8_t FONT_UPPER_W[8]  [static]
```

### Initial value:

```
= {  
    0x11,  
    0x11,  
    0x11,  
    0x15,  
    0x15,  
    0x1B,  
    0x11,  
    0x11  
}
```

**FONT\_UPPER\_X**

```
const uint8_t FONT_UPPER_X[8]  [static]
```

**Initial value:**

```
= {  
    0x11,  
    0x11,  
    0x0A,  
    0x0A,  
    0x04,  
    0x0A,  
    0x0A,  
    0x11  
}
```

**FONT\_UPPER\_Y**

```
const uint8_t FONT_UPPER_Y[8]  [static]
```

**Initial value:**

```
= {  
    0x11,  
    0x11,  
    0x11,  
    0x0A,  
    0x04,  
    0x04,  
    0x04,  
    0x04,  
    0x04  
}
```

**FONT\_UPPER\_Z**

```
const uint8_t FONT_UPPER_Z[8]  [static]
```

**Initial value:**

```
= {  
    0x1F,  
    0x01,  
    0x01,  
    0x02,  
    0x04,  
    0x08,  
    0x10,  
    0x1F  
}
```

**FONT\_LOWER\_A**

```
const uint8_t FONT_LOWER_A[8]  [static]
```

**Initial value:**

```
= {  
    0x00,  
    0x00,  
    0x0E,  
    0x01,  
    0x0F,  
    0x11,  
    0x0F,  
    0x00  
}
```

## FONT\_LOWER\_B

```
const uint8_t FONT_LOWER_B[8]  [static]
```

### Initial value:

```
= {  
    0x10,  
    0x10,  
    0x1E,  
    0x11,  
    0x11,  
    0x11,  
    0x1E,  
    0x00  
}
```

## FONT\_LOWER\_C

```
const uint8_t FONT_LOWER_C[8]  [static]
```

### Initial value:

```
= {  
    0x00,  
    0x00,  
    0x0E,  
    0x10,  
    0x10,  
    0x11,  
    0x0E,  
    0x00  
}
```

## FONT\_LOWER\_D

```
const uint8_t FONT_LOWER_D[8]  [static]
```

### Initial value:

```
= {  
    0x01,  
    0x01,  
    0x0F,  
    0x11,  
    0x11,  
    0x11,  
    0x0F,  
    0x00  
}
```

## FONT\_LOWER\_E

```
const uint8_t FONT_LOWER_E[8]  [static]
```

### Initial value:

```
= {  
    0x00,  
    0x00,  
    0x0E,  
    0x11,  
    0x1F,  
    0x10,  
    0x0E,  
    0x00  
}
```



**FONT\_LOWER\_F**

```
const uint8_t FONT_LOWER_F[8] [static]
```

**Initial value:**

```
= {  
    0x06,  
    0x09,  
    0x08,  
    0x1C,  
    0x08,  
    0x08,  
    0x08,  
    0x08,  
    0x00  
}
```

**FONT\_LOWER\_G**

```
const uint8_t FONT_LOWER_G[8] [static]
```

**Initial value:**

```
= {  
    0x00,  
    0x00,  
    0x0F,  
    0x11,  
    0x11,  
    0x0F,  
    0x01,  
    0x0E  
}
```

**FONT\_LOWER\_H**

```
const uint8_t FONT_LOWER_H[8] [static]
```

**Initial value:**

```
= {  
    0x10,  
    0x10,  
    0x1E,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x00  
}
```

**FONT\_LOWER\_I**

```
const uint8_t FONT_LOWER_I[8] [static]
```

**Initial value:**

```
= {  
    0x04,  
    0x00,  
    0x0C,  
    0x04,  
    0x04,  
    0x04,  
    0x0E,  
    0x00  
}
```

## FONT\_LOWER\_J

```
const uint8_t FONT_LOWER_J[8]  [static]
```

### Initial value:

```
= {  
    0x02,  
    0x00,  
    0x06,  
    0x02,  
    0x02,  
    0x12,  
    0x12,  
    0x0C  
}
```

## FONT\_LOWER\_K

```
const uint8_t FONT_LOWER_K[8]  [static]
```

### Initial value:

```
= {  
    0x10,  
    0x10,  
    0x12,  
    0x14,  
    0x18,  
    0x14,  
    0x12,  
    0x00  
}
```

## FONT\_LOWER\_L

```
const uint8_t FONT_LOWER_L[8]  [static]
```

### Initial value:

```
= {  
    0x0C,  
    0x04,  
    0x04,  
    0x04,  
    0x04,  
    0x04,  
    0x0E,  
    0x00  
}
```

## FONT\_LOWER\_M

```
const uint8_t FONT_LOWER_M[8]  [static]
```

### Initial value:

```
= {  
    0x00,  
    0x00,  
    0x1A,  
    0x15,  
    0x15,  
    0x11,  
    0x11,  
    0x00  
}
```

**FONT\_LOWER\_N**

```
const uint8_t FONT_LOWER_N[8]  [static]
```

**Initial value:**

```
= {  
    0x00,  
    0x00,  
    0x1E,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x00  
}
```

**FONT\_LOWER\_O**

```
const uint8_t FONT_LOWER_O[8]  [static]
```

**Initial value:**

```
= {  
    0x00,  
    0x00,  
    0x0E,  
    0x11,  
    0x11,  
    0x11,  
    0x0E,  
    0x00  
}
```

**FONT\_LOWER\_P**

```
const uint8_t FONT_LOWER_P[8]  [static]
```

**Initial value:**

```
= {  
    0x00,  
    0x00,  
    0x1E,  
    0x11,  
    0x11,  
    0x1E,  
    0x10,  
    0x10  
}
```

**FONT\_LOWER\_Q**

```
const uint8_t FONT_LOWER_Q[8]  [static]
```

**Initial value:**

```
= {  
    0x00,  
    0x00,  
    0x0F,  
    0x11,  
    0x11,  
    0x0F,  
    0x01,  
    0x01  
}
```

## FONT\_LOWER\_R

```
const uint8_t FONT_LOWER_R[8]  [static]
```

### Initial value:

```
= {  
    0x00,  
    0x00,  
    0x1A,  
    0x15,  
    0x10,  
    0x10,  
    0x10,  
    0x00  
}
```

## FONT\_LOWER\_S

```
const uint8_t FONT_LOWER_S[8]  [static]
```

### Initial value:

```
= {  
    0x00,  
    0x00,  
    0x0E,  
    0x10,  
    0x0E,  
    0x01,  
    0x0E,  
    0x00  
}
```

## FONT\_LOWER\_T

```
const uint8_t FONT_LOWER_T[8]  [static]
```

### Initial value:

```
= {  
    0x04,  
    0x04,  
    0x0E,  
    0x04,  
    0x04,  
    0x04,  
    0x02,  
    0x00  
}
```

## FONT\_LOWER\_U

```
const uint8_t FONT_LOWER_U[8]  [static]
```

### Initial value:

```
= {  
    0x00,  
    0x00,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x0F,  
    0x00  
}
```

**FONT\_LOWER\_V**

```
const uint8_t FONT_LOWER_V[8]  [static]
```

**Initial value:**

```
= {  
    0x00,  
    0x00,  
    0x11,  
    0x11,  
    0x11,  
    0x11,  
    0x0A,  
    0x04,  
    0x00  
}
```

**FONT\_LOWER\_W**

```
const uint8_t FONT_LOWER_W[8]  [static]
```

**Initial value:**

```
= {  
    0x00,  
    0x00,  
    0x11,  
    0x11,  
    0x15,  
    0x15,  
    0x0A,  
    0x00  
}
```

**FONT\_LOWER\_X**

```
const uint8_t FONT_LOWER_X[8]  [static]
```

**Initial value:**

```
= {  
    0x00,  
    0x00,  
    0x11,  
    0x0A,  
    0x04,  
    0x0A,  
    0x11,  
    0x00  
}
```

**FONT\_LOWER\_Y**

```
const uint8_t FONT_LOWER_Y[8]  [static]
```

**Initial value:**

```
= {  
    0x00,  
    0x00,  
    0x11,  
    0x11,  
    0x0F,  
    0x01,  
    0x0E,  
    0x00  
}
```

## FONT\_LOWER\_Z

```
const uint8_t FONT_LOWER_Z[8]  [static]
```

### Initial value:

```
= {
    0x00,
    0x00,
    0x1F,
    0x02,
    0x04,
    0x08,
    0x1F,
    0x00
}
```

## 4.1.4 QRS Detector

Module for analyzing ECG data to determine heart rate.

Collaboration diagram for QRS Detector:



## Files

- file [QRS.c](#)  
*Source code for QRS detection module.*
- file [QRS.h](#)  
*Header file for QRS detection module.*

## Macros

- **#define QRS\_NUM\_FID\_MARKS** 40
- **#define FLOAT\_COMPARE\_TOLERANCE** (float32\_t)(1E-5f)
- **#define IS\_GREATER**(X, Y) (bool) ((X - Y) > FLOAT\_COMPARE\_TOLERANCE)
- **#define IS\_PEAK**(X\_MINUS\_1, X, X\_PLUS\_1) (bool) (IS\_GREATER(X, X\_MINUS\_1) && IS\_GREATER(X, X\_PLUS\_1))
- **#define QRS\_SAMP\_FREQ** ((uint32\_t) 200)
- **#define QRS\_SAMP\_PERIOD\_SEC** ((float32\_t) 0.005f)
- **#define QRS\_NUM\_SAMP** ((uint16\_t) (1 << 11))

## Variables

- struct {
  - bool **isCalibrated**
  - float32\_t **signalLevel**  
*estimated signal level*
  - float32\_t **noiseLevel**  
*estimated noise level*
  - float32\_t **threshold**  
*amplitude threshold*
  - uint16\_t **fidMarkArray** [QRS\_NUM\_FID\_MARKS]
  - float32\_t **utilityBuffer1** [QRS\_NUM\_FID\_MARKS]  
*array to hold fidMark indices*
  - float32\_t **utilityBuffer2** [QRS\_NUM\_FID\_MARKS]
- } **Detector** = { false, 0.0f, 0.0f, 0.0f, { 0 }, { 0 }, { 0 } }

## Digital Filters

- enum {
  - NUM\_STAGES\_BANDPASS** = 4 , **NUM\_COEFF\_HIGHPASS** = NUM\_STAGES\_BANDPASS \* 5 , **STATE\_↵**
  - \_BUFF\_SIZE\_BANDPASS** = NUM\_STAGES\_BANDPASS \* 4 , **NUM\_COEFF\_DERFILT** = 5 ,
  - BLOCK\_SIZE\_DERFILT** = (1 << 8) , **STATE\_BUFF\_SIZE\_DERFILT** = NUM\_COEFF\_DERFILT + BLOCK\_↵
  - \_SIZE\_DERFILT - 1** , **NUM\_COEFF\_MOVAVG** = 10 , **BLOCK\_SIZE\_MOVAVG** = BLOCK\_SIZE\_DERFILT ,
  - STATE\_BUFF\_SIZE\_MOVAVG** = NUM\_COEFF\_MOVAVG + BLOCK\_SIZE\_MOVAVG - 1 }
- typedef arm\_biquad\_casd\_df1\_inst\_f32 **IIR\_Filt\_t**
- typedef arm\_fir\_instance\_f32 **FIR\_Filt\_t**
- static const float32\_t **COEFF\_BANDPASS** [NUM\_COEFF\_HIGHPASS]
- static const float32\_t **COEFF\_DERFILT** [NUM\_COEFF\_DERFILT]
- static const float32\_t **COEFF\_MOVAVG** [NUM\_COEFF\_MOVAVG]
- static float32\_t **stateBuffer\_bandPass** [STATE\_BUFF\_SIZE\_BANDPASS] = { 0 }
- static const IIR\_Filt\_t **bandpassFiltStruct** = { NUM\_STAGES\_BANDPASS, stateBuffer\_bandPass, COEFF\_↵
- \_BANDPASS** }
- static const IIR\_Filt\_t \*const **bandpassFilter** = &bandpassFiltStruct
- static float32\_t **stateBuffer\_DerFilt** [STATE\_BUFF\_SIZE\_DERFILT] = { 0 }
- static const FIR\_Filt\_t **derivativeFiltStruct** = { NUM\_COEFF\_DERFILT, stateBuffer\_DerFilt, COEFF\_↵
- DERFILT** }
- static const FIR\_Filt\_t \*const **derivativeFilter** = &derivativeFiltStruct
- static float32\_t **stateBuffer\_MovingAvg** [STATE\_BUFF\_SIZE\_MOVAVG] = { 0 }
- static const FIR\_Filt\_t **movingAvgFiltStruct** = { NUM\_COEFF\_MOVAVG, stateBuffer\_MovingAvg, COEFF\_↵
- \_MOVAVG** }
- static const FIR\_Filt\_t \*const **movingAverageFilter** = &movingAvgFiltStruct

## Implementation

- static uint8\_t **QRS\_findFiducialMarks** (const float32\_t yn[], uint16\_t fidMarkArray[])  
*Mark local peaks in the input signal  $y$  as potential candidates for QRS complexes (AKA "fiducial marks").*
- static void **QRS\_initLevels** (const float32\_t yn[], float32\_t \*sigLvlPtr, float32\_t \*noiseLvlPtr)  
*Initialize the signal and noise levels for the QRS detector using the initial block of input signal data.*
- static float32\_t **QRS\_updateLevel** (const float32\_t peakAmplitude, float32\_t level)  
*Update the signal level (if a fiducial mark is a confirmed peak) or the noise level (if a fiducial mark is rejected).*
- static float32\_t **QRS\_updateThreshold** (const float32\_t signalLevel, const float32\_t noiseLevel)  
*Update the amplitude threshold used to identify peaks based on the signal and noise levels.*

## Interface Functions

- void `QRS_Init` (void)  
*Initialize the QRS detector.*
- void `QRS_Preprocess` (const float32\_t xn[], float32\_t yn[])  
*Preprocess the ECG data to remove noise and/or exaggerate the signal characteristic(s) of interest.*
- float32\_t `QRS_applyDecisionRules` (const float32\_t yn[])  
*Calculate the average heart rate (HR) using predetermined decision rules.*
- float32\_t `QRS_runDetection` (const float32\_t xn[], float32\_t yn[])  
*Run the full algorithm (preprocessing and decision rules) on the inputted ECG data.*

### 4.1.4.1 Detailed Description

Module for analyzing ECG data to determine heart rate.

### 4.1.4.2 Function Documentation

#### QRS\_findFiducialMarks()

```
static uint8_t QRS_findFiducialMarks (
    const float32_t yn[],
    uint16_t fidMarkArray[] ) [static]
```

Mark local peaks in the input signal  $y$  as potential candidates for QRS complexes (AKA "fiducial marks").

#### Parameters

in	<i>yn</i>	Array containing the preprocessed ECG signal $y[n]$
in	<i>fidMarkArray</i>	Array to place the fiducial mark's sample indices into.
out	<i>numMarks</i>	Number of identified fiducial marks

#### Postcondition

*fidMarkArray* will hold the values of the fiducial marks.

The fiducial marks must be spaced apart by at least 200 [ms] (40 samples @  $f_s = 200$  [Hz]). If a peak is found within this range, the one with the largest amplitude is taken to be the correct peak and the other is ignored.

#### QRS\_initLevels()

```
static void QRS_initLevels (
    const float32_t yn[],
    float32_t * sigLvlPtr,
    float32_t * noiseLvlPtr ) [static]
```

Initialize the signal and noise levels for the QRS detector using the initial block of input signal data.



**Parameters**

in	<i>yn</i>	Array containing the preprocessed ECG signal $y[n]$
in	<i>sigLvIPtr</i>	Pointer to variable holding the signal level value.
in	<i>noiseLvIPtr</i>	Pointer to variable holding the noise level value.

**Postcondition**

The signal and noise levels are initialized.

**QRS\_updateLevel()**

```
static float32_t QRS_updateLevel (
    const float32_t peakAmplitude,
    float32_t level ) [static]
```

Update the signal level (if a fiducial mark is a confirmed peak) or the noise level (if a fiducial mark is rejected).

**Parameters**

in	<i>peakAmplitude</i>	Amplitude of the fiducial mark in signal $y[n]$
in	<i>level</i>	The current value of the signal level or noise level
out	<i>newLevel</i>	The updated value of the signal level or noise level

$$signalLevel_1 = f(peakAmplitude, signalLevel_0) = \frac{1}{8}peakAmplitude + \frac{7}{8}signalLevel_0$$

$$noiseLevel_1 = f(peakAmplitude, noiseLevel_0) = \frac{1}{8}peakAmplitude + \frac{7}{8}noiseLevel_0$$

**QRS\_updateThreshold()**

```
static float32_t QRS_updateThreshold (
    const float32_t signalLevel,
    const float32_t noiseLevel ) [static]
```

Update the amplitude threshold used to identify peaks based on the signal and noise levels.

**Parameters**

in	<i>signalLevel</i>	Current signal level.
in	<i>noiseLevel</i>	Current noise level.
out	<i>threshold</i>	New threshold to use for next comparison.

**See also**

[QRS\\_updateLevel\(\)](#), [QRS\\_applyDecisionRules](#)

$$threshold = f(signalLevel, noiseLevel) = noiseLevel + 0.25(signalLevel - noiseLevel)$$

**QRS\_Init()**

```
void QRS_Init (
    void )
```

Initialize the QRS detector.

**Note**

This function isn't necessary anymore, but I'm keeping it here just in case.

This function originally initialized the filter `structs` but now does nothing since those have been made `const` and their initialization functions have been removed entirely.

**QRS\_Preprocess()**

```
void QRS_Preprocess (
    const float32_t xn[],
    float32_t yn[] )
```

Preprocess the ECG data to remove noise and/or exaggerate the signal characteristic(s) of interest.

**Precondition**

Fill input buffer `xn` with raw or lightly preprocessed ECG data.

**Parameters**

in	<i>xn</i>	Array of raw ECG signal values.
in	<i>yn</i>	Array used to store preprocessed ECG signal values.

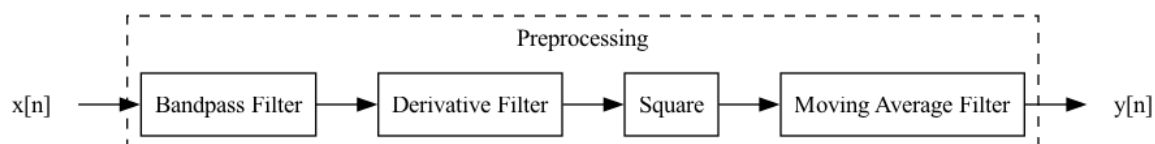
**Postcondition**

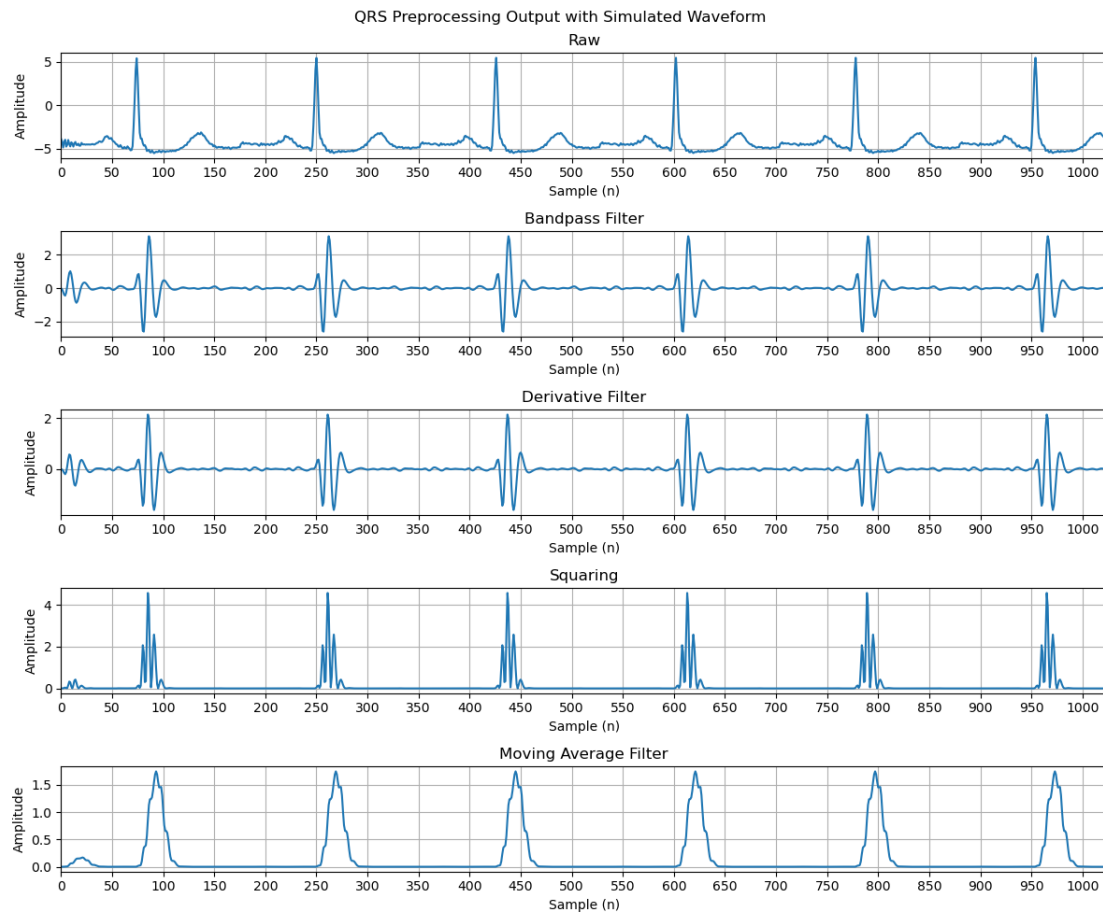
The preprocessed signal data  $y[n]$  is stored in `yn` and is ready to be analyzed to calculate the heart rate in [bpm].

**See also**

[QRS\\_applyDecisionRules\(\)](#)

This function uses the same overall preprocessing pipeline as the original Pan-Tompkins algorithm, but the high-pass and low-pass filters have been replaced with ones generated using Scipy.





**Figure 3** Output of each preprocessing step.

#### Note

The FIR filters are applied in blocks to decrease the amount of memory needed for their state buffers.

#### QRS\_applyDecisionRules()

```
float32_t QRS_applyDecisionRules (
    const float32_t yn[] )
```

Calculate the average heart rate (HR) using predetermined decision rules.

#### Precondition

Preprocess the raw ECG data.

#### Parameters

in	<i>yn</i>	Array of preprocessed ECG signal values.
out	<i>heartRate</i>	Average heart rate in [bpm].

**Postcondition**

Certain information (signal/noise levels, thresholds, etc.) is retained between calls and used to improve further detection.

**Warning**

The current implementation only processes one block at a time and discards the data immediately after, so peaks that are cut off between one block and another might not be being counted.

**See also**

[QRS\\_Preprocess\(\)](#)

**QRS\_runDetection()**

```
float32_t QRS_runDetection (
    const float32_t xn[],
    float32_t yn[] )
```

Run the full algorithm (preprocessing and decision rules) on the inputted ECG data.

This function simply combines the preprocessing and decision rules functions into a single function.

**Parameters**

in	<i>xn</i>	Array of raw ECG signal values.
in	<i>yn</i>	Array used to hold preprocessed ECG signal values.
out	<i>heartRate</i>	Average heart rate in [bpm].

**Postcondition**

*yn* will contain the preprocessed data.

Certain information (signal/noise levels, thresholds, etc.) is retained between calls.

**See also**

[QRS\\_Preprocess\(\)](#), [QRS\\_applyDecisionRules\(\)](#)

**4.1.4.3 Variable Documentation****COEFF\_BANDPASS**

```
const float32_t COEFF_BANDPASS[NUM_COEFF_HIGHPASS] [static]
```

**Initial value:**

```
= {
    0.002937758108600974f, 0.005875516217201948f, 0.002937758108600974f,
    1.0485996007919312f, -0.2961403429508209f,
```

```

1.0f, 2.0f, 1.0f,
1.3876197338104248f, -0.492422878742218f,

1.0f, -2.0f, 1.0f,
1.3209134340286255f, -0.6327387690544128f,

1.0f, -2.0f, 1.0f,
1.6299355030059814f, -0.7530401945114136f,
}

```

### COEFF\_DERFILT

```
const float32_t COEFF_DERFILT[NUM_COEFF_DERFILT] [static]
```

#### Initial value:

```

= {
    -0.125f, -0.25f, 0.0f, 0.25f, 0.125f
}

```

### COEFF\_MOVAVG

```
const float32_t COEFF_MOVAVG[NUM_COEFF_MOVAVG] [static]
```

#### Initial value:

```

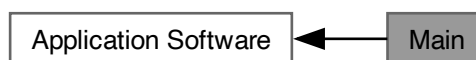
= {
    0.10000000149011612f, 0.10000000149011612f, 0.10000000149011612f,
    0.10000000149011612f, 0.10000000149011612f, 0.10000000149011612f,
    0.10000000149011612f, 0.10000000149011612f, 0.10000000149011612f, 0.10000000149011612f
}

```

## 4.1.5 Main

Main program file.

Collaboration diagram for Main:



### Files

- file [main.c](#)  
Main program file.

## Enumerations

- enum `ISR_VECTOR_NUMS` { `DAQ_VECTOR_NUM` = `INT_ADC0SS3` , `PROC_VECTOR_NUM` = `INT_CAN0` , `LCD_VECTOR_NUM` = `INT_TIMER1A` }
- enum `FIFO_INFO` {  
`DAQ_FIFO_CAP` = 3 , `DAQ_ARRAY_LEN` = `DAQ_FIFO_CAP` + 1 , `QRS_FIFO_CAP` = `QRS_NUM_SAMP` ,  
`QRS_ARRAY_LEN` = `QRS_FIFO_CAP` + 1 ,  
`LCD_FIFO_1_CAP` = `DAQ_FIFO_CAP` , `LCD_ARRAY_1_LEN` = `LCD_FIFO_1_CAP` + 1 , `LCD_FIFO_2_CAP`  
= 1 , `LCD_ARRAY_2_LEN` = `LCD_FIFO_2_CAP` + 1 }
- enum `LCD_INFO` {  
`LCD_TOP_LINE` = (`LCD_Y_MAX` - 24) , `LCD_WAVE_NUM_Y` = `LCD_TOP_LINE` , `LCD_WAVE_X_OFFSET`  
= 0 , `LCD_WAVE_Y_MIN` = (0 + `LCD_WAVE_X_OFFSET`) ,  
`LCD_WAVE_Y_MAX` = (`LCD_WAVE_NUM_Y` + `LCD_WAVE_X_OFFSET`) , `LCD_TEXT_LINE_NUM` = 28 ,  
`LCD_TEXT_COL_NUM` = 24 }

## Functions

- static void `DAQ_Handler` (void)  
*ISR for the data acquisition system.*
- static void `Processing_Handler` (void)  
*ISR for intermediate processing of the input data.*
- static void `LCD_Handler` (void)  
*ISR for plotting the waveform and outputting the heart rate to the LCD.*
- int `main` (void)  
*Main function for the project.*

## Variables

- static volatile `Fifo_t` `DAQ_Fifo` = 0
- static volatile `uint32_t` `DAQ_fifoBuffer` [`DAQ_ARRAY_LEN`] = { 0 }
- static volatile `Fifo_t` `QRS_Fifo` = 0
- static volatile `uint32_t` `QRS_fifoBuffer` [`QRS_ARRAY_LEN`] = { 0 }
- static volatile `Fifo_t` `LCD_Fifo1` = 0
- static volatile `uint32_t` `LCD_fifoBuffer1` [`LCD_ARRAY_1_LEN`] = { 0 }
- static volatile `Fifo_t` `LCD_Fifo2` = 0
- static volatile `uint32_t` `LCD_fifoBuffer2` [`LCD_ARRAY_2_LEN`] = { 0 }
- static volatile `bool` `qrsBufferIsFull` = false  
*flag for QRS detection to start*
- static volatile `bool` `heartRateIsReady` = false  
*flag for LCD to output heart rate*
- static `float32_t` `QRS_processingBuffer` [`QRS_ARRAY_LEN`] = { 0 }
- static `uint16_t` `LCD_prevSampleBuffer` [`LCD_X_MAX`] = { 0 }

### 4.1.5.1 Detailed Description

Main program file.

### 4.1.5.2 Enumeration Type Documentation

#### ISR\_VECTOR\_NUMS

enum `ISR_VECTOR_NUMS`

## Enumerator

DAQ_VECTOR_NUM	vector number for the <a href="#">DAQ_Handler()</a>
PROC_VECTOR_NUM	vector number for the <a href="#">Processing_Handler()</a>
LCD_VECTOR_NUM	vector number for the <a href="#">LCD_Handler()</a>

**FIFO\_INFO**enum [FIFO\\_INFO](#)

## Enumerator

DAQ_FIFO_CAP	capacity of DAQ's FIFO buffer
DAQ_ARRAY_LEN	actual size of underlying array
QRS_FIFO_CAP	capacity of QRS detector's FIFO buffer
QRS_ARRAY_LEN	actual size of underlying array
LCD_FIFO_1_CAP	capacity of LCD's waveform FIFO buffer
LCD_ARRAY_1_LEN	actual size of underlying array
LCD_FIFO_2_CAP	capacity of LCD's heart rate FIFO buffer
LCD_ARRAY_2_LEN	actual size of underlying array

**LCD\_INFO**enum [LCD\\_INFO](#)

## Enumerator

LCD_TOP_LINE	separates waveform from text
LCD_WAVE_NUM_Y	num. of y-vals available for plotting waveform
LCD_WAVE_X_OFFSET	waveform's offset from X axis
LCD_WAVE_Y_MIN	waveform's min y-value
LCD_WAVE_Y_MAX	waveform's max y-value
LCD_TEXT_LINE_NUM	line num. of text
LCD_TEXT_COL_NUM	starting col. num. for heart rate

**4.1.5.3 Function Documentation****DAQ\_Handler()**

```
static void DAQ_Handler (
    void ) [static]
```

ISR for the data acquisition system.

This ISR has a priority level of 1, is triggered when the ADC has finished capturing a sample, and also triggers the intermediate processing handler. It reads the 12-bit ADC output, converts it from an integer to a raw voltage sample, and sends it to the processing ISR via the DAQ\_Fifo.

**Precondition**

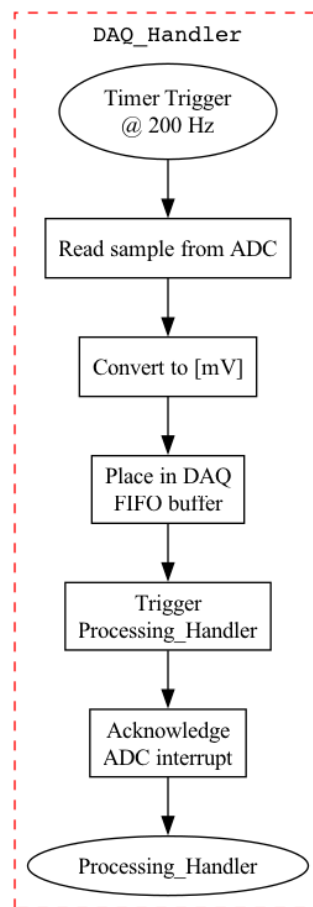
Initialize the DAQ module.

**Postcondition**

The converted sample is placed in the DAQ FIFO, and the processing ISR is triggered.

**See also**

[DAQ\\_Init\(\)](#), [Processing\\_Handler\(\)](#)



**Figure 4 Flowchart for the DAQ handler.**

**Processing\_Handler()**

```
static void Processing_Handler (
    void ) [static]
```

ISR for intermediate processing of the input data.

This ISR has a priority level of 1, is triggered by the DAQ ISR, and triggers the LCD handler. It removes baseline drift and power line interference (PLI) from a sample, and then moves it to the QRS\_Fifo and the LCD\_Fifo. It also notifies the superloop in [main\(\)](#) when the QRS buffer is full.



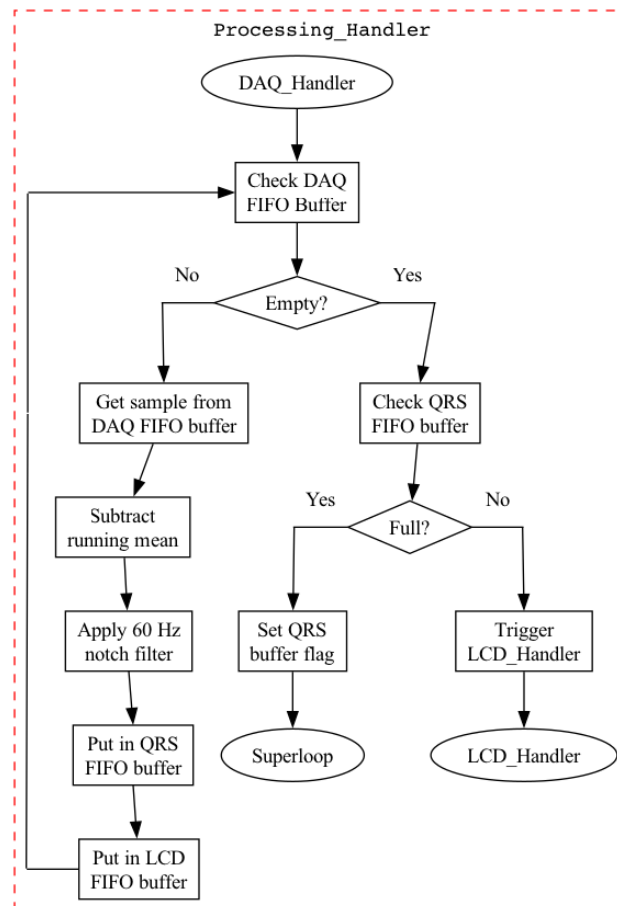
**Postcondition**

The converted sample is placed in the LCD FIFO, and the LCD ISR is triggered.

The converted sample is placed in the QRS FIFO, and the flag is set.

**See also**

[DAQ\\_Handler\(\)](#), [main\(\)](#), [LCD\\_Handler\(\)](#)



**Figure 5 Flowchart for the processing handler.**

**LCD\_Handler()**

```
static void LCD_Handler (
    void ) [static]
```

ISR for plotting the waveform and outputting the heart rate to the LCD.

This ISR has a priority level of 1 and is triggered by the Processing ISR. It applies a 0.5-40 [Hz] bandpass filter to the sample and plots it. It also outputs the heart rate.

**Precondition**

Initialize the LCD module.

**Postcondition**

The bandpass-filtered sample is plotted to the LCD.

The heart rate is updated after each block is analyzed.

**See also**

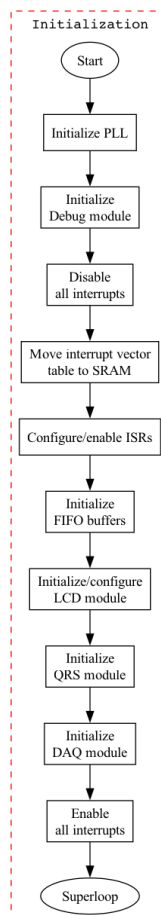
[LCD\\_Init\(\)](#), [Processing\\_Handler\(\)](#), [main\(\)](#)

**main()**

```
int main (
    void )
```

Main function for the project.

Moves the interrupt vector table to RAM; configures and enables the ISRs; initializes all modules and static variables; and performs QRS detection once the buffer has been filled.



**Figure 6 Flowchart for the initialization phase.**

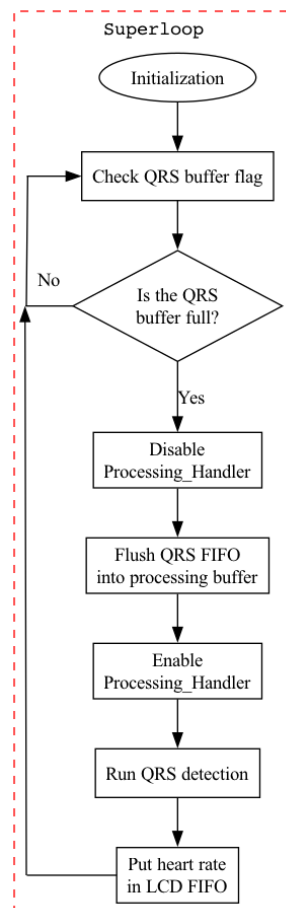
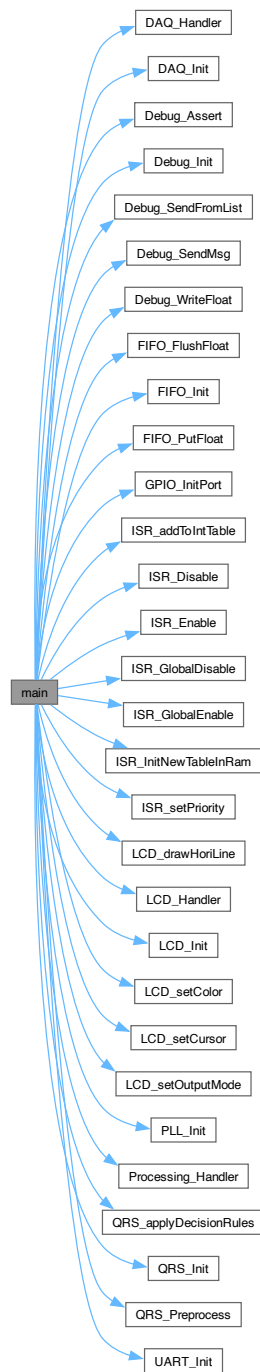


Figure 7 Flowchart for the superloop.

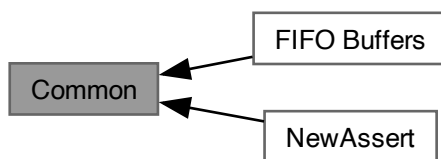
Here is the call graph for this function:



## 4.2 Common

Modules that are used by multiple layers and/or don't fit into any one layer.

Collaboration diagram for Common:



## Modules

- [FIFO Buffers](#)  
*Module for using the "first-in first-out (FIFO) buffer" data structure.*
- [NewAssert](#)  
*Module for using a custom `assert` implementation.*

## Files

- file [NewAssert.c](#)  
*Source code for custom `assert` implementation.*
- file [NewAssert.h](#)  
*Header file for custom `assert` implementation.*

## Functions

- void [Assert](#) (bool condition)  
*Custom `assert` implementation that is more lightweight than the one from `newlib`.*

### 4.2.1 Detailed Description

Modules that are used by multiple layers and/or don't fit into any one layer.

### 4.2.2 Function Documentation

#### **Assert()**

```
void Assert (
    bool condition )
```

Custom `assert` implementation that is more lightweight than the one from `newlib`.

#### Parameters

<code>in</code>	<code>condition</code>	Conditional to test.
-----------------	------------------------	----------------------

#### Postcondition

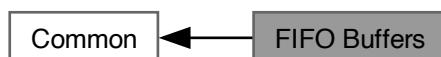
If `condition == true`, the function simply returns.

If `condition == false`, a breakpoint is initiated.

### 4.2.3 FIFO Buffers

Module for using the "first-in first-out (FIFO) buffer" data structure.

Collaboration diagram for FIFO Buffers:



#### Files

- file [Fifo.c](#)  
*Source code for FIFO buffer module.*
- file [Fifo.h](#)  
*Header file for FIFO buffer implementation.*

#### Data Structures

- struct [Fifo\\_t](#)

#### Macros

- `#define FIFO_POOL_SIZE 5`

#### Functions

- `Fifo_t` [FIFO\\_Init](#) (volatile uint32\_t buffer[], const uint32\_t N)  
*Initialize a FIFO buffer of length N.*
- void [FIFO\\_Reset](#) (volatile Fifo\_t fifo)  
*Reset the FIFO buffer.*

## Variables

- static `FifoStruct_t fifoPool` [FIFO\_POOL\_SIZE] = { 0 }  
*pre-allocated pool*
- static `uint8_t numFreeFifos` = FIFO\_POOL\_SIZE

## Basic Operations

- void `FIFO_Put` (volatile `Fifo_t` fifo, const `uint32_t` val)  
*Add a value to the end of the buffer.*
- `uint32_t` `FIFO_Get` (volatile `Fifo_t` fifo)  
*Remove the first value of the buffer.*
- void `FIFO_Flush` (volatile `Fifo_t` fifo, `uint32_t` outputBuffer[])  
*Empty the FIFO buffer's contents into an array.*
- void `FIFO_PutFloat` (volatile `Fifo_t` fifo, const `float` val)  
*Add a floating-point value to the end of the buffer.*
- `float` `FIFO_GetFloat` (volatile `Fifo_t` fifo)  
*Remove the first value of the buffer, and cast it to `float`.*
- void `FIFO_FlushFloat` (volatile `Fifo_t` fifo, `float` outputBuffer[])  
*Empty the FIFO buffer into an array of floating-point values.*

## Peeking

- `uint32_t` `FIFO_PeekOne` (volatile `Fifo_t` fifo)  
*See the first element in the FIFO without removing it.*
- void `FIFO_PeekAll` (volatile `Fifo_t` fifo, `uint32_t` outputBuffer[])  
*See the FIFO buffer's contents without removing them.*

## Status Checks

- bool `FIFO_isFull` (volatile `Fifo_t` fifo)  
*Check if the FIFO buffer is full.*
- bool `FIFO_isEmpty` (volatile `Fifo_t` fifo)  
*Check if the FIFO buffer is empty.*
- `uint32_t` `FIFO_getCurrSize` (volatile `Fifo_t` fifo)  
*Get the current size of the FIFO buffer.*

### 4.2.3.1 Detailed Description

Module for using the "first-in first-out (FIFO) buffer" data structure.

### 4.2.3.2 Function Documentation

#### FIFO\_Init()

```
Fifo_t FIFO_Init (
    volatile uint32_t buffer[],
    const uint32_t N )
```

Initialize a FIFO buffer of length N.

**Parameters**

in	<i>buffer</i>	Array of size <i>N</i> to be used as FIFO buffer
in	<i>N</i>	Length of <i>buffer</i> . Usable length is $N - 1$ .
out	<i>fifo</i>	pointer to the FIFO buffer

**Postcondition**

The number of available FIFO buffers is reduced by 1.

TODO: Add details

**FIFO\_Reset()**

```
void FIFO_Reset (
    volatile Fifo_t fifo )
```

Reset the FIFO buffer.

**Parameters**

in	<i>fifo</i>	Pointer to FIFO buffer.
----	-------------	-------------------------

**Postcondition**

The FIFO is now considered empty. The underlying buffer's contents are not affected.

**FIFO\_Put()**

```
void FIFO_Put (
    volatile Fifo_t fifo,
    const uint32_t val )
```

Add a value to the end of the buffer.

**Parameters**

in	<i>fifo</i>	Pointer to FIFO object
in	<i>val</i>	Value to add to the buffer.

**Postcondition**

If the FIFO is not full, *val* is placed in the buffer. If the FIFO is full, nothing happens.

**See also**

[FIFO\\_PutFloat\(\)](#)



**FIFO\_Get()**

```
uint32_t FIFO_Get (
    volatile Fifo_t fifo )
```

Remove the first value of the buffer.

**Parameters**

in	<i>fifo</i>	Pointer to FIFO object
out	<i>val</i>	First sample in the FIFO.

**Postcondition**

If the FIFO is not empty, the next value is returned. If the FIFO is empty, 0 is returned.

**See also**

[FIFO\\_GetFloat\(\)](#)

**FIFO\_Flush()**

```
void FIFO_Flush (
    volatile Fifo_t fifo,
    uint32_t outputBuffer[] )
```

Empty the FIFO buffer's contents into an array.

**Parameters**

in	<i>fifo</i>	Pointer to source FIFO buffer.
in	<i>outputBuffer</i>	Array to output values to. Should be the same length as the FIFO buffer.

**Postcondition**

The FIFO buffer's contents are transferred to the output buffer.

**See also**

[FIFO\\_FlushFloat\(\)](#)

**FIFO\_PutFloat()**

```
void FIFO_PutFloat (
    volatile Fifo_t fifo,
    const float val )
```

Add a floating-point value to the end of the buffer.

**Parameters**

in	<i>fifo</i>	Pointer to FIFO object
in	<i>val</i>	Value to add to the buffer.

**Postcondition**

If the FIFO is not full, `val` is placed in the buffer. If the FIFO is full, nothing happens.

**Note**

This was added to avoid needing to type-pun floating-point values.

```
// type-punning example
float num = 4.252603;
FIFO_Put(fifo, *((uint32_t *) &num));
FIFO_PutFloat(fifo, num); // same thing, but cleaner
```

**See also**

[FIFO\\_Put\(\)](#)

**Remarks**

To properly use floating-point values, type-punning is necessary.

**FIFO\_GetFloat()**

```
float FIFO_GetFloat (
    volatile Fifo_t fifo )
```

Remove the first value of the buffer, and cast it to `float`.

**Parameters**

in	<i>fifo</i>	Pointer to FIFO object
out	<i>val</i>	First sample in the FIFO.

**Postcondition**

If the FIFO is not empty, the next value is returned. If the FIFO is empty, 0 is returned.

**Note**

This was added to avoid needing to type-pun floating-point values.

```
// type-punning example
float num;
*((uint32_t *) &num) = FIFO_Get(fifo);
num = FIFO_GetFloat(fifo);
```

See also

[FIFO\\_Get\(\)](#)

#### Remarks

To properly use floating-point values, type-punning is necessary.

### FIFO\_FlushFloat()

```
void FIFO_FlushFloat (
    volatile Fifo_t fifo,
    float outputBuffer[] )
```

Empty the FIFO buffer into an array of floating-point values.

#### Parameters

in	<i>fifo</i>	Pointer to source FIFO buffer.
in	<i>outputBuffer</i>	Array to output values to. Should be the same length as the FIFO buffer.

#### Postcondition

The FIFO buffer's contents are transferred to the output buffer.

#### Note

This was added to avoid needing to type-pun floating-point values.

```
// type-punning example
FIFO_Flush(fifo, (uint32_t *) outputBuffer);
FIFO_FlushFloat(fifo, outputBuffer); // same thing, but cleaner
```

See also

[FIFO\\_Flush\(\)](#)

### FIFO\_PeekOne()

```
uint32_t FIFO_PeekOne (
    volatile Fifo_t fifo )
```

See the first element in the FIFO without removing it.

#### Parameters

in	<i>fifo</i>	Pointer to FIFO object
out	<i>val</i>	First sample in the FIFO.

**FIFO\_PeekAll()**

```
void FIFO_PeekAll (
    volatile Fifo_t fifo,
    uint32_t outputBuffer[] )
```

See the FIFO buffer's contents without removing them.

**Parameters**

in	<i>fifo</i>	Pointer to source FIFO buffer.
in	<i>outputBuffer</i>	Array to output values to. Should be the same length as the FIFO buffer.

**Postcondition**

The FIFO buffer's contents are copied to the output buffer.

**FIFO\_isFull()**

```
bool FIFO_isFull (
    volatile Fifo_t fifo )
```

Check if the FIFO buffer is full.

**Parameters**

in	<i>fifo</i>	Pointer to the FIFO buffer.
out	<i>true</i>	The FIFO buffer is full.
out	<i>false</i>	The FIFO buffer is not full.

**FIFO\_isEmpty()**

```
bool FIFO_isEmpty (
    volatile Fifo_t fifo )
```

Check if the FIFO buffer is empty.

**Parameters**

in	<i>fifo</i>	Pointer to the FIFO buffer.
out	<i>true</i>	The FIFO buffer is empty.
out	<i>false</i>	The FIFO buffer is not empty.

**FIFO\_getCurrSize()**

```
uint32_t FIFO_getCurrSize (
    volatile Fifo_t fifo )
```

Get the current size of the FIFO buffer.

#### Parameters

in	<i>fifo</i>	Pointer to the FIFO buffer.
out	<i>size</i>	Current number of values in the FIFO buffer.

#### 4.2.4 NewAssert

Module for using a custom `assert` implementation.

Collaboration diagram for NewAssert:

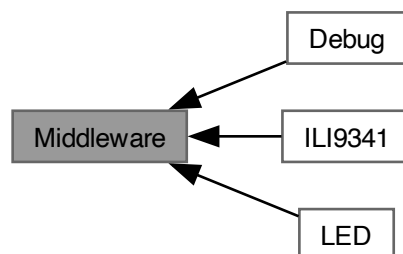


Module for using a custom `assert` implementation.

### 4.3 Middleware

High-level device driver modules.

Collaboration diagram for Middleware:



## Modules

- [Debug](#)  
*Module for debugging functions, including serial output and assertions.*
- [ILI9341](#)  
*Functions for interfacing an ILI9341-based 240RGBx320 LCD via [Serial Peripheral Interface \(SPI\)](#).*
- [LED](#)  
*Functions for driving light-emitting diodes (LEDs) via [General-Purpose Input/Output \(GPIO\)](#).*

### 4.3.1 Detailed Description

High-level device driver modules.

These modules contain functions for interfacing with external devices/peripherals using low-level drivers.

### 4.3.2 Debug

Module for debugging functions, including serial output and assertions.

Collaboration diagram for Debug:



## Files

- file [Debug.h](#)  
*Functions to output debugging information to a serial port via UART.*

## Serial Output

- enum **Msg\_t** { **DEBUG\_DAQ\_INIT** , **DEBUG\_QRS\_INIT** , **DEBUG\_LCD\_INIT** , **DEBUG\_QRS\_START** }
- void [Debug\\_SendMsg](#) (void \*message)  
*Send a message to the serial port.*
- void [Debug\\_SendFromList](#) (Msg\_t msg)  
*Send a message from the message list.*
- void [Debug\\_WriteFloat](#) (double value)  
*Write a floating-point value to the serial port.*

## Initialization

- void [Debug\\_Init](#) (Uart\_t uart)  
*Initialize the Debug module.*

## Assertions

- void `Debug_Assert` (bool condition)

*Stops program if `condition` is `true`. Useful for bug detection during debugging.*

### 4.3.2.1 Detailed Description

Module for debugging functions, including serial output and assertions.

### 4.3.2.2 Function Documentation

#### Debug\_Init()

```
void Debug_Init (
    Uart_t uart )
```

Initialize the Debug module.

##### Parameters

in	<code>uart</code>	UART to use for serial output.
----	-------------------	--------------------------------

##### Postcondition

An initialization message is sent to the serial port.

#### Debug\_SendMsg()

```
void Debug_SendMsg (
    void * message )
```

Send a message to the serial port.

##### Precondition

Initialize the Debug module.

##### Parameters

<code>message</code>	(Pointer to) array of ASCII characters.
----------------------	---

##### Postcondition

A floating point value is written to the serial port.

See also

[Debug\\_SendMsg\(\)](#)

### Debug\_SendFromList()

```
void Debug_SendFromList (
    Msg_t msg )
```

Send a message from the message list.

#### Precondition

Initialize the Debug module.

#### Parameters

in	<i>msg</i>	An entry from the enumeration.
----	------------	--------------------------------

#### Postcondition

The corresponding message is sent to the serial port.

See also

[Debug\\_SendMsg\(\)](#)

### Debug\_WriteFloat()

```
void Debug_WriteFloat (
    double value )
```

Write a floating-point value to the serial port.

#### Precondition

Initialize the Debug module.

#### Parameters

in	<i>value</i>	Floating-point value.
----	--------------	-----------------------

#### Postcondition

A floating point value is written to the serial port.



See also

[Debug\\_SendMsg\(\)](#)

### Debug\_Assert()

```
void Debug_Assert (
    bool condition )
```

Stops program if `condition` is `true`. Useful for bug detection during debugging.

#### Precondition

Initialize the Debug module.

#### Parameters

in	<i>condition</i>	Conditional statement to evaluate.
----	------------------	------------------------------------

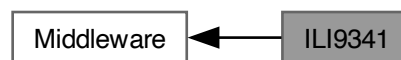
#### Postcondition

If `condition == true`, the program continues normally. If `condition == false`, a message is sent and a breakpoint is activated.

### 4.3.3 ILI9341

Functions for interfacing an ILI9341-based 240RGBx320 LCD via [Serial Peripheral Interface \(SPI\)](#).

Collaboration diagram for ILI9341:



#### Files

- file [ILI9341.c](#)  
*Source code for ILI9341 module.*
- file [ILI9341.h](#)  
*Driver module for interfacing with an ILI9341 LCD driver.*

## Enumerations

- enum { `ILI9341_NUM_COLS` = 240 , `ILI9341_NUM_ROWS` = 320 }
- enum `Cmd_t` {  
`NOP` = 0x00 , `SWRESET` = 0x01 , `SPLIN` = 0x10 , `SPLOUT` = 0x11 ,  
`PTLON` = 0x12 , `NORON` = 0x13 , `DINVOFF` = 0x20 , `DINVON` = 0x21 ,  
`CASET` = 0x2A , `PASET` = 0x2B , `RAMWR` = 0x2C , `DISPOFF` = 0x28 ,  
`DISPON` = 0x29 , `PLTAR` = 0x30 , `VSCRDEF` = 0x33 , `MADCTL` = 0x36 ,  
`VSCRSADD` = 0x37 , `IDMOFF` = 0x38 , `IDMON` = 0x39 , `PIXSET` = 0x3A ,  
`FRMCTR1` = 0xB1 , `FRMCTR2` = 0xB2 , `FRMCTR3` = 0xB3 , `PRCTR` = 0xB5 ,  
`IFCTL` = 0xF6 }
- enum `sleepMode_t` { `SLEEP_ON` = `SPLIN` , `SLEEP_OFF` = `SPLOUT` }
- enum `displayArea_t` { `NORMAL_AREA` = `NORON` , `PARTIAL_AREA` = `PTLON` }
- enum `colorExpr_t` { `FULL_COLORS` = `IDMOFF` , `PARTIAL_COLORS` = `IDMON` }
- enum `invertMode_t` { `INVERT_ON` = `DINVON` , `INVERT_OFF` = `DINVOFF` }
- enum `outputMode_t` { `OUTPUT_ON` = `DISPON` , `OUTPUT_OFF` = `DISPOFF` }
- enum `colorDepth_t` { `COLORDEPTH_16BIT` = 0x55 , `COLORDEPTH_18BIT` = 0x66 }

## Functions

- static void `ILI9341_setMode` (`uint8_t` param)
- static void `ILI9341_setAddress` (`uint16_t` start\_address, `uint16_t` end\_address, bool is\_row)
- static void `ILI9341_sendParams` (`Cmd_t` cmd)  
*Send a command and/or the data within the FIFO buffer. A command is only sent when cmd != NOP (where NOP = 0). Data is only sent if the FIFO buffer is not empty.*
- void `ILI9341_Init` (`Timer_t` timer)  
*Initialize the LCD driver and the SPI module.*
- void `ILI9341_setInterface` (void)  
*Sets the interface for the ILI9341.*
- void `ILI9341_resetHard` (`Timer_t` timer)  
*Perform a hardware reset of the LCD driver.*
- void `ILI9341_resetSoft` (`Timer_t` timer)  
*Perform a software reset of the LCD driver.*
- void `ILI9341_setSleepMode` (`sleepMode_t` sleepMode, `Timer_t` timer)  
*Enter or exit sleep mode (ON by default).*
- void `ILI9341_setDisplayArea` (`displayArea_t` displayArea)  
*Set the display area.*
- void `ILI9341_setColorExpression` (`colorExpr_t` colorExpr)  
*Set the color expression (FULL\_COLORS by default).*
- void `ILI9341_setPartialArea` (`uint16_t` rowStart, `uint16_t` rowEnd)  
*Set the display area for partial mode. Call before activating partial mode.*
- void `ILI9341_setDispInversion` (`invertMode_t` invertMode)  
*Toggle display inversion (OFF by default).*
- void `ILI9341_setDispOutput` (`outputMode_t` outputMode)  
*Change whether the IC is outputting to the display for not.*
- void `ILI9341_setMemAccessCtrl` (bool areRowsFlipped, bool areColsFlipped, bool areRowsAndCols↔ Switched, bool isVertRefreshFlipped, bool isColorOrderFlipped, bool isHorRefreshFlipped)  
*Set how data is converted from memory to display.*
- void `ILI9341_setColorDepth` (`colorDepth_t` colorDepth)  
*Set the color depth for the display.*
- void `ILI9341_setFrameRate` (`uint8_t` divisionRatio, `uint8_t` clocksPerLine)  
*TODO: Write brief.*

- void [ILI9341\\_setRowAddress](#) (uint16\_t startRow, uint16\_t endRow)  
*Sets the start/end rows to be written to.*
- void [ILI9341\\_setColAddress](#) (uint16\_t startCol, uint16\_t endCol)  
*Sets the start/end columns to be written to.*
- void [ILI9341\\_writeMemCmd](#) (void)  
*Signal to the driver that pixel data is incoming and should be written to memory.*
- void [ILI9341\\_writePixel](#) (uint8\_t red, uint8\_t green, uint8\_t blue)  
*Write a single pixel to frame memory.*

## Variables

- static uint32\_t **ILI9341\_Buffer** [8]
- static Fifo\_t **ILI9341\_Fifo**
- struct {  
    sleepMode\_t **sleepMode**  
    displayArea\_t **displayArea**  
    colorExpr\_t **colorExpression**  
    invertMode\_t **invertMode**  
    outputMode\_t **outputMode**  
    colorDepth\_t **colorDepth**  
    bool **isInit**  
} **ili9341** = { SLEEP\_ON, NORMAL\_AREA, FULL\_COLORS, INVERT\_OFF, OUTPUT\_ON, COLORDEPTH\_16BIT, false }

### 4.3.3.1 Detailed Description

Functions for interfacing an ILI9341-based 240RGBx320 LCD via [Serial Peripheral Interface \(SPI\)](#).

### 4.3.3.2 Enumeration Type Documentation

#### anonymous enum

anonymous enum

#### Enumerator

ILI9341_NUM_COLS	<b>4.3.3.3 of columns available on the display</b>
ILI9341_NUM_ROWS	<b>4.3.3.4 of rows available on the display</b>

#### Cmd\_t

enum [Cmd\\_t](#)

### Enumerator

NOP	No Operation.
SWRESET	Software Reset.
SPLIN	Enter Sleep Mode.
SPLOUT	Sleep Out (i.e. Exit Sleep Mode)
PTLON	Partial Display Mode ON.
NORON	Normal Display Mode ON.
DINVOFF	Display Inversion OFF.
DINVON	Display Inversion ON.
CASET	Column Address Set.
PASET	Page Address Set.
RAMWR	Memory Write.
DISPOFF	Display OFF.
DISPON	Display ON.
PLTAR	Partial Area.
VSCRDEF	Vertical Scrolling Definition.
MADCTL	Memory Access Control.
VSCRSADD	Vertical Scrolling Start Address.
IDMOFF	Idle Mode OFF.
IDMON	Idle Mode ON.
PIXSET	Pixel Format Set.
FRMCTR1	Frame Rate Control Set (Normal Mode)
FRMCTR2	Frame Rate Control Set (Idle Mode)
FRMCTR3	Frame Rate Control Set (Partial Mode)
PRCTR	Blanking Porch Control.
IFCTL	Interface Control.

### 4.3.3.5 Function Documentation

#### ILI9341\_setMode()

```
static void ILI9341_setMode (
    uint8_t param ) [static]
```

This function simply groups each of the configuration functions into one to reduce code duplication.

#### ILI9341\_setAddress()

```
static void ILI9341_setAddress (
    uint16_t start_address,
    uint16_t end_address,
    bool is_row ) [static]
```

This function implements the "Column Address Set" (CASET) and "Page Address Set" (PASET) commands from p. 110-113 of the ILI9341 datasheet.

The input parameters represent the first and last addresses to be written to when [ILI9341\\_writePixel\(\)](#) is called.

To work correctly, `startAddress` must be no greater than `endAddress`, and `endAddress` cannot be greater than the max number of rows/columns.

**ILI9341\_sendParams()**

```
static void ILI9341_sendParams (
    Cmd_t cmd ) [static]
```

Send a command and/or the data within the FIFO buffer. A command is only sent when `cmd != NOP` (where `NOP = 0`). Data is only sent if the FIFO buffer is not empty.

**Parameters**

in	<i>cmd</i>	Command to send.
----	------------	------------------

**ILI9341\_Init()**

```
void ILI9341_Init (
    Timer_t timer )
```

Initialize the LCD driver and the SPI module.

**Parameters**

in	<i>timer</i>	Hardware timer to use during initialization.
----	--------------	--

**ILI9341\_setInterface()**

```
void ILI9341_setInterface (
    void )
```

Sets the interface for the ILI9341.

The parameters for this command are hard-coded, so it only needs to be called once upon initialization.

This function implements the "Interface Control" (IFCTL) command from p. 192-194 of the ILI9341 datasheet, which controls how the LCD driver handles 16-bit data and what interfaces (internal or external) are used.

Name	Bit #	Param #	Effect when set = 1
MY_EOR	7	0	flips value of corresponding MADCTL bit
MX_EOR	6		flips value of corresponding MADCTL bit
MV_EOR	5		flips value of corresponding MADCTL bit
BGR_EOR	3		flips value of corresponding MADCTL bit
WEMODE	0		overflowing pixel data is not ignored
EPF[1:0]	5:4	1	controls 16 to 18-bit pixel data conversion
MDT[1:0]	1:0		controls display data transfer method
ENDIAN	5	2	host sends LSB first
DM[1:0]	3:2		selects display operation mode
RM	1		selects GRAM interface mode
RIM	0		specifies RGB interface-specific details

The first param's bits are cleared so that the corresponding MADCTL bits (ILI9341\_setMemoryAccessCtrl()) are unaffected and overflowing pixel data is ignored. The EPF bits are cleared so that the LSB of the R and B values is copied from the MSB when using 16-bit color depth. The TM4C123 sends the MSB first, so the ENDIAN bit is cleared. The other bits are cleared and/or irrelevant since the RGB and VSYNC interfaces aren't used.

### ILI9341\_resetHard()

```
void ILI9341_resetHard (
    Timer_t timer )
```

Perform a hardware reset of the LCD driver.

#### Parameters

in	<i>timer</i>	Hardware timer to use during reset.
----	--------------	-------------------------------------

The LCD driver's RESET pin requires a negative logic (i.e. active `LOW`) signal for  $\geq 10$  [us] and an additional 5 [ms] before further commands can be sent.

### ILI9341\_resetSoft()

```
void ILI9341_resetSoft (
    Timer_t timer )
```

Perform a software reset of the LCD driver.

#### Parameters

in	<i>timer</i>	Hardware timer to use during reset.
----	--------------	-------------------------------------

the driver needs 5 [ms] before another command

### ILI9341\_setSleepMode()

```
void ILI9341_setSleepMode (
    sleepMode_t sleepMode,
    Timer_t timer )
```

Enter or exit sleep mode (ON by default).

#### Parameters

in	<i>sleepMode</i>	SLEEP_ON or SLEEP_OFF
in	<i>timer</i>	Hardware timer to use for a slight delay after the mode change.

#### Postcondition

The IC will be in or out of sleep mode depending on the value of `sleepMode`.

The MCU must wait  $\geq 5$  [ms] before sending further commands regardless of the selected mode.

It's also necessary to wait 120 [ms] before sending `SPL_OUT` after sending `SPL_IN` or a reset, so this function waits 120 [ms] regardless of the preceding event.

### ILI9341\_setDisplayArea()

```
void ILI9341_setDisplayArea (
    displayArea_t displayArea )
```

Set the display area.

#### Precondition

If using partial mode, set the partial area first.

#### Parameters

in	<i>displayArea</i>	NORMAL_AREA or PARTIAL_AREA
----	--------------------	-----------------------------

#### See also

[ILI9341\\_setPartialArea\(\)](#)

### ILI9341\_setColorExpression()

```
void ILI9341_setColorExpression (
    colorExpr_t colorExpr )
```

Set the color expression (FULL\_COLORS by default).

#### Parameters

in	<i>colorExpr</i>	FULL_COLORS or PARTIAL_COLORS
----	------------------	-------------------------------

#### Postcondition

With partial color expression, the display only uses 8 colors. Otherwise, the color depth determines the number of colors available.

### ILI9341\_setPartialArea()

```
void ILI9341_setPartialArea (
    uint16_t rowStart,
    uint16_t rowEnd )
```

Set the display area for partial mode. Call before activating partial mode.

**Parameters**

in	<i>rowStart</i>	
in	<i>rowEnd</i>	

**See also**

[ILI9341\\_setDisplayArea\(\)](#)

**ILI9341\_setDispInversion()**

```
void ILI9341_setDispInversion (
    invertMode_t invertMode )
```

Toggle display inversion (OFF by default).

**Parameters**

in	<i>invertMode</i>	INVERT_ON or INVERT_OFF
----	-------------------	-------------------------

**Postcondition**

When inversion is ON, the display colors are inverted. (e.g. BLACK -> WHITE, GREEN -> PURPLE)

**ILI9341\_setDispOutput()**

```
void ILI9341_setDispOutput (
    outputMode_t outputMode )
```

Change whether the IC is outputting to the display for not.

**Parameters**

in	<i>outputMode</i>	OUTPUT_ON or OUTPUT_OFF
----	-------------------	-------------------------

**Postcondition**

If ON, the IC outputs data from its memory to the display. If OFF, the display is cleared and the IC stops outputting data.

TODO: Write description

**ILI9341\_setMemAccessCtrl()**

```
void ILI9341_setMemAccessCtrl (
    bool areRowsFlipped,
```



```

bool areColsFlipped,
bool areRowsAndColsSwitched,
bool isVertRefreshFlipped,
bool isColorOrderFlipped,
bool isHorRefreshFlipped )

```

Set how data is converted from memory to display.

#### Parameters

in	<i>areRowsFlipped</i>	
in	<i>areColsFlipped</i>	
in	<i>areRowsAndColsSwitched</i>	
in	<i>isVertRefreshFlipped</i>	
in	<i>isColorOrderFlipped</i>	
in	<i>isHorRefreshFlipped</i>	

This function implements the "Memory Access Control" (MADCTL) command from p. 127-128 of the ILI9341 datasheet, which controls how the LCD driver displays data upon writing to memory.

Name	Bit #	Effect when set = 1
MY	7	flip row (AKA "page") addresses
MX	6	flip column addresses
MV	5	exchange rows and column addresses
ML	4	reverse horizontal refresh order
BGR	3	reverse color input order (RGB -> BGR)
MH	2	reverse vertical refresh order

All bits are clear after powering on or HWRESET.

#### ILI9341\_setColorDepth()

```

void ILI9341_setColorDepth (
    colorDepth_t colorDepth )

```

Set the color depth for the display.

#### Parameters

in	<i>colorDepth</i>	COLORDEPTH_16BIT or COLORDEPTH_18BIT
----	-------------------	--------------------------------------

#### Postcondition

16BIT mode allows for ~65K ( $2^{16}$ ) colors and requires 2 transfers. 18BIT mode allows for ~262K ( $2^{18}$ ) colors but requires 3 transfers.

#### ILI9341\_setFrameRate()

```

void ILI9341_setFrameRate (

```

```
uint8_t divisionRatio,  
uint8_t clocksPerLine )
```

TODO: Write brief.

TODO: Write description

### ILI9341\_setRowAddress()

```
void ILI9341_setRowAddress (   
    uint16_t startRow,  
    uint16_t endRow )
```

Sets the start/end rows to be written to.

#### Parameters

in		
----	--	--

$0 \leq \text{startRow} \leq \text{endRow}$

#### Parameters

in		
----	--	--

$\text{startRow} \leq \text{endRow} \leq 240$

#### See also

[ILI9341\\_setRowAddress](#), [ILI9341\\_writePixel\(\)](#)

This function is simply an interface to [ILI9341\\_setAddress\(\)](#). To work correctly, `start_row` must be no greater than `end_row`, and `end_row` cannot be greater than the max row number (default 320).

### ILI9341\_setColAddress()

```
void ILI9341_setColAddress (   
    uint16_t startCol,  
    uint16_t endCol )
```

Sets the start/end columns to be written to.

#### Parameters

in		
----	--	--

$0 \leq \text{startCol} \leq \text{endCol}$

**Parameters**

in		
----	--	--

`startCol` ≤ `endCol` < 240

**See also**

[ILI9341\\_setColAddress](#), [ILI9341\\_writePixel\(\)](#)

This function is simply an interface to [ILI9341\\_setAddress\(\)](#). To work correctly, `start_col` must be no greater than `end_col`, and `end_col` cannot be greater than the max column number (default 240).

**ILI9341\_writeMemCmd()**

```
void ILI9341_writeMemCmd (
    void )
```

Signal to the driver that pixel data is incoming and should be written to memory.

**Precondition**

Set the row and/or column addresses.

**Postcondition**

The LCD driver is ready to accept pixel data.

**See also**

[ILI9341\\_setRowAddress](#), [ILI9341\\_setColAddress\(\)](#), [ILI9341\\_writePixel\(\)](#)

**ILI9341\_writePixel()**

```
void ILI9341_writePixel (
    uint8_t red,
    uint8_t green,
    uint8_t blue )
```

Write a single pixel to frame memory.

**Precondition**

Send the "Write Memory" command.

Set the desired color depth for the display.

## Parameters

in	<i>red</i>	5 or 6-bit R value
in	<i>green</i>	5 or 6-bit G value
in	<i>blue</i>	5 or 6-bit B value

## See also

[ILI9341\\_setColorDepth](#), [ILI9341\\_writeMemCmd\(\)](#), [ILI9341\\_writePixel\(\)](#)

This function sends one pixel to the display. Because the serial interface (SPI) is used, each pixel requires 2 transfers in 16-bit mode and 3 transfers in 18-bit mode.

The following table (adapted from p. 63 of the datasheet) visualizes how the RGB data is sent to the display when using 16-bit color depth.

Transfer	1								2							
Bit #	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Value	R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0

The following table (adapted from p. 64 of the datasheet) visualizes how the RGB data is sent to the display when using 18-bit color depth.

Transfer	1								2		
Bit #	7	6	5	4	3	2	1	0	7	6	...
Value	R5	R4	R3	R2	R1	R0	0/1	0/1	G5	G4	...

## 4.3.4 LED

Functions for driving light-emitting diodes (LEDs) via [General-Purpose Input/Output \(GPIO\)](#).

Collaboration diagram for LED:



## Files

- file [Led.c](#)  
*Source code for LED module.*
- file [Led.h](#)  
*Interface for LED module.*

## Data Structures

- struct [Led\\_t](#)

## Macros

- `#define LED_POOL_SIZE 1`

## Variables

- static [LedStruct\\_t](#) [Led\\_ObjPool](#) [LED\_POOL\_SIZE] = { 0 }
- static uint8\_t [num\\_free\\_leds](#) = LED\_POOL\_SIZE

## Initialization & Configuration

- [Led\\_t](#) [Led\\_Init](#) ([GpioPort\\_t](#) gpioPort, [GpioPin\\_t](#) pin)  
*Initialize a light-emitting diode (LED) as an [Led\\_t](#).*
- [GpioPort\\_t](#) [Led\\_GetPort](#) ([Led\\_t](#) led)  
*Get the GPIO port associated with the LED.*
- [GpioPin\\_t](#) [Led\\_GetPin](#) ([Led\\_t](#) led)  
*Get the GPIO pin associated with the LED.*

## Status Checking

- bool [Led\\_isInit](#) ([Led\\_t](#) led)  
*Check if an LED is initialized.*
- bool [Led\\_isOn](#) ([Led\\_t](#) led)  
*Check the LED's status.*

## Operations

- void [Led\\_TurnOn](#) ([Led\\_t](#) led)  
*Turn an LED ON.*
- void [Led\\_TurnOff](#) ([Led\\_t](#) led)  
*Turn an LED OFF.*
- void [Led\\_Toggle](#) ([Led\\_t](#) led)  
*Toggle an LED.*

### 4.3.4.1 Detailed Description

Functions for driving light-emitting diodes (LEDs) via [General-Purpose Input/Output \(GPIO\)](#).

### 4.3.4.2 Function Documentation

#### [Led\\_Init\(\)](#)

```
Led_t Led_Init (  
    GpioPort_t gpioPort,  
    GpioPin_t pin )
```

Initialize a light-emitting diode (LED) as an [Led\\_t](#).

**Parameters**

in	<i>gpioPort</i>	Pointer to a <code>struct</code> representing a GPIO port.
in	<i>pin</i>	GPIO pin to use.
out	<i>led</i>	Pointer to LED data structure.

**Led\_GetPort()**

```
GpioPort_t Led_GetPort (  
    Led_t led )
```

Get the GPIO port associated with the LED.

**Precondition**

Initialize the LED.

**Parameters**

in	<i>led</i>	Pointer to LED data structure.
out	<i>gpioPort</i>	Pointer to a GPIO port data structure.

**See also**

[Led\\_Init\(\)](#), [Led\\_GetPin\(\)](#)

**Led\_GetPin()**

```
GpioPin_t Led_GetPin (  
    Led_t led )
```

Get the GPIO pin associated with the LED.

**Precondition**

Initialize the LED.

**Parameters**

in	<i>led</i>	Pointer to LED data structure.
out	<i>pin</i>	GPIO pin associated with the LED.

**See also**

[Led\\_Init\(\)](#), [Led\\_GetPort\(\)](#)

**Led\_isInit()**

```
bool Led_isInit (
    Led_t led )
```

Check if an LED is initialized.

**Parameters**

in	<i>led</i>	Pointer to LED data structure.
out	<i>true</i>	The LED is initialized.
out	<i>false</i>	The LED is not initialized.

**See also**

[Led\\_Init\(\)](#)

**Led\_isOn()**

```
bool Led_isOn (
    Led_t led )
```

Check the LED's status.

**Precondition**

Initialize the LED.

**Parameters**

in	<i>led</i>	Pointer to LED data structure.
out	<i>true</i>	the LED is ON.
out	<i>false</i>	the LED is OFF.

**See also**

[Led\\_TurnOn\(\)](#), [Led\\_TurnOff\(\)](#), [Led\\_Toggle\(\)](#)

**Led\_TurnOn()**

```
void Led_TurnOn (
    Led_t led )
```

Turn an LED ON.

**Precondition**

Initialize the LED.

**Parameters**

in	<i>led</i>	Pointer to LED data structure.
----	------------	--------------------------------

**Postcondition**

The LED is turned ON.

**See also**

[Led\\_TurnOff\(\)](#), [Led\\_Toggle\(\)](#)

**Led\_TurnOff()**

```
void Led_TurnOff (
    Led_t led )
```

Turn an LED OFF.

**Precondition**

Initialize the LED.

**Parameters**

in	<i>led</i>	Pointer to LED data structure.
----	------------	--------------------------------

**Postcondition**

The LED is turned OFF.

**See also**

[Led\\_TurnOn\(\)](#), [Led\\_Toggle\(\)](#)

**Led\_Toggle()**

```
void Led_Toggle (
    Led_t led )
```

Toggle an LED.

**Precondition**

Initialize the LED.



## Parameters

in	/led	Pointer to LED data structure.
----	------	--------------------------------

## Postcondition

The LED's state is flipped (i.e. ON -> OFF or OFF -> ON).

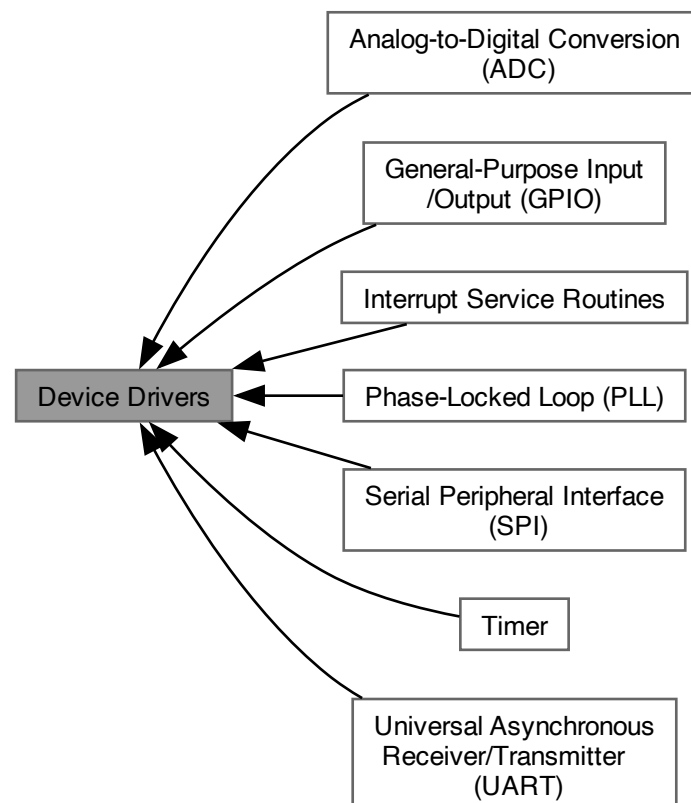
## See also

[Led\\_TurnOn\(\)](#), [Led\\_TurnOff\(\)](#)

## 4.4 Device Drivers

Low level device driver modules.

Collaboration diagram for Device Drivers:



## Modules

- [Analog-to-Digital Conversion \(ADC\)](#)  
*Functions for analog-to-digital conversion.*
- [General-Purpose Input/Output \(GPIO\)](#)  
*Functions for using GPIO ports.*
- [Phase-Locked Loop \(PLL\)](#)  
*Function for initializing the phase-locked loop.*
- [Serial Peripheral Interface \(SPI\)](#)  
*Functions for SPI-based communication via the SSI peripheral.*
- [Timer](#)  
*Functions for using hardware timers.*
- [Universal Asynchronous Receiver/Transmitter \(UART\)](#)  
*Functions for serial communication via the UART peripheral.*
- [Interrupt Service Routines](#)  
*Functions for manipulating the interrupt vector table and setting up interrupt handlers via the NVIC.*

### 4.4.1 Detailed Description

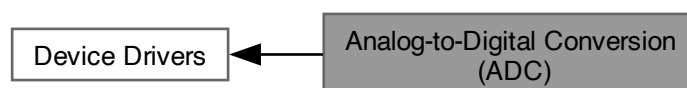
Low level device driver modules.

These modules contain functions for interfacing with the TM4C123 microcontroller's built-in peripherals.

### 4.4.2 Analog-to-Digital Conversion (ADC)

Functions for analog-to-digital conversion.

Collaboration diagram for Analog-to-Digital Conversion (ADC):



## Files

- file [ADC.c](#)  
*Source code for analog-to-digital conversion (ADC) module.*
- file [ADC.h](#)  
*Header file for analog-to-digital conversion (ADC) module.*

## Functions

- void [ADC\\_Init](#) (void)  
*Initialize ADC0 as a single-input analog-to-digital converter.*

#### 4.4.2.1 Detailed Description

Functions for analog-to-digital conversion.

#### 4.4.2.2 Function Documentation

##### ADC\_Init()

```
void ADC_Init (
    void )
```

Initialize ADC0 as a single-input analog-to-digital converter.

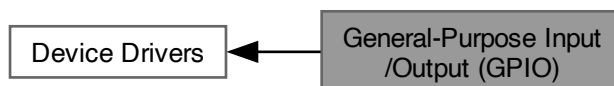
##### Postcondition

Analog input 8 (Ain8) – AKA GPIO pin PE5 – captures samples when triggered by one of the hardware timers, and initiates an interrupt once sample capture is complete.

#### 4.4.3 General-Purpose Input/Output (GPIO)

Functions for using GPIO ports.

Collaboration diagram for General-Purpose Input/Output (GPIO):



##### Files

- file [GPIO.c](#)  
*Source code for GPIO module.*
- file [GPIO.h](#)  
*Header file for general-purpose input/output (GPIO) device driver.*

##### Data Structures

- struct [GpioPort\\_t](#)

##### Macros

- `#define GPIO_NUM_PORTS 6`

## Enumerations

- enum {  
**GPIO\_PORTA\_BASE\_ADDRESS** = (uint32\_t) 0x40004000 , **GPIO\_PORTB\_BASE\_ADDRESS** = (uint32\_t) 0x40005000 , **GPIO\_PORTC\_BASE\_ADDRESS** = (uint32\_t) 0x40006000 , **GPIO\_PORTD\_BASE\_ADDRESS** = (uint32\_t) 0x40007000 ,  
**GPIO\_PORTE\_BASE\_ADDRESS** = (uint32\_t) 0x40024000 , **GPIO\_PORTF\_BASE\_ADDRESS** = (uint32\_t) 0x40025000 }
- enum {  
**GPIO\_DATA\_R\_OFFSET** = (uint32\_t) 0x03FC , **GPIO\_DIR\_R\_OFFSET** = (uint32\_t) 0x0400 , **GPIO\_IS\_R\_OFFSET** = (uint32\_t) 0x0404 , **GPIO\_IBE\_R\_OFFSET** = (uint32\_t) 0x0408 ,  
**GPIO\_IEV\_R\_OFFSET** = (uint32\_t) 0x040C , **GPIO\_IM\_R\_OFFSET** = (uint32\_t) 0x0410 , **GPIO\_ICR\_R\_OFFSET** = (uint32\_t) 0x041C , **GPIO\_AFSEL\_R\_OFFSET** = (uint32\_t) 0x0420 ,  
**GPIO\_DR2R\_R\_OFFSET** = (uint32\_t) 0x0500 , **GPIO\_DR4R\_R\_OFFSET** = (uint32\_t) 0x0504 , **GPIO\_DR8R\_R\_OFFSET** = (uint32\_t) 0x0508 , **GPIO\_PUR\_R\_OFFSET** = (uint32\_t) 0x0510 ,  
**GPIO\_PDR\_R\_OFFSET** = (uint32\_t) 0x0518 , **GPIO\_DEN\_R\_OFFSET** = (uint32\_t) 0x051C , **GPIO\_LOCK\_R\_OFFSET** = (uint32\_t) 0x0520 , **GPIO\_COMMIT\_R\_OFFSET** = (uint32\_t) 0x0524 ,  
**GPIO\_AMSEL\_R\_OFFSET** = (uint32\_t) 0x0528 , **GPIO\_PCTL\_R\_OFFSET** = (uint32\_t) 0x052C }
- enum **GPIO\_PortName\_t** {  
**GPIO\_PORT\_A** , **GPIO\_PORT\_B** , **GPIO\_PORT\_C** , **GPIO\_PORT\_D** ,  
**GPIO\_PORT\_E** , **GPIO\_PORT\_F** , **A** = **GPIO\_PORT\_A** , **B** = **GPIO\_PORT\_B** ,  
**C** = **GPIO\_PORT\_C** , **D** = **GPIO\_PORT\_D** , **E** = **GPIO\_PORT\_E** , **F** = **GPIO\_PORT\_F** }
- enum **GpioPin\_t** {  
**GPIO\_PIN0** = ((uint8\_t) 1) , **GPIO\_PIN1** = ((uint8\_t) (1 << 1)) , **GPIO\_PIN2** = ((uint8\_t) (1 << 2)) , **GPIO\_PIN3** = ((uint8\_t) (1 << 3)) ,  
**GPIO\_PIN4** = ((uint8\_t) (1 << 4)) , **GPIO\_PIN5** = ((uint8\_t) (1 << 5)) , **GPIO\_PIN6** = ((uint8\_t) (1 << 6)) ,  
**GPIO\_PIN7** = ((uint8\_t) (1 << 7)) ,  
**GPIO\_ALL\_PINS** = ((uint8\_t) (0xFF)) }
- enum **GPIO\_LAUNCHPAD\_LEDS** {  
**LED\_RED** = **GPIO\_PIN1** , **LED\_GREEN** = **GPIO\_PIN3** , **LED\_BLUE** = **GPIO\_PIN2** , **LED\_YELLOW** = (**LED\_RED** + **LED\_GREEN**) ,  
**LED\_CYAN** = (**LED\_BLUE** + **LED\_GREEN**) , **LED\_PURPLE** = (**LED\_RED** + **LED\_BLUE**) , **LED\_WHITE** = (**LED\_RED** + **LED\_BLUE** + **LED\_GREEN**) }

## Functions

- GpioPort\_t **GPIO\_InitPort** (GPIO\_PortName\_t portName)  
Initialize a GPIO Port and return a pointer to its struct.
- bool **GPIO\_isPortInit** (GpioPort\_t gpioPort)  
Check if the GPIO port is initialized.
- uint32\_t **GPIO\_getBaseAddr** (GpioPort\_t gpioPort)  
Get the base address of a GPIO port.
- void **GPIO\_ConfigDirOutput** (GpioPort\_t gpioPort, GpioPin\_t pinMask)  
Configure the direction of the specified GPIO pins. All pins are configured to *INPUT* by default, so this function should only be called to specify *OUTPUT* pins.
- void **GPIO\_ConfigDirInput** (GpioPort\_t gpioPort, GpioPin\_t pinMask)  
Configure the specified GPIO pins as *INPUT* pins. All pins are configured to *INPUT* by default, so this function is technically unnecessary, but useful for code readability.
- void **GPIO\_ConfigPullUp** (GpioPort\_t gpioPort, GpioPin\_t pinMask)  
Activate the specified pins' internal pull-up resistors.
- void **GPIO\_ConfigPullDown** (GpioPort\_t gpioPort, GpioPin\_t pinMask)  
Activate the specified pins' internal pull-down resistors.
- void **GPIO\_ConfigDriveStrength** (GpioPort\_t gpioPort, GpioPin\_t pinMask, uint8\_t drive\_mA)  
Configure the specified pins' drive strength. Pins are initialized with 2[mA] drive strength, so this is only needed for a drive strength of 4[mA] or 8[mA].

- void [GPIO\\_EnableDigital](#) (GpioPort\_t gpioPort, GpioPin\_t pinMask)  
*Enable digital I/O for the specified pins.*
- void [GPIO\\_DisableDigital](#) (GpioPort\_t gpioPort, GpioPin\_t pinMask)  
*Disable digital I/O for the specified pins.*
- void [GPIO\\_ConfigInterrupts\\_Edge](#) (GpioPort\_t gpioPort, GpioPin\_t pinMask, bool risingEdge)  
*Configure the specified GPIO pins to trigger an interrupt on the rising or falling edge of an input.*
- void [GPIO\\_ConfigInterrupts\\_BothEdges](#) (GpioPort\_t gpioPort, GpioPin\_t pinMask)  
*Configure the specified GPIO pins to trigger an interrupt on both edges of an input.*
- void [GPIO\\_ConfigInterrupts\\_LevelTrig](#) (GpioPort\_t gpioPort, GpioPin\_t pinMask, bool highLevel)  
*Configure the specified GPIO pins to trigger an interrupt on a high level or low level pulse.*
- void [GPIO\\_ConfigNVIC](#) (GpioPort\_t gpioPort, uint8\_t priority)  
*Configure interrupts for the selected port in the NVIC.*
- volatile uint32\_t \* [GPIO\\_getDataRegister](#) (GpioPort\_t gpioPort)  
*Get the address of a GPIO port's data register.*
- uint8\_t [GPIO\\_ReadPins](#) (GpioPort\_t gpioPort, GpioPin\_t pinMask)  
*Read from the specified GPIO pin.*
- void [GPIO\\_WriteHigh](#) (GpioPort\_t gpioPort, GpioPin\_t pinMask)  
*Write a 1 to the specified GPIO pins.*
- void [GPIO\\_WriteLow](#) (GpioPort\_t gpioPort, GpioPin\_t pinMask)  
*Write a 0 to the specified GPIO pins.*
- void [GPIO\\_Toggle](#) (GpioPort\_t gpioPort, GpioPin\_t pinMask)  
*Toggle the specified GPIO pins.*
- void [GPIO\\_ConfigAltMode](#) (GpioPort\_t gpioPort, GpioPin\_t pinMask)  
*Activate the alternate mode for the specified pins.*
- void [GPIO\\_ConfigPortCtrl](#) (GpioPort\_t gpioPort, GpioPin\_t pinMask, uint8\_t fieldEncoding)  
*Specify the alternate mode to use for the specified pins.*
- void [GPIO\\_ConfigAnalog](#) (GpioPort\_t gpioPort, GpioPin\_t pinMask)  
*Activate analog mode for the specified GPIO pins.*

## Variables

- static [GpioPortStruct\\_t GPIO\\_PTR\\_ARR](#) [6]

### 4.4.3.1 Detailed Description

Functions for using GPIO ports.

### 4.4.3.2 Enumeration Type Documentation

#### GPIO\_LAUNCHPAD\_LEDS

enum [GPIO\\_LAUNCHPAD\\_LEDS](#)

Enumerator

LED_RED	PF1.
LED_GREEN	PF3.
LED_BLUE	PF2.

#### 4.4.3.3 Function Documentation

##### GPIO\_InitPort()

```
GpioPort_t GPIO_InitPort (
    GPIO_PortName_t portName )
```

Initialize a GPIO Port and return a pointer to its struct.

###### Parameters

in	<i>portName</i>	Name of the chosen port.
out	<i>gpioPort</i>	Pointer to the specified GPIO port.

##### GPIO\_isPortInit()

```
bool GPIO_isPortInit (
    GpioPort_t gpioPort )
```

Check if the GPIO port is initialized.

###### Parameters

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
out	<i>true</i>	The GPIO port is initialized.
out	<i>false</i>	The GPIO port has not been initialized.

##### GPIO\_getBaseAddr()

```
uint32_t GPIO_getBaseAddr (
    GpioPort_t gpioPort )
```

Get the base address of a GPIO port.

###### Parameters

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
out	<i>baseAddress</i>	Base address of the GPIO port.

##### GPIO\_ConfigDirOutput()

```
void GPIO_ConfigDirOutput (
    GpioPort_t gpioPort,
    GpioPin_t pinMask )
```

Configure the direction of the specified GPIO pins. All pins are configured to `INPUT` by default, so this function should only be called to specify `OUTPUT` pins.

## Parameters

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
in	<i>bitMask</i>	Bit mask corresponding to the intended OUTPUT pin(s).

**GPIO\_ConfigDirInput()**

```
void GPIO_ConfigDirInput (
    GpioPort_t gpioPort,
    GpioPin_t pinMask )
```

Configure the specified GPIO pins as INPUT pins. All pins are configured to INPUT by default, so this function is technically unnecessary, but useful for code readability.

## Parameters

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
in	<i>bitMask</i>	Bit mask corresponding to the intended INPUT pin(s).

**GPIO\_ConfigPullUp()**

```
void GPIO_ConfigPullUp (
    GpioPort_t gpioPort,
    GpioPin_t pinMask )
```

Activate the specified pins' internal pull-up resistors.

## Parameters

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
in	<i>pinMask</i>	Bit mask corresponding to the intended pin(s).

**GPIO\_ConfigPullDown()**

```
void GPIO_ConfigPullDown (
    GpioPort_t gpioPort,
    GpioPin_t pinMask )
```

Activate the specified pins' internal pull-down resistors.

## Parameters

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
in	<i>pinMask</i>	Bit mask corresponding to the intended pin(s).

**GPIO\_ConfigDriveStrength()**

```
void GPIO_ConfigDriveStrength (
    GpioPort_t gpioPort,
    GpioPin_t pinMask,
    uint8_t drive_mA )
```

Configure the specified pins' drive strength. Pins are initialized with 2[mA] drive strength, so this is only needed for a drive strength of 4[mA] or 8[mA].

**Parameters**

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
in	<i>pinMask</i>	Bit mask corresponding to the intended pin(s).
in	<i>drive_mA</i>	Drive strength in [mA]. Should be 2, 4, or 8 [mA].

**GPIO\_EnableDigital()**

```
void GPIO_EnableDigital (
    GpioPort_t gpioPort,
    GpioPin_t pinMask )
```

Enable digital I/O for the specified pins.

**Parameters**

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
in	<i>pinMask</i>	Bit mask corresponding to the intended pin(s).

**GPIO\_DisableDigital()**

```
void GPIO_DisableDigital (
    GpioPort_t gpioPort,
    GpioPin_t pinMask )
```

Disable digital I/O for the specified pins.

**Parameters**

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
in	<i>pinMask</i>	Bit mask corresponding to the intended pin(s).

**GPIO\_ConfigInterrupts\_Edge()**

```
void GPIO_ConfigInterrupts_Edge (
    GpioPort_t gpioPort,
```



```
GpioPin_t pinMask,
bool risingEdge )
```

Configure the specified GPIO pins to trigger an interrupt on the rising or falling edge of an input.

#### Parameters

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
in	<i>pinMask</i>	Bit mask corresponding to the intended pin(s).
in	<i>risingEdge</i>	true for rising edge, false for falling edge

### GPIO\_ConfigInterrupts\_BothEdges()

```
void GPIO_ConfigInterrupts_BothEdges (
    GpioPort_t gpioPort,
    GpioPin_t pinMask )
```

Configure the specified GPIO pins to trigger an interrupt on both edges of an input.

#### Parameters

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
in	<i>pinMask</i>	Bit mask corresponding to the intended pin(s).

### GPIO\_ConfigInterrupts\_LevelTrig()

```
void GPIO_ConfigInterrupts_LevelTrig (
    GpioPort_t gpioPort,
    GpioPin_t pinMask,
    bool highLevel )
```

Configure the specified GPIO pins to trigger an interrupt on a high level or low level pulse.

#### Parameters

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
in	<i>pinMask</i>	Bit mask corresponding to the intended pin(s).
in	<i>highLevel</i>	true for high level, false for low level

### GPIO\_ConfigNVIC()

```
void GPIO_ConfigNVIC (
    GpioPort_t gpioPort,
    uint8_t priority )
```

Configure interrupts for the selected port in the NVIC.

**Parameters**

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
in	<i>priority</i>	Priority number between 0 (highest) and 7 (lowest).

**GPIO\_getDataRegister()**

```
volatile uint32_t * GPIO_getDataRegister (
    GpioPort_t gpioPort )
```

Get the address of a GPIO port's data register.

**Parameters**

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
out	<i>dataRegister</i>	Address of the GPIO port's data register.

**GPIO\_ReadPins()**

```
uint8_t GPIO_ReadPins (
    GpioPort_t gpioPort,
    GpioPin_t pinMask )
```

Read from the specified GPIO pin.

**Parameters**

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
in	<i>pinMask</i>	Bit mask corresponding to the intended pin(s).

**GPIO\_WriteHigh()**

```
void GPIO_WriteHigh (
    GpioPort_t gpioPort,
    GpioPin_t pinMask )
```

Write a 1 to the specified GPIO pins.

**Parameters**

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
in	<i>pinMask</i>	Bit mask corresponding to the intended pin(s).

**GPIO\_WriteLow()**

```
void GPIO_WriteLow (
    GpioPort_t gpioPort,
    GpioPin_t pinMask )
```

Write a 0 to the specified GPIO pins.

**Parameters**

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
in	<i>pinMask</i>	Bit mask corresponding to the intended pin(s).

**GPIO\_Toggle()**

```
void GPIO_Toggle (
    GpioPort_t gpioPort,
    GpioPin_t pinMask )
```

Toggle the specified GPIO pins.

**Parameters**

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
in	<i>pinMask</i>	Bit mask corresponding to the intended pin(s).

**GPIO\_ConfigAltMode()**

```
void GPIO_ConfigAltMode (
    GpioPort_t gpioPort,
    GpioPin_t pinMask )
```

Activate the alternate mode for the specified pins.

**Parameters**

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
in	<i>pinMask</i>	Bit mask corresponding to the intended pin(s).

**GPIO\_ConfigPortCtrl()**

```
void GPIO_ConfigPortCtrl (
    GpioPort_t gpioPort,
    GpioPin_t pinMask,
    uint8_t fieldEncoding )
```

Specify the alternate mode to use for the specified pins.

**Parameters**

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
in	<i>pinMask</i>	Bit mask corresponding to the intended pin(s).
in	<i>fieldEncoding</i>	Number corresponding to intended alternate mode.

**GPIO\_ConfigAnalog()**

```
void GPIO_ConfigAnalog (
    GpioPort_t gpioPort,
    GpioPin_t pinMask )
```

Activate analog mode for the specified GPIO pins.

**Parameters**

in	<i>gpioPort</i>	Pointer to the specified GPIO port.
in	<i>pinMask</i>	Bit mask corresponding to the intended pin(s).

**4.4.3.4 Variable Documentation****GPIO\_PTR\_ARR**

```
GpioPortStruct_t GPIO_PTR_ARR[6] [static]
```

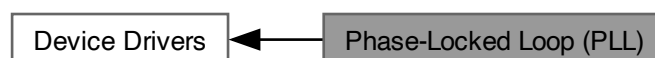
**Initial value:**

```
= {
    { GPIO_PORTA_BASE_ADDRESS, (GPIO_PORTA_BASE_ADDRESS + GPIO_DATA_R_OFFSET), false },
    { GPIO_PORTB_BASE_ADDRESS, (GPIO_PORTB_BASE_ADDRESS + GPIO_DATA_R_OFFSET), false },
    { GPIO_PORTC_BASE_ADDRESS, (GPIO_PORTC_BASE_ADDRESS + GPIO_DATA_R_OFFSET), false },
    { GPIO_PORTD_BASE_ADDRESS, (GPIO_PORTD_BASE_ADDRESS + GPIO_DATA_R_OFFSET), false },
    { GPIO_PORTE_BASE_ADDRESS, (GPIO_PORTE_BASE_ADDRESS + GPIO_DATA_R_OFFSET), false },
    { GPIO_PORTF_BASE_ADDRESS, (GPIO_PORTF_BASE_ADDRESS + GPIO_DATA_R_OFFSET), false },
}
```

**4.4.4 Phase-Locked Loop (PLL)**

Function for initializing the phase-locked loop.

Collaboration diagram for Phase-Locked Loop (PLL):



## Files

- file [PLL.c](#)  
*Implementation details for phase-lock-loop (PLL) functions.*
- file [PLL.h](#)  
*Driver module for activating the phase-locked-loop (PLL).*

## Functions

- void [PLL\\_Init](#) (void)  
*Initialize the phase-locked-loop to change the bus frequency.*

### 4.4.4.1 Detailed Description

Function for initializing the phase-locked loop.

### 4.4.4.2 Function Documentation

#### PLL\_Init()

```
void PLL_Init (  
    void )
```

Initialize the phase-locked-loop to change the bus frequency.

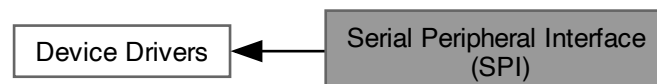
#### Postcondition

The bus frequency is now running at 80 [MHz].

### 4.4.5 Serial Peripheral Interface (SPI)

Functions for SPI-based communication via the SSI peripheral.

Collaboration diagram for Serial Peripheral Interface (SPI):



## Files

- file [SPI.c](#)  
*Source code for serial peripheral interface (SPI) module.*
- file [SPI.h](#)  
*Header file for serial peripheral interface (SPI) module.*

## Macros

- `#define SPI_IS_BUSY (SSI0_SR_R & 0x10)`
- `#define SPI_TX_ISNOTFULL (SSI0_SR_R & 0x02)`
- `#define SPI_CLEAR_RESET() (GPIO_PORTA_DATA_R &= ~(0x80))`
- `#define SPI_SET_RESET() (GPIO_PORTA_DATA_R |= 0x80)`

## Enumerations

- enum {  
    **SPI\_CLK\_PIN** = GPIO\_PIN2 , **SPI\_CS\_PIN** = GPIO\_PIN3 , **SPI\_RX\_PIN** = GPIO\_PIN4 , **SPI\_TX\_PIN** = GPIO\_PIN5 ,  
    **SPI\_DC\_PIN** = GPIO\_PIN6 , **SPI\_RESET\_PIN** = GPIO\_PIN7 , **SPI\_SSI0\_PINS** = (SPI\_CLK\_PIN | SPI\_CS\_PIN | SPI\_RX\_PIN | SPI\_TX\_PIN) , **SPI\_GPIO\_PINS** = (SPI\_DC\_PIN | SPI\_RESET\_PIN) ,  
    **SPI\_ALL\_PINS** = (SPI\_SSI0\_PINS | SPI\_GPIO\_PINS) }

## Functions

- void [SPI\\_Init](#) (void)  
*Initialize SSI0 to act as an SPI Controller (AKA Master) in mode 0.*
- uint8\_t [SPI\\_Read](#) (void)  
*Read data from the serial port.*
- void [SPI\\_WriteCmd](#) (uint8\_t cmd)  
*Write a command to the serial port.*
- void [SPI\\_WriteData](#) (uint8\_t data)  
*Write data to the serial port.*

## Variables

- static register\_t **gpioPortReg** = 0

### 4.4.5.1 Detailed Description

Functions for SPI-based communication via the SSI peripheral.

#### 4.4.5.2 Function Documentation

##### SPI\_Init()

```
void SPI_Init (
    void )
```

Initialize SSI0 to act as an SPI Controller (AKA Master) in mode 0.

The bit rate  $BR$  is set using the (positive, even-numbered) clock prescale divisor  $CPSDVSR$  and the  $SCR$  field in the SSI Control 0 ( $CR0$ ) register:

$$BR = f_{bus} / (CPSDVSR * (1 + SCR))$$

The ILI9341 driver has a min. read cycle of 150 [ns] and a min. write cycle of 100 [ns], so the bit rate  $BR$  is set to be equal to the bus frequency (  $f_{bus} = 80[MHz]$  ) divided by 8, allowing a bit rate of 10 [MHz], or a period of 100 [ns].

##### SPI\_Read()

```
uint8_t SPI_Read (
    void )
```

Read data from the serial port.

##### Precondition

Initialize the SPI module.

##### Parameters

out	data	8-bit data received from the hardware's receive FIFO.
-----	------	---

##### SPI\_WriteCmd()

```
void SPI_WriteCmd (
    uint8_t cmd )
```

Write a command to the serial port.

##### Precondition

Initialize the SPI module.

##### Parameters

in	cmd	8-bit command to write.
----	-----	-------------------------

**Postcondition**

The D/C pin is cleared.

The data is added to the hardware's transmit FIFO.

**SPI\_WriteData()**

```
void SPI_WriteData (
    uint8_t data )
```

Write data to the serial port.

**Precondition**

Initialize the SPI module.

**Parameters**

in	data	8-bit data to write.
----	------	----------------------

**Postcondition**

The D/C pin is set.

The data is added to the hardware's transmit FIFO.

**4.4.6 Timer**

Functions for using hardware timers.

Collaboration diagram for Timer:

**Files**

- file [Timer.c](#)  
*Source code for Timer module.*
- file [Timer.h](#)  
*Device driver for general-purpose timer modules.*



## Data Structures

- struct `Timer_t`

## Enumerations

- enum {  
**TIMER0\_BASE** = 0x40030000 , **TIMER1\_BASE** = 0x40031000 , **TIMER2\_BASE** = 0x40032000 , **TIMER3**←  
**\_BASE** = 0x40033000 ,  
**TIMER4\_BASE** = 0x40034000 , **TIMER5\_BASE** = 0x40035000 }
- enum **REGISTER\_OFFSETS** {  
**CONFIG** = 0x00 , **MODE** = 0x04 , **CTRL** = 0x0C , **INT\_MASK** = 0x18 ,  
**INT\_CLEAR** = 0x24 , **INTERVAL** = 0x28 , **VALUE** = 0x054 }
- enum **timerName\_t** {  
**TIMER0** , **TIMER1** , **TIMER2** , **TIMER3** ,  
**TIMER4** , **TIMER5** }
- enum **timerMode\_t** { **ONESHOT** , **PERIODIC** }
- enum **timerDirection\_t** { **UP** , **DOWN** }

## Functions

- `Timer_t Timer_Init` (timerName\_t timerName)  
Initialize a hardware timer.
- void `Timer_Deinit` (Timer\_t timer)  
De-initialize a hardware timer.
- timerName\_t `Timer_getName` (Timer\_t timer)  
Get the name of a timer object.
- bool `Timer_isInit` (Timer\_t timer)  
Check if a timer object is initialized.
- void `Timer_setMode` (Timer\_t timer, timerMode\_t timerMode, timerDirection\_t timerDirection)  
Set the mode for the timer.
- void `Timer_enableAdcTrigger` (Timer\_t timer)  
Set the timer to trigger ADC sample capture once it reaches timeout (i.e. down to 0 or up to its reload value).
- void `Timer_disableAdcTrigger` (Timer\_t timer)  
Disable ADC sample capture on timeout.
- void `Timer_enableInterruptOnTimeout` (Timer\_t timer)  
Set the timer to trigger an interrupt on timeout.
- void `Timer_disableInterruptOnTimeout` (Timer\_t timer)  
Stop the timer from triggering interrupts on timeout.
- void `Timer_clearInterruptFlag` (Timer\_t timer)  
Clear the timer's interrupt flag to acknowledge the interrupt.
- void `Timer_setInterval_ms` (Timer\_t timer, uint32\_t time\_ms)  
Set the interval to use.
- uint32\_t **Timer\_getCurrentValue** (Timer\_t timer)
- void `Timer_Start` (Timer\_t timer)  
Start the timer.
- void `Timer_Stop` (Timer\_t timer)  
Stop the timer.
- bool `Timer_isCounting` (Timer\_t timer)  
Check if the timer is currently counting.
- void `Timer_Wait1ms` (Timer\_t timer, uint32\_t time\_ms)  
Initiate a time delay.

## Variables

- static [TimerStruct\\_t](#) [TIMER\\_POOL](#) [6]

### 4.4.6.1 Detailed Description

Functions for using hardware timers.

### 4.4.6.2 Enumeration Type Documentation

#### timerMode\_t

```
enum timerMode_t
```

##### Enumerator

ONESHOT	the timer runs once, then stops
PERIODIC	the timer runs continuously once started

#### timerDirection\_t

```
enum timerDirection_t
```

##### Enumerator

UP	the timer starts at 0 and counts to the reload value
DOWN	the timer starts at its reload value and counts down

### 4.4.6.3 Function Documentation

#### Timer\_Init()

```
Timer_t Timer_Init (
    timerName_t timerName )
```

Initialize a hardware timer.

##### Parameters

in	<i>timerName</i>	Name of the hardware timer to use.
out	<i>timer</i>	Pointer to timer object.

##### Postcondition

The timer is ready to be configured and used.

See also

[Timer\\_isInit\(\)](#), [Timer\\_Deinit\(\)](#)

### Timer\_Deinit()

```
void Timer_Deinit (
    Timer_t timer )
```

De-initialize a hardware timer.

#### Parameters

in	<i>timerName</i>	Name of the hardware timer to use.
----	------------------	------------------------------------

#### Postcondition

The hardware timer is no longer initialized or receiving power.

See also

[Timer\\_Init\(\)](#), [Timer\\_isInit\(\)](#)

### Timer\_getName()

```
timerName_t Timer_getName (
    Timer_t timer )
```

Get the name of a timer object.

#### Parameters

in	<i>timer</i>	Pointer to timer object.
out	<i>timerName_t</i>	Name of the hardware timer being used.

### Timer\_isInit()

```
bool Timer_isInit (
    Timer_t timer )
```

Check if a timer object is initialized.

#### Parameters

in	<i>timer</i>	Pointer to timer object.
out	<i>true</i>	The timer is initialized.
out	<i>false</i>	The timer is not initialized.

See also

[Timer\\_Init\(\)](#), [Timer\\_Deinit\(\)](#)

### Timer\_setMode()

```
void Timer_setMode (
    Timer_t timer,
    timerMode_t timerMode,
    timerDirection_t timerDirection )
```

Set the mode for the timer.

#### Parameters

in	<i>timer</i>	Pointer to timer object.
in	<i>timerMode</i>	Mode for hardware timer to use.
in	<i>timerDirection</i>	Direction to count towards.

### Timer\_enableAdcTrigger()

```
void Timer_enableAdcTrigger (
    Timer_t timer )
```

Set the timer to trigger ADC sample capture once it reaches timeout (i.e. down to 0 or up to its reload value).

#### Precondition

Initialize and configure an ADC module to be timer-triggered.

#### Parameters

in	<i>timer</i>	Pointer to timer object.
----	--------------	--------------------------

#### Postcondition

A timeout event triggers ADC sample capture.

See also

[Timer\\_disableAdcTrigger\(\)](#)

### Timer\_disableAdcTrigger()

```
void Timer_disableAdcTrigger (
    Timer_t timer )
```

Disable ADC sample capture on timeout.

**Precondition**

Initialize and configure an ADC module to be timer-triggered.

**Parameters**

in	<i>timer</i>	Pointer to timer object.
----	--------------	--------------------------

**Postcondition**

A timeout event no longer triggers ADC sample capture.

**See also**

[Timer\\_enableAdcTrigger\(\)](#)

**Timer\_enableInterruptOnTimeout()**

```
void Timer_enableInterruptOnTimeout (
    Timer_t timer )
```

Set the timer to trigger an interrupt on timeout.

**Precondition**

Configure the interrupt service routine using the ISR module.

**Parameters**

in	<i>timer</i>	Pointer to timer object.
----	--------------	--------------------------

**Postcondition**

Upon timeout, an interrupt is triggered.

**See also**

[Timer\\_disableInterruptOnTimeout\(\)](#)

**Timer\_disableInterruptOnTimeout()**

```
void Timer_disableInterruptOnTimeout (
    Timer_t timer )
```

Stop the timer from triggering interrupts on timeout.

**Parameters**

in	<i>timer</i>	Pointer to timer object.
----	--------------	--------------------------

**Postcondition**

Timeout no longer triggers ADC sample capture.

**See also**

[Timer\\_enableInterruptOnTimeout\(\)](#)

**Timer\_clearInterruptFlag()**

```
void Timer_clearInterruptFlag (
    Timer_t timer )
```

Clear the timer's interrupt flag to acknowledge the interrupt.

**Precondition**

Call this during a timer's interrupt service routine (ISR).

**Parameters**

in	<i>timer</i>	Pointer to timer object.
----	--------------	--------------------------

**Timer\_setInterval\_ms()**

```
void Timer_setInterval_ms (
    Timer_t timer,
    uint32_t time_ms )
```

Set the interval to use.

**Precondition**

Initialize and configure the timer.

**Parameters**

in	<i>timer</i>	Pointer to timer object.
in	<i>time_ms</i>	Time in [ms].

**Postcondition**

Upon starting, the Timer counts down from or up to this value.

**See also**

[Timer\\_Init\(\)](#), [Timer\\_setMode\(\)](#)

**Timer\_Start()**

```
void Timer_Start (
    Timer_t timer )
```

Start the timer.

**Precondition**

Initialize and configure the timer.

**Parameters**

in	<i>timer</i>	Pointer to timer object.
----	--------------	--------------------------

**Postcondition**

The timer is counting.

**See also**

[Timer\\_Stop\(\)](#), [Timer\\_isCounting\(\)](#)

**Timer\_Stop()**

```
void Timer_Stop (
    Timer_t timer )
```

Stop the timer.

**Precondition**

Start the timer.

**Parameters**

in	<i>timer</i>	Pointer to timer object.
----	--------------	--------------------------

**Postcondition**

The timer is no longer counting.

**See also**

[Timer\\_Start\(\)](#), [Timer\\_isCounting\(\)](#)



**Timer\_isCounting()**

```
bool Timer_isCounting (
    Timer_t timer )
```

Check if the timer is currently counting.

**Parameters**

in	<i>timer</i>	Pointer to timer object.
out	<i>true</i>	The timer is counting.
out	<i>false</i>	The timer is not counting.

See also

[Timer\\_Start\(\)](#), [Timer\\_Stop\(\)](#)

**Timer\_Wait1ms()**

```
void Timer_Wait1ms (
    Timer_t timer,
    uint32_t time_ms )
```

Initiate a time delay.

**Precondition**

Initialize and configure the timer.

**Parameters**

in	<i>timer</i>	Pointer to timer object.
in	<i>time_ms</i>	Time in [ms] to wait for.

**Postcondition**

The program is delayed for the desired time.

**4.4.6.4 Variable Documentation****TIMER\_POOL**

```
TimerStruct_t TIMER_POOL[6] [static]
```

**Initial value:**

```
= {
    { TIMER0, TIMER0_BASE, (register_t) (TIMER0_BASE + CTRL), (register_t) (TIMER0_BASE + INTERVAL),
      (register_t) (TIMER0_BASE + INT_CLEAR), false },
    { TIMER1, TIMER1_BASE, (register_t) (TIMER1_BASE + CTRL), (register_t) (TIMER1_BASE + INTERVAL),
      (register_t) (TIMER1_BASE + INT_CLEAR), false },
```

```

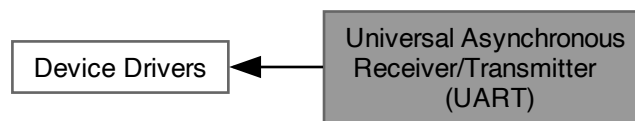
{ TIMER2, TIMER2_BASE, (register_t) (TIMER2_BASE + CTRL), (register_t) (TIMER2_BASE + INTERVAL),
  (register_t) (TIMER2_BASE + INT_CLEAR), false },
{ TIMER3, TIMER3_BASE, (register_t) (TIMER3_BASE + CTRL), (register_t) (TIMER3_BASE + INTERVAL),
  (register_t) (TIMER3_BASE + INT_CLEAR), false },
{ TIMER4, TIMER4_BASE, (register_t) (TIMER4_BASE + CTRL), (register_t) (TIMER4_BASE + INTERVAL),
  (register_t) (TIMER4_BASE + INT_CLEAR), false },
{ TIMER5, TIMER5_BASE, (register_t) (TIMER5_BASE + CTRL), (register_t) (TIMER5_BASE + INTERVAL),
  (register_t) (TIMER5_BASE + INT_CLEAR), false }
}

```

#### 4.4.7 Universal Asynchronous Receiver/Transmitter (UART)

Functions for serial communication via the UART peripheral.

Collaboration diagram for Universal Asynchronous Receiver/Transmitter (UART):



#### Files

- file [UART.c](#)  
Source code for UART module.
- file [UART.h](#)  
Driver module for serial communication via UART0 and UART 1.

#### Data Structures

- struct [Uart\\_t](#)

#### Macros

- #define **ASCII\_CONVERSION** 0x30

#### Enumerations

- enum **GPIO\_BASE\_ADDRESSES** {  
**GPIO\_PORTA\_BASE** = (uint32\_t) 0x40004000 , **GPIO\_PORTB\_BASE** = (uint32\_t) 0x40005000 , **GPIO\_PORTC\_BASE** = (uint32\_t) 0x40006000 , **GPIO\_PORTD\_BASE** = (uint32\_t) 0x40007000 ,  
**GPIO\_PORTE\_BASE** = (uint32\_t) 0x40024000 , **GPIO\_PORTF\_BASE** = (uint32\_t) 0x40025000 }
- enum **UART\_BASE\_ADDRESSES** {  
**UART0\_BASE** = (uint32\_t) 0x4000C000 , **UART1\_BASE** = (uint32\_t) 0x4000D000 , **UART2\_BASE** = (uint32\_t) 0x4000E000 , **UART3\_BASE** = (uint32\_t) 0x4000F000 ,  
**UART4\_BASE** = (uint32\_t) 0x40010000 , **UART5\_BASE** = (uint32\_t) 0x40011000 , **UART6\_BASE** = (uint32\_t) 0x40012000 , **UART7\_BASE** = (uint32\_t) 0x40013000 }
- enum **UART\_REG\_OFFSETS** {  
**UART\_FR\_R\_OFFSET** = (uint32\_t) 0x18 , **IBRD\_R\_OFFSET** = (uint32\_t) 0x24 , **FBRD\_R\_OFFSET** = (uint32\_t) 0x28 , **LCRH\_R\_OFFSET** = (uint32\_t) 0x2C ,  
**CTL\_R\_OFFSET** = (uint32\_t) 0x30 , **CC\_R\_OFFSET** = (uint32\_t) 0xFC8 }
- enum **uartNum\_t** {  
**UART0** , **UART1** , **UART2** , **UART3** ,  
**UART4** , **UART5** , **UART6** , **UART7** }

## Functions

- `Uart_t UART_Init` (GpioPort\_t port, uartNum\_t uartNum)  
*Initialize the specified UART peripheral.*
- `bool UART_IsInit` (Uart\_t uart)  
*Check if the UART object is initialized.*
- `unsigned char UART_ReadChar` (Uart\_t uart)  
*Read a single ASCII character from the UART.*
- `void UART_WriteChar` (Uart\_t uart, unsigned char inputChar)  
*Write a single character to the UART.*
- `void UART_WriteStr` (Uart\_t uart, void \*inputStr)  
*Write a C string to the UART.*
- `void UART_WriteInt` (Uart\_t uart, int32\_t n)  
*Write a 32-bit unsigned integer the UART.*
- `void UART_WriteFloat` (Uart\_t uart, double n, uint8\_t numDecimals)  
*Write a floating-point number the UART.*

## Variables

- static `UartStruct_t UART_ARR` [8]

### 4.4.7.1 Detailed Description

Functions for serial communication via the UART peripheral.

### 4.4.7.2 Function Documentation

#### UART\_Init()

```
Uart_t UART_Init (
    GpioPort_t port,
    uartNum_t uartNum )
```

Initialize the specified UART peripheral.

#### Parameters

in	<i>port</i>	GPIO port to use.
in	<i>uartNum</i>	UART number. Should be either one of the enumerated constants or an int in range [0, 7].
out	<i>uart</i>	(Pointer to) initialized UART peripheral.

Given the bus frequency ( $f_{bus}$ ) and desired baud rate (BR), the baud rate divisor (BRD) can be calculated:  
 $BRD = f_{bus} / (16 * BR)$

The integer BRD (IBRD) is simply the integer part of the BRD:  $IBRD = int(BRD)$

The fractional BRD (FBRD) is calculated using the fractional part ( $mod(BRD, 1)$ ) of the BRD:  $FBRD = int((mod(BRD, 1) * 64) + 0.5)$

**UART\_isInit()**

```
bool UART_isInit (
    Uart_t uart )
```

Check if the UART object is initialized.

**Parameters**

in	<i>uart</i>	UART to check.
out	<i>true</i>	The UART object is initialized.
out	<i>false</i>	The UART object is not initialized.

**UART\_ReadChar()**

```
unsigned char UART_ReadChar (
    Uart_t uart )
```

Read a single ASCII character from the UART.

**Parameters**

in	<i>uart</i>	UART to read from.
out	<i>unsigned</i>	char ASCII character from sender.

**UART\_WriteChar()**

```
void UART_WriteChar (
    Uart_t uart,
    unsigned char inputChar )
```

Write a single character to the UART.

**Parameters**

in	<i>uart</i>	UART to write to.
in	<i>input_char</i>	ASCII character to send.

**UART\_WriteStr()**

```
void UART_WriteStr (
    Uart_t uart,
    void * inputStr )
```

Write a C string to the UART.

## Parameters

in	<i>uart</i>	UART to write to.
in	<i>input_str</i>	Array of ASCII characters.

**UART\_WriteInt()**

```
void UART_WriteInt (
    Uart_t uart,
    int32_t n )
```

Write a 32-bit unsigned integer the UART.

## Parameters

in	<i>uart</i>	UART to write to.
in	<i>n</i>	Unsigned 32-bit <code>int</code> to be converted and transmitted.

**UART\_WriteFloat()**

```
void UART_WriteFloat (
    Uart_t uart,
    double n,
    uint8_t numDecimals )
```

Write a floating-point number the UART.

## Parameters

in	<i>uart</i>	UART to write to.
in	<i>n</i>	Floating-point number to be converted and transmitted.
in	<i>num_decimals</i>	Number of digits after the decimal point to include.

**4.4.7.3 Variable Documentation****UART\_ARR**

```
UartStruct_t UART_ARR[8] [static]
```

**Initial value:**

```
= {
    { UART0_BASE, ((register_t) (UART0_BASE + UART_FR_R_OFFSET)), 0, GPIO_PIN0, GPIO_PIN1, false },
    { UART1_BASE, ((register_t) (UART1_BASE + UART_FR_R_OFFSET)), 0, GPIO_PIN0, GPIO_PIN1, false },
    { UART2_BASE, ((register_t) (UART2_BASE + UART_FR_R_OFFSET)), 0, GPIO_PIN6, GPIO_PIN7, false },
    { UART3_BASE, ((register_t) (UART3_BASE + UART_FR_R_OFFSET)), 0, GPIO_PIN6, GPIO_PIN7, false },
    { UART4_BASE, ((register_t) (UART4_BASE + UART_FR_R_OFFSET)), 0, GPIO_PIN4, GPIO_PIN5, false },
    { UART5_BASE, ((register_t) (UART5_BASE + UART_FR_R_OFFSET)), 0, GPIO_PIN4, GPIO_PIN5, false },
    { UART6_BASE, ((register_t) (UART6_BASE + UART_FR_R_OFFSET)), 0, GPIO_PIN4, GPIO_PIN5, false },
    { UART7_BASE, ((register_t) (UART7_BASE + UART_FR_R_OFFSET)), 0, GPIO_PIN0, GPIO_PIN1, false }
}
```

#### 4.4.8 Interrupt Service Routines

Functions for manipulating the interrupt vector table and setting up interrupt handlers via the NVIC.

Collaboration diagram for Interrupt Service Routines:



#### Files

- file [ISR.c](#)  
*Source code for interrupt service routine (ISR) configuration module.*
- file [ISR.h](#)  
*Header file for interrupt service routine (ISR) configuration module.*

#### Macros

- `#define VECTOR_TABLE_BASE_ADDR ((uint32_t) 0x00000000)`
- `#define VECTOR_TABLE_SIZE ((uint32_t) 155)`
- `#define VECTOR_TABLE_ALIGNMENT ((uint32_t) (1 << 10))`
- `#define NVIC_EN_BASE_ADDR ((uint32_t) 0xE000E100)`
- `#define NVIC_DIS_BASE_ADDR ((uint32_t) 0xE000E180)`
- `#define NVIC_PRI_BASE_ADDR ((uint32_t) 0xE000E400)`
- `#define NVIC_UNPEND_BASE_ADDR ((uint32_t) 0xE000E280)`

#### Typedefs

- `typedef void(* ISR_t) (void)`  
*Type definition for function pointers representing ISRs.*

#### Functions

- `static void ISR_setStatus (const uint8_t vectorNum, const bool isEnabled)`
- `void ISR\_GlobalDisable (void)`  
*Disable all interrupts globally.*
- `void ISR\_GlobalEnable (void)`  
*Enable all interrupts globally.*
- `static ISR\_t newVectorTable[VECTOR_TABLE_SIZE] __attribute__ ((aligned(VECTOR_TABLE_  
ALIGNMENT)))`
- `void ISR\_InitNewTableInRam (void)`  
*Relocate the vector table to RAM.*
- `void ISR\_addToIntTable (ISR\_t isr, const uint8_t vectorNum)`

- Add an ISR to the interrupt table.*
  - void [ISR\\_setPriority](#) (const uint8\_t vectorNum, const uint8\_t priority)
- Set the priority for an interrupt.*
  - void [ISR\\_Enable](#) (const uint8\_t vectorNum)
- Enable an interrupt in the NVIC.*
  - void [ISR\\_Disable](#) (const uint8\_t vectorNum)
- Disable an interrupt in the NVIC.*
  - void [ISR\\_triggerInterrupt](#) (const uint8\_t vectorNum)
- Generate a software-generated interrupt (SGI).*

## Variables

- static bool **interruptsAreEnabled** = true
- void(\*const **interruptVectorTable** [ ])(void)
- static bool **isTableCopiedToRam** = false

### 4.4.8.1 Detailed Description

Functions for manipulating the interrupt vector table and setting up interrupt handlers via the NVIC.

### 4.4.8.2 Function Documentation

#### ISR\_GlobalDisable()

```
void ISR_GlobalDisable (  
    void )
```

Disable all interrupts globally.

See also

[ISR\\_GlobalEnable\(\)](#)

#### ISR\_GlobalEnable()

```
void ISR_GlobalEnable (  
    void )
```

Enable all interrupts globally.

See also

[ISR\\_GlobalDisable\(\)](#)

### ISR\_InitNewTableInRam()

```
void ISR_InitNewTableInRam (
    void )
```

Relocate the vector table to RAM.

#### Precondition

Disable interrupts globally before calling this.

#### Postcondition

The vector table is now located in RAM, allowing the ISRs listed in the startup file to be replaced.

#### See also

[ISR\\_GlobalDisable\(\)](#), [ISR\\_addToIntTable\(\)](#)

### ISR\_addToIntTable()

```
void ISR_addToIntTable (
    ISR_t isr,
    const uint8_t vectorNum )
```

Add an ISR to the interrupt table.

#### Precondition

Initialize a new vector table in RAM before calling this function.

#### Parameters

in	<i>isr</i>	Name of the ISR to add.
in	<i>vectorNum</i>	ISR's vector number (i.e. offset from the top of the table). Should be in range [16, 154].

#### Postcondition

The ISR is now added to the vector table and available to be called.

#### See also

[ISR\\_InitNewTableInRam\(\)](#)

### ISR\_setPriority()

```
void ISR_setPriority (
    const uint8_t vectorNum,
    const uint8_t priority )
```

Set the priority for an interrupt.



**Precondition**

Disable the interrupt before adjusting its priority.

**Parameters**

in	<i>vectorNum</i>	ISR's vector number (i.e. offset from the top of the table). Should be in range [16, 154].
in	<i>priority</i>	Priority to assign. Highest priority is 0, lowest is 7.

**Postcondition**

The interrupt's priority has now been changed in the NVIC.

**See also**

[ISR\\_Disable\(\)](#)

**ISR\_Enable()**

```
void ISR_Enable (
    const uint8_t vectorNum )
```

Enable an interrupt in the NVIC.

**Precondition**

If needed, add the interrupt to the vector table.

If needed, set the interrupt's priority (default 0, or highest priority) before calling this.

**Parameters**

in	<i>vectorNum</i>	ISR's vector number (i.e. offset from the top of the table). Should be in range [16, 154].
----	------------------	--

**Postcondition**

The interrupt is now enabled in the NVIC.

**See also**

[ISR\\_addToIntTable\(\)](#), [ISR\\_setPriority\(\)](#), [ISR\\_Disable\(\)](#)

**ISR\_Disable()**

```
void ISR_Disable (
    const uint8_t vectorNum )
```

Disable an interrupt in the NVIC.

**Parameters**

in	<i>vectorNum</i>	ISR's vector number (i.e. offset from the top of the table). Should be in range [16, 154].
----	------------------	--

**Postcondition**

The interrupt is now disabled in the NVIC.

**See also**

[ISR\\_Enable\(\)](#)

**ISR\_triggerInterrupt()**

```
void ISR_triggerInterrupt (
    const uint8_t vectorNum )
```

Generate a software-generated interrupt (SGI).

**Precondition**

Enable the ISR (and set priority as needed).

Enable all interrupts.

**Parameters**

in	<i>vectorNum</i>	ISR's vector number (i.e. offset from the top of the table). Should be in range [16, 154].
----	------------------	--

**Postcondition**

The ISR should trigger once any higher priority ISRs return.

**See also**

[ISR\\_clearPending\(\)](#)

## 5 Data Structure Documentation

### 5.1 Fifo\_t Struct Reference

**Data Fields**

- volatile uint32\_t \* **buffer**  
*(pointer to) array to use as FIFO buffer*
- volatile uint32\_t **N**  
*length of buffer*

- volatile uint32\_t **frontIdx**  
*idx of front of FIFO*
- volatile uint32\_t **backIdx**  
*idx of back of FIFO*

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

- [Fifo.c](#)

## 5.2 GpioPort\_t Struct Reference

### Data Fields

- const uint32\_t **BASE\_ADDRESS**
- const uint32\_t **DATA\_REGISTER**
- bool **isInit**

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

- [GPIO.c](#)

## 5.3 Led\_t Struct Reference

### Data Fields

- GpioPort\_t **GPIO\_PORT\_PTR**  
*pointer to GPIO port data structure*
- GpioPin\_t **GPIO\_PIN**  
*GPIO pin number.*
- volatile uint32\_t \* **gpioDataRegister**
- bool **isOn**  
*state indicator*
- bool **isInit**

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

- [Led.c](#)

## 5.4 Timer\_t Struct Reference

### Data Fields

- const timerName\_t **NAME**
- const uint32\_t **BASE\_ADDR**
- register\_t **controlRegister**
- register\_t **intervalLoadRegister**
- register\_t **interruptClearRegister**
- bool **isInit**

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

- [Timer.c](#)

## 5.5 Uart\_t Struct Reference

### Data Fields

- const uint32\_t **BASE\_ADDRESS**
- register\_t **FLAG\_R\_ADDRESS**
- GpioPort\_t **GPIO\_PORT**  
*pointer to GPIO port data structure*
- GpioPin\_t **RX\_PIN\_NUM**  
*GPIO pin number.*
- GpioPin\_t **TX\_PIN\_NUM**  
*GPIO pin number.*
- bool **isInit**

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

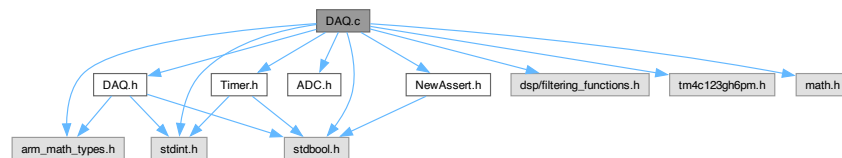
- [UART.c](#)

## 6 File Documentation

### 6.1 DAQ.c File Reference

Source code for DAQ module.

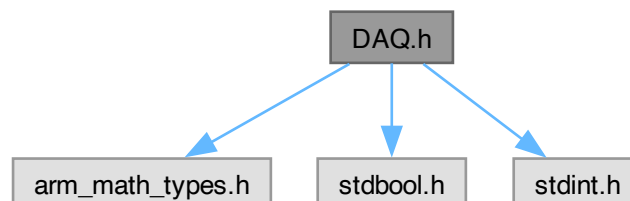
Include dependency graph for DAQ.c:



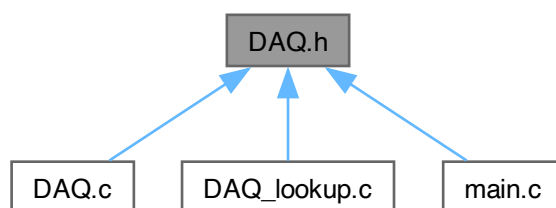
### 6.2 DAQ.h File Reference

Application software for handling data acquisition (DAQ) functions.

Include dependency graph for DAQ.h:



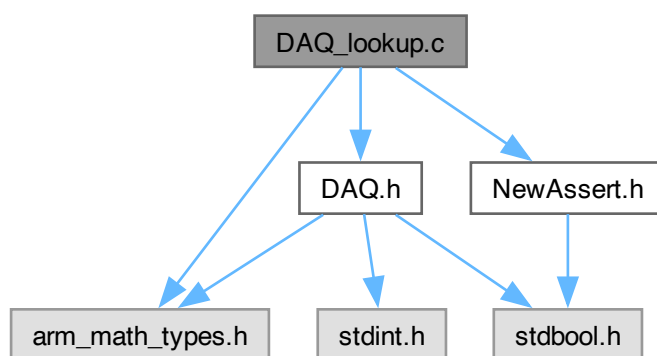
This graph shows which files directly or indirectly include this file:



### 6.3 DAQ\_lookup.c File Reference

Source code for DAQ module's lookup table.

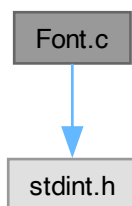
Include dependency graph for DAQ\_lookup.c:



### 6.4 Font.c File Reference

Contains bitmaps for a selection of ASCII characters.

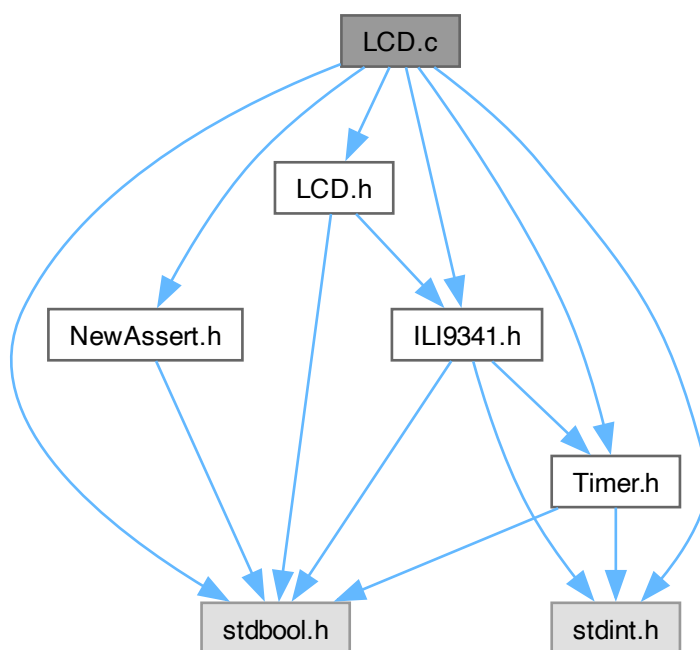
Include dependency graph for Font.c:



## 6.5 LCD.c File Reference

Source code for LCD module.

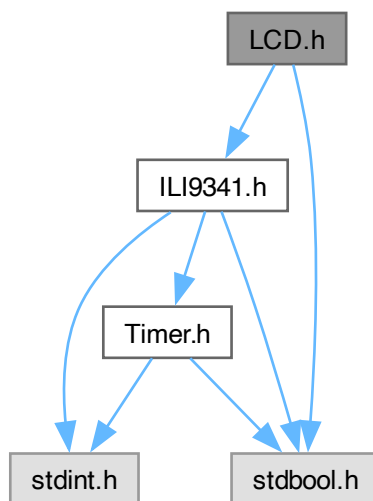
Include dependency graph for LCD.c:



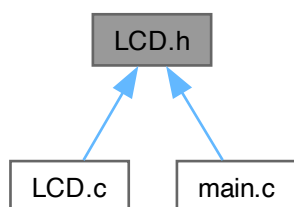
## 6.6 LCD.h File Reference

Header file for LCD module.

Include dependency graph for LCD.h:



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



## 6.7 QRS.c File Reference

Source code for QRS detection module.

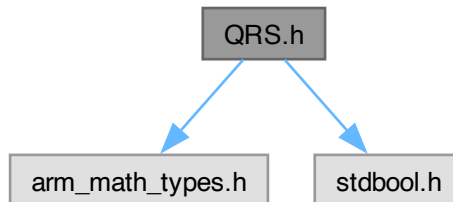
Include dependency graph for QRS.c:



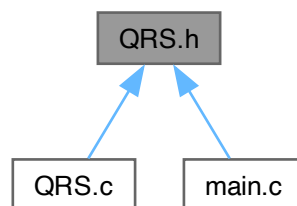
## 6.8 QRS.h File Reference

Header file for QRS detection module.

Include dependency graph for QRS.h:



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

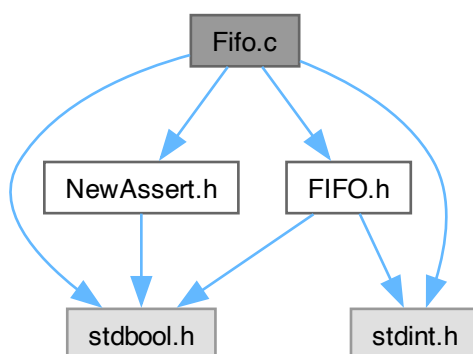


## 6.9 Fifo.c File Reference

Source code for FIFO buffer module.



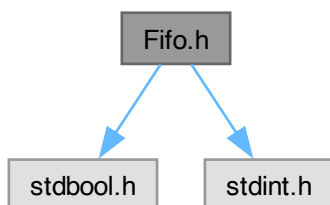
Include dependency graph for Fifo.c:



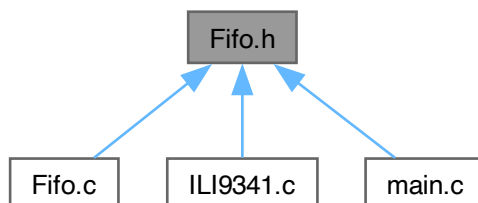
## 6.10 Fifo.h File Reference

Header file for FIFO buffer implementation.

Include dependency graph for Fifo.h:



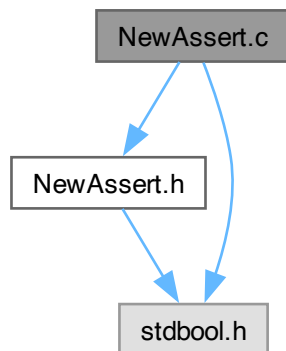
This graph shows which files directly or indirectly include this file:



## 6.11 NewAssert.c File Reference

Source code for custom `assert` implementation.

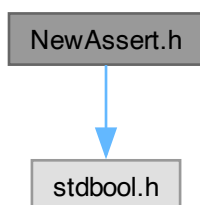
Include dependency graph for NewAssert.c:



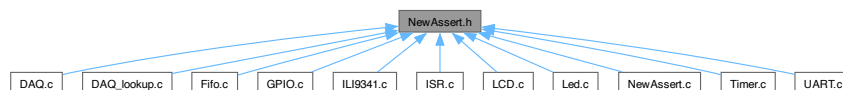
## 6.12 NewAssert.h File Reference

Header file for custom `assert` implementation.

Include dependency graph for NewAssert.h:



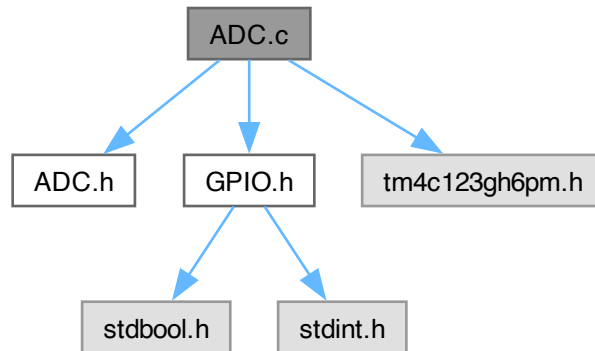
This graph shows which files directly or indirectly include this file:



## 6.13 ADC.c File Reference

Source code for analog-to-digital conversion (ADC) module.

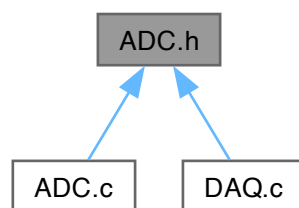
Include dependency graph for ADC.c:



## 6.14 ADC.h File Reference

Header file for analog-to-digital conversion (ADC) module.

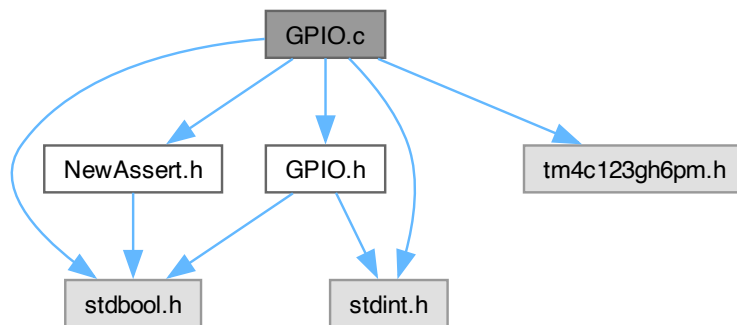
This graph shows which files directly or indirectly include this file:



## 6.15 GPIO.c File Reference

Source code for GPIO module.

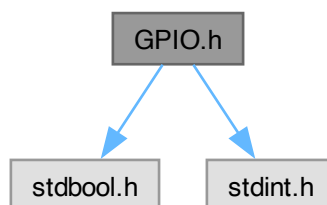
Include dependency graph for GPIO.c:



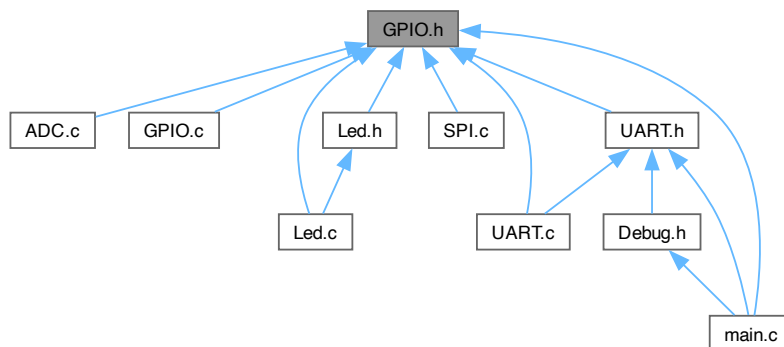
## 6.16 GPIO.h File Reference

Header file for general-purpose input/output (GPIO) device driver.

Include dependency graph for GPIO.h:



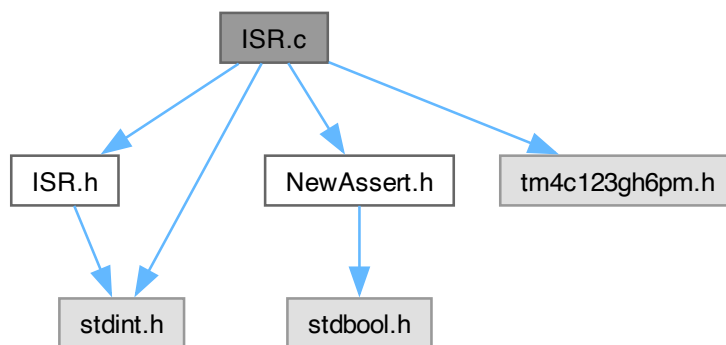
This graph shows which files directly or indirectly include this file:



## 6.17 ISR.c File Reference

Source code for interrupt service routine (ISR) configuration module.

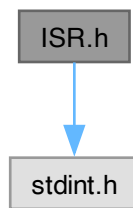
Include dependency graph for `ISR.c`:



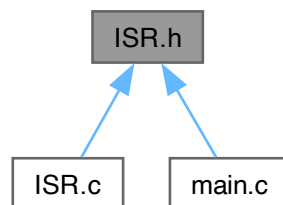
## 6.18 ISR.h File Reference

Header file for interrupt service routine (ISR) configuration module.

Include dependency graph for ISR.h:



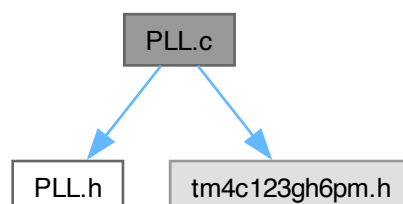
This graph shows which files directly or indirectly include this file:



## 6.19 PLL.c File Reference

Implementation details for phase-lock-loop (PLL) functions.

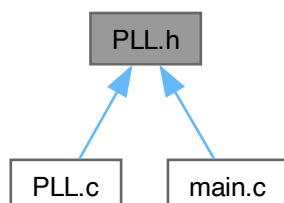
Include dependency graph for PLL.c:



## 6.20 PLL.h File Reference

Driver module for activating the phase-locked-loop (PLL).

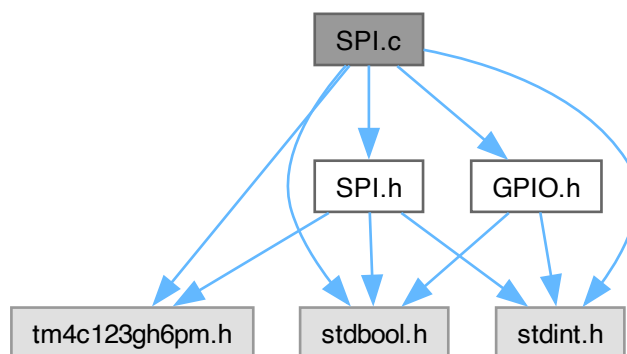
This graph shows which files directly or indirectly include this file:



## 6.21 SPI.c File Reference

Source code for serial peripheral interface (SPI) module.

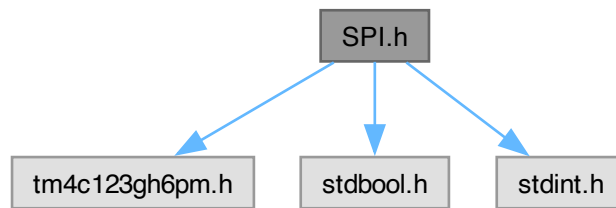
Include dependency graph for `SPI.c`:



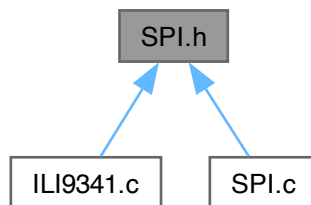
## 6.22 SPI.h File Reference

Header file for serial peripheral interface (SPI) module.

Include dependency graph for SPI.h:



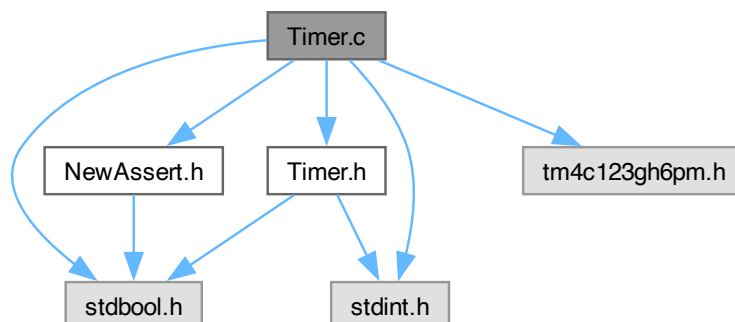
This graph shows which files directly or indirectly include this file:



## 6.23 Timer.c File Reference

Source code for Timer module.

Include dependency graph for Timer.c:

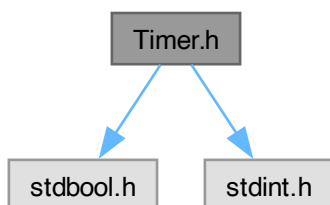




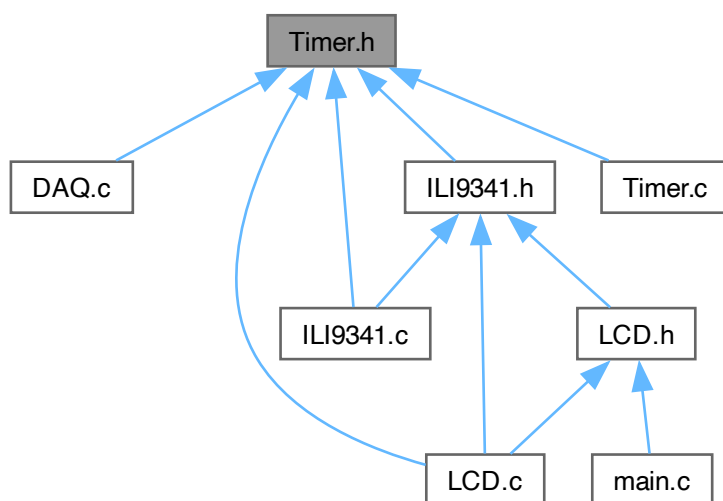
## 6.24 Timer.h File Reference

Device driver for general-purpose timer modules.

Include dependency graph for Timer.h:



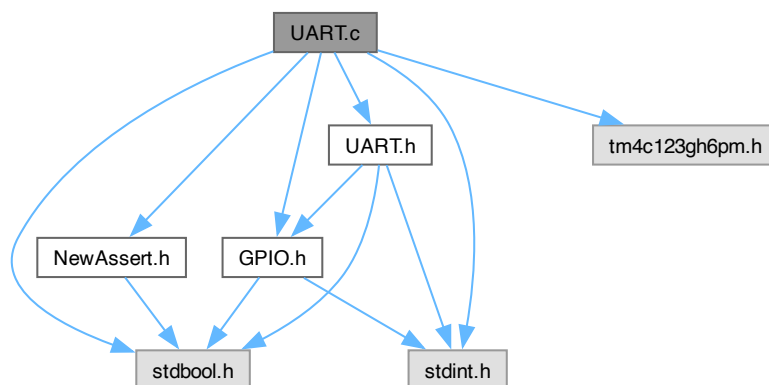
This graph shows which files directly or indirectly include this file:



## 6.25 UART.c File Reference

Source code for UART module.

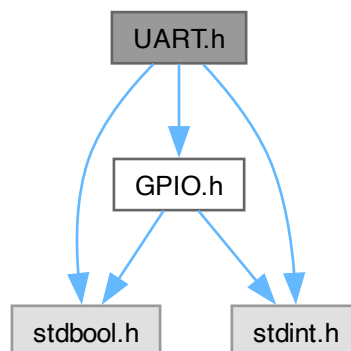
Include dependency graph for UART.c:



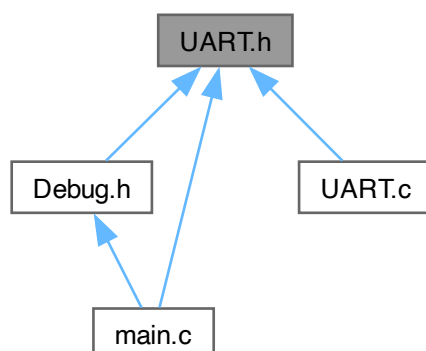
## 6.26 UART.h File Reference

Driver module for serial communication via UART0 and UART 1.

Include dependency graph for UART.h:



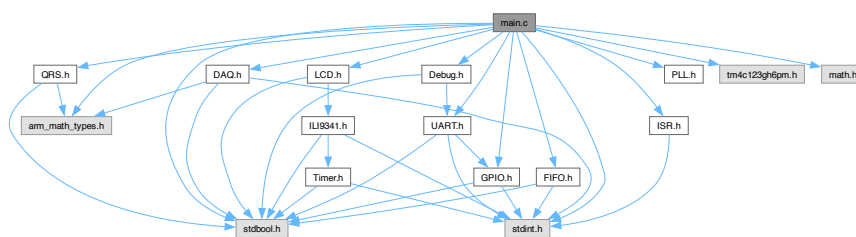
This graph shows which files directly or indirectly include this file:



## 6.27 main.c File Reference

Main program file.

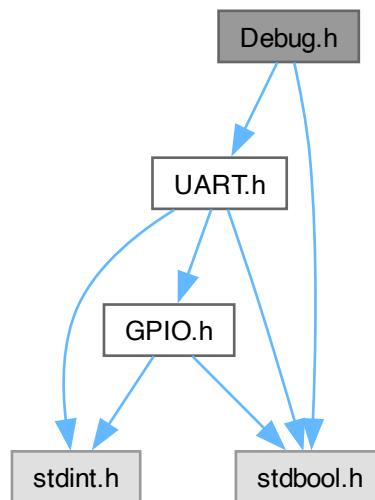
Include dependency graph for `main.c`:



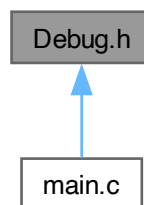
## 6.28 Debug.h File Reference

Functions to output debugging information to a serial port via UART.

Include dependency graph for Debug.h:



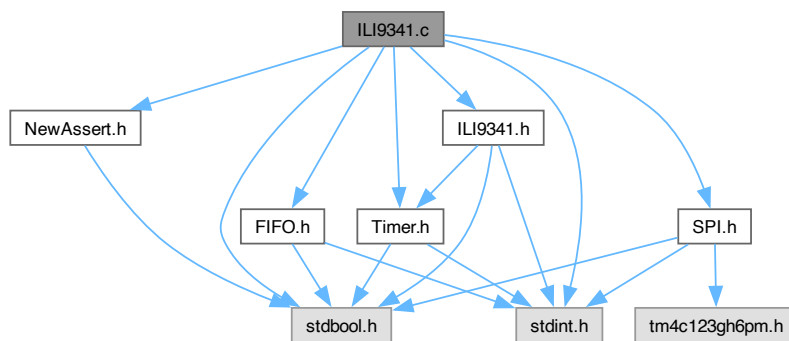
This graph shows which files directly or indirectly include this file:



## 6.29 ILI9341.c File Reference

Source code for ILI9341 module.

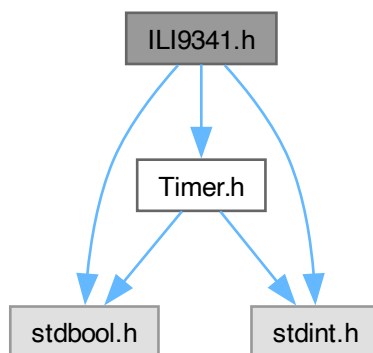
Include dependency graph for ILI9341.c:



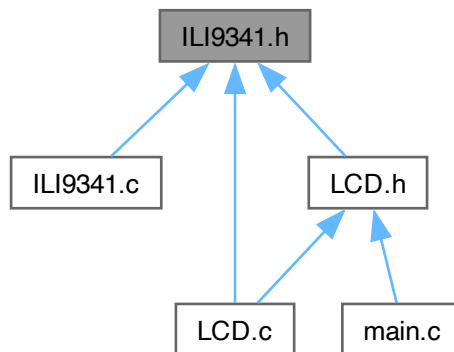
## 6.30 ILI9341.h File Reference

Driver module for interfacing with an ILI9341 LCD driver.

Include dependency graph for ILI9341.h:



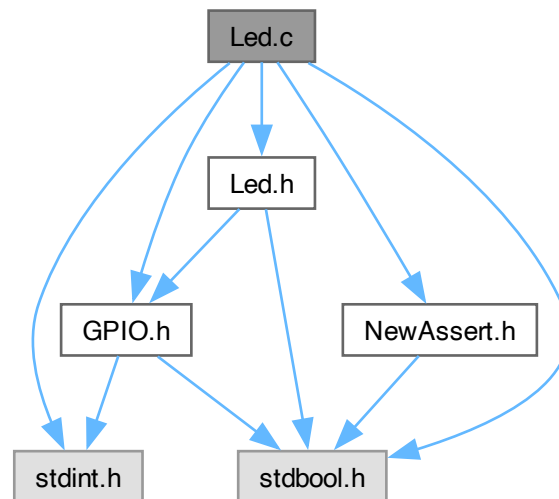
This graph shows which files directly or indirectly include this file:



### 6.31 Led.c File Reference

Source code for LED module.

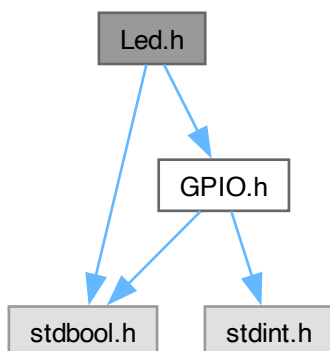
Include dependency graph for `Led.c`:



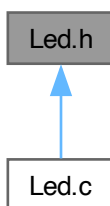
### 6.32 Led.h File Reference

Interface for LED module.

Include dependency graph for Led.h:



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







## Index

- ADC.c, [121](#)
- ADC.h, [121](#)
- ADC\_Init
  - Analog-to-Digital Conversion (ADC), [81](#)
- Analog-to-Digital Conversion (ADC), [80](#)
  - ADC\_Init, [81](#)
- Application Software, [4](#)
- Assert
  - Common, [51](#)
- bandpassFiltStruct
  - Data Acquisition (DAQ), [11](#)
- CASET
  - ILI9341, [66](#)
- Cmd\_t
  - ILI9341, [65](#)
- COEFF\_BANDPASS
  - QRS Detector, [42](#)
- COEFF\_DERFILT
  - QRS Detector, [43](#)
- COEFF\_MOVAVG
  - QRS Detector, [43](#)
- COEFFS\_BANDPASS
  - Data Acquisition (DAQ), [11](#)
- COEFFS\_NOTCH
  - Data Acquisition (DAQ), [11](#)
- Common, [50](#)
  - Assert, [51](#)
- DAQ.c, [114](#)
- DAQ.h, [114](#)
- DAQ\_acknowledgeInterrupt
  - Data Acquisition (DAQ), [8](#)
- DAQ\_ARRAY\_LEN
  - Main, [45](#)
- DAQ\_BandpassFilter
  - Data Acquisition (DAQ), [9](#)
- DAQ\_convertToMilliVolts
  - Data Acquisition (DAQ), [10](#)
- DAQ\_FIFO\_CAP
  - Main, [45](#)
- DAQ\_Handler
  - Main, [45](#)
- DAQ\_Init
  - Data Acquisition (DAQ), [7](#)
- DAQ\_lookup.c, [115](#)
- DAQ\_NotchFilter
  - Data Acquisition (DAQ), [8](#)
- DAQ\_readSample
  - Data Acquisition (DAQ), [7](#)
- DAQ\_VECTOR\_NUM
  - Main, [45](#)
- Data Acquisition (DAQ), [5](#)
  - bandpassFiltStruct, [11](#)
  - COEFFS\_BANDPASS, [11](#)
  - COEFFS\_NOTCH, [11](#)
  - DAQ\_acknowledgeInterrupt, [8](#)
  - DAQ\_BandpassFilter, [9](#)
  - DAQ\_convertToMilliVolts, [10](#)
  - DAQ\_Init, [7](#)
  - DAQ\_NotchFilter, [8](#)
  - DAQ\_readSample, [7](#)
- Debug, [60](#)
  - Debug\_Assert, [63](#)
  - Debug\_Init, [61](#)
  - Debug\_SendFromList, [62](#)
  - Debug\_SendMsg, [61](#)
  - Debug\_WriteFloat, [62](#)
- Debug.h, [129](#)
- Debug\_Assert
  - Debug, [63](#)
- Debug\_Init
  - Debug, [61](#)
- Debug\_SendFromList
  - Debug, [62](#)
- Debug\_SendMsg
  - Debug, [61](#)
- Debug\_WriteFloat
  - Debug, [62](#)
- Device Drivers, [79](#)
- DINVOFF
  - ILI9341, [66](#)
- DINVON
  - ILI9341, [66](#)
- DISPOFF
  - ILI9341, [66](#)
- DISPON
  - ILI9341, [66](#)
- DOWN
  - Timer, [96](#)
- FIFO Buffers, [52](#)
  - FIFO\_Flush, [55](#)
  - FIFO\_FlushFloat, [57](#)
  - FIFO\_Get, [54](#)
  - FIFO\_getCurrSize, [58](#)
  - FIFO\_GetFloat, [56](#)
  - FIFO\_Init, [53](#)
  - FIFO\_isEmpty, [58](#)
  - FIFO\_isFull, [58](#)
  - FIFO\_PeekAll, [57](#)
  - FIFO\_PeekOne, [57](#)
  - FIFO\_Put, [54](#)
  - FIFO\_PutFloat, [55](#)
  - FIFO\_Reset, [54](#)
- Fifo.c, [118](#)
- Fifo.h, [119](#)
- FIFO\_Flush
  - FIFO Buffers, [55](#)
- FIFO\_FlushFloat

- FIFO Buffers, [57](#)
- FIFO\_Get
  - FIFO Buffers, [54](#)
- FIFO\_getCurrSize
  - FIFO Buffers, [58](#)
- FIFO\_GetFloat
  - FIFO Buffers, [56](#)
- FIFO\_INFO
  - Main, [45](#)
- FIFO\_Init
  - FIFO Buffers, [53](#)
- FIFO\_isEmpty
  - FIFO Buffers, [58](#)
- FIFO\_isFull
  - FIFO Buffers, [58](#)
- FIFO\_PeekAll
  - FIFO Buffers, [57](#)
- FIFO\_PeekOne
  - FIFO Buffers, [57](#)
- FIFO\_Put
  - FIFO Buffers, [54](#)
- FIFO\_PutFloat
  - FIFO Buffers, [55](#)
- FIFO\_Reset
  - FIFO Buffers, [54](#)
- Fifo\_t, [112](#)
- Font.c, [115](#)
- FONT\_0
  - Liquid Crystal Display (LCD), [20](#)
- FONT\_1
  - Liquid Crystal Display (LCD), [20](#)
- FONT\_2
  - Liquid Crystal Display (LCD), [21](#)
- FONT\_3
  - Liquid Crystal Display (LCD), [21](#)
- FONT\_4
  - Liquid Crystal Display (LCD), [21](#)
- FONT\_5
  - Liquid Crystal Display (LCD), [21](#)
- FONT\_6
  - Liquid Crystal Display (LCD), [22](#)
- FONT\_7
  - Liquid Crystal Display (LCD), [22](#)
- FONT\_8
  - Liquid Crystal Display (LCD), [22](#)
- FONT\_9
  - Liquid Crystal Display (LCD), [22](#)
- FONT\_COLON
  - Liquid Crystal Display (LCD), [20](#)
- FONT\_LOWER\_A
  - Liquid Crystal Display (LCD), [29](#)
- FONT\_LOWER\_B
  - Liquid Crystal Display (LCD), [29](#)
- FONT\_LOWER\_C
  - Liquid Crystal Display (LCD), [30](#)
- FONT\_LOWER\_D
  - Liquid Crystal Display (LCD), [30](#)
- FONT\_LOWER\_E
  - Liquid Crystal Display (LCD), [30](#)
- FONT\_LOWER\_F
  - Liquid Crystal Display (LCD), [30](#)
- FONT\_LOWER\_G
  - Liquid Crystal Display (LCD), [31](#)
- FONT\_LOWER\_H
  - Liquid Crystal Display (LCD), [31](#)
- FONT\_LOWER\_I
  - Liquid Crystal Display (LCD), [31](#)
- FONT\_LOWER\_J
  - Liquid Crystal Display (LCD), [31](#)
- FONT\_LOWER\_K
  - Liquid Crystal Display (LCD), [32](#)
- FONT\_LOWER\_L
  - Liquid Crystal Display (LCD), [32](#)
- FONT\_LOWER\_M
  - Liquid Crystal Display (LCD), [32](#)
- FONT\_LOWER\_N
  - Liquid Crystal Display (LCD), [32](#)
- FONT\_LOWER\_O
  - Liquid Crystal Display (LCD), [33](#)
- FONT\_LOWER\_P
  - Liquid Crystal Display (LCD), [33](#)
- FONT\_LOWER\_Q
  - Liquid Crystal Display (LCD), [33](#)
- FONT\_LOWER\_R
  - Liquid Crystal Display (LCD), [33](#)
- FONT\_LOWER\_S
  - Liquid Crystal Display (LCD), [34](#)
- FONT\_LOWER\_T
  - Liquid Crystal Display (LCD), [34](#)
- FONT\_LOWER\_U
  - Liquid Crystal Display (LCD), [34](#)
- FONT\_LOWER\_V
  - Liquid Crystal Display (LCD), [34](#)
- FONT\_LOWER\_W
  - Liquid Crystal Display (LCD), [35](#)
- FONT\_LOWER\_X
  - Liquid Crystal Display (LCD), [35](#)
- FONT\_LOWER\_Y
  - Liquid Crystal Display (LCD), [35](#)
- FONT\_LOWER\_Z
  - Liquid Crystal Display (LCD), [35](#)
- FONT\_PERIOD
  - Liquid Crystal Display (LCD), [20](#)
- FONT\_SPACE
  - Liquid Crystal Display (LCD), [20](#)
- FONT\_UPPER\_A
  - Liquid Crystal Display (LCD), [23](#)
- FONT\_UPPER\_B
  - Liquid Crystal Display (LCD), [23](#)
- FONT\_UPPER\_C
  - Liquid Crystal Display (LCD), [23](#)
- FONT\_UPPER\_D
  - Liquid Crystal Display (LCD), [23](#)
- FONT\_UPPER\_E
  - Liquid Crystal Display (LCD), [24](#)
- FONT\_UPPER\_F
  - Liquid Crystal Display (LCD), [24](#)

- Liquid Crystal Display (LCD), [24](#)
- FONT\_UPPER\_G
  - Liquid Crystal Display (LCD), [24](#)
- FONT\_UPPER\_H
  - Liquid Crystal Display (LCD), [24](#)
- FONT\_UPPER\_I
  - Liquid Crystal Display (LCD), [25](#)
- FONT\_UPPER\_J
  - Liquid Crystal Display (LCD), [25](#)
- FONT\_UPPER\_K
  - Liquid Crystal Display (LCD), [25](#)
- FONT\_UPPER\_L
  - Liquid Crystal Display (LCD), [25](#)
- FONT\_UPPER\_M
  - Liquid Crystal Display (LCD), [26](#)
- FONT\_UPPER\_N
  - Liquid Crystal Display (LCD), [26](#)
- FONT\_UPPER\_O
  - Liquid Crystal Display (LCD), [26](#)
- FONT\_UPPER\_P
  - Liquid Crystal Display (LCD), [26](#)
- FONT\_UPPER\_Q
  - Liquid Crystal Display (LCD), [27](#)
- FONT\_UPPER\_R
  - Liquid Crystal Display (LCD), [27](#)
- FONT\_UPPER\_S
  - Liquid Crystal Display (LCD), [27](#)
- FONT\_UPPER\_T
  - Liquid Crystal Display (LCD), [27](#)
- FONT\_UPPER\_U
  - Liquid Crystal Display (LCD), [28](#)
- FONT\_UPPER\_V
  - Liquid Crystal Display (LCD), [28](#)
- FONT\_UPPER\_W
  - Liquid Crystal Display (LCD), [28](#)
- FONT\_UPPER\_X
  - Liquid Crystal Display (LCD), [28](#)
- FONT\_UPPER\_Y
  - Liquid Crystal Display (LCD), [29](#)
- FONT\_UPPER\_Z
  - Liquid Crystal Display (LCD), [29](#)
- FRMCTR1
  - ILI9341, [66](#)
- FRMCTR2
  - ILI9341, [66](#)
- FRMCTR3
  - ILI9341, [66](#)
- General-Purpose Input/Output (GPIO), [81](#)
  - GPIO\_ConfigAltMode, [89](#)
  - GPIO\_ConfigAnalog, [90](#)
  - GPIO\_ConfigDirInput, [85](#)
  - GPIO\_ConfigDirOutput, [84](#)
  - GPIO\_ConfigDriveStrength, [85](#)
  - GPIO\_ConfigInterrupts\_BothEdges, [87](#)
  - GPIO\_ConfigInterrupts\_Edge, [86](#)
  - GPIO\_ConfigInterrupts\_LevelTrig, [87](#)
  - GPIO\_ConfigNVIC, [87](#)
  - GPIO\_ConfigPortCtrl, [89](#)
  - GPIO\_ConfigPullDown, [85](#)
  - GPIO\_ConfigPullUp, [85](#)
  - GPIO\_DisableDigital, [86](#)
  - GPIO\_EnableDigital, [86](#)
  - GPIO\_getBaseAddr, [84](#)
  - GPIO\_getDataRegister, [88](#)
  - GPIO\_InitPort, [84](#)
  - GPIO\_isPortInit, [84](#)
  - GPIO\_LAUNCHPAD\_LEDS, [83](#)
  - GPIO\_PTR\_ARR, [90](#)
  - GPIO\_ReadPins, [88](#)
  - GPIO\_Toggle, [89](#)
  - GPIO\_WriteHigh, [88](#)
  - GPIO\_WriteLow, [88](#)
  - LED\_BLUE, [83](#)
  - LED\_GREEN, [83](#)
  - LED\_RED, [83](#)
- GPIO.c, [121](#)
- GPIO.h, [122](#)
- GPIO\_ConfigAltMode
  - General-Purpose Input/Output (GPIO), [89](#)
- GPIO\_ConfigAnalog
  - General-Purpose Input/Output (GPIO), [90](#)
- GPIO\_ConfigDirInput
  - General-Purpose Input/Output (GPIO), [85](#)
- GPIO\_ConfigDirOutput
  - General-Purpose Input/Output (GPIO), [84](#)
- GPIO\_ConfigDriveStrength
  - General-Purpose Input/Output (GPIO), [85](#)
- GPIO\_ConfigInterrupts\_BothEdges
  - General-Purpose Input/Output (GPIO), [87](#)
- GPIO\_ConfigInterrupts\_Edge
  - General-Purpose Input/Output (GPIO), [86](#)
- GPIO\_ConfigInterrupts\_LevelTrig
  - General-Purpose Input/Output (GPIO), [87](#)
- GPIO\_ConfigNVIC
  - General-Purpose Input/Output (GPIO), [87](#)
- GPIO\_ConfigPortCtrl
  - General-Purpose Input/Output (GPIO), [89](#)
- GPIO\_ConfigPullDown
  - General-Purpose Input/Output (GPIO), [85](#)
- GPIO\_ConfigPullUp
  - General-Purpose Input/Output (GPIO), [85](#)
- GPIO\_DisableDigital
  - General-Purpose Input/Output (GPIO), [86](#)
- GPIO\_EnableDigital
  - General-Purpose Input/Output (GPIO), [86](#)
- GPIO\_getBaseAddr
  - General-Purpose Input/Output (GPIO), [84](#)
- GPIO\_getDataRegister
  - General-Purpose Input/Output (GPIO), [88](#)
- GPIO\_InitPort
  - General-Purpose Input/Output (GPIO), [84](#)
- GPIO\_isPortInit
  - General-Purpose Input/Output (GPIO), [84](#)
- GPIO\_LAUNCHPAD\_LEDS
  - General-Purpose Input/Output (GPIO), [83](#)
- GPIO\_PTR\_ARR

- General-Purpose Input/Output (GPIO), 90
- GPIO\_ReadPins
  - General-Purpose Input/Output (GPIO), 88
- GPIO\_Toggle
  - General-Purpose Input/Output (GPIO), 89
- GPIO\_WriteHigh
  - General-Purpose Input/Output (GPIO), 88
- GPIO\_WriteLow
  - General-Purpose Input/Output (GPIO), 88
- GpioPort\_t, 113
- IDMOFF
  - ILI9341, 66
- IDMON
  - ILI9341, 66
- IFCTL
  - ILI9341, 66
- ILI9341, 63
  - CASET, 66
  - Cmd\_t, 65
  - DINVOFF, 66
  - DINVON, 66
  - DISPOFF, 66
  - DISPON, 66
  - FRMCTR1, 66
  - FRMCTR2, 66
  - FRMCTR3, 66
  - IDMOFF, 66
  - IDMON, 66
  - IFCTL, 66
  - ILI9341\_Init, 67
  - ILI9341\_NUM\_COLS, 65
  - ILI9341\_NUM\_ROWS, 65
  - ILI9341\_resetHard, 68
  - ILI9341\_resetSoft, 68
  - ILI9341\_sendParams, 66
  - ILI9341\_setAddress, 66
  - ILI9341\_setColAddress, 72
  - ILI9341\_setColorDepth, 71
  - ILI9341\_setColorExpression, 69
  - ILI9341\_setDisplInversion, 70
  - ILI9341\_setDisplayArea, 69
  - ILI9341\_setDispOutput, 70
  - ILI9341 setFrameRate, 71
  - ILI9341\_setInterface, 67
  - ILI9341\_setMemAccessCtrl, 70
  - ILI9341\_setMode, 66
  - ILI9341\_setPartialArea, 69
  - ILI9341\_setRowAddress, 72
  - ILI9341\_setSleepMode, 68
  - ILI9341\_writeMemCmd, 73
  - ILI9341\_writePixel, 73
- MADCTL, 66
- NOP, 66
- NORON, 66
- PASET, 66
- PIXSET, 66
- PLTAR, 66
- PRCTR, 66
- PTLON, 66
- RAMWR, 66
- SPLIN, 66
- SPLOUT, 66
- SWRESET, 66
- VSCRDEF, 66
- VSCRSADD, 66
- ILI9341.c, 130
- ILI9341.h, 131
- ILI9341\_Init
  - ILI9341, 67
- ILI9341\_NUM\_COLS
  - ILI9341, 65
- ILI9341\_NUM\_ROWS
  - ILI9341, 65
- ILI9341\_resetHard
  - ILI9341, 68
- ILI9341\_resetSoft
  - ILI9341, 68
- ILI9341\_sendParams
  - ILI9341, 66
- ILI9341\_setAddress
  - ILI9341, 66
- ILI9341\_setColAddress
  - ILI9341, 72
- ILI9341\_setColorDepth
  - ILI9341, 71
- ILI9341\_setColorExpression
  - ILI9341, 69
- ILI9341\_setDisplInversion
  - ILI9341, 70
- ILI9341\_setDisplayArea
  - ILI9341, 69
- ILI9341\_setDispOutput
  - ILI9341, 70
- ILI9341 setFrameRate
  - ILI9341, 71
- ILI9341\_setInterface
  - ILI9341, 67
- ILI9341\_setMemAccessCtrl
  - ILI9341, 70
- ILI9341\_setMode
  - ILI9341, 66
- ILI9341\_setPartialArea
  - ILI9341, 69
- ILI9341\_setRowAddress
  - ILI9341, 72
- ILI9341\_setSleepMode
  - ILI9341, 68
- ILI9341\_writeMemCmd
  - ILI9341, 73
- ILI9341\_writePixel
  - ILI9341, 73
- Interrupt Service Routines, 108
  - ISR\_addToIntTable, 110
  - ISR\_Disable, 111
  - ISR\_Enable, 111
  - ISR\_GlobalDisable, 109

- ISR\_GlobalEnable, [109](#)
- ISR\_InitNewTableInRam, [109](#)
- ISR\_setPriority, [110](#)
- ISR\_triggerInterrupt, [112](#)
- ISR.c, [123](#)
- ISR.h, [123](#)
- ISR\_addToIntTable
  - Interrupt Service Routines, [110](#)
- ISR\_Disable
  - Interrupt Service Routines, [111](#)
- ISR\_Enable
  - Interrupt Service Routines, [111](#)
- ISR\_GlobalDisable
  - Interrupt Service Routines, [109](#)
- ISR\_GlobalEnable
  - Interrupt Service Routines, [109](#)
- ISR\_InitNewTableInRam
  - Interrupt Service Routines, [109](#)
- ISR\_setPriority
  - Interrupt Service Routines, [110](#)
- ISR\_triggerInterrupt
  - Interrupt Service Routines, [112](#)
- ISR\_VECTOR\_NUMS
  - Main, [44](#)
- LCD.c, [116](#)
- LCD.h, [116](#)
- LCD\_ARRAY\_1\_LEN
  - Main, [45](#)
- LCD\_ARRAY\_2\_LEN
  - Main, [45](#)
- LCD\_Draw
  - Liquid Crystal Display (LCD), [17](#)
- LCD\_drawHoriLine
  - Liquid Crystal Display (LCD), [17](#)
- LCD\_drawLine
  - Liquid Crystal Display (LCD), [15](#)
- LCD\_drawRectangle
  - Liquid Crystal Display (LCD), [18](#)
- LCD\_drawVertLine
  - Liquid Crystal Display (LCD), [18](#)
- LCD\_FIFO\_1\_CAP
  - Main, [45](#)
- LCD\_FIFO\_2\_CAP
  - Main, [45](#)
- LCD\_Fill
  - Liquid Crystal Display (LCD), [17](#)
- LCD\_Handler
  - Main, [47](#)
- LCD\_INFO
  - Main, [45](#)
- LCD\_Init
  - Liquid Crystal Display (LCD), [15](#)
- LCD\_plotSample
  - Liquid Crystal Display (LCD), [19](#)
- LCD\_setColor
  - Liquid Crystal Display (LCD), [17](#)
- LCD\_setCursor
  - Liquid Crystal Display (LCD), [19](#)
- LCD\_setOutputMode
  - Liquid Crystal Display (LCD), [16](#)
- LCD\_setX
  - Liquid Crystal Display (LCD), [16](#)
- LCD\_setY
  - Liquid Crystal Display (LCD), [16](#)
- LCD\_TEXT\_COL\_NUM
  - Main, [45](#)
- LCD\_TEXT\_LINE\_NUM
  - Main, [45](#)
- LCD\_TOP\_LINE
  - Main, [45](#)
- LCD\_VECTOR\_NUM
  - Main, [45](#)
- LCD\_WAVE\_NUM\_Y
  - Main, [45](#)
- LCD\_WAVE\_X\_OFFSET
  - Main, [45](#)
- LCD\_WAVE\_Y\_MAX
  - Main, [45](#)
- LCD\_WAVE\_Y\_MIN
  - Main, [45](#)
- LED, [74](#)
  - Led\_GetPin, [76](#)
  - Led\_GetPort, [76](#)
  - Led\_Init, [75](#)
  - Led\_isInit, [76](#)
  - Led\_isOn, [77](#)
  - Led\_Toggle, [78](#)
  - Led\_TurnOff, [78](#)
  - Led\_TurnOn, [77](#)
- Led.c, [132](#)
- Led.h, [132](#)
- LED\_BLUE
  - General-Purpose Input/Output (GPIO), [83](#)
- Led\_GetPin
  - LED, [76](#)
- Led\_GetPort
  - LED, [76](#)
- LED\_GREEN
  - General-Purpose Input/Output (GPIO), [83](#)
- Led\_Init
  - LED, [75](#)
- Led\_isInit
  - LED, [76](#)
- Led\_isOn
  - LED, [77](#)
- LED\_RED
  - General-Purpose Input/Output (GPIO), [83](#)
- Led\_t, [113](#)
- Led\_Toggle
  - LED, [78](#)
- Led\_TurnOff
  - LED, [78](#)
- Led\_TurnOn
  - LED, [77](#)
- Liquid Crystal Display (LCD), [12](#)
  - FONT\_0, [20](#)

FONT\_1, [20](#)  
 FONT\_2, [21](#)  
 FONT\_3, [21](#)  
 FONT\_4, [21](#)  
 FONT\_5, [21](#)  
 FONT\_6, [22](#)  
 FONT\_7, [22](#)  
 FONT\_8, [22](#)  
 FONT\_9, [22](#)  
 FONT\_COLON, [20](#)  
 FONT\_LOWER\_A, [29](#)  
 FONT\_LOWER\_B, [29](#)  
 FONT\_LOWER\_C, [30](#)  
 FONT\_LOWER\_D, [30](#)  
 FONT\_LOWER\_E, [30](#)  
 FONT\_LOWER\_F, [30](#)  
 FONT\_LOWER\_G, [31](#)  
 FONT\_LOWER\_H, [31](#)  
 FONT\_LOWER\_I, [31](#)  
 FONT\_LOWER\_J, [31](#)  
 FONT\_LOWER\_K, [32](#)  
 FONT\_LOWER\_L, [32](#)  
 FONT\_LOWER\_M, [32](#)  
 FONT\_LOWER\_N, [32](#)  
 FONT\_LOWER\_O, [33](#)  
 FONT\_LOWER\_P, [33](#)  
 FONT\_LOWER\_Q, [33](#)  
 FONT\_LOWER\_R, [33](#)  
 FONT\_LOWER\_S, [34](#)  
 FONT\_LOWER\_T, [34](#)  
 FONT\_LOWER\_U, [34](#)  
 FONT\_LOWER\_V, [34](#)  
 FONT\_LOWER\_W, [35](#)  
 FONT\_LOWER\_X, [35](#)  
 FONT\_LOWER\_Y, [35](#)  
 FONT\_LOWER\_Z, [35](#)  
 FONT\_PERIOD, [20](#)  
 FONT\_SPACE, [20](#)  
 FONT\_UPPER\_A, [23](#)  
 FONT\_UPPER\_B, [23](#)  
 FONT\_UPPER\_C, [23](#)  
 FONT\_UPPER\_D, [23](#)  
 FONT\_UPPER\_E, [24](#)  
 FONT\_UPPER\_F, [24](#)  
 FONT\_UPPER\_G, [24](#)  
 FONT\_UPPER\_H, [24](#)  
 FONT\_UPPER\_I, [25](#)  
 FONT\_UPPER\_J, [25](#)  
 FONT\_UPPER\_K, [25](#)  
 FONT\_UPPER\_L, [25](#)  
 FONT\_UPPER\_M, [26](#)  
 FONT\_UPPER\_N, [26](#)  
 FONT\_UPPER\_O, [26](#)  
 FONT\_UPPER\_P, [26](#)  
 FONT\_UPPER\_Q, [27](#)  
 FONT\_UPPER\_R, [27](#)  
 FONT\_UPPER\_S, [27](#)  
 FONT\_UPPER\_T, [27](#)  
 FONT\_UPPER\_U, [28](#)  
 FONT\_UPPER\_V, [28](#)  
 FONT\_UPPER\_W, [28](#)  
 FONT\_UPPER\_X, [28](#)  
 FONT\_UPPER\_Y, [29](#)  
 FONT\_UPPER\_Z, [29](#)  
 LCD\_Draw, [17](#)  
 LCD\_drawHoriLine, [17](#)  
 LCD\_drawLine, [15](#)  
 LCD\_drawRectangle, [18](#)  
 LCD\_drawVertLine, [18](#)  
 LCD\_Fill, [17](#)  
 LCD\_Init, [15](#)  
 LCD\_plotSample, [19](#)  
 LCD\_setColor, [17](#)  
 LCD\_setCursor, [19](#)  
 LCD\_setOutputMode, [16](#)  
 LCD\_setX, [16](#)  
 LCD\_setY, [16](#)  
 MADCTL  
     ILI9341, [66](#)  
 Main, [43](#)  
     DAQ\_ARRAY\_LEN, [45](#)  
     DAQ\_FIFO\_CAP, [45](#)  
     DAQ\_Handler, [45](#)  
     DAQ\_VECTOR\_NUM, [45](#)  
     FIFO\_INFO, [45](#)  
     ISR\_VECTOR\_NUMS, [44](#)  
     LCD\_ARRAY\_1\_LEN, [45](#)  
     LCD\_ARRAY\_2\_LEN, [45](#)  
     LCD\_FIFO\_1\_CAP, [45](#)  
     LCD\_FIFO\_2\_CAP, [45](#)  
     LCD\_Handler, [47](#)  
     LCD\_INFO, [45](#)  
     LCD\_TEXT\_COL\_NUM, [45](#)  
     LCD\_TEXT\_LINE\_NUM, [45](#)  
     LCD\_TOP\_LINE, [45](#)  
     LCD\_VECTOR\_NUM, [45](#)  
     LCD\_WAVE\_NUM\_Y, [45](#)  
     LCD\_WAVE\_X\_OFFSET, [45](#)  
     LCD\_WAVE\_Y\_MAX, [45](#)  
     LCD\_WAVE\_Y\_MIN, [45](#)  
     main, [48](#)  
     PROC\_VECTOR\_NUM, [45](#)  
     Processing\_Handler, [46](#)  
     QRS\_ARRAY\_LEN, [45](#)  
     QRS\_FIFO\_CAP, [45](#)  
 main  
     Main, [48](#)  
 main.c, [129](#)  
 Middleware, [59](#)  
 NewAssert, [59](#)  
 NewAssert.c, [120](#)  
 NewAssert.h, [120](#)  
 NOP  
     ILI9341, [66](#)  
 NORON

- ILI9341, 66
- ONESHOT
  - Timer, 96
- PASET
  - ILI9341, 66
- PERIODIC
  - Timer, 96
- Phase-Locked Loop (PLL), 90
  - PLL\_Init, 91
- PIXSET
  - ILI9341, 66
- PLL.c, 124
- PLL.h, 125
- PLL\_Init
  - Phase-Locked Loop (PLL), 91
- PLTAR
  - ILI9341, 66
- PRCTR
  - ILI9341, 66
- PROC\_VECTOR\_NUM
  - Main, 45
- Processing\_Handler
  - Main, 46
- PTLON
  - ILI9341, 66
- QRS Detector, 36
  - COEFF\_BANDPASS, 42
  - COEFF\_DERFILT, 43
  - COEFF\_MOVAVG, 43
  - QRS\_applyDecisionRules, 41
  - QRS\_findFiducialMarks, 38
  - QRS\_Init, 39
  - QRS\_initLevels, 38
  - QRS\_Preprocess, 40
  - QRS\_runDetection, 42
  - QRS\_updateLevel, 39
  - QRS\_updateThreshold, 39
- QRS.c, 117
- QRS.h, 118
- QRS\_applyDecisionRules
  - QRS Detector, 41
- QRS\_ARRAY\_LEN
  - Main, 45
- QRS\_FIFO\_CAP
  - Main, 45
- QRS\_findFiducialMarks
  - QRS Detector, 38
- QRS\_Init
  - QRS Detector, 39
- QRS\_initLevels
  - QRS Detector, 38
- QRS\_Preprocess
  - QRS Detector, 40
- QRS\_runDetection
  - QRS Detector, 42
- QRS\_updateLevel
- QRS Detector, 39
  - QRS Detector, 39
- RAMWR
  - ILI9341, 66
- Serial Peripheral Interface (SPI), 91
  - SPI\_Init, 93
  - SPI\_Read, 93
  - SPI\_WriteCmd, 93
  - SPI\_WriteData, 94
- SPI.c, 125
- SPI.h, 125
- SPI\_Init
  - Serial Peripheral Interface (SPI), 93
- SPI\_Read
  - Serial Peripheral Interface (SPI), 93
- SPI\_WriteCmd
  - Serial Peripheral Interface (SPI), 93
- SPI\_WriteData
  - Serial Peripheral Interface (SPI), 94
- SPLIN
  - ILI9341, 66
- SPLOUT
  - ILI9341, 66
- SWRESET
  - ILI9341, 66
- Timer, 94
  - DOWN, 96
  - ONESHOT, 96
  - PERIODIC, 96
  - Timer\_clearInterruptFlag, 101
  - Timer\_Deinit, 97
  - Timer\_disableAdcTrigger, 98
  - Timer\_disableInterruptOnTimeout, 100
  - Timer\_enableAdcTrigger, 98
  - Timer\_enableInterruptOnTimeout, 100
  - Timer\_getName, 97
  - Timer\_Init, 96
  - Timer\_isCounting, 102
  - Timer\_isInit, 97
  - TIMER\_POOL, 103
  - Timer\_setInterval\_ms, 101
  - Timer\_setMode, 98
  - Timer\_Start, 101
  - Timer\_Stop, 102
  - Timer\_Wait1ms, 103
  - timerDirection\_t, 96
  - timerMode\_t, 96
  - UP, 96
- Timer.c, 126
- Timer.h, 127
- Timer\_clearInterruptFlag
  - Timer, 101
- Timer\_Deinit
  - Timer, 97
- Timer\_disableAdcTrigger

- Timer, [98](#)
- Timer\_disableInterruptOnTimeout
  - Timer, [100](#)
- Timer\_enableAdcTrigger
  - Timer, [98](#)
- Timer\_enableInterruptOnTimeout
  - Timer, [100](#)
- Timer\_getName
  - Timer, [97](#)
- Timer\_Init
  - Timer, [96](#)
- Timer\_isCounting
  - Timer, [102](#)
- Timer\_isInit
  - Timer, [97](#)
- TIMER\_POOL
  - Timer, [103](#)
- Timer\_setInterval\_ms
  - Timer, [101](#)
- Timer\_setMode
  - Timer, [98](#)
- Timer\_Start
  - Timer, [101](#)
- Timer\_Stop
  - Timer, [102](#)
- Timer\_t, [113](#)
- Timer\_Wait1ms
  - Timer, [103](#)
- timerDirection\_t
  - Timer, [96](#)
- timerMode\_t
  - Timer, [96](#)
- UART.c, [127](#)
- UART.h, [128](#)
- UART\_ARR
  - Universal Asynchronous Receiver/Transmitter (UART), [107](#)
- UART\_Init
  - Universal Asynchronous Receiver/Transmitter (UART), [105](#)
- UART\_isInit
  - Universal Asynchronous Receiver/Transmitter (UART), [105](#)
- UART\_ReadChar
  - Universal Asynchronous Receiver/Transmitter (UART), [106](#)
- Uart\_t, [114](#)
- UART\_WriteChar
  - Universal Asynchronous Receiver/Transmitter (UART), [106](#)
- UART\_WriteFloat
  - Universal Asynchronous Receiver/Transmitter (UART), [107](#)
- UART\_WriteInt
  - Universal Asynchronous Receiver/Transmitter (UART), [107](#)
- UART\_WriteStr
  - Universal Asynchronous Receiver/Transmitter (UART), [106](#)
- UP
  - Timer, [96](#)
- VSCRDEF
  - ILI9341, [66](#)
- VSCRSADD
  - ILI9341, [66](#)