

uHeartMonitor

Generated by Doxygen 1.9.8

| | |
|--------------------------------------------------------------------|-----------|
| 1 Bug List | 1 |
| 2 Topic Index | 2 |
| 2.1 Topics | 2 |
| 3 Data Structure Index | 2 |
| 3.1 Data Structures | 2 |
| 4 File Index | 3 |
| 4.1 File List | 3 |
| 5 Topic Documentation | 4 |
| 5.1 Application Software | 4 |
| 5.1.1 Detailed Description | 5 |
| 5.1.2 Data Acquisition (DAQ) | 5 |
| 5.1.3 Liquid Crystal Display (LCD) | 10 |
| 5.1.4 QRS Detector | 17 |
| 5.2 Common | 23 |
| 5.2.1 Detailed Description | 24 |
| 5.2.2 Function Documentation | 24 |
| 5.2.3 FIFO Buffers | 24 |
| 5.2.4 NewAssert | 30 |
| 5.3 Device Drivers | 30 |
| 5.3.1 Detailed Description | 31 |
| 5.3.2 Analog-to-Digital Conversion (ADC) | 32 |
| 5.3.3 General-Purpose Input/Output (GPIO) | 33 |
| 5.3.4 Phase-Locked Loop (PLL) | 33 |
| 5.3.5 Serial Peripheral Interface (SPI) | 34 |
| 5.3.6 System Tick (SysTick) | 37 |
| 5.3.7 Timer | 38 |
| 5.3.8 Universal Asynchronous Receiver/Transmitter (UART) | 47 |
| 5.3.9 Interrupt Service Routines | 51 |
| 5.4 Middleware | 55 |
| 5.4.1 Detailed Description | 56 |
| 5.4.2 Debug | 56 |
| 5.4.3 ILI9341 | 59 |
| 5.4.4 LED | 70 |
| 5.5 Main | 73 |
| 5.5.1 Detailed Description | 74 |
| 5.5.2 Enumeration Type Documentation | 74 |
| 5.5.3 Function Documentation | 75 |
| 6 Data Structure Documentation | 79 |
| 6.1 Fifo_t Struct Reference | 79 |

| | |
|-------------------------------------------|-----------|
| 6.2 GpioPort_t Struct Reference | 80 |
| 6.3 Led_t Struct Reference | 80 |
| 6.4 Timer_t Struct Reference | 81 |
| 6.5 Uart_t Struct Reference | 81 |
| 7 File Documentation | 81 |
| 7.1 DAQ.c File Reference | 81 |
| 7.1.1 Detailed Description | 83 |
| 7.2 DAQ.h File Reference | 83 |
| 7.2.1 Detailed Description | 84 |
| 7.3 DAQ_lookup.c File Reference | 84 |
| 7.3.1 Detailed Description | 85 |
| 7.4 LCD.c File Reference | 85 |
| 7.4.1 Detailed Description | 87 |
| 7.5 LCD.h File Reference | 87 |
| 7.5.1 Detailed Description | 89 |
| 7.6 QRS.c File Reference | 89 |
| 7.6.1 Detailed Description | 91 |
| 7.7 QRS.h File Reference | 92 |
| 7.7.1 Detailed Description | 93 |
| 7.8 Fifo.c File Reference | 93 |
| 7.8.1 Detailed Description | 94 |
| 7.9 Fifo.h File Reference | 95 |
| 7.9.1 Detailed Description | 96 |
| 7.10 NewAssert.c File Reference | 96 |
| 7.10.1 Detailed Description | 97 |
| 7.11 NewAssert.h File Reference | 97 |
| 7.11.1 Detailed Description | 98 |
| 7.12 ADC.c File Reference | 98 |
| 7.12.1 Detailed Description | 99 |
| 7.13 ADC.h File Reference | 99 |
| 7.13.1 Detailed Description | 99 |
| 7.14 GPIO.c File Reference | 100 |
| 7.14.1 Detailed Description | 102 |
| 7.14.2 Function Documentation | 102 |
| 7.14.3 Variable Documentation | 110 |
| 7.15 GPIO.h File Reference | 110 |
| 7.15.1 Detailed Description | 113 |
| 7.15.2 Function Documentation | 113 |
| 7.16 ISR.c File Reference | 119 |
| 7.16.1 Detailed Description | 120 |
| 7.17 ISR.h File Reference | 121 |

| | |
|-----------------------------------------|------------|
| 7.17.1 Detailed Description | 122 |
| 7.18 PLL.c File Reference | 122 |
| 7.18.1 Detailed Description | 122 |
| 7.19 PLL.h File Reference | 123 |
| 7.19.1 Detailed Description | 123 |
| 7.20 SPI.c File Reference | 123 |
| 7.20.1 Detailed Description | 125 |
| 7.21 SPI.h File Reference | 125 |
| 7.21.1 Detailed Description | 126 |
| 7.22 SysTick.c File Reference | 126 |
| 7.22.1 Detailed Description | 127 |
| 7.23 SysTick.h File Reference | 127 |
| 7.23.1 Detailed Description | 128 |
| 7.24 Timer.c File Reference | 128 |
| 7.24.1 Detailed Description | 129 |
| 7.25 Timer.h File Reference | 130 |
| 7.25.1 Detailed Description | 131 |
| 7.26 UART.c File Reference | 132 |
| 7.26.1 Detailed Description | 133 |
| 7.27 UART.h File Reference | 133 |
| 7.27.1 Detailed Description | 134 |
| 7.28 main.c File Reference | 135 |
| 7.28.1 Detailed Description | 136 |
| 7.29 Debug.h File Reference | 136 |
| 7.29.1 Detailed Description | 137 |
| 7.30 ILI9341.c File Reference | 138 |
| 7.30.1 Detailed Description | 139 |
| 7.31 ILI9341.h File Reference | 139 |
| 7.31.1 Detailed Description | 141 |
| 7.32 Led.c File Reference | 142 |
| 7.32.1 Detailed Description | 143 |
| 7.33 Led.h File Reference | 143 |
| 7.33.1 Detailed Description | 144 |
| Index | 145 |

1 Bug List

Global **FIFO_Get** (volatile **Fifo_t** fifo)

To use `floats` (AKA `float32_t`), type-punning is necessary.

Global **FIFO_Put** (volatile **Fifo_t** fifo, **const uint32_t** val)

To use `floats` (AKA `float32_t`), type-punning is necessary.

2 Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

| | |
|-----------------------------------------------------------|-----------|
| Application Software | 4 |
| Data Acquisition (DAQ) | 5 |
| Liquid Crystal Display (LCD) | 10 |
| QRS Detector | 17 |
| Common | 23 |
| FIFO Buffers | 24 |
| NewAssert | 30 |
| Device Drivers | 30 |
| Analog-to-Digital Conversion (ADC) | 32 |
| General-Purpose Input/Output (GPIO) | 33 |
| Phase-Locked Loop (PLL) | 33 |
| Serial Peripheral Interface (SPI) | 34 |
| System Tick (SysTick) | 37 |
| Timer | 38 |
| Universal Asynchronous Receiver/Transmitter (UART) | 47 |
| Interrupt Service Routines | 51 |
| Middleware | 55 |
| Debug | 56 |
| ILI9341 | 59 |
| LED | 70 |
| Main | 73 |

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

| | |
|-------------------|-----------|
| Fifo_t | 79 |
| GpioPort_t | 80 |

| | |
|-------------------------|----|
| Led_t | 80 |
| Timer_t | 81 |
| Uart_t | 81 |

4 File Index

4.1 File List

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

| | | |
|--------------------------------------------------------------------|--|-----|
| DAQ.c | | |
| Source code for DAQ module | | 81 |
| DAQ.h | | |
| Application software for handling data acquisition (DAQ) functions | | 83 |
| DAQ_lookup.c | | |
| Source code for DAQ module's lookup table | | 84 |
| LCD.c | | |
| Source code for LCD module | | 85 |
| LCD.h | | |
| Header file for LCD module | | 87 |
| QRS.c | | |
| Source code for QRS detection module | | 89 |
| QRS.h | | |
| Header file for QRS detection module | | 92 |
| Fifo.c | | |
| Source code for FIFO buffer module | | 93 |
| Fifo.h | | |
| Header file for FIFO buffer implementation | | 95 |
| NewAssert.c | | |
| Source code for custom <code>assert</code> implementation | | 96 |
| NewAssert.h | | |
| Header file for custom <code>assert</code> implementation | | 97 |
| ADC.c | | |
| Source code for analog-to-digital conversion (ADC) module | | 98 |
| ADC.h | | |
| Header file for analog-to-digital conversion (ADC) module | | 99 |
| GPIO.c | | |
| Source code for GPIO module | | 100 |
| GPIO.h | | |
| Header file for general-purpose input/output (GPIO) device driver | | 110 |

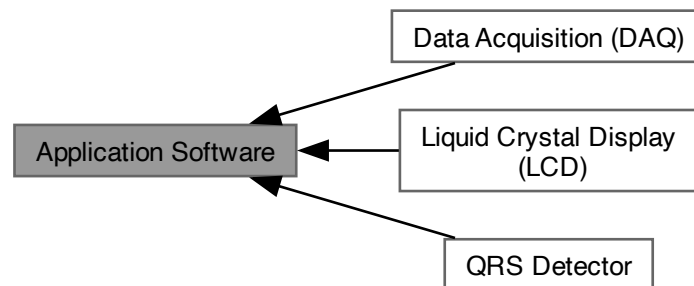
| | | |
|------------------|----------------------------------------------------------------------|-----|
| ISR.c | Source code for interrupt service routine (ISR) configuration module | 119 |
| ISR.h | Header file for interrupt service routine (ISR) configuration module | 121 |
| PLL.c | Implementation details for phase-lock-loop (PLL) functions | 122 |
| PLL.h | Driver module for activating the phase-locked-loop (PLL) | 123 |
| SPI.c | Source code for serial peripheral interface (SPI) module | 123 |
| SPI.h | Header file for serial peripheral interface (SPI) module | 125 |
| SysTick.c | Implementation details for SysTick functions | 126 |
| SysTick.h | Driver module for using SysTick-based timing and/or interrupts | 127 |
| Timer.c | Source code for Timer module | 128 |
| Timer.h | Device driver for general-purpose timer modules | 130 |
| UART.c | Source code for UART module | 132 |
| UART.h | Driver module for serial communication via UART0 and UART 1 | 133 |
| main.c | Main program file | 135 |
| Debug.h | Functions to output debugging information to a serial port via UART | 136 |
| ILI9341.c | Source code for ILI9341 module | 138 |
| ILI9341.h | Driver module for interfacing with an ILI9341 LCD driver | 139 |
| Led.c | Source code for LED module | 142 |
| Led.h | Interface for LED module | 143 |

5 Topic Documentation

5.1 Application Software

Application-specific software modules.

Collaboration diagram for Application Software:



Modules

- [Data Acquisition \(DAQ\)](#)
- [Liquid Crystal Display \(LCD\)](#)
- [QRS Detector](#)

5.1.1 Detailed Description

Application-specific software modules.

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

5.1.2 Data Acquisition (DAQ)

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 ($T_s = 1/f_s$)
- `#define DAQ_LOOKUP_MAX ((float32_t) 5.5f)`
- `#define DAQ_LOOKUP_MIN ((float32_t) (-5.5f))`

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** = ¬chFiltStruct
- 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.

5.1.2.1 Detailed Description

Module for managing data acquisition (DAQ) functions.

5.1.2.2 Function Documentation

DAQ_Init()

```
void DAQ_Init (
    void )
```

Initialize the data acquisition (DAQ) module.

Postcondition

The ADC and Timer are initialized, and 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_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 | xn | Raw input sample |
| out | yn | Filtered output sample |

Postcondition

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

See also

[DAQ_BandpassFilter\(\)](#)

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\(\)](#)

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, 0xFFFF] |
| 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\(\)](#)

5.1.2.3 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_DAQ_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

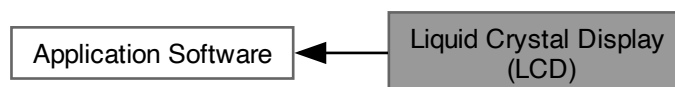
```
const Filter_t bandpassFiltStruct [static]
```

Initial value:

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

5.1.3 Liquid Crystal Display (LCD)

Collaboration diagram for Liquid Crystal Display (LCD):



Files

- 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_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.
- static void [LCD_plotSample](#) (uint16_t x, uint16_t y, LCD_Color_t color)
Plot a sample at coordinates (x, y).

Variables

- 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**
 - uint16_t **colNum**
 - uint8_t **R_val**
5 or 6-bit R value
 - uint8_t **G_val**
6-bit G value
 - uint8_t **B_val**
5 or 6-bit B value
 - bool **isInit**
if true, LCD has been initialized
 } **lcd** = { 0 }
- const uint8_t *const **FONT_ARRAY** [128]

Init./Config. Functions

- enum { **LCD_X_MAX** = ILI9341_NUM_ROWS - 1 , **LCD_Y_MAX** = ILI9341_NUM_COLS - 1 }
- enum **LCD_Color_t** {
 - 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** (LCD_Color_t color)
Set the color value.

Writing Functions

- enum { **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_writeln** (int32_t num)
- void **LCD_writeFloat** (float num)

Drawing Functions

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

5.1.3.1 Detailed Description

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

5.1.3.2 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 (
    LCD_Color_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\(\)](#)

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,
bool isFilled )

```

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 |
| in | <i>isFilled</i> | true to fill the rectangle, false to leave it unfilled |

See also

[LCD_Draw\(\)](#), [LCD_Fill\(\)](#), [LCD_drawHoriLine\(\)](#), [LCD_drawVertLine\(\)](#)

LCD_plotSample()

```

static void LCD_plotSample (
    uint16_t x,
    uint16_t y,
    LCD_Color_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 $[0, 64)$. |

5.1.4 QRS Detector

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 , **STATE_BUFF_SIZE_DERFILT** = NUM_COEFF_DERFILT + BLOCK_SIZE_↵
_DERFILT - 1 , **BLOCK_SIZE_MOVAVG** = 1 , **NUM_COEFF_MOVAVG** = 10 ,
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-specific Functions

- 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.

5.1.4.1 Detailed Description

Module for analyzing ECG data to determine heart rate.

5.1.4.2 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 | yn | Array containing the preprocessed ECG signal $y[n]$ |
| in | $fidMarkArray$ | Array to place the fiducial mark's sample indices into. |
| out | $numMarks$ | 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 @ $fs = 200$ [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 | yn | Array containing the preprocessed ECG signal $y[n]$ |
| in | $sigLvlPtr$ | Pointer to variable holding the signal level value. |
| in | $noiseLvlPtr$ | 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.

Warning

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. |

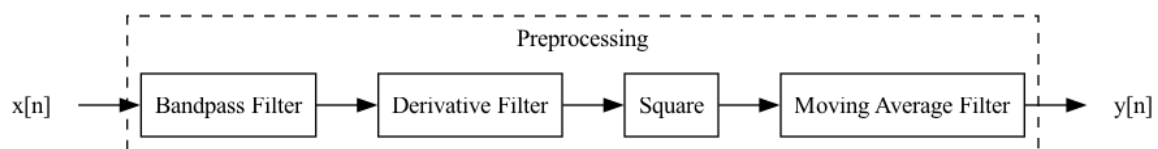
Postcondition

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.

**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\(\)](#)

5.1.4.3 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

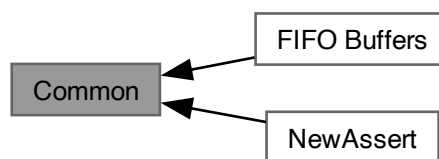
```
const float32_t COEFF_MOVAVG[NUM_COEFF_MOVAVG] [static]
```

Initial value:

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

5.2 Common

Collaboration diagram for Common:

**Modules**

- [FIFO Buffers](#)
- [NewAssert](#)

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.

5.2.1 Detailed Description

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

5.2.2 Function Documentation

Assert()

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

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

Parameters

| | | |
|----|------------------|----------------------|
| in | <i>condition</i> | Conditional to test. |
|----|------------------|----------------------|

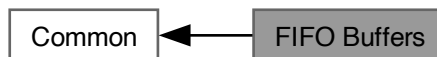
Postcondition

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

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

5.2.3 FIFO Buffers

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.

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_TransferOne` (volatile Fifo_t srcFifo, volatile Fifo_t destFifo)
Transfer a value from one FIFO buffer to another.

Bulk Removal

- void `FIFO_Flush` (volatile Fifo_t fifo, uint32_t outputBuffer[])
Empty the FIFO buffer's contents into an array.
- void `FIFO_Reset` (volatile Fifo_t fifo)
Reset the FIFO buffer.
- void `FIFO_TransferAll` (volatile Fifo_t srcFifo, volatile Fifo_t destFifo)
Transfer the contents of one FIFO buffer to another.

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.

5.2.3.1 Detailed Description

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

5.2.3.2 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 <i>N</i> - 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_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.

Bug To use floats (AKA `float32_t`), type-punning is necessary.

```
// type-punning example
float num = 4.252603;
FIFO_Put(fifo, *((uint32_t *) &num));
```

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 return If the FIFO is empty, 0 is returned.

Bug To use floats (AKA float32_t), type-punning is necessary.

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

FIFO_TransferOne()

```
void FIFO_TransferOne (
    volatile Fifo_t srcFifo,
    volatile Fifo_t destFifo )
```

Transfer a value from one FIFO buffer to another.

Precondition

Initialize both FIFO buffers.

Parameters

| | | |
|----|-----------------|-------------------------------------|
| in | <i>srcFifo</i> | Pointer to source FIFO buffer. |
| in | <i>destFifo</i> | Pointer to destination FIFO buffer. |

Postcondition

A value is removed from *srcFifo* and placed in *destFifo*.

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.

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_TransferAll()

```
void FIFO_TransferAll (
    volatile Fifo_t srcFifo,
    volatile Fifo_t destFifo )
```

Transfer the contents of one FIFO buffer to another.

Parameters

| | | |
|----|-----------------|-------------------------------------|
| in | <i>srcFifo</i> | Pointer to source FIFO buffer. |
| in | <i>destFifo</i> | Pointer to destination FIFO buffer. |

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. |

5.2.4 NewAssert

Collaboration diagram for NewAssert:

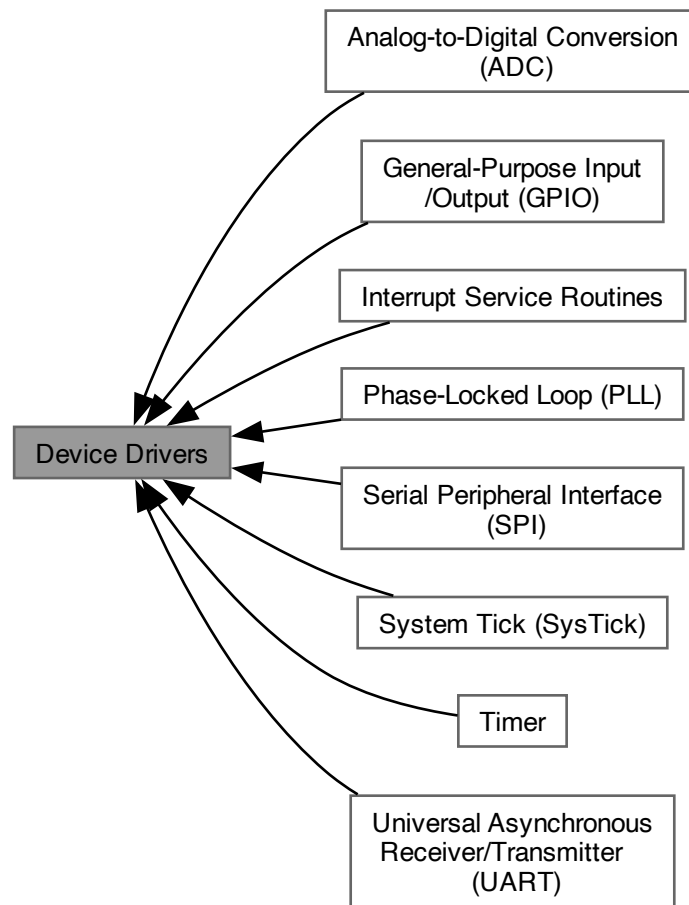


Module for using a custom `assert` implementation.

5.3 Device Drivers

Low level device driver modules.

Collaboration diagram for Device Drivers:



Modules

- [Analog-to-Digital Conversion \(ADC\)](#)
- [General-Purpose Input/Output \(GPIO\)](#)
- [Phase-Locked Loop \(PLL\)](#)
- [Serial Peripheral Interface \(SPI\)](#)
- [System Tick \(SysTick\)](#)
- [Timer](#)
- [Universal Asynchronous Receiver/Transmitter \(UART\)](#)
- [Interrupt Service Routines](#)

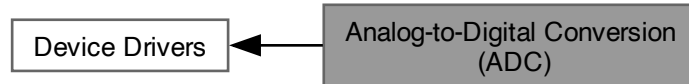
5.3.1 Detailed Description

Low level device driver modules.

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

5.3.2 Analog-to-Digital Conversion (ADC)

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.

5.3.2.1 Detailed Description

Functions for differential-input analog-to-digital conversion.

5.3.2.2 Function Documentation

ADC_Init()

```
void ADC_Init (  
    void )
```

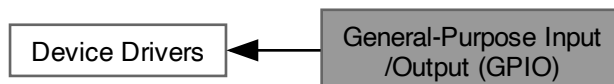
Initialize ADC0 as a single-input analog-to-digital converter.

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.

5.3.3 General-Purpose Input/Output (GPIO)

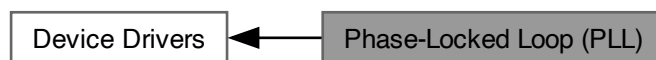
Collaboration diagram for General-Purpose Input/Output (GPIO):



Functions for using general-purpose input/output (GPIO) ports.

5.3.4 Phase-Locked Loop (PLL)

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.

5.3.4.1 Detailed Description

Function for initializing the phase-locked loop.

5.3.4.2 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].

5.3.5 Serial Peripheral Interface (SPI)

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_SET_DC()` (GPIO_PORTA_DATA_R |= 0x40)
- `#define SPI_CLEAR_DC()` (GPIO_PORTA_DATA_R &= ~(0x40))
- `#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.

5.3.5.1 Detailed Description

Functions for SPI-based communication via SSI0 peripheral.

5.3.5.2 Macro Definition Documentation

SPI_SET_DC

```
#define SPI_SET_DC( ) (GPIO_PORTA_DATA_R |= 0x40)
```

| TM4C Pin | Function | ILI9341 Pin | Description |
|----------|----------|-------------|-----------------------------------------------|
| PA2 | SSI0Clk | CLK | Serial clock signal |
| PA3 | SSI0Fss | CS | Chip select signal |
| PA4 | SSI0Rx | MISO | TM4C (M) input, LCD (S) output |
| PA5 | SSI0Tx | MOSI | TM4C (M) output, LCD (S) input |
| PA6 | GPIO | D/C | Data = 1, Command = 0 |
| PA7 | GPIO | RESET | Reset the display (negative logic/active LOW) |

Clk. Polarity = steady state low (0)

Clk. Phase = rising clock edge (0)

5.3.5.3 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 | <i>data</i> | 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 | <i>cmd</i> | 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.

5.3.6 System Tick (SysTick)

Collaboration diagram for System Tick (SysTick):



Files

- file [SysTick.c](#)
Implementation details for SysTick functions.
- file [SysTick.h](#)
Driver module for using SysTick-based timing and/or interrupts.

Functions

- void **SysTick_Timer_Init** (void)
Initialize SysTick for timing purposes.
- void **SysTick_Wait1ms** (uint32_t delay_ms)
Delay for specified amount of time in [ms]. Assumes f_bus = 80[MHz].
- void [SysTick_Interrupt_Init](#) (uint32_t time_ms)
Initialize SysTick for interrupts.

5.3.6.1 Detailed Description

Functions for timing and periodic interrupts via SysTick.

5.3.6.2 Function Documentation

SysTick_Interrupt_Init()

```
void SysTick_Interrupt_Init (
    uint32_t time_ms )
```

Initialize SysTick for interrupts.

Parameters

| | |
|----------------------|---------------------------------------------------------------|
| <code>time_ms</code> | Time in [ms] between interrupts. Cannot be more than 200[ms]. |
|----------------------|---------------------------------------------------------------|

5.3.7 Timer

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]

5.3.7.1 Detailed Description

Functions for timing and periodic interrupts via general-purpose timer modules (GPTM).

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

5.3.7.3 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>timer</i> ↔ <i>Name_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.

5.3.7.4 Variable Documentation**TIMER_POOL**

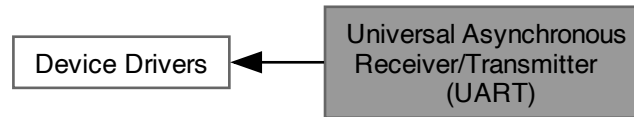
```
TimerStruct_t TIMER_POOL[6] [static]
```

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 }
}
```

5.3.8 Universal Asynchronous Receiver/Transmitter (UART)

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 to the UART.
- `void UART_WriteFloat` (Uart_t uart, double n, uint8_t numDecimals)
Write a floating-point number to the UART.

Variables

- static `UartStruct_t UART_ARR` [8]

5.3.8.1 Detailed Description

Functions for UART-based communication.

5.3.8.2 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. |

5.3.8.3 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 }
}
```

5.3.9 Interrupt Service Routines

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

5.3.9.1 Detailed Description

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

5.3.9.2 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.

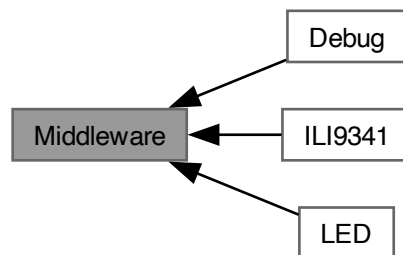
See also

[ISR_clearPending\(\)](#)

5.4 Middleware

High-level device driver modules.

Collaboration diagram for Middleware:



Modules

- [Debug](#)
- [ILI9341](#)
- [LED](#)

5.4.1 Detailed Description

High-level device driver modules.

These modules contain functions for interfacing with external devices/peripherals via the use of low-level drivers.

5.4.2 Debug

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.

5.4.2.1 Detailed Description

Module for debugging functions, including serial output and assertions.

5.4.2.2 Function Documentation

Debug_Init()

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

Initialize the Debug module.

Parameters

| | | |
|----|-------------|--------------------------------|
| in | <i>uart</i> | 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

| | |
|----------------|-----------------------------------------|
| <i>message</i> | (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

| | | |
|-----------|------------|--------------------------------|
| <i>in</i> | <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.

5.4.3 ILI9341

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 ,
[VSCRADD](#) = 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 }

5.4.3.1 Detailed Description

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

5.4.3.2 Enumeration Type Documentation

anonymous enum

anonymous enum

Enumerator

| | |
|------------------|----------------------------------------------------|
| ILI9341_NUM_COLS | 5.4.3.3 of columns available on the display |
| ILI9341_NUM_ROWS | 5.4.3.4 of rows available on the display |

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. |

5.4.3.5 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.

Currently unused commands `#define RDDST (uint8_t) 0x09` /// Read Display Status `#define RDDMADCTL (uint8_t) 0x0B` /// Read Display MADCTL `#define RDDCOLMOD (uint8_t) 0x0C` /// Read Display Pixel Format `#define RGBSET (uint8_t) 0x2D` /// Color Set `#define RAMRD (uint8_t) 0x2E` /// Memory Read `#define WRITE_MEMORY_CONTINUE (uint8_t) 0x3C` /// Write_Memory_Continue `#define READ_MEMORY_CONTINUE (uint8_t) 0x3E` /// Read_Memory_Continue `#define WRDISBV (uint8_t) 0x51` /// Write Display Brightness `#define RDDISBV (uint8_t) 0x52` /// Read Display Brightness `#define IFMODE (uint8_t) 0xB0` /// RGB Interface Signal Control (i.e. Interface Mode Control) `#define INVTR (uint8_t) 0xB4` /// Display Inversion Control

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 | timer | Hardware timer to use during reset. |
|----|-------|-------------------------------------|

The LCD driver's RESET pin requires a negative logic (i.e. active LOW) signal for ≥ 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 | timer | 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 ≥ 5 [ms] before sending further commands regardless of the selected mode.

It's also necessary to wait 120 [ms] before sending `SPLOUT` after sending `SPLIN` 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 | | |
|----|--|--|

$\text{startCol} \leq \text{endCol} < 240$

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 | ... |

5.4.4 LED

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`

Functions

- `Led_t Led_Init` (GpioPort_t gpioPort, GPIO_Pin_t pin)
Initialize a light-emitting diode (LED) as an `Led_t`.
- `bool Led_isInit` (Led_t led)
- `GpioPort_t Led_GetPort` (Led_t led)
Get the GPIO port associated with the LED.
- `GPIO_Pin_t Led_GetPin` (Led_t led)
Get the GPIO pin associated with the LED.
- `bool Led_isOn` (Led_t led)
Check the LED's status.
- `void Led_TurnOn` (Led_t led)
Turn the LED ON.
- `void Led_TurnOff` (Led_t led)
Turn the LED OFF.
- `void Led_Toggle` (Led_t led)
Toggle the LED (i.e. OFF -> ON or ON -> OFF).

Variables

- `static LedStruct_t Led_ObjPool` [LED_POOL_SIZE] = { 0 }
- `static uint8_t num_free_leds` = LED_POOL_SIZE

5.4.4.1 Detailed Description

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

5.4.4.2 Function Documentation

Led_Init()

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

Initialize a light-emitting diode (LED) as an `Led_t`.

Parameters

| | | |
|-----|-----------------|-----------------------------------------------|
| in | <i>gpioPort</i> | Pointer to a struct representing a GPIO port. |
| in | <i>pin</i> | GPIO pin to use. |
| out | <i>Led_t*</i> | Pointer to LED data structure. |

Led_GetPort()

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

Get the GPIO port associated with the LED.

Parameters

| | | |
|-----|---------------------|----------------------------------------|
| in | <i>led</i> | Pointer to LED data structure. |
| out | <i>GPIO_Port_t*</i> | Pointer to a GPIO port data structure. |

Led_GetPin()

```
GPIO_Pin_t Led_GetPin (
    Led_t led )
```

Get the GPIO pin associated with the LED.

Parameters

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

Led_isOn()

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

Check the LED's status.

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. |

Led_TurnOn()

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

Turn the LED ON.

Parameters

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

Led_TurnOff()

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

Turn the LED OFF.

Parameters

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

Led_Toggle()

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

Toggle the LED (i.e. OFF -> ON or ON -> OFF).

Parameters

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

5.5 Main**Files**

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

Enumerations

- enum { **DAQ_VECTOR_NUM** = INT_ADC0SS3 , **PROC_VECTOR_NUM** = INT_CAN0 , **LCD_VECTOR_NUM** = INT_TIMER1A }
- enum { **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_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) }

Functions

- static void [DAQ_Handler](#) (void)
Reads ADC output, converts to raw voltage sample, and sends to next FIFO.
- static void [Processing_Handler](#) (void)
Removes baseline drift and PLI from a sample, and moves it to the QRS/LCD FIFOs.
- static void [LCD_Handler](#) (void)
Applies a 0.5-40 [Hz] bandpass filter and plots the sample to the waveform.
- 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 bool [QRS_bufferIsFull](#) = false
- 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 [LCD_heartRateIsReady](#) = false
- static float32_t [QRS_processingBuffer](#) [[QRS_ARRAY_LEN](#)] = { 0 }
- static uint16_t [LCD_prevSampleBuffer](#) [[LCD_X_MAX](#)] = { 0 }

5.5.1 Detailed Description

5.5.2 Enumeration Type Documentation

anonymous enum

anonymous enum

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 |

anonymous enum

anonymous enum

Enumerator

| | |
|---------------------|-------------------------------------------------|
| LCD_TOP_LINE | separates waveform from text |
| LCD_WAVE_NUM↔ _Y | num. of y-val's available for plotting waveform |

5.5.3 Function Documentation

DAQ_Handler()

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

Reads ADC output, converts to raw voltage sample, and sends to next FIFO.

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.

Precondition

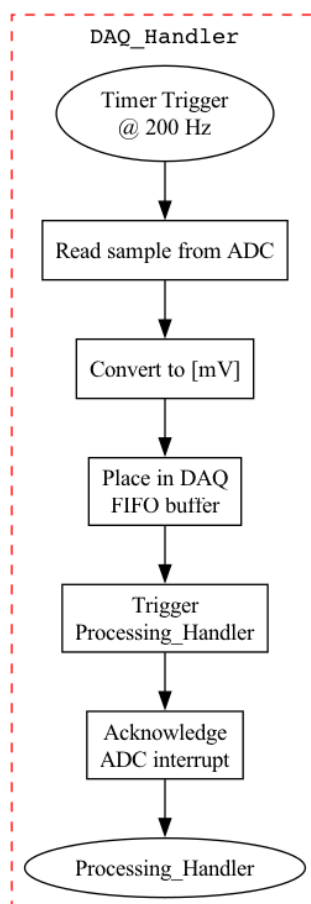
Initialize the DAQ module.

Postcondition

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

See also

[DAQ_Init\(\)](#), [Processing_Handler\(\)](#)



Processing_Handler()

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

Removes baseline drift and PLI from a sample, and moves it to the QRS/LCD FIFOs.

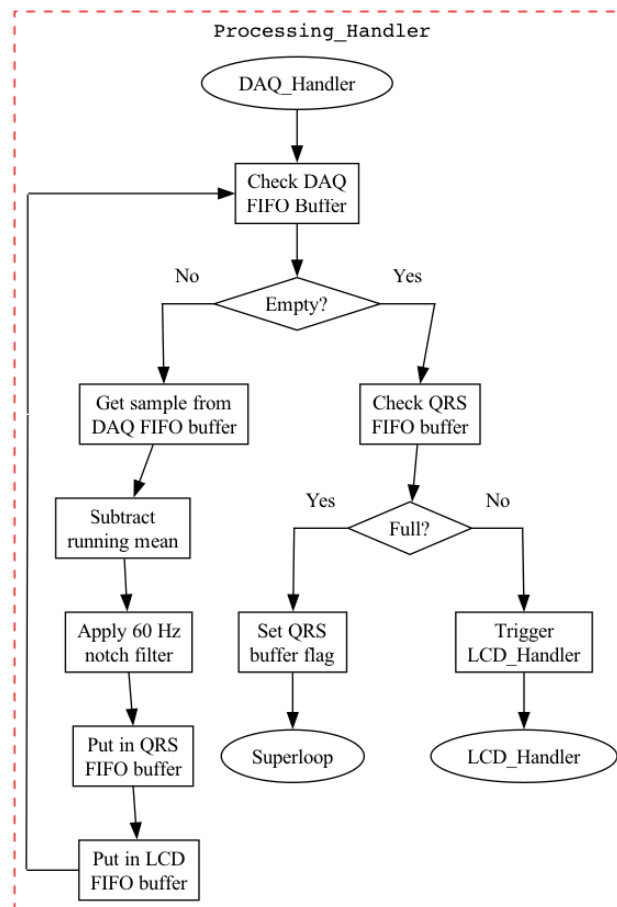
This ISR has a priority level of 1, is triggered by the DAQ ISR, and triggers the LCD Handler. It also notifies the superloop in [main\(\)](#) when the QRS buffer is full.

Postcondition

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

See also

[DAQ_Handler\(\)](#), [main\(\)](#), [LCD_Handler\(\)](#)



LCD_Handler()

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

Applies a 0.5-40 [Hz] bandpass filter and plots the sample to the waveform.

This ISR has a priority level of 1 and is triggered by the Processing ISR.

Precondition

Initialize the LCD module.

Postcondition

The bandpass-filtered sample is plotted to the LCD.

See also

[LCD_Init\(\)](#), [Processing_Handler\(\)](#)

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.

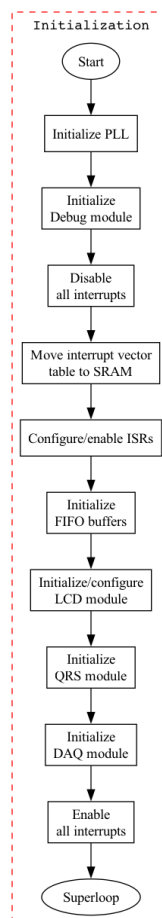


Figure 1 Flowchart for the initialization phase.

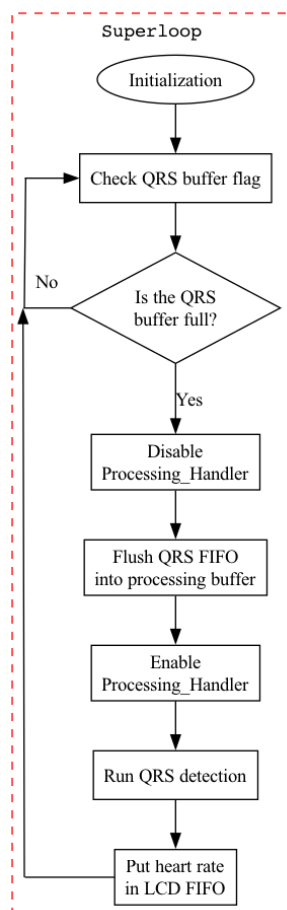
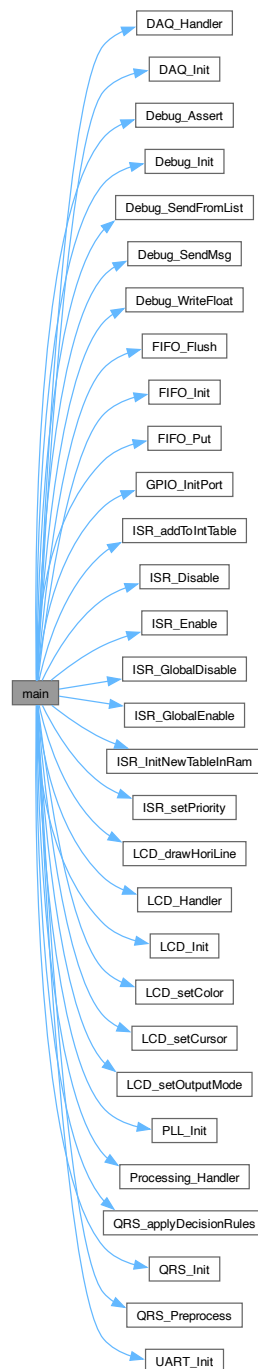


Figure 2 Flowchart for the superloop.

Here is the call graph for this function:



6 Data Structure Documentation

6.1 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](#)

6.2 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](#)

6.3 Led_t Struct Reference

Data Fields

- GpioPort_t **GPIO_PORT_PTR**
pointer to GPIO port data structure
- GPIO_Pin_t **GPIO_PIN**
GPIO pin number.
- bool **is_ON**
state indicator
- bool **isInit**

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

- [Led.c](#)

6.4 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](#)

6.5 Uart_t Struct Reference

Data Fields

- const uint32_t **BASE_ADDRESS**
- register_t const **FLAG_R_ADDRESS**
- GpioPort_t **GPIO_PORT**
pointer to GPIO port data structure
- GPIO_Pin_t **RX_PIN_NUM**
GPIO pin number.
- GPIO_Pin_t **TX_PIN_NUM**
GPIO pin number.
- bool **isInit**

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

- [UART.c](#)

7 File Documentation

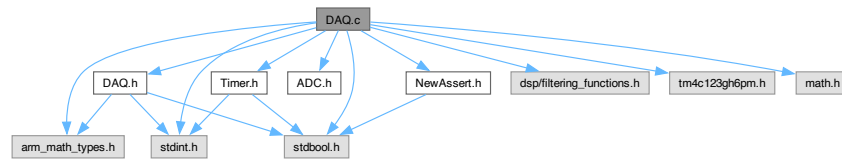
7.1 DAQ.c File Reference

Source code for DAQ module.

```
#include "DAQ.h"
#include "ADC.h"
#include "Timer.h"
#include "NewAssert.h"
#include "arm_math_types.h"
#include "dsp/filtering_functions.h"
#include "tm4c123gh6pm.h"
#include <math.h>
#include <stdbool.h>
```

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

Include dependency graph for DAQ.c:



Macros

- `#define SAMPLING_PERIOD_MS 5`
sampling period in ms ($T_s = 1/f_s$)

Functions

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.

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.

Digital Filters

- enum {
 NUM_STAGES_NOTCH = 6 , **NUM_COEFFS_NOTCH** = NUM_STAGES_NOTCH * 5 , **STATE_BUFFER_SIZE_NOTCH** = NUM_STAGES_NOTCH * 4 , **NUM_STAGES_BANDPASS** = 4 ,
 NUM_COEFFS_DAQ_BANDPASS = NUM_STAGES_BANDPASS * 5 , **STATE_BUFFER_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_BUFFER_SIZE_NOTCH]
• static const Filter_t **notchFiltStruct** = { NUM_STAGES_NOTCH, stateBuffer_Notch, COEFFS_NOTCH }
• static const Filter_t *const **notchFilter** = ¬chFiltStruct
• static float32_t **stateBuffer_Bandpass** [STATE_BUFFER_SIZE_BANDPASS]
• static const Filter_t **bandpassFiltStruct**
• static const Filter_t *const **bandpassFilter** = &bandpassFiltStruct

7.1.1 Detailed Description

Source code for DAQ module.

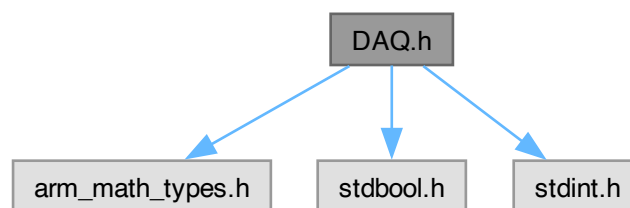
Author

Bryan McElvy

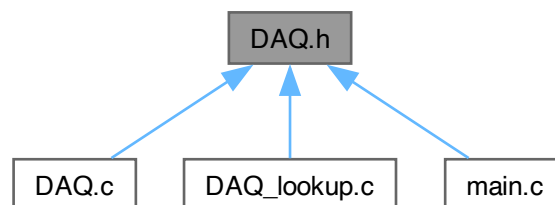
7.2 DAQ.h File Reference

Application software for handling data acquisition (DAQ) functions.

```
#include "arm_math_types.h"  
#include <stdbool.h>  
#include <stdint.h>  
Include dependency graph for DAQ.h:
```



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



Macros

- #define **DAQ_LOOKUP_MAX** ((float32_t) 5.5f)
- #define **DAQ_LOOKUP_MIN** ((float32_t) (-5.5f))

Functions

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.
- float32_t **DAQ_convertToMilliVolts** (uint16_t sample)
Convert a 12-bit ADC sample to a floating-point voltage value via LUT.
- void **DAQ_acknowledgeInterrupt** (void)
Acknowledge the ADC interrupt.

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.

7.2.1 Detailed Description

Application software for handling data acquisition (DAQ) functions.

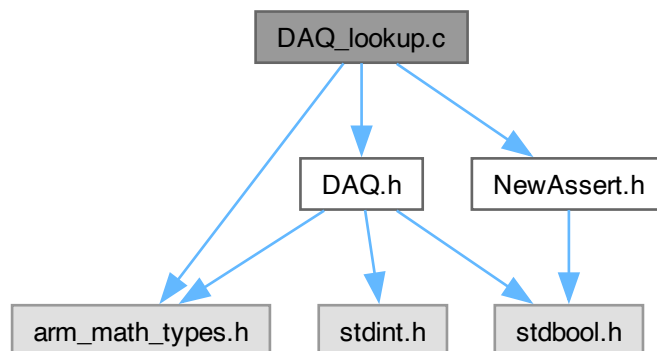
Author

Bryan McElvy

7.3 DAQ_lookup.c File Reference

Source code for DAQ module's lookup table.

```
#include "DAQ.h"
#include "NewAssert.h"
#include "arm_math_types.h"
Include dependency graph for DAQ_lookup.c:
```



Functions

Reading Input Data

- float32_t [DAQ_convertToMilliVolts](#) (uint16_t sample)
Convert a 12-bit ADC sample to a floating-point voltage value via LUT.

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.

7.3.1 Detailed Description

Source code for DAQ module's lookup table.

Author

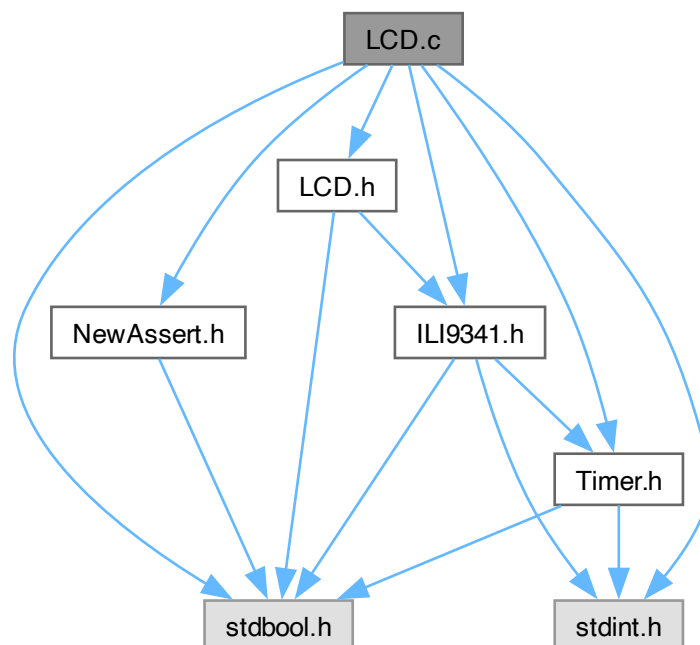
Bryan McElvy

7.4 LCD.c File Reference

Source code for LCD module.

```
#include "LCD.h"
#include "ILI9341.h"
#include "Timer.h"
#include "NewAssert.h"
#include <stdbool.h>
#include <stdint.h>
```

Include dependency graph for LCD.c:



Macros

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

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.
- static void `LCD_plotSample` (uint16_t x, uint16_t y, LCD_Color_t color)
Plot a sample at coordinates (x, y).

Init./Config. Functions

- 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` (LCD_Color_t color)
Set the color value.

Drawing Functions

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

Writing Functions

- 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)

Variables

- 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**
 - uint16_t **colNum**
 - uint8_t **R_val**
5 or 6-bit R value
 - uint8_t **G_val**
6-bit G value
 - uint8_t **B_val**
5 or 6-bit B value
 - bool **isInit**
if true, LCD has been initialized
 } **lcd** = { 0 }
- const uint8_t *const **FONT_ARRAY** [128]

7.4.1 Detailed Description

Source code for LCD module.

Author

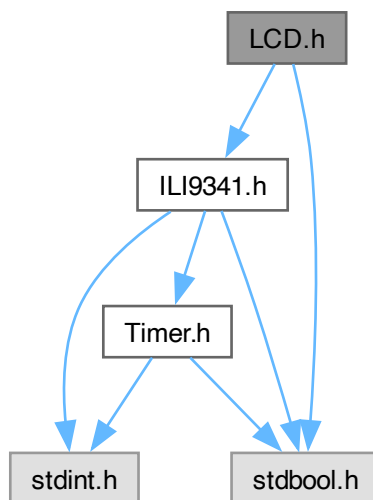
Bryan McElvy

7.5 LCD.h File Reference

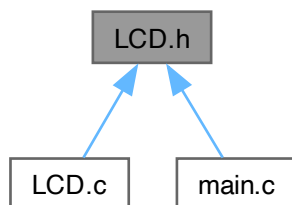
Header file for LCD module.

```
#include "ILI9341.h"
#include <stdbool.h>
```

Include dependency graph for LCD.h:



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



Functions

Drawing Functions

- 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, bool isFilled)
Draw a rectangle of size dx x dy onto the display. The bottom-left corner will be located at (x1, y1).
- static void [LCD_plotSample](#) (uint16_t x, uint16_t y, LCD_Color_t color)

Init./Config. Functions

- enum { **LCD_X_MAX** = ILI9341_NUM_ROWS - 1 , **LCD_Y_MAX** = ILI9341_NUM_COLS - 1 }
- enum **LCD_Color_t** {
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** (LCD_Color_t color)
Set the color value.

Writing Functions

- enum { **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)

7.5.1 Detailed Description

Header file for LCD module.

Author

Bryan McElvy

This module is essentially a higher-level interface to the ILI9341 module.

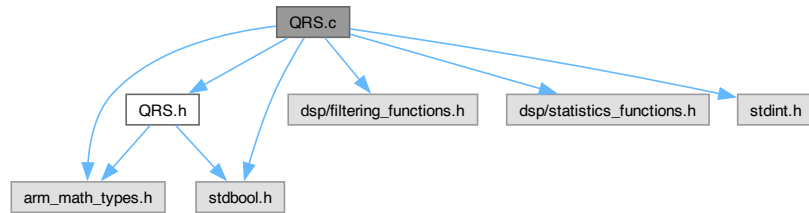
7.6 QRS.c File Reference

Source code for QRS detection module.

```
#include "QRS.h"
#include "arm_math_types.h"
#include "dsp/filtering_functions.h"
#include "dsp/statistics_functions.h"
#include <stdbool.h>
```

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

Include dependency graph for QRS.c:



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))`

Functions

Implementation-specific Functions

- 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.

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 , **STATE_BUFF_SIZE_DERFILT** = NUM_COEFF_DERFILT + BLOCK_SIZE_↵
 - _DERFILT - 1** , **BLOCK_SIZE_MOVAVG** = 1 , **NUM_COEFF_MOVAVG** = 10 ,
 - 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

7.6.1 Detailed Description

Source code for QRS detection module.

Author

Bryan McElvy

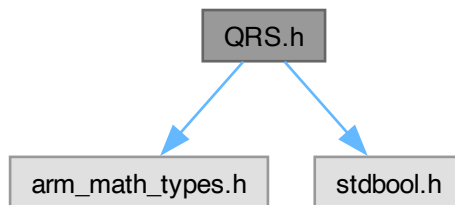
The algorithm used in this file is a simplified version of the Pan-Tompkins algorithm. Specifically, this version currently only uses the integrated signal for the thresholding, and also completely omits the searchback and T wave discrimination parts of the original.

7.7 QRS.h File Reference

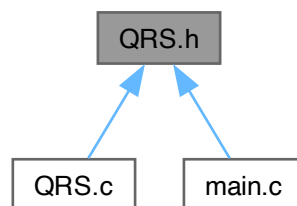
Header file for QRS detection module.

```
#include "arm_math_types.h"
#include <stdbool.h>
```

Include dependency graph for QRS.h:



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



Macros

- `#define QRS_SAMP_FREQ ((uint32_t) 200)`
- `#define QRS_SAMP_PERIOD_SEC ((float32_t) 0.005f)`
- `#define QRS_NUM_SAMP ((uint16_t) (1 << 11))`

Functions

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.

7.7.1 Detailed Description

Header file for QRS detection module.

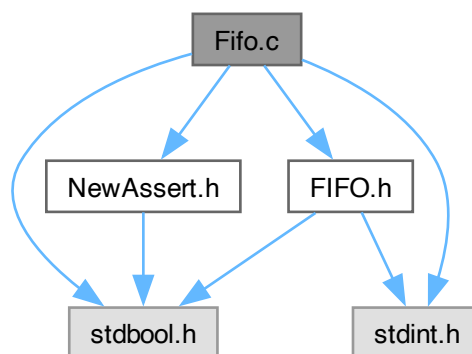
Author

Bryan McElvy

7.8 Fifo.c File Reference

Source code for FIFO buffer module.

```
#include "FIFO.h"
#include "NewAssert.h"
#include <stdbool.h>
#include <stdint.h>
Include dependency graph for Fifo.c:
```



Data Structures

- struct [Fifo_t](#)

Functions

- `Fifo_t` [FIFO_Init](#) (volatile uint32_t buffer[], const uint32_t N)
Initialize a FIFO buffer of length N.

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_TransferOne` (volatile `Fifo_t` srcFifo, volatile `Fifo_t` destFifo)
Transfer a value from one FIFO buffer to another.

Bulk Removal

- void `FIFO_Flush` (volatile `Fifo_t` fifo, `uint32_t` outputBuffer[])
Empty the FIFO buffer's contents into an array.
- void `FIFO_Reset` (volatile `Fifo_t` fifo)
Reset the FIFO buffer.
- void `FIFO_TransferAll` (volatile `Fifo_t` srcFifo, volatile `Fifo_t` destFifo)
Transfer the contents of one FIFO buffer to another.

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.

Variables

- static `FifoStruct_t` `fifoPool` [FIFO_POOL_SIZE] = { 0 }
pre-allocated pool
- static `uint8_t` `numFreeFifos` = FIFO_POOL_SIZE

7.8.1 Detailed Description

Source code for FIFO buffer module.

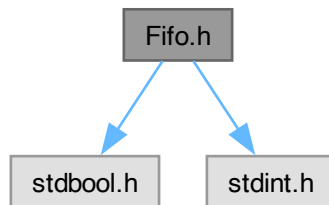
Author

Bryan McElvy

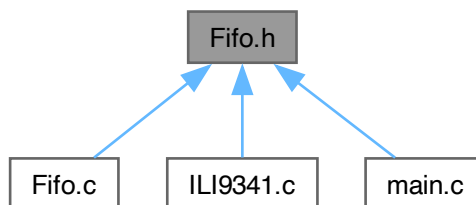
7.9 Fifo.h File Reference

Header file for FIFO buffer implementation.

```
#include <stdbool.h>
#include <stdint.h>
Include dependency graph for Fifo.h:
```



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



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.

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_TransferOne](#) (volatile Fifo_t srcFifo, volatile Fifo_t destFifo)
Transfer a value from one FIFO buffer to another.

Bulk Removal

- void [FIFO_Flush](#) (volatile Fifo_t fifo, uint32_t outputBuffer[])
Empty the FIFO buffer's contents into an array.
- void [FIFO_Reset](#) (volatile Fifo_t fifo)
Reset the FIFO buffer.
- void [FIFO_TransferAll](#) (volatile Fifo_t srcFifo, volatile Fifo_t destFifo)
Transfer the contents of one FIFO buffer to another.

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.

7.9.1 Detailed Description

Header file for FIFO buffer implementation.

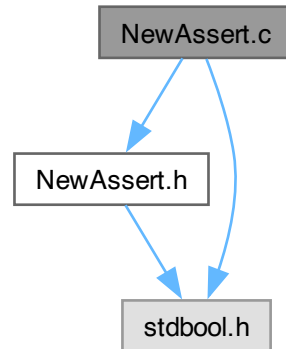
Author

Bryan McElvy

7.10 NewAssert.c File Reference

Source code for custom `assert` implementation.

```
#include "NewAssert.h"
#include <stdbool.h>
Include dependency graph for NewAssert.c:
```



Functions

- void [Assert](#) (bool condition)

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

7.10.1 Detailed Description

Source code for custom `assert` implementation.

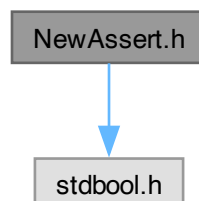
Author

Bryan McElvy

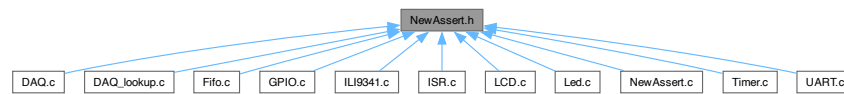
7.11 NewAssert.h File Reference

Header file for custom `assert` implementation.

```
#include <stdbool.h>
Include dependency graph for NewAssert.h:
```



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



Functions

- void [Assert](#) (bool condition)

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

7.11.1 Detailed Description

Header file for custom `assert` implementation.

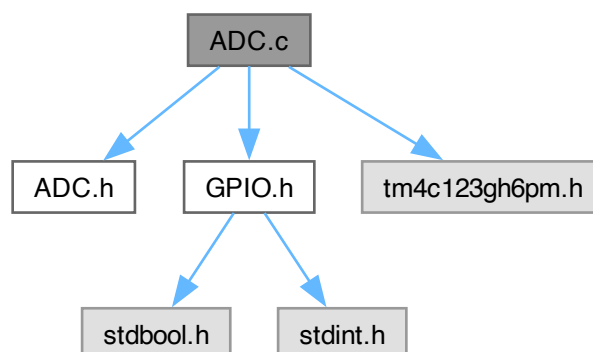
Author

Bryan McElvy

7.12 ADC.c File Reference

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

```
#include "ADC.h"
#include "GPIO.h"
#include "tm4c123gh6pm.h"
Include dependency graph for ADC.c:
```



Functions

- void [ADC_Init](#) (void)
Initialize ADC0 as a single-input analog-to-digital converter.

7.12.1 Detailed Description

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

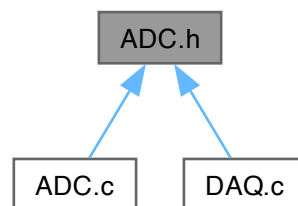
Author

Bryan McElvy

7.13 ADC.h File Reference

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

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



Functions

- void [ADC_Init](#) (void)
Initialize ADC0 as a single-input analog-to-digital converter.

7.13.1 Detailed Description

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

Author

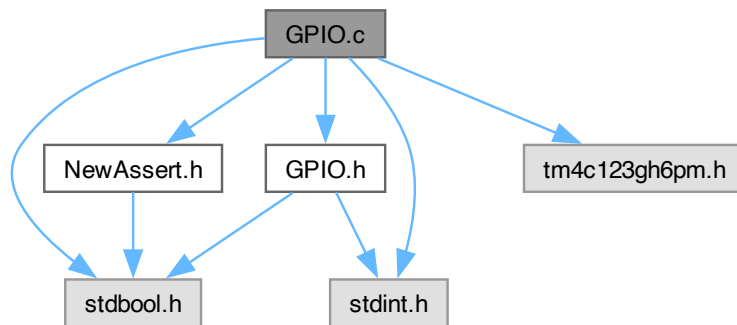
Bryan McElvy

7.14 GPIO.c File Reference

Source code for GPIO module.

```
#include "GPIO.h"
#include <NewAssert.h>
#include "tm4c123gh6pm.h"
#include <stdbool.h>
#include <stdint.h>
```

Include dependency graph for GPIO.c:



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 }

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, `GPIO_Pin_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, `GPIO_Pin_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, `GPIO_Pin_t` pinMask)
Activate the specified pins' internal pull-up resistors.
- `void` [GPIO_ConfigPullDown](#) (`GpioPort_t` gpioPort, `GPIO_Pin_t` pinMask)
Activate the specified pins' internal pull-down resistors.
- `void` [GPIO_ConfigDriveStrength](#) (`GpioPort_t` gpioPort, `GPIO_Pin_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, `GPIO_Pin_t` pinMask)
Enable digital I/O for the specified pins.
- `void` [GPIO_DisableDigital](#) (`GpioPort_t` gpioPort, `GPIO_Pin_t` pinMask)
Disable digital I/O for the specified pins.
- `void` [GPIO_ConfigInterrupts_Edge](#) (`GpioPort_t` gpioPort, `GPIO_Pin_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, `GPIO_Pin_t` pinMask)
Configure the specified GPIO pins to trigger an interrupt on both edges of an input.
- `void` [GPIO_ConfigInterrupts_LevelTrig](#) (`GpioPort_t` gpioPort, `GPIO_Pin_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.
- `uint32_t` [GPIO_getDataRegister](#) (`GpioPort_t` gpioPort)
Get the address of a GPIO port's data register.
- `uint8_t` [GPIO_ReadPins](#) (`GpioPort_t` gpioPort, `GPIO_Pin_t` pinMask)
Read from the specified GPIO pin.
- `void` [GPIO_WriteHigh](#) (`GpioPort_t` gpioPort, `GPIO_Pin_t` pinMask)
Write a 1 to the specified GPIO pins.
- `void` [GPIO_WriteLow](#) (`GpioPort_t` gpioPort, `GPIO_Pin_t` pinMask)
Write a 0 to the specified GPIO pins.
- `void` [GPIO_Toggle](#) (`GpioPort_t` gpioPort, `GPIO_Pin_t` pinMask)
Toggle the specified GPIO pins.
- `void` [GPIO_ConfigAltMode](#) (`GpioPort_t` gpioPort, `GPIO_Pin_t` pinMask)
Activate the alternate mode for the specified pins.
- `void` [GPIO_ConfigPortCtrl](#) (`GpioPort_t` gpioPort, `GPIO_Pin_t` pinMask, `uint8_t` fieldEncoding)
Specify the alternate mode to use for the specified pins.
- `void` [GPIO_ConfigAnalog](#) (`GpioPort_t` gpioPort, `GPIO_Pin_t` pinMask)
Activate analog mode for the specified GPIO pins.

Variables

- static [GpioPortStruct_t](#) [GPIO_PTR_ARR](#) [6]

7.14.1 Detailed Description

Source code for GPIO module.

Author

Bryan McElvy

7.14.2 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. |
|----|-----------------|--------------------------|

Returns

`GPIO_Port_t*` Pointer to the GPIO port's `struct`.

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,
    GPIO_Pin_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 <code>OUTPUT</code> pin(s). |

GPIO_ConfigDirInput()

```
void GPIO_ConfigDirInput (
    GpioPort_t gpioPort,
    GPIO_Pin_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 <code>INPUT</code> pin(s). |

GPIO_ConfigPullUp()

```
void GPIO_ConfigPullUp (
    GpioPort_t gpioPort,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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()

```
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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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). |

7.14.3 Variable Documentation**GPIO_PTR_ARR**

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

Initial value:

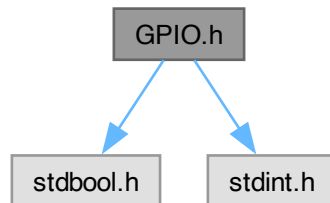
```
= {
    { GPIO_PORTA_BASE_ADDRESS, (GPIO_PORTA_BASE_ADDRESS + GPIO_DATA_R_OFFSET), false },
    { GPIO_PORTB_BASE_ADDRESS, (GPIO_PORTB_BASE_ADDRESS + GPIO_DATA_R_OFFSET), false },
    { GPIO_PORTC_BASE_ADDRESS, (GPIO_PORTC_BASE_ADDRESS + GPIO_DATA_R_OFFSET), false },
    { GPIO_PORTD_BASE_ADDRESS,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

7.15 GPIO.h File Reference

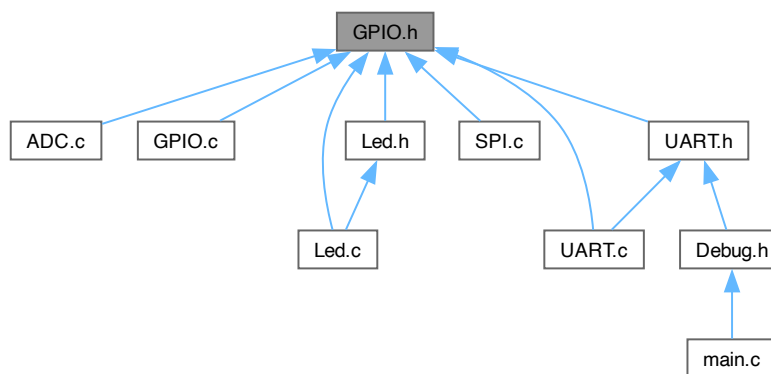
Header file for general-purpose input/output (GPIO) device driver.

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

Include dependency graph for GPIO.h:



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



Enumerations

- enum **GPIO_PortName_t** {
A, **B**, **C**, **D**,
E, **F** }
- enum **GPIO_Pin_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 {
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`.
- `uint32_t` [GPIO_getBaseAddr](#) (`GpioPort_t` gpioPort)
Get the base address of a GPIO port.
- `bool` [GPIO_isPortInit](#) (`GpioPort_t` gpioPort)
Check if the GPIO port is initialized.
- `void` [GPIO_ConfigDirOutput](#) (`GpioPort_t` gpioPort, `GPIO_Pin_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, `GPIO_Pin_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, `GPIO_Pin_t` pinMask)
Activate the specified pins' internal pull-up resistors.
- `void` [GPIO_ConfigPullDown](#) (`GpioPort_t` gpioPort, `GPIO_Pin_t` pinMask)
Activate the specified pins' internal pull-down resistors.
- `void` [GPIO_ConfigDriveStrength](#) (`GpioPort_t` gpioPort, `GPIO_Pin_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, `GPIO_Pin_t` pinMask)
Enable digital I/O for the specified pins.
- `void` [GPIO_DisableDigital](#) (`GpioPort_t` gpioPort, `GPIO_Pin_t` pinMask)
Disable digital I/O for the specified pins.
- `void` [GPIO_ConfigInterrupts_Edge](#) (`GpioPort_t` gpioPort, `GPIO_Pin_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, `GPIO_Pin_t` pinMask)
Configure the specified GPIO pins to trigger an interrupt on both edges of an input.
- `void` [GPIO_ConfigInterrupts_LevelTrig](#) (`GpioPort_t` gpioPort, `GPIO_Pin_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.
- `uint32_t` [GPIO_getDataRegister](#) (`GpioPort_t` gpioPort)
Get the address of a GPIO port's data register.
- `uint8_t` [GPIO_ReadPins](#) (`GpioPort_t` gpioPort, `GPIO_Pin_t` pinMask)
Read from the specified GPIO pin.
- `void` [GPIO_WriteHigh](#) (`GpioPort_t` gpioPort, `GPIO_Pin_t` pinMask)
Write a 1 to the specified GPIO pins.
- `void` [GPIO_WriteLow](#) (`GpioPort_t` gpioPort, `GPIO_Pin_t` pinMask)
Write a 0 to the specified GPIO pins.
- `void` [GPIO_Toggle](#) (`GpioPort_t` gpioPort, `GPIO_Pin_t` pinMask)
Toggle the specified GPIO pins.
- `void` [GPIO_ConfigAltMode](#) (`GpioPort_t` gpioPort, `GPIO_Pin_t` pinMask)
Activate the alternate mode for the specified pins.
- `void` [GPIO_ConfigPortCtrl](#) (`GpioPort_t` gpioPort, `GPIO_Pin_t` pinMask, `uint8_t` fieldEncoding)
Specify the alternate mode to use for the specified pins.
- `void` [GPIO_ConfigAnalog](#) (`GpioPort_t` gpioPort, `GPIO_Pin_t` pinMask)
Activate analog mode for the specified GPIO pins.

7.15.1 Detailed Description

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

Author

Bryan McElvy

7.15.2 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. |
|----|-----------------|--------------------------|

Returns

GPIO_Port_t* Pointer to the GPIO port's struct.

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_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_ConfigDirOutput()

```
void GPIO_ConfigDirOutput (
    GpioPort_t gpioPort,
    GPIO_Pin_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 <code>OUTPUT</code> pin(s). |

GPIO_ConfigDirInput()

```
void GPIO_ConfigDirInput (
    GpioPort_t gpioPort,
    GPIO_Pin_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 <code>INPUT</code> pin(s). |

GPIO_ConfigPullUp()

```
void GPIO_ConfigPullUp (
    GpioPort_t gpioPort,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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()

```
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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_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,
    GPIO_Pin_t pinMask )
```

Activate analog mode for the specified GPIO pins.

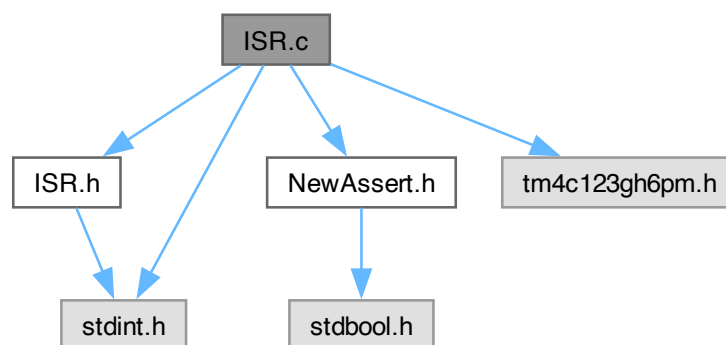
Parameters

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

7.16 ISR.c File Reference

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

```
#include "ISR.h"
#include "NewAssert.h"
#include "tm4c123gh6pm.h"
#include <stdint.h>
Include dependency graph for ISR.c:
```



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)`

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

7.16.1 Detailed Description

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

Author

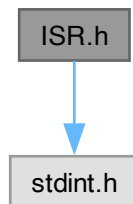
Bryan McElvy

7.17 ISR.h File Reference

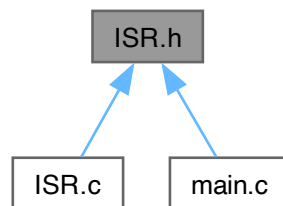
Header file for interrupt service routine (ISR) configuration module.

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

Include dependency graph for ISR.h:



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



Typedefs

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

Functions

- void **ISR_GlobalDisable** (void)
Disable all interrupts globally.
- void **ISR_GlobalEnable** (void)
Enable all interrupts globally.
- 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).*

7.17.1 Detailed Description

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

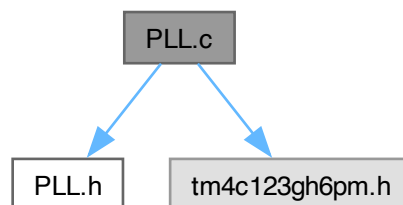
Author

Bryan McElvy

7.18 PLL.c File Reference

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

```
#include "PLL.h"
#include "tm4c123gh6pm.h"
Include dependency graph for PLL.c:
```



Functions

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

7.18.1 Detailed Description

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

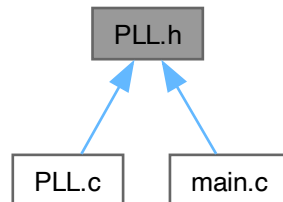
Author

Bryan McElvy

7.19 PLL.h File Reference

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

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



Functions

- void `PLL_Init` (void)
Initialize the phase-locked-loop to change the bus frequency.

7.19.1 Detailed Description

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

Author

Bryan McElvy

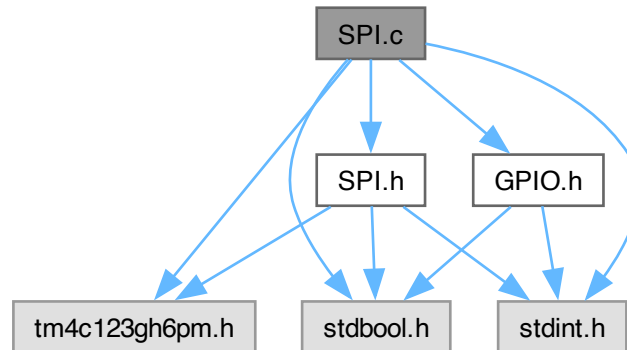
7.20 SPI.c File Reference

Source code for serial peripheral interface (SPI) module.

```
#include "SPI.h"  
#include "GPIO.h"  
#include "tm4c123gh6pm.h"  
#include <stdbool.h>
```

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

Include dependency graph for SPI.c:



Macros

- `#define SPI_SET_DC()` (GPIO_PORTA_DATA_R |= 0x40)
- `#define SPI_CLEAR_DC()` (GPIO_PORTA_DATA_R &= ~(0x40))
- `#define SPI_IS_BUSY` (SSI0_SR_R & 0x10)
- `#define SPI_TX_ISNOTFULL` (SSI0_SR_R & 0x02)

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.

7.20.1 Detailed Description

Source code for serial peripheral interface (SPI) module.

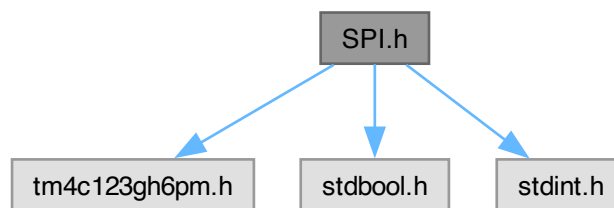
Author

Bryan McElvy

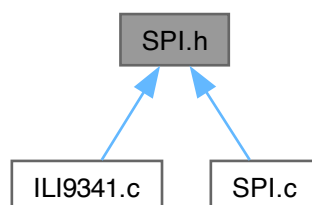
7.21 SPI.h File Reference

Header file for serial peripheral interface (SPI) module.

```
#include "tm4c123gh6pm.h"  
#include <stdbool.h>  
#include <stdint.h>  
Include dependency graph for SPI.h:
```



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



Macros

- `#define SPI_CLEAR_RESET()` (GPIO_PORTA_DATA_R &= ~(0x80))
- `#define SPI_SET_RESET()` (GPIO_PORTA_DATA_R |= 0x80)

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.

7.21.1 Detailed Description

Header file for serial peripheral interface (SPI) module.

Author

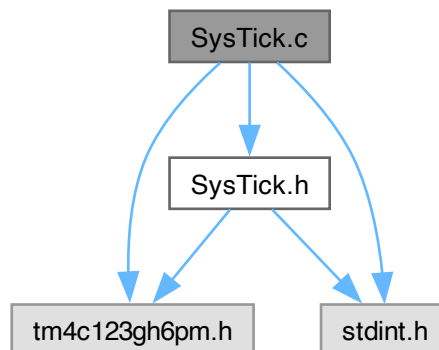
Bryan McElvy

7.22 SysTick.c File Reference

Implementation details for SysTick functions.

```
#include "SysTick.h"
#include "tm4c123gh6pm.h"
#include <stdint.h>
```

Include dependency graph for SysTick.c:



Functions

- void **SysTick_Timer_Init** (void)
Initialize SysTick for timing purposes.
- void **SysTick_Wait1ms** (uint32_t delay_ms)
Delay for specified amount of time in [ms]. Assumes f_bus = 80[MHz].
- void **SysTick_Interrupt_Init** (uint32_t time_ms)
Initialize SysTick for interrupts.

7.22.1 Detailed Description

Implementation details for SysTick functions.

Author

Bryan McElvy

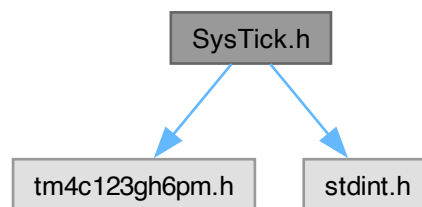
7.23 SysTick.h File Reference

Driver module for using SysTick-based timing and/or interrupts.

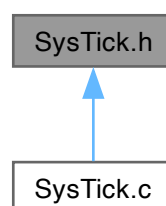
```
#include "tm4c123gh6pm.h"
```

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

Include dependency graph for SysTick.h:



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



Functions

- void **SysTick_Timer_Init** (void)
Initialize SysTick for timing purposes.
- void **SysTick_Wait1ms** (uint32_t delay_ms)
Delay for specified amount of time in [ms]. Assumes f_bus = 80[MHz].
- void **SysTick_Interrupt_Init** (uint32_t time_ms)
Initialize SysTick for interrupts.

7.23.1 Detailed Description

Driver module for using SysTick-based timing and/or interrupts.

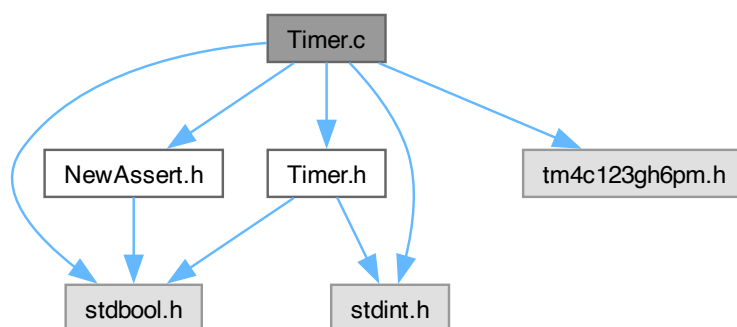
Author

Bryan McElvy

7.24 Timer.c File Reference

Source code for Timer module.

```
#include "Timer.h"
#include "NewAssert.h"
#include "tm4c123gh6pm.h"
#include <stdbool.h>
#include <stdint.h>
Include dependency graph for Timer.c:
```



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 }

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]

7.24.1 Detailed Description

Source code for Timer module.

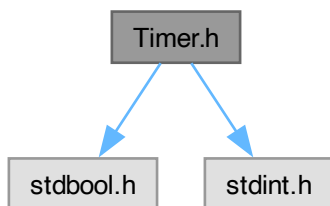
Author

Bryan McElvy

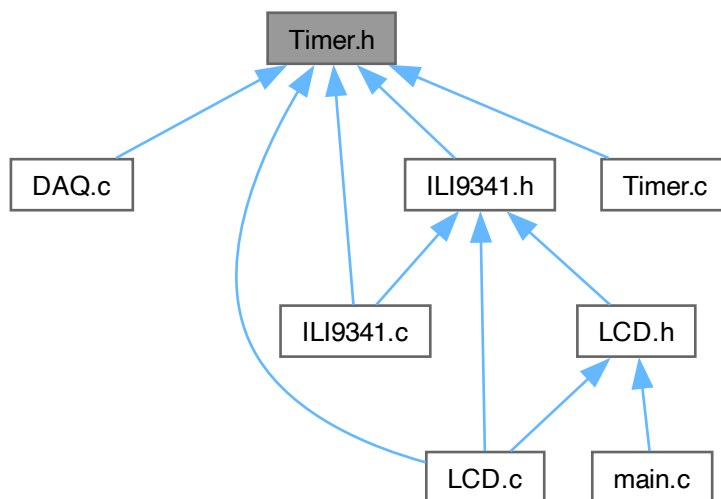
7.25 Timer.h File Reference

Device driver for general-purpose timer modules.

```
#include <stdbool.h>
#include <stdint.h>
Include dependency graph for Timer.h:
```



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



Enumerations

- 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.

7.25.1 Detailed Description

Device driver for general-purpose timer modules.

Author

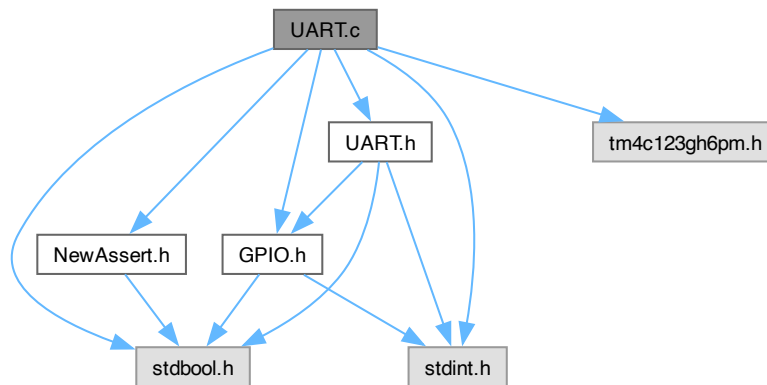
Bryan McElvy

7.26 UART.c File Reference

Source code for UART module.

```
#include "UART.h"
#include "GPIO.h"
#include "NewAssert.h"
#include "tm4c123gh6pm.h"
#include <stdbool.h>
#include <stdint.h>
```

Include dependency graph for UART.c:



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 }

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 to the UART.
- `void UART_WriteFloat` (Uart_t uart, double n, uint8_t numDecimals)
Write a floating-point number to the UART.

Variables

- static `UartStruct_t UART_ARR` [8]

7.26.1 Detailed Description

Source code for UART module.

Author

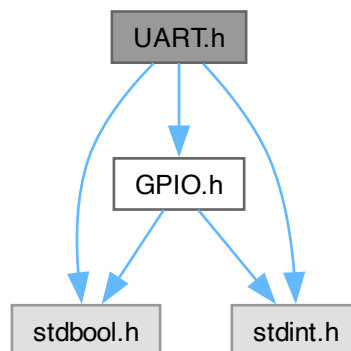
Bryan McElvy

7.27 UART.h File Reference

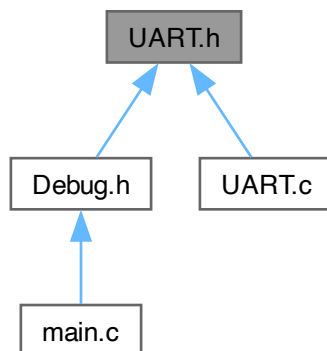
Driver module for serial communication via UART0 and UART 1.

```
#include "GPIO.h"
#include <stdbool.h>
#include <stdint.h>
```

Include dependency graph for UART.h:



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



Enumerations

- 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.

7.27.1 Detailed Description

Driver module for serial communication via UART0 and UART 1.

Author

Bryan McElvy

UART0 uses PA0 and PA1, which are not broken out but can connect to a PC's serial port via USB.

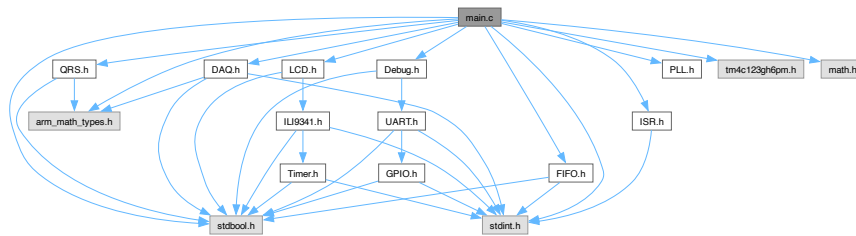
UART1 uses PB0 (Rx) and PB1 (Tx), which are broken out but do not connect to a serial port.

7.28 main.c File Reference

Main program file.

```
#include "DAQ.h"
#include "Debug.h"
#include "LCD.h"
#include "QRS.h"
#include "FIFO.h"
#include "ISR.h"
#include "PLL.h"
#include "arm_math_types.h"
#include "tm4c123gh6pm.h"
#include <math.h>
#include <stdbool.h>
#include <stdint.h>
```

Include dependency graph for main.c:



Enumerations

- enum { **DAQ_VECTOR_NUM** = INT_ADC0SS3 , **PROC_VECTOR_NUM** = INT_CAN0 , **LCD_VECTOR_NUM** = INT_TIMER1A }
- enum { **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_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) }

Functions

- static void **DAQ_Handler** (void)
Reads ADC output, converts to raw voltage sample, and sends to next FIFO.
- static void **Processing_Handler** (void)
Removes baseline drift and PLI from a sample, and moves it to the QRS/LCD FIFOs.
- static void **LCD_Handler** (void)
Applies a 0.5-40 [Hz] bandpass filter and plots the sample to the waveform.
- 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 `bool` **QRS_bufferIsFull** = false
- 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` **LCD_heartRateIsReady** = false
- static `float32_t` **QRS_processingBuffer** [[QRS_ARRAY_LEN](#)] = { 0 }
- static `uint16_t` **LCD_prevSampleBuffer** [[LCD_X_MAX](#)] = { 0 }

7.28.1 Detailed Description

Main program file.

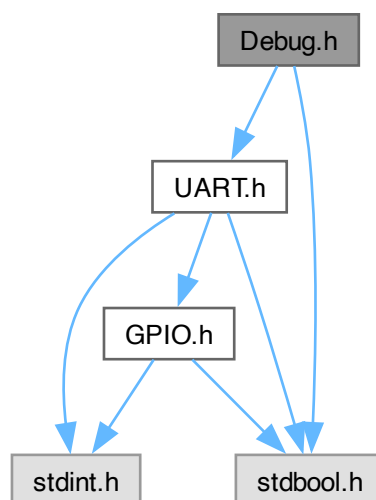
Author

Bryan McElvy

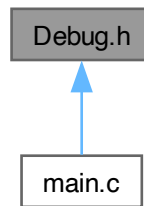
7.29 Debug.h File Reference

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

```
#include "UART.h"
#include <stdbool.h>
Include dependency graph for Debug.h:
```



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



Functions

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.

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.

7.29.1 Detailed Description

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

Author

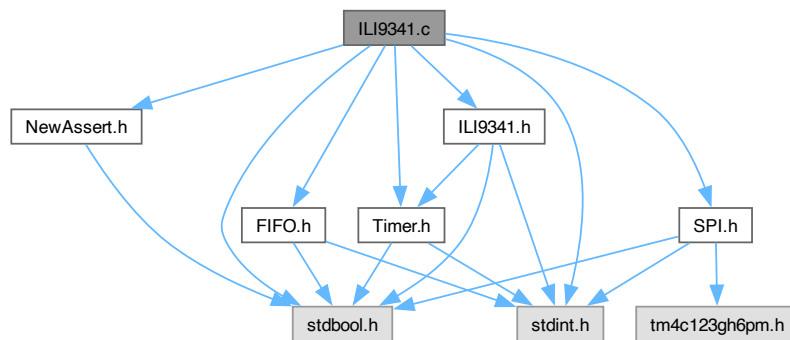
Bryan McElvy

7.30 ILI9341.c File Reference

Source code for ILI9341 module.

```
#include "ILI9341.h"
#include "SPI.h"
#include "Timer.h"
#include "FIFO.h"
#include "NewAssert.h"
#include <stdbool.h>
#include <stdint.h>
```

Include dependency graph for ILI9341.c:



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_setDisInversion](#) (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 }

7.30.1 Detailed Description

Source code for ILI9341 module.

Author

Bryan McElvy

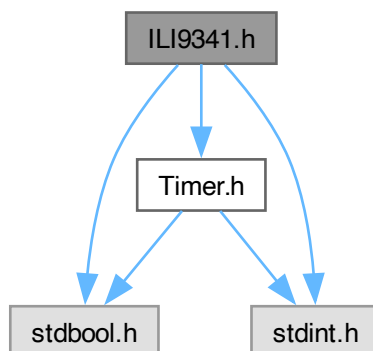
7.31 ILI9341.h File Reference

Driver module for interfacing with an ILI9341 LCD driver.

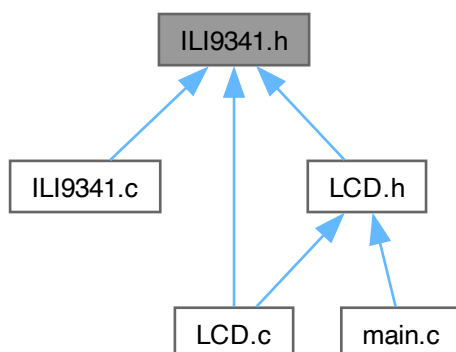
```
#include "Timer.h"
#include <stdbool.h>
```

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

Include dependency graph for ILI9341.h:



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



Enumerations

- enum { **ILI9341_NUM_COLS** = 240 , **ILI9341_NUM_ROWS** = 320 }
- enum **Cmd_t** {
NOP = 0x00 , **SWRESET** = 0x01 , **SPLIN** = 0x10 , **SPROUT** = 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 ,
VSCRSAADD = 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** = SPROUT }

- 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

- 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_setPartialArea** (uint16_t rowStart, uint16_t rowEnd)
Set the display area for partial mode. Call before activating partial mode.
- void **ILI9341_setColorExpression** (colorExpr_t colorExpr)
Set the color expression (FULL_COLORS by default).
- 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.

7.31.1 Detailed Description

Driver module for interfacing with an ILI9341 LCD driver.

Author

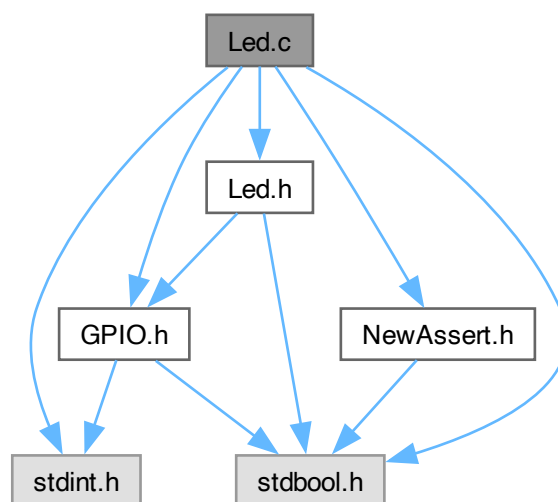
Bryan McElvy

This module contains functions for initializing and outputting graphical data to a 240RGBx320 resolution, 262K color-depth liquid crystal display (LCD). The module interfaces the LaunchPad (or any other board featuring the TM4C123GH6PM microcontroller) with an ILI9341 LCD driver chip via the serial peripheral interface (SPI) protocol.

7.32 Led.c File Reference

Source code for LED module.

```
#include "Led.h"
#include "GPIO.h"
#include "NewAssert.h"
#include <stdbool.h>
#include <stdint.h>
Include dependency graph for Led.c:
```



Data Structures

- struct [Led_t](#)

Functions

- [Led_t](#) [Led_Init](#) (GpioPort_t gpioPort, GPIO_Pin_t pin)
Initialize a light-emitting diode (LED) as an `Led_t`.
- bool [Led_isInit](#) (Led_t led)
- GpioPort_t [Led_GetPort](#) (Led_t led)
Get the GPIO port associated with the LED.
- GPIO_Pin_t [Led_GetPin](#) (Led_t led)
Get the GPIO pin associated with the LED.
- bool [Led_isOn](#) (Led_t led)
Check the LED's status.
- void [Led_TurnOn](#) (Led_t led)
Turn the LED ON.
- void [Led_TurnOff](#) (Led_t led)
Turn the LED OFF.
- void [Led_Toggle](#) (Led_t led)
Toggle the LED (i.e. OFF -> ON or ON -> OFF).

Variables

- static `LedStruct_t` `Led_ObjPool` [`LED_POOL_SIZE`] = { 0 }
- static `uint8_t` `num_free_leds` = `LED_POOL_SIZE`

7.32.1 Detailed Description

Source code for LED module.

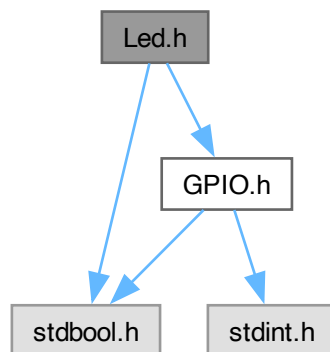
Author

Bryan McElvy

7.33 Led.h File Reference

Interface for LED module.

```
#include "GPIO.h"
#include <stdbool.h>
Include dependency graph for Led.h:
```



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



Macros

- `#define LED_POOL_SIZE 1`

Functions

- `Led_t Led_Init` (GpioPort_t gpioPort, GPIO_Pin_t pin)
Initialize a light-emitting diode (LED) as an `Led_t`.
- `bool Led_isInit` (Led_t led)
- `GpioPort_t Led_GetPort` (Led_t led)
Get the GPIO port associated with the LED.
- `GPIO_Pin_t Led_GetPin` (Led_t led)
Get the GPIO pin associated with the LED.
- `bool Led_isOn` (Led_t led)
Check the LED's status.
- `void Led_TurnOn` (Led_t led)
Turn the LED ON.
- `void Led_TurnOff` (Led_t led)
Turn the LED OFF.
- `void Led_Toggle` (Led_t led)
Toggle the LED (i.e. OFF -> ON or ON -> OFF).

7.33.1 Detailed Description

Interface for LED module.

Author

Bryan McElvy

Index

- ADC.c, [98](#)
- ADC.h, [99](#)
- ADC_Init
 - Analog-to-Digital Conversion (ADC), [32](#)
- Analog-to-Digital Conversion (ADC), [32](#)
 - ADC_Init, [32](#)
- Application Software, [4](#)
- Assert
 - Common, [24](#)
- bandpassFiltStruct
 - Data Acquisition (DAQ), [9](#)
- Bug List, [1](#)
- CASET
 - ILI9341, [62](#)
- Cmd_t
 - ILI9341, [62](#)
- COEFF_BANDPASS
 - QRS Detector, [22](#)
- COEFF_DERFILT
 - QRS Detector, [22](#)
- COEFF_MOVAVG
 - QRS Detector, [23](#)
- COEFFS_BANDPASS
 - Data Acquisition (DAQ), [9](#)
- COEFFS_NOTCH
 - Data Acquisition (DAQ), [9](#)
- Common, [23](#)
 - Assert, [24](#)
- DAQ.c, [81](#)
- DAQ.h, [83](#)
- DAQ_ARRAY_LEN
 - Main, [74](#)
- DAQ_BandpassFilter
 - Data Acquisition (DAQ), [8](#)
- DAQ_convertToMilliVolts
 - Data Acquisition (DAQ), [8](#)
- DAQ_FIFO_CAP
 - Main, [74](#)
- DAQ_Handler
 - Main, [75](#)
- DAQ_Init
 - Data Acquisition (DAQ), [7](#)
- DAQ_lookup.c, [84](#)
- DAQ_NotchFilter
 - Data Acquisition (DAQ), [7](#)
- DAQ_readSample
 - Data Acquisition (DAQ), [7](#)
- Data Acquisition (DAQ), [5](#)
 - bandpassFiltStruct, [9](#)
 - COEFFS_BANDPASS, [9](#)
 - COEFFS_NOTCH, [9](#)
 - DAQ_BandpassFilter, [8](#)
 - DAQ_convertToMilliVolts, [8](#)
 - DAQ_Init, [7](#)
 - DAQ_NotchFilter, [7](#)
 - DAQ_readSample, [7](#)
- Debug, [56](#)
 - Debug_Assert, [59](#)
 - Debug_Init, [57](#)
 - Debug_SendFromList, [58](#)
 - Debug_SendMsg, [57](#)
 - Debug_WriteFloat, [58](#)
- Debug.h, [136](#)
- Debug_Assert
 - Debug, [59](#)
- Debug_Init
 - Debug, [57](#)
- Debug_SendFromList
 - Debug, [58](#)
- Debug_SendMsg
 - Debug, [57](#)
- Debug_WriteFloat
 - Debug, [58](#)
- Device Drivers, [30](#)
- DINVOFF
 - ILI9341, [62](#)
- DINVON
 - ILI9341, [62](#)
- DISPOFF
 - ILI9341, [62](#)
- DISPON
 - ILI9341, [62](#)
- DOWN
 - Timer, [40](#)
- FIFO Buffers, [24](#)
 - FIFO_Flush, [27](#)
 - FIFO_Get, [26](#)
 - FIFO_getCurrSize, [29](#)
 - FIFO_Init, [26](#)
 - FIFO_isEmpty, [29](#)
 - FIFO_isFull, [29](#)
 - FIFO_PeekAll, [29](#)
 - FIFO_PeekOne, [28](#)
 - FIFO_Put, [26](#)
 - FIFO_Reset, [28](#)
 - FIFO_TransferAll, [28](#)
 - FIFO_TransferOne, [27](#)
- Fifo.c, [93](#)
- Fifo.h, [95](#)
- FIFO_Flush
 - FIFO Buffers, [27](#)
- FIFO_Get
 - FIFO Buffers, [26](#)
- FIFO_getCurrSize
 - FIFO Buffers, [29](#)
- FIFO_Init
 - FIFO Buffers, [26](#)

- FIFO_isEmpty
 - FIFO Buffers, [29](#)
- FIFO_isFull
 - FIFO Buffers, [29](#)
- FIFO_PeekAll
 - FIFO Buffers, [29](#)
- FIFO_PeekOne
 - FIFO Buffers, [28](#)
- FIFO_Put
 - FIFO Buffers, [26](#)
- FIFO_Reset
 - FIFO Buffers, [28](#)
- Fifo_t, [79](#)
- FIFO_TransferAll
 - FIFO Buffers, [28](#)
- FIFO_TransferOne
 - FIFO Buffers, [27](#)
- FRMCTR1
 - ILI9341, [62](#)
- FRMCTR2
 - ILI9341, [62](#)
- FRMCTR3
 - ILI9341, [62](#)
- General-Purpose Input/Output (GPIO), [33](#)
- GPIO.c, [100](#)
 - GPIO_ConfigAltMode, [109](#)
 - GPIO_ConfigAnalog, [110](#)
 - GPIO_ConfigDirInput, [104](#)
 - GPIO_ConfigDirOutput, [104](#)
 - GPIO_ConfigDriveStrength, [105](#)
 - GPIO_ConfigInterrupts_BothEdges, [107](#)
 - GPIO_ConfigInterrupts_Edge, [107](#)
 - GPIO_ConfigInterrupts_LevelTrig, [107](#)
 - GPIO_ConfigNVIC, [108](#)
 - GPIO_ConfigPortCtrl, [109](#)
 - GPIO_ConfigPullDown, [104](#)
 - GPIO_ConfigPullUp, [104](#)
 - GPIO_DisableDigital, [105](#)
 - GPIO_EnableDigital, [105](#)
 - GPIO_getBaseAddr, [102](#)
 - GPIO_getDataRegister, [108](#)
 - GPIO_InitPort, [102](#)
 - GPIO_isPortInit, [102](#)
 - GPIO_PTR_ARR, [110](#)
 - GPIO_ReadPins, [108](#)
 - GPIO_Toggle, [109](#)
 - GPIO_WriteHigh, [108](#)
 - GPIO_WriteLow, [109](#)
- GPIO.h, [110](#)
 - GPIO_ConfigAltMode, [118](#)
 - GPIO_ConfigAnalog, [119](#)
 - GPIO_ConfigDirInput, [114](#)
 - GPIO_ConfigDirOutput, [114](#)
 - GPIO_ConfigDriveStrength, [115](#)
 - GPIO_ConfigInterrupts_BothEdges, [116](#)
 - GPIO_ConfigInterrupts_Edge, [115](#)
 - GPIO_ConfigInterrupts_LevelTrig, [116](#)
 - GPIO_ConfigNVIC, [116](#)
- GPIO_ConfigPortCtrl, [118](#)
- GPIO_ConfigPullDown, [114](#)
- GPIO_ConfigPullUp, [114](#)
- GPIO_DisableDigital, [115](#)
- GPIO_EnableDigital, [115](#)
- GPIO_getBaseAddr, [113](#)
- GPIO_getDataRegister, [117](#)
- GPIO_InitPort, [113](#)
- GPIO_isPortInit, [113](#)
- GPIO_ReadPins, [117](#)
- GPIO_Toggle, [118](#)
- GPIO_WriteHigh, [117](#)
- GPIO_WriteLow, [117](#)

- GPIO_getDataRegister
 - GPIO.c, [108](#)
 - GPIO.h, [117](#)
- GPIO_InitPort
 - GPIO.c, [102](#)
 - GPIO.h, [113](#)
- GPIO_isPortInit
 - GPIO.c, [102](#)
 - GPIO.h, [113](#)
- GPIO_PTR_ARR
 - GPIO.c, [110](#)
- GPIO_ReadPins
 - GPIO.c, [108](#)
 - GPIO.h, [117](#)
- GPIO_Toggle
 - GPIO.c, [109](#)
 - GPIO.h, [118](#)
- GPIO_WriteHigh
 - GPIO.c, [108](#)
 - GPIO.h, [117](#)
- GPIO_WriteLow
 - GPIO.c, [109](#)
 - GPIO.h, [117](#)
- GpioPort_t, [80](#)
- IDMOFF
 - ILI9341, [62](#)
- IDMON
 - ILI9341, [62](#)
- IFCTL
 - ILI9341, [62](#)
- ILI9341, [59](#)
 - CASET, [62](#)
 - Cmd_t, [62](#)
 - DINVOFF, [62](#)
 - DINVON, [62](#)
 - DISPOFF, [62](#)
 - DISPON, [62](#)
 - FRMCTR1, [62](#)
 - FRMCTR2, [62](#)
 - FRMCTR3, [62](#)
 - IDMOFF, [62](#)
 - IDMON, [62](#)
 - IFCTL, [62](#)
 - ILI9341_Init, [63](#)
 - ILI9341_NUM_COLS, [61](#)
 - ILI9341_NUM_ROWS, [61](#)
 - ILI9341_resetHard, [64](#)
 - ILI9341_resetSoft, [64](#)
 - ILI9341_sendParams, [63](#)
 - ILI9341_setAddress, [62](#)
 - ILI9341_setColAddress, [68](#)
 - ILI9341_setColorDepth, [67](#)
 - ILI9341_setColorExpression, [65](#)
 - ILI9341_setDisplInversion, [66](#)
 - ILI9341_setDisplayArea, [65](#)
 - ILI9341_setDispOutput, [66](#)
 - ILI9341 setFrameRate, [68](#)
 - ILI9341_setInterface, [63](#)
 - ILI9341_setMemAccessCtrl, [67](#)
 - ILI9341_setMode, [62](#)
 - ILI9341_setPartialArea, [66](#)
 - ILI9341_setRowAddress, [68](#)
 - ILI9341_setSleepMode, [65](#)
 - ILI9341_writeMemCmd, [69](#)
 - ILI9341_writePixel, [69](#)
 - MADCTL, [62](#)
 - NOP, [62](#)
 - NORON, [62](#)
 - PASET, [62](#)
 - PIXSET, [62](#)
 - PLTAR, [62](#)
 - PRCTR, [62](#)
 - PTLON, [62](#)
 - RAMWR, [62](#)
 - SPLIN, [62](#)
 - SPLOUT, [62](#)
 - SWRESET, [62](#)
 - VSCRDEF, [62](#)
 - VSCRSAADD, [62](#)
- ILI9341.c, [138](#)
- ILI9341.h, [139](#)
- ILI9341_Init
 - ILI9341, [63](#)
- ILI9341_NUM_COLS
 - ILI9341, [61](#)
- ILI9341_NUM_ROWS
 - ILI9341, [61](#)
- ILI9341_resetHard
 - ILI9341, [64](#)
- ILI9341_resetSoft
 - ILI9341, [64](#)
- ILI9341_sendParams
 - ILI9341, [63](#)
- ILI9341_setAddress
 - ILI9341, [62](#)
- ILI9341_setColAddress
 - ILI9341, [68](#)
- ILI9341_setColorDepth
 - ILI9341, [67](#)
- ILI9341_setColorExpression
 - ILI9341, [65](#)
- ILI9341_setDisplInversion
 - ILI9341, [66](#)
- ILI9341_setDisplayArea
 - ILI9341, [65](#)
- ILI9341_setDispOutput
 - ILI9341, [66](#)
- ILI9341 setFrameRate
 - ILI9341, [68](#)
- ILI9341_setInterface
 - ILI9341, [63](#)
- ILI9341_setMemAccessCtrl
 - ILI9341, [67](#)
- ILI9341_setMode
 - ILI9341, [62](#)
- ILI9341_setPartialArea

- ILI9341, 66
- ILI9341_setRowAddress
 - ILI9341, 68
- ILI9341_setSleepMode
 - ILI9341, 65
- ILI9341_writeMemCmd
 - ILI9341, 69
- ILI9341_writePixel
 - ILI9341, 69
- Interrupt Service Routines, 51
 - ISR_addToIntTable, 53
 - ISR_Disable, 54
 - ISR_Enable, 54
 - ISR_GlobalDisable, 52
 - ISR_GlobalEnable, 52
 - ISR_InitNewTableInRam, 52
 - ISR_setPriority, 53
 - ISR_triggerInterrupt, 55
- ISR.c, 119
- ISR.h, 121
- ISR_addToIntTable
 - Interrupt Service Routines, 53
- ISR_Disable
 - Interrupt Service Routines, 54
- ISR_Enable
 - Interrupt Service Routines, 54
- ISR_GlobalDisable
 - Interrupt Service Routines, 52
- ISR_GlobalEnable
 - Interrupt Service Routines, 52
- ISR_InitNewTableInRam
 - Interrupt Service Routines, 52
- ISR_setPriority
 - Interrupt Service Routines, 53
- ISR_triggerInterrupt
 - Interrupt Service Routines, 55
- LCD.c, 85
- LCD.h, 87
- LCD_ARRAY_1_LEN
 - Main, 74
- LCD_ARRAY_2_LEN
 - Main, 74
- LCD_Draw
 - Liquid Crystal Display (LCD), 14
- LCD_drawHoriLine
 - Liquid Crystal Display (LCD), 14
- LCD_drawLine
 - Liquid Crystal Display (LCD), 12
- LCD_drawRectangle
 - Liquid Crystal Display (LCD), 15
- LCD_drawVertLine
 - Liquid Crystal Display (LCD), 15
- LCD_FIFO_1_CAP
 - Main, 74
- LCD_FIFO_2_CAP
 - Main, 74
- LCD_Fill
 - Liquid Crystal Display (LCD), 14
- LCD_Handler
 - Main, 76
- LCD_Init
 - Liquid Crystal Display (LCD), 12
- LCD_plotSample
 - Liquid Crystal Display (LCD), 16
- LCD_setColor
 - Liquid Crystal Display (LCD), 14
- LCD_setCursor
 - Liquid Crystal Display (LCD), 16
- LCD_setOutputMode
 - Liquid Crystal Display (LCD), 12
- LCD_setX
 - Liquid Crystal Display (LCD), 13
- LCD_setY
 - Liquid Crystal Display (LCD), 13
- LCD_TOP_LINE
 - Main, 75
- LCD_WAVE_NUM_Y
 - Main, 75
- LED, 70
 - Led_GetPin, 72
 - Led_GetPort, 71
 - Led_Init, 71
 - Led_isOn, 72
 - Led_Toggle, 73
 - Led_TurnOff, 73
 - Led_TurnOn, 72
- Led.c, 142
- Led.h, 143
- Led_GetPin
 - LED, 72
- Led_GetPort
 - LED, 71
- Led_Init
 - LED, 71
- Led_isOn
 - LED, 72
- Led_t, 80
- Led_Toggle
 - LED, 73
- Led_TurnOff
 - LED, 73
- Led_TurnOn
 - LED, 72
- Liquid Crystal Display (LCD), 10
 - LCD_Draw, 14
 - LCD_drawHoriLine, 14
 - LCD_drawLine, 12
 - LCD_drawRectangle, 15
 - LCD_drawVertLine, 15
 - LCD_Fill, 14
 - LCD_Init, 12
 - LCD_plotSample, 16
 - LCD_setColor, 14
 - LCD_setCursor, 16
 - LCD_setOutputMode, 12
 - LCD_setX, 13

- LCD_setY, [13](#)
- MADCTL
 - ILI9341, [62](#)
- Main, [73](#)
 - DAQ_ARRAY_LEN, [74](#)
 - DAQ_FIFO_CAP, [74](#)
 - DAQ_Handler, [75](#)
 - LCD_ARRAY_1_LEN, [74](#)
 - LCD_ARRAY_2_LEN, [74](#)
 - LCD_FIFO_1_CAP, [74](#)
 - LCD_FIFO_2_CAP, [74](#)
 - LCD_Handler, [76](#)
 - LCD_TOP_LINE, [75](#)
 - LCD_WAVE_NUM_Y, [75](#)
 - main, [77](#)
 - Processing_Handler, [75](#)
 - QRS_ARRAY_LEN, [74](#)
 - QRS_FIFO_CAP, [74](#)
- main
 - Main, [77](#)
- main.c, [135](#)
- Middleware, [55](#)
- NewAssert, [30](#)
- NewAssert.c, [96](#)
- NewAssert.h, [97](#)
- NOP
 - ILI9341, [62](#)
- NORON
 - ILI9341, [62](#)
- ONESHOT
 - Timer, [40](#)
- PASET
 - ILI9341, [62](#)
- PERIODIC
 - Timer, [40](#)
- Phase-Locked Loop (PLL), [33](#)
 - PLL_Init, [34](#)
- PIXSET
 - ILI9341, [62](#)
- PLL.c, [122](#)
- PLL.h, [123](#)
- PLL_Init
 - Phase-Locked Loop (PLL), [34](#)
- PLTAR
 - ILI9341, [62](#)
- PRCTR
 - ILI9341, [62](#)
- Processing_Handler
 - Main, [75](#)
- PTLON
 - ILI9341, [62](#)
- QRS Detector, [17](#)
 - COEFF_BANDPASS, [22](#)
 - COEFF_DERFILT, [22](#)
 - COEFF_MOVAVG, [23](#)
 - QRS_applyDecisionRules, [21](#)
 - QRS_findFiducialMarks, [19](#)
 - QRS_Init, [20](#)
 - QRS_initLevels, [19](#)
 - QRS_Preprocess, [20](#)
 - QRS_runDetection, [22](#)
 - QRS_updateLevel, [19](#)
 - QRS_updateThreshold, [20](#)
- QRS.c, [89](#)
- QRS.h, [92](#)
- QRS_applyDecisionRules
 - QRS Detector, [21](#)
- QRS_ARRAY_LEN
 - Main, [74](#)
- QRS_FIFO_CAP
 - Main, [74](#)
- QRS_findFiducialMarks
 - QRS Detector, [19](#)
- QRS_Init
 - QRS Detector, [20](#)
- QRS_initLevels
 - QRS Detector, [19](#)
- QRS_Preprocess
 - QRS Detector, [20](#)
- QRS_runDetection
 - QRS Detector, [22](#)
- QRS_updateLevel
 - QRS Detector, [19](#)
- QRS_updateThreshold
 - QRS Detector, [20](#)
- RAMWR
 - ILI9341, [62](#)
- Serial Peripheral Interface (SPI), [34](#)
 - SPI_Init, [35](#)
 - SPI_Read, [35](#)
 - SPI_SET_DC, [35](#)
 - SPI_WriteCmd, [36](#)
 - SPI_WriteData, [36](#)
- SPI.c, [123](#)
- SPI.h, [125](#)
- SPI_Init
 - Serial Peripheral Interface (SPI), [35](#)
- SPI_Read
 - Serial Peripheral Interface (SPI), [35](#)
- SPI_SET_DC
 - Serial Peripheral Interface (SPI), [35](#)
- SPI_WriteCmd
 - Serial Peripheral Interface (SPI), [36](#)
- SPI_WriteData
 - Serial Peripheral Interface (SPI), [36](#)
- SPLIN
 - ILI9341, [62](#)
- SPLOUT
 - ILI9341, [62](#)
- SWRESET
 - ILI9341, [62](#)

- System Tick (SysTick), 37
 - SysTick_Interrupt_Init, 37
- SysTick.c, 126
- SysTick.h, 127
- SysTick_Interrupt_Init
 - System Tick (SysTick), 37
- Timer, 38
 - DOWN, 40
 - ONESHOT, 40
 - PERIODIC, 40
 - Timer_clearInterruptFlag, 44
 - Timer_Deinit, 40
 - Timer_disableAdcTrigger, 42
 - Timer_disableInterruptOnTimeout, 43
 - Timer_enableAdcTrigger, 42
 - Timer_enableInterruptOnTimeout, 43
 - Timer_getName, 41
 - Timer_Init, 40
 - Timer_isCounting, 45
 - Timer_isInit, 41
 - TIMER_POOL, 46
 - Timer_setInterval_ms, 44
 - Timer_setMode, 41
 - Timer_Start, 44
 - Timer_Stop, 45
 - Timer_Wait1ms, 46
 - timerDirection_t, 40
 - timerMode_t, 39
 - UP, 40
- Timer.c, 128
- Timer.h, 130
- Timer_clearInterruptFlag
 - Timer, 44
- Timer_Deinit
 - Timer, 40
- Timer_disableAdcTrigger
 - Timer, 42
- Timer_disableInterruptOnTimeout
 - Timer, 43
- Timer_enableAdcTrigger
 - Timer, 42
- Timer_enableInterruptOnTimeout
 - Timer, 43
- Timer_getName
 - Timer, 41
- Timer_Init
 - Timer, 40
- Timer_isCounting
 - Timer, 45
- Timer_isInit
 - Timer, 41
- TIMER_POOL
 - Timer, 46
- Timer_setInterval_ms
 - Timer, 44
- Timer_setMode
 - Timer, 41
- Timer_Start
 - Timer, 44
- Timer_Stop
 - Timer, 45
- Timer_t, 81
- Timer_Wait1ms
 - Timer, 46
- timerDirection_t
 - Timer, 40
- timerMode_t
 - Timer, 39
- UART.c, 132
- UART.h, 133
- UART_ARR
 - Universal Asynchronous Receiver/Transmitter (UART), 50
- UART_Init
 - Universal Asynchronous Receiver/Transmitter (UART), 48
- UART_isInit
 - Universal Asynchronous Receiver/Transmitter (UART), 48
- UART_ReadChar
 - Universal Asynchronous Receiver/Transmitter (UART), 49
- Uart_t, 81
- UART_WriteChar
 - Universal Asynchronous Receiver/Transmitter (UART), 49
- UART_WriteFloat
 - Universal Asynchronous Receiver/Transmitter (UART), 50
- UART_WriteInt
 - Universal Asynchronous Receiver/Transmitter (UART), 50
- UART_WriteStr
 - Universal Asynchronous Receiver/Transmitter (UART), 49
- Universal Asynchronous Receiver/Transmitter (UART), 47
 - UART_ARR, 50
 - UART_Init, 48
 - UART_isInit, 48
 - UART_ReadChar, 49
 - UART_WriteChar, 49
 - UART_WriteFloat, 50
 - UART_WriteInt, 50
 - UART_WriteStr, 49
- UP
 - Timer, 40
- VSCRDEF
 - ILI9341, 62
- VSCRSADD
 - ILI9341, 62