

## ECG-HRM

Generated by Doxygen 1.9.8

<b>1 Bug List</b>	<b>1</b>
<b>2 Topic Index</b>	<b>1</b>
2.1 Topics	1
<b>3 Data Structure Index</b>	<b>2</b>
3.1 Data Structures	2
<b>4 File Index</b>	<b>2</b>
4.1 File List	2
<b>5 Topic Documentation</b>	<b>5</b>
5.1 Device Drivers	5
5.1.1 Detailed Description	6
5.1.2 Analog-to-Digital Conversion (ADC)	6
5.1.3 GPIO	7
5.1.4 Phase-Locked Loop (PLL)	7
5.1.5 Serial Peripheral Interface (SPI)	8
5.1.6 System Tick (SysTick)	11
5.1.7 Timer	12
5.1.8 Universal Asynchronous Receiver/Transmitter (UART)	20
5.1.9 Interrupt Service Routines	24
5.2 Middleware	29
5.2.1 Detailed Description	29
5.2.2 ILI9341	29
5.2.3 LED	40
5.3 Application Software	43
5.3.1 Detailed Description	44
5.3.2 Data Acquisition (DAQ)	44
5.3.3 Debug	49
5.3.4 LCD	52
5.3.5 QRS	58
5.4 Common	64
5.4.1 Detailed Description	65
5.4.2 Function Documentation	65
5.4.3 FIFO	65
5.4.4 NewAssert	71
5.5 Main	71
5.5.1 Detailed Description	72
5.5.2 Enumeration Type Documentation	72
5.5.3 Function Documentation	73
<b>6 Data Structure Documentation</b>	<b>75</b>
6.1 Fifo_t Struct Reference	75

6.2 GPIO_Port_t Struct Reference . . . . .	76
6.3 Led_t Struct Reference . . . . .	76
6.4 Timer_t Struct Reference . . . . .	77
6.5 UART_t Struct Reference . . . . .	77
<b>7 File Documentation</b> . . . . .	<b>77</b>
7.1 DAQ.c File Reference . . . . .	77
7.1.1 Detailed Description . . . . .	79
7.2 DAQ.h File Reference . . . . .	79
7.2.1 Detailed Description . . . . .	80
7.3 LCD.c File Reference . . . . .	81
7.3.1 Detailed Description . . . . .	82
7.4 LCD.h File Reference . . . . .	83
7.4.1 Detailed Description . . . . .	84
7.5 lookup.c File Reference . . . . .	84
7.5.1 Detailed Description . . . . .	85
7.6 lookup.h File Reference . . . . .	85
7.6.1 Detailed Description . . . . .	86
7.7 QRS.c File Reference . . . . .	87
7.7.1 Detailed Description . . . . .	88
7.8 QRS.h File Reference . . . . .	89
7.8.1 Detailed Description . . . . .	90
7.9 Fifo.c File Reference . . . . .	90
7.9.1 Detailed Description . . . . .	91
7.10 Fifo.h File Reference . . . . .	92
7.10.1 Detailed Description . . . . .	93
7.11 NewAssert.c File Reference . . . . .	93
7.11.1 Detailed Description . . . . .	94
7.12 NewAssert.h File Reference . . . . .	94
7.12.1 Detailed Description . . . . .	95
7.13 ADC.c File Reference . . . . .	95
7.13.1 Detailed Description . . . . .	95
7.14 ADC.h File Reference . . . . .	96
7.14.1 Detailed Description . . . . .	96
7.15 GPIO.c File Reference . . . . .	96
7.15.1 Detailed Description . . . . .	99
7.15.2 Function Documentation . . . . .	99
7.15.3 Variable Documentation . . . . .	105
7.16 GPIO.h File Reference . . . . .	105
7.16.1 Detailed Description . . . . .	107
7.16.2 Function Documentation . . . . .	107
7.17 ISR.c File Reference . . . . .	113

---

7.17.1 Detailed Description . . . . .	114
7.18 ISR.h File Reference . . . . .	114
7.18.1 Detailed Description . . . . .	115
7.19 PLL.c File Reference . . . . .	116
7.19.1 Detailed Description . . . . .	116
7.20 PLL.h File Reference . . . . .	116
7.20.1 Detailed Description . . . . .	117
7.21 SPI.c File Reference . . . . .	117
7.21.1 Detailed Description . . . . .	118
7.22 SPI.h File Reference . . . . .	118
7.22.1 Detailed Description . . . . .	119
7.23 SysTick.c File Reference . . . . .	119
7.23.1 Detailed Description . . . . .	120
7.24 SysTick.h File Reference . . . . .	120
7.24.1 Detailed Description . . . . .	121
7.25 Timer.c File Reference . . . . .	121
7.25.1 Detailed Description . . . . .	122
7.26 Timer.h File Reference . . . . .	123
7.26.1 Detailed Description . . . . .	124
7.27 UART.c File Reference . . . . .	124
7.27.1 Detailed Description . . . . .	126
7.28 UART.h File Reference . . . . .	126
7.28.1 Detailed Description . . . . .	127
7.29 main.c File Reference . . . . .	128
7.29.1 Detailed Description . . . . .	129
7.30 Debug.h File Reference . . . . .	129
7.30.1 Detailed Description . . . . .	130
7.31 ILI9341.c File Reference . . . . .	131
7.31.1 Detailed Description . . . . .	132
7.32 ILI9341.h File Reference . . . . .	132
7.32.1 Detailed Description . . . . .	134
7.33 Led.c File Reference . . . . .	135
7.33.1 Detailed Description . . . . .	136
7.34 Led.h File Reference . . . . .	136
7.34.1 Detailed Description . . . . .	137
7.35 test_adc.c File Reference . . . . .	137
7.35.1 Detailed Description . . . . .	138
7.36 test_daq.c File Reference . . . . .	138
7.36.1 Detailed Description . . . . .	139
7.37 test_debug.c File Reference . . . . .	139
7.37.1 Detailed Description . . . . .	140
7.38 test_fifo.c File Reference . . . . .	140

7.38.1 Detailed Description . . . . .	141
7.39 test_lcd_image.c File Reference . . . . .	142
7.39.1 Detailed Description . . . . .	143
7.40 test_pll.c File Reference . . . . .	143
7.40.1 Detailed Description . . . . .	143
7.41 test_qrs.c File Reference . . . . .	144
7.41.1 Detailed Description . . . . .	145
7.42 test_spi.c File Reference . . . . .	145
7.42.1 Detailed Description . . . . .	145
7.43 test_systick_int.c File Reference . . . . .	146
7.43.1 Detailed Description . . . . .	146
7.44 test_timer1_int.c File Reference . . . . .	147
7.44.1 Detailed Description . . . . .	147
7.45 test_uart_la.c File Reference . . . . .	148
7.45.1 Detailed Description . . . . .	148
7.46 test_uart_write.c File Reference . . . . .	149
7.46.1 Detailed Description . . . . .	149
7.47 test_userctrl.c File Reference . . . . .	150
7.47.1 Detailed Description . . . . .	150
<b>Index</b>	<b>151</b>

## 1 Bug List

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

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 are not being counted.

## 2 Topic Index

### 2.1 Topics

Here is a list of all topics with brief descriptions:

<b>Device Drivers</b>	<b>5</b>
<b>Analog-to-Digital Conversion (ADC)</b>	<b>6</b>
<b>GPIO</b>	<b>7</b>
<b>Phase-Locked Loop (PLL)</b>	<b>7</b>
<b>Serial Peripheral Interface (SPI)</b>	<b>8</b>
<b>System Tick (SysTick)</b>	<b>11</b>

Timer	12
Universal Asynchronous Receiver/Transmitter (UART)	20
Interrupt Service Routines	24
Middleware	29
ILI9341	29
LED	40
Application Software	43
Data Acquisition (DAQ)	44
Debug	49
LCD	52
QRS	58
Common	64
FIFO	65
NewAssert	71
Main	71

## 3 Data Structure Index

### 3.1 Data Structures

Here are the data structures with brief descriptions:

Fifo_t	75
GPIO_Port_t	76
Led_t	76
Timer_t	77
UART_t	77

## 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	77

<b>DAQ.h</b>	
Application software for handling data acquisition (DAQ) functions	79
<b>LCD.c</b>	
Source code for LCD module	81
<b>LCD.h</b>	
Header file for LCD module	83
<b>lookup.c</b>	
Source code for DAQ module's lookup table	84
<b>lookup.h</b>	
Lookup table for DAQ module	85
<b>QRS.c</b>	
Source code for QRS detection module	87
<b>QRS.h</b>	
Header file for QRS detection module	89
<b>Fifo.c</b>	
Source code for FIFO buffer module	90
<b>Fifo.h</b>	
Header file for FIFO buffer implementation	92
<b>NewAssert.c</b>	
Source code for custom <code>assert</code> implementation	93
<b>NewAssert.h</b>	
Header file for custom <code>assert</code> implementation	94
<b>ADC.c</b>	
Source code for analog-to-digital conversion (ADC) module	95
<b>ADC.h</b>	
Header file for analog-to-digital conversion (ADC) module	96
<b>GPIO.c</b>	
Source code for GPIO module	96
<b>GPIO.h</b>	
Header file for general-purpose input/output (GPIO) device driver	105
<b>ISR.c</b>	
Source code for interrupt service routine (ISR) configuration module	113
<b>ISR.h</b>	
Header file for interrupt service routine (ISR) configuration module	114
<b>PLL.c</b>	
Implementation details for phase-lock-loop (PLL) functions	116
<b>PLL.h</b>	
Driver module for activating the phase-locked-loop (PLL)	116
<b>SPI.c</b>	
Source code for serial peripheral interface (SPI) module	117
<b>SPI.h</b>	
Header file for serial peripheral interface (SPI) module	118

<a href="#">SysTick.c</a>	
Implementation details for SysTick functions	119
<a href="#">SysTick.h</a>	
Driver module for using SysTick-based timing and/or interrupts	120
<a href="#">Timer.c</a>	
Source code for Timer module	121
<a href="#">Timer.h</a>	
Device driver for general-purpose timer modules	123
<a href="#">UART.c</a>	
Source code for UART module	124
<a href="#">UART.h</a>	
Driver module for serial communication via UART0 and UART 1	126
<a href="#">main.c</a>	
Main program file	128
<a href="#">Debug.h</a>	
Functions to output debugging information to a serial port via UART	129
<a href="#">ILI9341.c</a>	
Source code for ILI9341 module	131
<a href="#">ILI9341.h</a>	
Driver module for interfacing with an ILI9341 LCD driver	132
<a href="#">Led.c</a>	
Source code for LED module	135
<a href="#">Led.h</a>	
Interface for LED module	136
<a href="#">test_adc.c</a>	
Test script for analog-to-digital conversion (ADC) module	137
<a href="#">test_daq.c</a>	
Test script for the data acquisition (DAQ) module	138
<a href="#">test_debug.c</a>	
Test script for Debug module	139
<a href="#">test_fifo.c</a>	
Test script for FIFO buffer	140
<a href="#">test_lcd_image.c</a>	
Test script for writing images onto the display	142
<a href="#">test_pll.c</a>	
Test script for the PLL module	143
<a href="#">test_qrs.c</a>	
QRS detector test script	144
<a href="#">test_spi.c</a>	
Test script for initializing SSI0 and writing data/commands via SPI	145
<a href="#">test_systick_int.c</a>	
Test script for SysTick interrupts	146



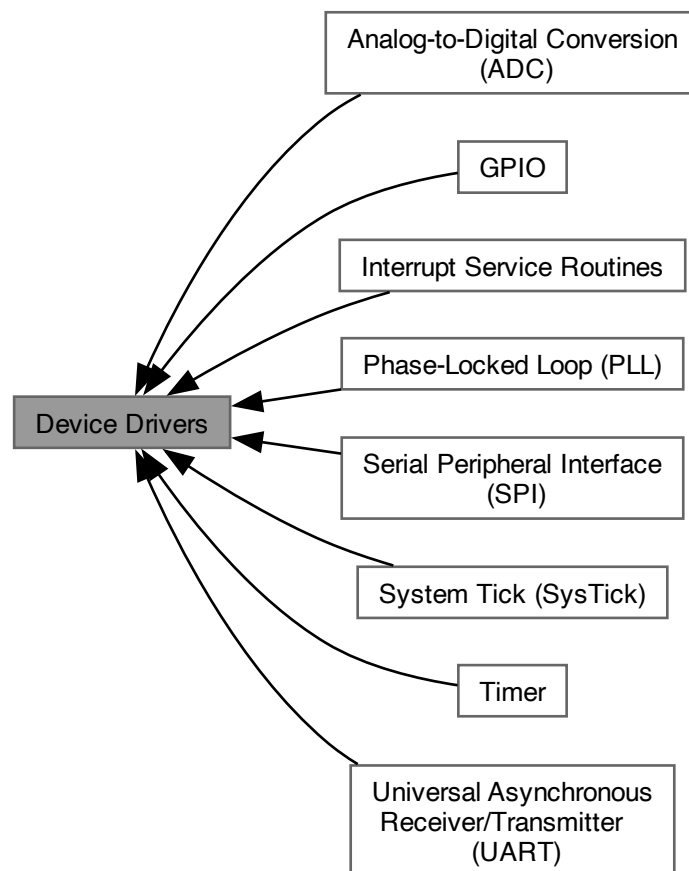
<a href="#">test_timer1_int.c</a>	147
Test script for relocating the vector table to RAM	
<a href="#">test_uart_la.c</a>	148
Test script for using a USB logic analyzer to decode UART signals	
<a href="#">test_uart_write.c</a>	149
Test script for writing to serial port via UART0	
<a href="#">test_userctrl.c</a>	150
Test file for GPIO/UserCtrl modules and GPIO interrupts	

## 5 Topic Documentation

### 5.1 Device Drivers

Low level device driver modules.

Collaboration diagram for Device Drivers:



## Modules

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

### 5.1.1 Detailed Description

Low level device driver modules.

These modules contain functions for interfacing with peripherals available on the TM4C123GH6PM microcontroller.

### 5.1.2 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.*

### 5.1.2.1 Detailed Description

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

### 5.1.2.2 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.

### 5.1.3 GPIO

Collaboration diagram for GPIO:



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

### 5.1.4 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.*

### 5.1.4.1 Detailed Description

Function for initializing the phase-locked loop.

### 5.1.4.2 Function Documentation

#### PLL\_Init()

```
void PLL_Init (  
    void )
```

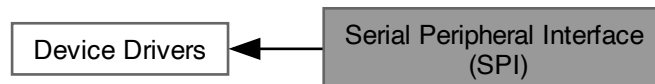
Initialize the phase-locked-loop to change the bus frequency.

#### Postcondition

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

## 5.1.5 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.

### 5.1.5.1 Detailed Description

Functions for SPI-based communication via SSI0 peripheral.

### 5.1.5.2 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)

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

### **SPI\_WriteCmd()**

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

Write a command to the serial port.

#### **Precondition**

Initialize the SPI module.

#### **Parameters**

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

#### **Postcondition**

The D/C pin is cleared.

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

### **SPI\_WriteData()**

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

Write data to the serial port.

#### Precondition

Initialize the SPI module.

#### Parameters

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

#### Postcondition

The D/C pin is set.

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

### 5.1.6 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.*

#### 5.1.6.1 Detailed Description

Functions for timing and periodic interrupts via SysTick.

### 5.1.6.2 Function Documentation

#### SysTick\_Interrupt\_Init()

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

Initialize SysTick for interrupts.

#### Parameters

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

### 5.1.7 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](#)

#### Typedefs

- typedef volatile uint32\_t \* **register\_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.*
- 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]

### 5.1.7.1 Detailed Description

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

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

### 5.1.7.3 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\_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\_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.

## 5.1.7.4 Variable Documentation

### TIMER\_POOL

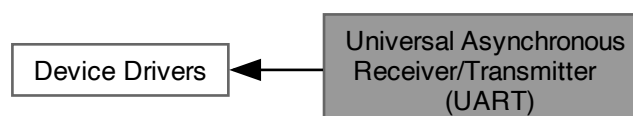
```
TimerStruct_t TIMER_POOL[6] [static]
```

#### 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 }
}
```

## 5.1.8 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`

## Typedefs

- `typedef volatile uint32_t * register_t`

## 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) 0xFC }
- enum **UART\_Num\_t** {  
  **UART0** , **UART1** , **UART2** , **UART3** ,  
  **UART4** , **UART5** , **UART6** , **UART7** }

## Functions

- [UART\\_t \\* UART\\_Init](#) ([GPIO\\_Port\\_t](#) \*port, [UART\\_Num\\_t](#) uartNum)  
*Initialize the specified UART peripheral.*
- unsigned char [UART\\_ReadChar](#) ([UART\\_t](#) \*uart)  
*Read a single ASCII character from the UART.*
- void [UART\\_WriteChar](#) ([UART\\_t](#) \*uart, unsigned char input\_char)  
*Write a single character to the UART.*
- void [UART\\_WriteStr](#) ([UART\\_t](#) \*uart, void \*input\_str)  
*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](#) num\_decimals)  
*Write a floating-point number the UART.*

## Variables

- static `UART_t UART_ARR` [8]

### 5.1.8.1 Detailed Description

Functions for UART-based communication.

### 5.1.8.2 Function Documentation

#### UART\_Init()

```
UART_t * UART_Init (
    GPIO_Port_t * port,
    UART_Num_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_t*</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\_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 input_char )

```

Write a single character to the UART.

#### Parameters

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

### UART\_WriteStr()

```

void UART_WriteStr (
    UART_t * uart,
    void * input_str )

```

Write a C string to the UART.

#### Parameters

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

### UART\_WriteFloat()

```

void UART_WriteFloat (
    UART_t * uart,
    double n,
    uint8_t num_decimals )

```

Write a floating-point number the UART.

#### Parameters

in	<i>uart</i>	UART to read from.
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.

### 5.1.8.3 Variable Documentation

#### UART\_ARR

```
UART_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 }
}
```

### 5.1.9 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 volatile uint32_t * register_t`
- `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

### 5.1.9.1 Detailed Description

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

### 5.1.9.2 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.



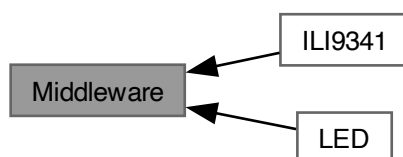
See also

`ISR_clearPending()`

## 5.2 Middleware

High-level device driver modules.

Collaboration diagram for Middleware:



### Modules

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

### 5.2.1 Detailed Description

High-level device driver modules.

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

### 5.2.2 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 }

#### 5.2.2.1 Detailed Description

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

#### 5.2.2.2 Enumeration Type Documentation

##### anonymous enum

anonymous enum

##### Enumerator

ILI9341_NUM_COLS	<b>5.2.2.3 of columns available on the display</b>
ILI9341_NUM_ROWS	<b>5.2.2.4 of rows available on the display</b>

**Cmd\_t**

enum Cmd\_t

**Enumerator**

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

**5.2.2.5 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  $\geq 10$  [us] and an additional 5 [ms] before further commands can be sent.

### ILI9341\_resetSoft()

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

Perform a software reset of the LCD driver.

#### Parameters

in	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  $\geq 5$  [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	...

## 5.2.3 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 (GPIO_Port_t *gpioPort, GPIO_Pin_t pin)`  
*Initialize a light-emitting diode (LED) as an `Led_t`.*
- `bool Led_isInit (Led_t led)`
- `GPIO_Port_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`

### 5.2.3.1 Detailed Description

Functions for driving light-emitting diodes (LEDs) via [GPIO](#).

### 5.2.3.2 Function Documentation

#### Led\_Init()

```
Led_t Led_Init (
    GPIO_Port_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()

```
GPIO_Port_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).

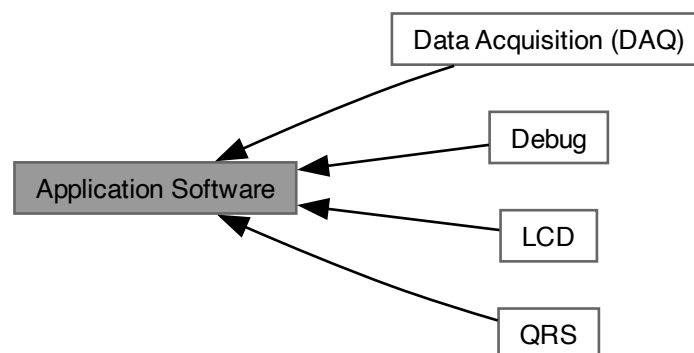
**Parameters**

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

## 5.3 Application Software

Application-specific software modules.

Collaboration diagram for Application Software:



## Modules

- [Data Acquisition \(DAQ\)](#)
- [Debug](#)
- [LCD](#)
- [QRS](#)

### 5.3.1 Detailed Description

Application-specific software modules.

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

### 5.3.2 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 [lookup.c](#)  
*Source code for DAQ module's lookup table.*
- file [lookup.h](#)  
*Lookup table for DAQ module.*

## Macros

- `#define SAMPLING_PERIOD_MS 5`  
*sampling period in ms ( $T_s = 1/f_s$ )*
- `#define LOOKUP_DAQ_MAX (float32_t) 5.5`
- `#define LOOKUP_DAQ_MIN (float32_t)(-5.5)`

## Functions

- `const float32_t * Lookup_GetPtr (void)`  
*Return a pointer to the DAQ lookup table.*



## Variables

- static const float32\_t \* **DAQ\_LOOKUP\_TABLE** = 0
- static const float32\_t **LOOKUP\_DAQ\_TABLE** [4096]

*Lookup table for converting ADC data from unsigned 12-bit integer values to 32-bit floating point values.*

## Digital Filters

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

## Initialization

- void **DAQ\_Init** (void)  
*pointer to lookup table*

## 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.*

### 5.3.2.1 Detailed Description

Module for managing data acquisition (DAQ) functions.

### 5.3.2.2 Function Documentation

#### DAQ\_Init()

```
void DAQ_Init (
    void )
```

pointer to lookup table

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\_convertToMilliVolts()

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

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

#### Precondition

Read a sample from the ADC.

**Parameters**

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

**Postcondition**

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

**See also**

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

**DAQ\_NotchFilter()**

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

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

**Precondition**

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

**Parameters**

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

**Postcondition**

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

**See also**

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

**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\(\)](#)

**Lookup\_GetPtr()**

```
const float32_t * Lookup_GetPtr (
    void )
```

Return a pointer to the DAQ lookup table.

**Parameters**

out	<i>lutPtr</i>	Pointer to the lookup table (LUT).
-----	---------------	------------------------------------

**5.3.2.3 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,
}
```

**COEFS\_BANDPASS**

```
const float32_t COEFS_BANDPASS[NUM_COEFS_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**

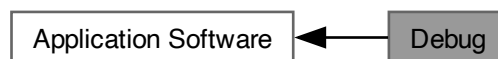
```
const Filter_t bandpassFiltStruct [static]
```

**Initial value:**

```
= { NUM_STAGES_BANDPASS, stateBuffer_Bandpass,
                                         COEFS_BANDPASS }
```

**5.3.3 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` (void)  
*Initialize the Debug module.*

## Assertions

- void `Debug_Assert` (bool condition)  
*Stops program if `condition` is `true`. Useful for bug detection during debugging.*

### 5.3.3.1 Detailed Description

Module for debugging functions, including serial output and assertion.

### 5.3.3.2 Function Documentation

#### `Debug_Init()`

```
void Debug_Init (  
    void )
```

Initialize the Debug module.

#### Postcondition

An initialization message is sent to the serial port (UART0).

#### `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

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

#### Postcondition

The corresponding message is sent to the serial port.

See also

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

### Debug\_WriteFloat()

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

Write a floating-point value to the serial port.

#### Precondition

Initialize the Debug module.

#### Parameters

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

#### Postcondition

A floating point value is written to the serial port.

See also

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

## Debug\_Assert()

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

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

### Precondition

Initialize the Debug module.

### Parameters

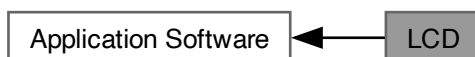
in	<i>condition</i>	Conditional statement to evaluate.
----	------------------	------------------------------------

### Postcondition

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

## 5.3.4 LCD

Collaboration diagram for LCD:



### Files

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

### Functions

- static void [LCD\\_drawLine](#) (uint16\_t center, uint16\_t lineWidth, bool is\_horizontal)  
*Helper function for drawing straight lines.*
- 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)
    - 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**

### 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.

### 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).

### 5.3.4.1 Detailed Description

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

### 5.3.4.2 Function Documentation

#### LCD\_drawLine()

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

Helper function for drawing straight lines.

##### Parameters

<i>center</i>	Row or column that the line is centered on. <i>center</i> 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>	true for horizontal line, false 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

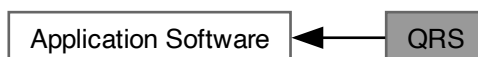
in	x	x-coordinate (i.e. sample number) in range [0, X_MAX]
in	y	y-coordinate (i.e. amplitude) in range [0, Y_MAX]
in	color	Color to use

See also

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

### 5.3.5 QRS

Collaboration diagram for QRS:



#### 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 20`
- `#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) (1200))`

### 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 ,
  - STATE\_BUFF\_SIZE\_DERFILT** = NUM\_COEFF\_DERFILT + QRS\_NUM\_SAMP - 1 , **NUM\_COEFF\_↵**  
**MOVAVG** = 10 , **STATE\_BUFF\_SIZE\_MOVAVG** = NUM\_COEFF\_MOVAVG + QRS\_NUM\_SAMP - 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.*

### 5.3.5.1 Detailed Description

Module for analyzing ECG data to determine heart rate.

### 5.3.5.2 Function Documentation

#### QRS\_findFiducialMarks()

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

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

#### Parameters

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

#### Postcondition

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

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

#### QRS\_initLevels()

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

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



**Parameters**

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

**Postcondition**

The signal and noise levels are initialized.

**QRS\_updateLevel()**

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

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

**Parameters**

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

$$\begin{aligned} signalLevel_1 &= f(peakAmplitude, signalLevel_0) = \frac{1}{8}peakAmplitude + \frac{7}{8}signalLevel_0 \\ noiseLevel_1 &= f(peakAmplitude, noiseLevel_0) = \frac{1}{8}peakAmplitude + \frac{7}{8}noiseLevel_0 \end{aligned}$$

**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.

**Bug** 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 are not 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\(\)](#)

**5.3.5.3 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**

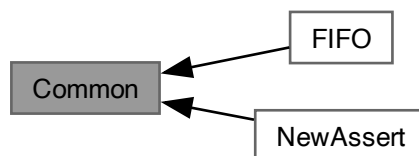
```
const float32_t COEFF_MOVAVG[NUM_COEFF_MOVAVG] [static]
```

**Initial value:**

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

**5.4 Common**

Collaboration diagram for Common:

**Modules**

- [FIFO](#)
- [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.*

### 5.4.1 Detailed Description

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

### 5.4.2 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.

### 5.4.3 FIFO

Collaboration diagram for FIFO:



## 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 buffer_pool [FIFO_POOL_SIZE] = { 0 }`  
*pre-allocated pool*
- static `uint8_t free_buffers = FIFO_POOL_SIZE`  
*no. of remaining buffers*

## 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.*

#### 5.4.3.1 Detailed Description

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

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

##### Postcondition

The number of available FIFO buffers is reduced by 1.

TODO: Add details

##### FIFO\_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.

**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.

**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.

**5.4.4 NewAssert**

Collaboration diagram for NewAssert:



Module for using a custom `assert` implementation.

**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\_CAPACITY** = 3 , **DAQ\_BUFFER\_SIZE** = DAQ\_FIFO\_CAPACITY + 1 , **QRS\_BUFFER\_SIZE** = QRS\_NUM\_SAMP + 1 , **LCD\_FIFO\_CAPACITY** = DAQ\_FIFO\_CAPACITY , **LCD\_BUFFER\_SIZE** = LCD\_FIFO\_CAPACITY + 1 }
- enum { **LCD\_TOP\_LINE** = (LCD\_Y\_MAX - 48) , **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\_BUFFER\_SIZE] = { 0 }
- static volatile `Fifo_t QRS_Fifo` = 0
- static volatile `uint32_t QRS_fifoBuffer` [QRS\_BUFFER\_SIZE] = { 0 }
- static volatile `bool QRS_bufferIsFull` = false
- static volatile `Fifo_t LCD_Fifo` = 0
- static volatile `uint32_t LCD_fifoBuffer` [LCD\_BUFFER\_SIZE] = { 0 }
- static `float32_t QRS_processingBuffer` [QRS\_BUFFER\_SIZE] = { 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_CAPACITY	capacity of DAQ's FIFO buffer
DAQ_BUFFER_SIZE	actual size of underlying array
LCD_FIFO_CAPACITY	capacity of LCD's FIFO buffer
LCD_BUFFER_SIZE	actual size of underlying array

#### anonymous enum

anonymous enum

#### Enumerator

LCD_TOP_LINE	separates waveform from text
LCD_WAVE_NUM↔ _Y	num. of y-vals 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\(\)](#)

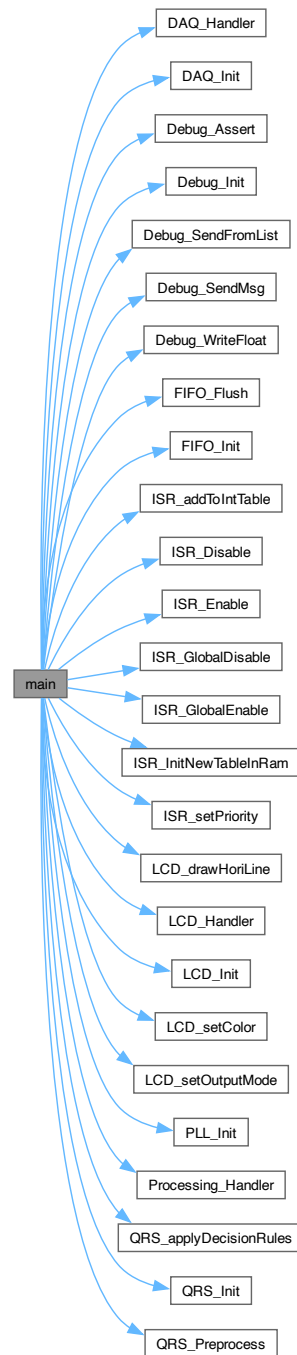
**main()**

```
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. 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 **front\_idx**  
*idx of front of FIFO*
- volatile uint32\_t **back\_idx**  
*idx of back of FIFO*

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

- [Fifo.c](#)

## 6.2 GPIO\_Port\_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

- [GPIO\\_Port\\_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**
- [GPIO\\_Port\\_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

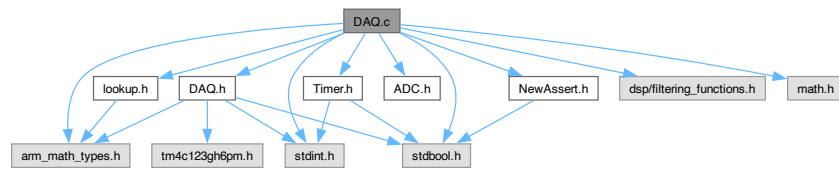
### 7.1 DAQ.c File Reference

Source code for DAQ module.

```
#include "DAQ.h"
#include "lookup.h"
#include "ADC.h"
#include "Timer.h"
#include "NewAssert.h"
#include "arm_math_types.h"
#include "dsp/filtering_functions.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)  
pointer to lookup table

### 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.

## Variables

- static const float32\_t \* `DAQ_LOOKUP_TABLE` = 0

## Digital Filters

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

### 7.1.1 Detailed Description

Source code for DAQ module.

Author

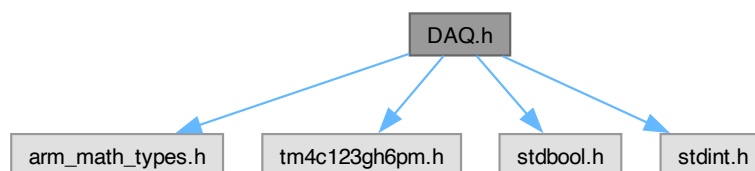
Bryan McElvy

## 7.2 DAQ.h File Reference

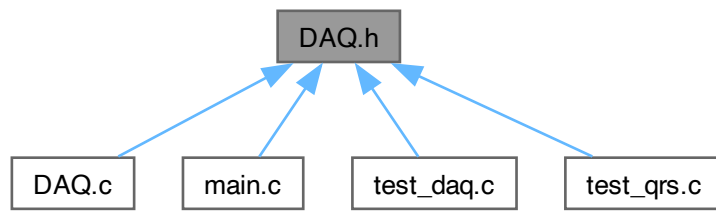
Application software for handling data acquisition (DAQ) functions.

```
#include "arm_math_types.h"
#include "tm4c123gh6pm.h"
#include <stdbool.h>
#include <stdint.h>
```

Include dependency graph for DAQ.h:



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



## Functions

### Initialization

- void [DAQ\\_Init](#) (void)  
*pointer to lookup table*

### 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.

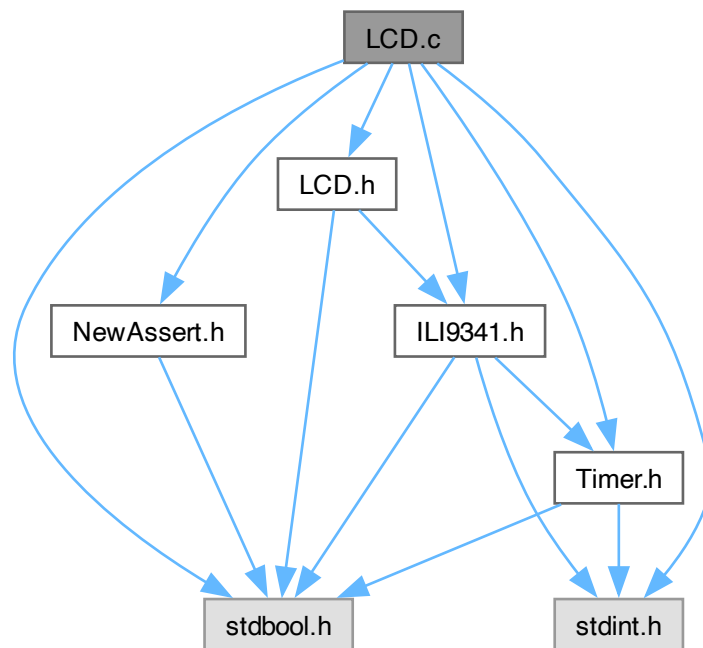
### Author

Bryan McElvy

## 7.3 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:
```



### Functions

- static void **LCD\_drawLine** (uint16\_t center, uint16\_t lineWidth, bool is\_horizontal)  
*Helper function for drawing straight lines.*
- 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  $dx \times dy$  onto the display. The bottom-left corner will be located at  $(x1, y1)$ .*

### 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  $[0, NUM\_COLS]$*   
    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`

#### 7.3.1 Detailed Description

Source code for LCD module.

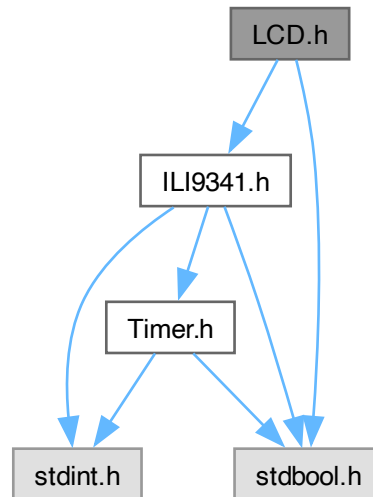
Author

Bryan McElvy

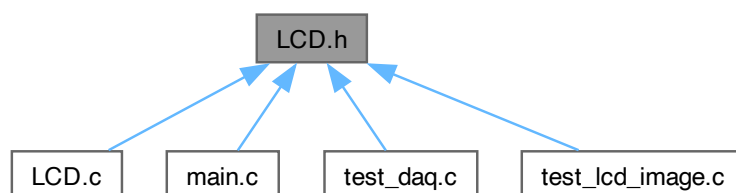
## 7.4 LCD.h File Reference

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  $dx \times dy$  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` =  $\sim(0x00) \& 0x07$  , `LCD_RED` =  $\sim(0x04) \& 0x07$  , `LCD_GREEN` =  $\sim(0x02) \& 0x07$  , `LCD_BLUE` =  $\sim(0x01) \& 0x07$  ,  
`LCD_YELLOW` =  $\sim(0x06) \& 0x07$  , `LCD_CYAN` =  $\sim(0x03) \& 0x07$  , `LCD_PURPLE` =  $\sim(0x05) \& 0x07$  ,  
`LCD_WHITE` =  $\sim(0x07) \& 0x07$  }
- 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.*

#### 7.4.1 Detailed Description

Header file for LCD module.

#### Author

Bryan McElvy

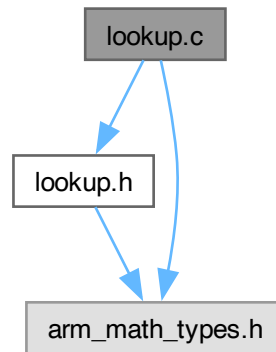
This module is essentially a higher-level interface to the ILI9341 module.

### 7.5 lookup.c File Reference

Source code for DAQ module's lookup table.



```
#include "lookup.h"
#include "arm_math_types.h"
Include dependency graph for lookup.c:
```



## Functions

- `const float32_t * Lookup\_GetPtr (void)`  
*Return a pointer to the DAQ lookup table.*

## Variables

- `static const float32_t LOOKUP_DAQ_TABLE [4096]`  
*Lookup table for converting ADC data from unsigned 12-bit integer values to 32-bit floating point values.*

### 7.5.1 Detailed Description

Source code for DAQ module's lookup table.

#### Author

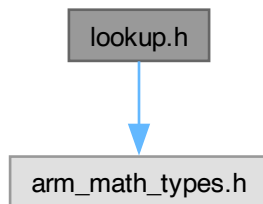
Bryan McElvy

## 7.6 lookup.h File Reference

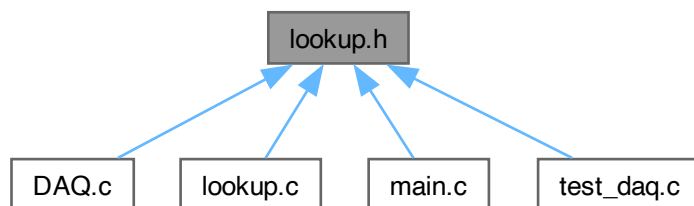
Lookup table for DAQ module.

```
#include "arm_math_types.h"
```

Include dependency graph for lookup.h:



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



## Macros

- `#define LOOKUP_DAQ_MAX (float32_t) 5.5`
- `#define LOOKUP_DAQ_MIN (float32_t)(-5.5)`

## Functions

- `const float32_t * Lookup\_GetPtr (void)`  
*Return a pointer to the DAQ lookup table.*

### 7.6.1 Detailed Description

Lookup table for DAQ module.

Author

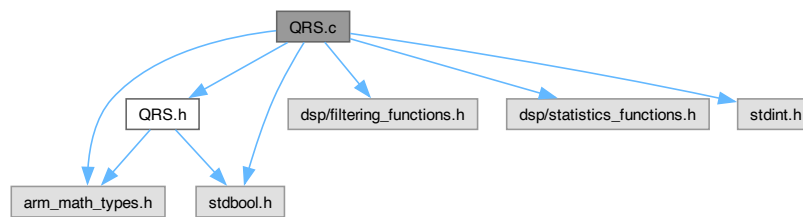
Bryan McElvy

## 7.7 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 20`
- `#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 ,
  - STATE\_BUFF\_SIZE\_DERFILT** = NUM\_COEFF\_DERFILT + QRS\_NUM\_SAMP - 1 , **NUM\_COEFF\_↵**  
**MOVAVG** = 10 , **STATE\_BUFF\_SIZE\_MOVAVG** = NUM\_COEFF\_MOVAVG + QRS\_NUM\_SAMP - 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

### 7.7.1 Detailed Description

Source code for QRS detection module.

#### Author

Bryan McElvy

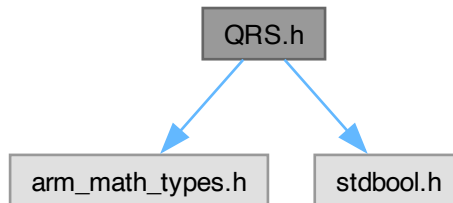
The algorithm used in this file is a simplified version of the Pan-Tompkins algorithm. Specifically, this version currently only uses the integrated signal for the thresholding, and also completely omits the searchback and T wave discrimination parts of the original.

## 7.8 QRS.h File Reference

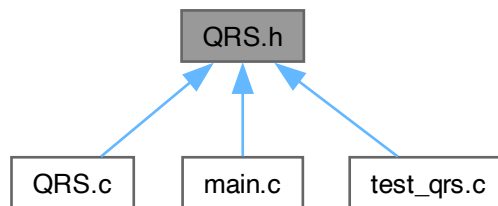
Header file for QRS detection module.

```
#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) (1200))`

### 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.*

### 7.8.1 Detailed Description

Header file for QRS detection module.

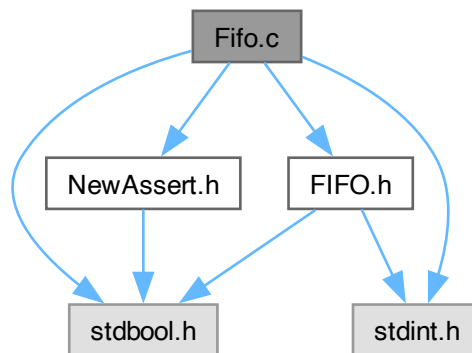
Author

Bryan McElvy

## 7.9 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](#) **buffer\_pool** [FIFO\_POOL\_SIZE] = { 0 }  
*pre-allocated pool*
- static uint8\_t **free\_buffers** = FIFO\_POOL\_SIZE  
*no. of remaining buffers*

#### 7.9.1 Detailed Description

Source code for FIFO buffer module.

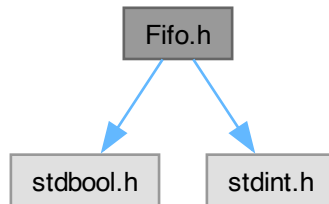
#### Author

Bryan McElvy

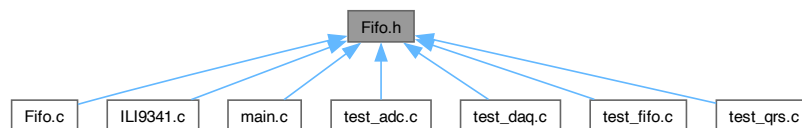
## 7.10 Fifo.h File Reference

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.*

**7.10.1 Detailed Description**

Header file for FIFO buffer implementation.

Author

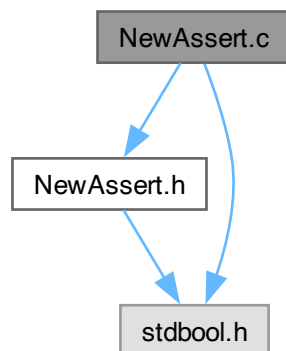
Bryan McElvy

**7.11 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.*

### 7.11.1 Detailed Description

Source code for custom `assert` implementation.

#### Author

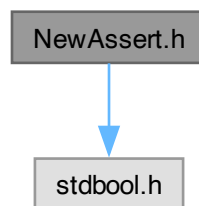
Bryan McElvy

## 7.12 NewAssert.h File Reference

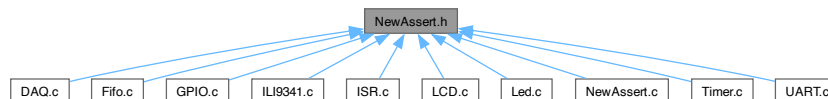
Header file for custom `assert` implementation.

```
#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.*

### 7.12.1 Detailed Description

Header file for custom `assert` implementation.

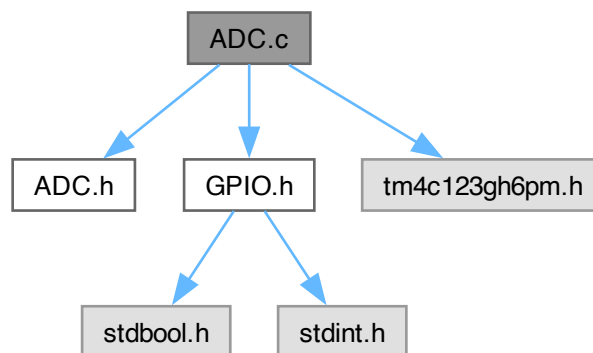
Author

Bryan McElvy

## 7.13 ADC.c File Reference

Source code for 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.*

### 7.13.1 Detailed Description

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

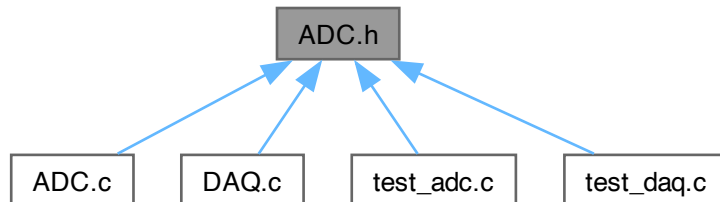
Author

Bryan McElvy

## 7.14 ADC.h File Reference

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

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



### Functions

- void `ADC_Init` (void)  
*Initialize ADC0 as a single-input analog-to-digital converter.*

#### 7.14.1 Detailed Description

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

#### Author

Bryan McElvy

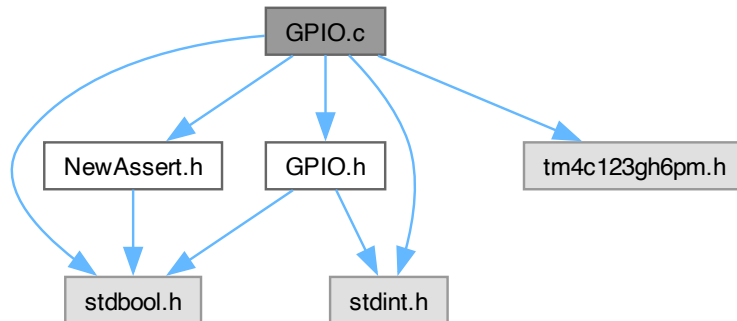
## 7.15 GPIO.c File Reference

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 [GPIO\\_Port\\_t](#)

## Macros

- `#define GPIO_NUM_PORTS 6`

## Typedefs

- `typedef volatile uint32_t * register_t`

## 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

- **GPIO\_Port\_t \* GPIO\_InitPort** (GPIO\_PortName\_t portName)  
*Initialize a GPIO Port and return a pointer to its struct.*
- **bool GPIO\_isPortInit** (GPIO\_Port\_t \*gpioPort)  
*Check if the GPIO port is initialized.*
- **uint32\_t GPIO\_getBaseAddr** (GPIO\_Port\_t \*gpioPort)
- **void GPIO\_ConfigDirOutput** (GPIO\_Port\_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** (GPIO\_Port\_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** (GPIO\_Port\_t \*gpioPort, GPIO\_Pin\_t pinMask)  
*Activate the specified pins' internal pull-up resistors.*
- **void GPIO\_ConfigPullDown** (GPIO\_Port\_t \*gpioPort, GPIO\_Pin\_t pinMask)  
*Activate the specified pins' internal pull-down resistors.*
- **void GPIO\_ConfigDriveStrength** (GPIO\_Port\_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** (GPIO\_Port\_t \*gpioPort, GPIO\_Pin\_t pinMask)  
*Enable digital I/O for the specified pins.*
- **void GPIO\_DisableDigital** (GPIO\_Port\_t \*gpioPort, GPIO\_Pin\_t pinMask)  
*Disable digital I/O for the specified pins.*
- **void GPIO\_ConfigInterrupts\_Edge** (GPIO\_Port\_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** (GPIO\_Port\_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** (GPIO\_Port\_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** (GPIO\_Port\_t \*gpioPort, uint8\_t priority)  
*Configure interrupts for the selected port in the NVIC.*
- **uint8\_t GPIO\_ReadPins** (GPIO\_Port\_t \*gpioPort, GPIO\_Pin\_t pinMask)  
*Read from the specified GPIO pin.*
- **void GPIO\_WriteHigh** (GPIO\_Port\_t \*gpioPort, GPIO\_Pin\_t pinMask)  
*Write a 1 to the specified GPIO pins.*
- **void GPIO\_WriteLow** (GPIO\_Port\_t \*gpioPort, GPIO\_Pin\_t pinMask)  
*Write a 0 to the specified GPIO pins.*
- **void GPIO\_Toggle** (GPIO\_Port\_t \*gpioPort, GPIO\_Pin\_t pinMask)  
*Toggle the specified GPIO pins.*
- **void GPIO\_ConfigAltMode** (GPIO\_Port\_t \*gpioPort, GPIO\_Pin\_t pinMask)  
*Activate the alternate mode for the specified pins.*
- **void GPIO\_ConfigPortCtrl** (GPIO\_Port\_t \*gpioPort, GPIO\_Pin\_t pinMask, uint8\_t fieldEncoding)  
*Specify the alternate mode to use for the specified pins.*
- **void GPIO\_ConfigAnalog** (GPIO\_Port\_t \*gpioPort, GPIO\_Pin\_t pinMask)  
*Activate analog mode for the specified GPIO pins.*

## Variables

- static **GPIO\_Port\_t GPIO\_PTR\_ARR** [6]

### 7.15.1 Detailed Description

Source code for GPIO module.

Author

Bryan McElvy

### 7.15.2 Function Documentation

#### GPIO\_InitPort()

```
GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 OUTPUT pin(s).

**GPIO\_ConfigDirInput()**

```
void GPIO_ConfigDirInput (
    GPIO_Port_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 INPUT pin(s).

**GPIO\_ConfigPullUp()**

```
void GPIO_ConfigPullUp (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (  
    GPIO_Port_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 (  
    GPIO_Port_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 (  
    GPIO_Port_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\_ReadPins()**

```
uint8_t GPIO_ReadPins (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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).

## 7.15.3 Variable Documentation

## GPIO\_PTR\_ARR

```
GPIO_Port_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 },
}
```

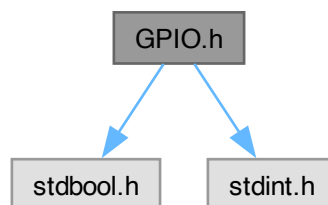
## 7.16 GPIO.h File Reference

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

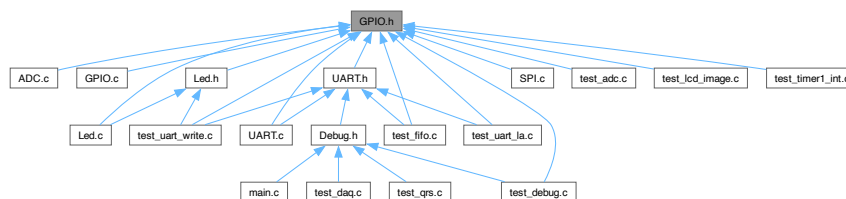
```
#include <stdbool.h>
```

```
#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

- **GPIO\_Port\_t \* GPIO\_InitPort** (GPIO\_PortName\_t portName)  
Initialize a GPIO Port and return a pointer to its *struct*.
- uint32\_t **GPIO\_getBaseAddr** (GPIO\_Port\_t \*gpioPort)
- bool **GPIO\_isPortInit** (GPIO\_Port\_t \*gpioPort)  
Check if the GPIO port is initialized.
- void **GPIO\_ConfigDirOutput** (GPIO\_Port\_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** (GPIO\_Port\_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** (GPIO\_Port\_t \*gpioPort, GPIO\_Pin\_t pinMask)  
Activate the specified pins' internal pull-up resistors.
- void **GPIO\_ConfigPullDown** (GPIO\_Port\_t \*gpioPort, GPIO\_Pin\_t pinMask)  
Activate the specified pins' internal pull-down resistors.
- void **GPIO\_ConfigDriveStrength** (GPIO\_Port\_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** (GPIO\_Port\_t \*gpioPort, GPIO\_Pin\_t pinMask)  
Enable digital I/O for the specified pins.
- void **GPIO\_DisableDigital** (GPIO\_Port\_t \*gpioPort, GPIO\_Pin\_t pinMask)  
Disable digital I/O for the specified pins.
- void **GPIO\_ConfigInterrupts\_Edge** (GPIO\_Port\_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** (GPIO\_Port\_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** (GPIO\_Port\_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** (GPIO\_Port\_t \*gpioPort, uint8\_t priority)  
Configure interrupts for the selected port in the NVIC.
- uint8\_t **GPIO\_ReadPins** (GPIO\_Port\_t \*gpioPort, GPIO\_Pin\_t pinMask)  
Read from the specified GPIO pin.
- void **GPIO\_WriteHigh** (GPIO\_Port\_t \*gpioPort, GPIO\_Pin\_t pinMask)  
Write a 1 to the specified GPIO pins.

- void [GPIO\\_WriteLow](#) ([GPIO\\_Port\\_t](#) \*gpioPort, [GPIO\\_Pin\\_t](#) pinMask)  
*Write a 0 to the specified GPIO pins.*
- void [GPIO\\_Toggle](#) ([GPIO\\_Port\\_t](#) \*gpioPort, [GPIO\\_Pin\\_t](#) pinMask)  
*Toggle the specified GPIO pins.*
- void [GPIO\\_ConfigAltMode](#) ([GPIO\\_Port\\_t](#) \*gpioPort, [GPIO\\_Pin\\_t](#) pinMask)  
*Activate the alternate mode for the specified pins.*
- void [GPIO\\_ConfigPortCtrl](#) ([GPIO\\_Port\\_t](#) \*gpioPort, [GPIO\\_Pin\\_t](#) pinMask, [uint8\\_t](#) fieldEncoding)  
*Specify the alternate mode to use for the specified pins.*
- void [GPIO\\_ConfigAnalog](#) ([GPIO\\_Port\\_t](#) \*gpioPort, [GPIO\\_Pin\\_t](#) pinMask)  
*Activate analog mode for the specified GPIO pins.*

### 7.16.1 Detailed Description

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

#### Author

Bryan McElvy

### 7.16.2 Function Documentation

#### GPIO\_InitPort()

```
GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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\_ReadPins()**

```
uint8_t GPIO_ReadPins (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_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 (
    GPIO_Port_t * gpioPort,
    GPIO_Pin_t pinMask )
```

Activate analog mode for the specified GPIO pins.

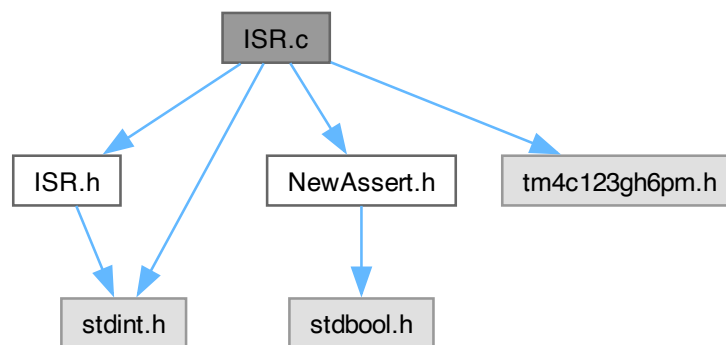
## Parameters

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

## 7.17 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)`

## Typedefs

- `typedef volatile uint32_t * register_t`

## 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

### 7.17.1 Detailed Description

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

Author

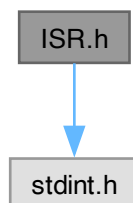
Bryan McElvy

## 7.18 ISR.h File Reference

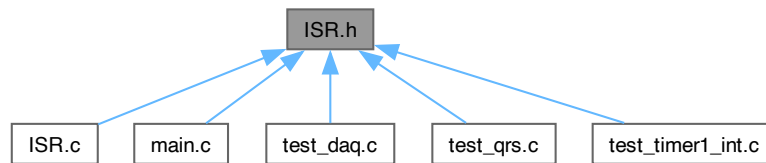
Header file for interrupt service routine (ISR) configuration module.

```
#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).*

### 7.18.1 Detailed Description

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

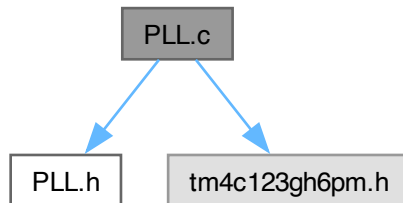
#### Author

Bryan McElvy

## 7.19 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.*

#### 7.19.1 Detailed Description

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

#### Author

Bryan McElvy

## 7.20 PLL.h File Reference

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

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



### Functions

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



### 7.20.1 Detailed Description

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

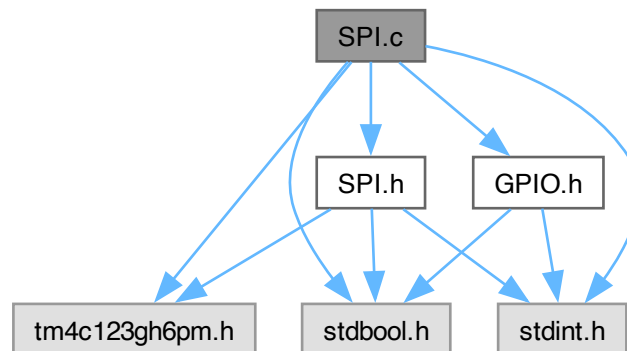
Author

Bryan McElvy

## 7.21 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 SSIO 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

Source code for serial peripheral interface (SPI) module.

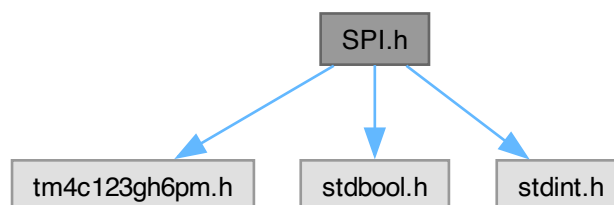
Author

Bryan McElvy

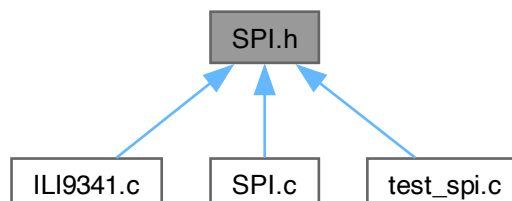
### 7.22 SPI.h File Reference

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.*

### 7.22.1 Detailed Description

Header file for serial peripheral interface (SPI) module.

#### Author

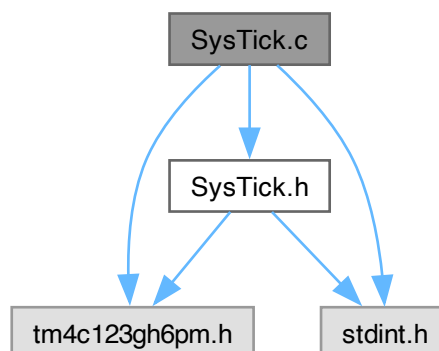
Bryan McElvy

## 7.23 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.*

### 7.23.1 Detailed Description

Implementation details for SysTick functions.

#### Author

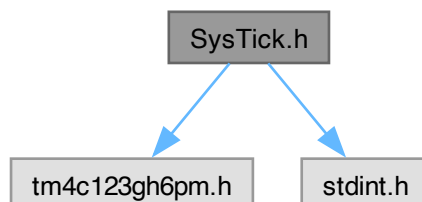
Bryan McElvy

## 7.24 SysTick.h File Reference

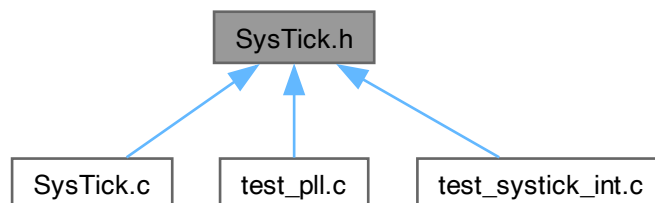
Driver module for using SysTick-based timing and/or interrupts.

```
#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.*

### 7.24.1 Detailed Description

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

#### Author

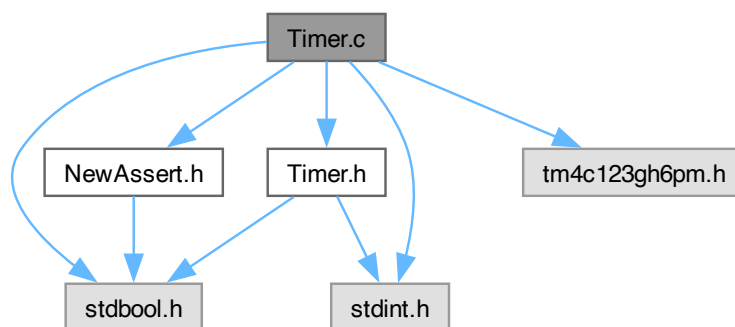
Bryan McElvy

## 7.25 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**

## Typedefs

- typedef volatile uint32\_t \* **register\_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.*
- 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]

### 7.25.1 Detailed Description

Source code for Timer module.

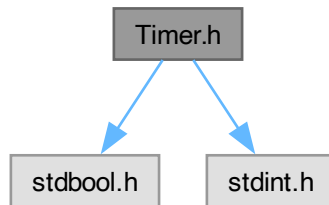
#### Author

Bryan McElvy

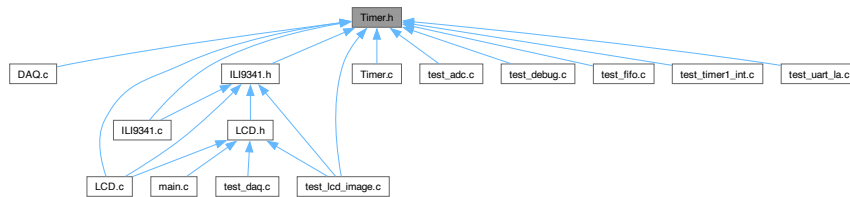
## 7.26 Timer.h File Reference

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.*
- 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.*

### 7.26.1 Detailed Description

Device driver for general-purpose timer modules.

#### Author

Bryan McElvy

## 7.27 UART.c File Reference

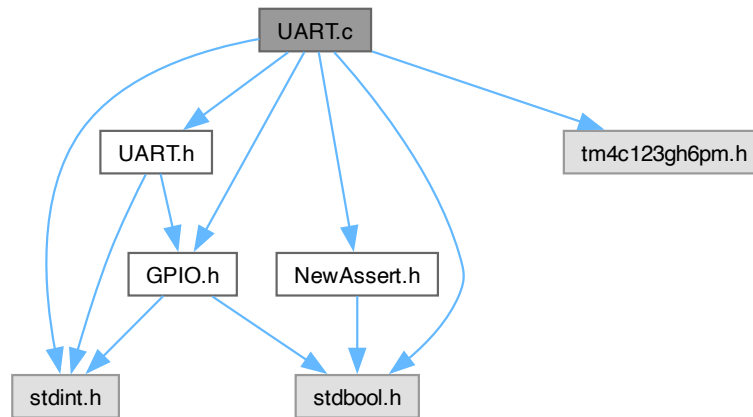
Source code for UART module.

```
#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`

## Typedefs

- `typedef volatile uint32_t * register_t`

## 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 (GPIO_Port_t *port, UART_Num_t uartNum)`  
*Initialize the specified UART peripheral.*
- `unsigned char UART_ReadChar (UART_t *uart)`  
*Read a single ASCII character from the UART.*
- `void UART_WriteChar (UART_t *uart, unsigned char input_char)`  
*Write a single character to the UART.*
- `void UART_WriteStr (UART_t *uart, void *input_str)`  
*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 num_decimals)`  
*Write a floating-point number the UART.*

## Variables

- `static UART_t UART_ARR [8]`

### 7.27.1 Detailed Description

Source code for UART module.

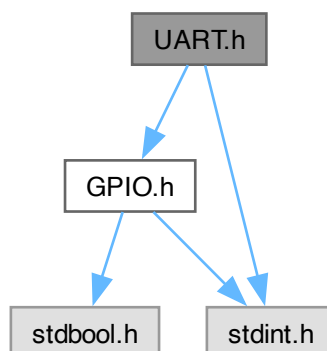
Author

Bryan McElvy

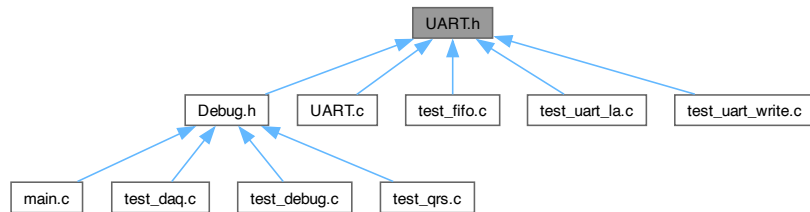
## 7.28 UART.h File Reference

Driver module for serial communication via UART0 and UART 1.

```
#include "GPIO.h"
#include <stdint.h>
Include dependency graph for UART.h:
```



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



## Enumerations

- enum **UART\_Num\_t** {  
**UART0** , **UART1** , **UART2** , **UART3** ,  
**UART4** , **UART5** , **UART6** , **UART7** }

## Functions

- UART\_t \* UART\_Init** (GPIO\_Port\_t \*port, UART\_Num\_t uartNum)  
*Initialize the specified UART peripheral.*
- unsigned char **UART\_ReadChar** (UART\_t \*uart)  
*Read a single ASCII character from the UART.*
- void **UART\_WriteChar** (UART\_t \*uart, unsigned char input\_char)  
*Write a single character to the UART.*
- void **UART\_WriteStr** (UART\_t \*uart, void \*input\_str)  
*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 num\_decimals)  
*Write a floating-point number the UART.*

### 7.28.1 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.

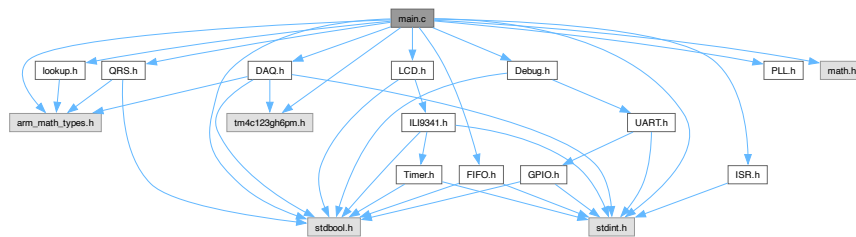
UART1 uses PB0 (Rx) and PB1 (Tx), which are broken out but do not connect to a serial port.

## 7.29 main.c File Reference

Main program file.

```
#include "DAQ.h"
#include "Debug.h"
#include "LCD.h"
#include "lookup.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\_CAPACITY** = 3 , **DAQ\_BUFFER\_SIZE** = DAQ\_FIFO\_CAPACITY + 1 , **QRS\_BUFFER\_SIZE** = QRS\_NUM\_SAMP + 1 , **LCD\_FIFO\_CAPACITY** = DAQ\_FIFO\_CAPACITY , **LCD\_BUFFER\_SIZE** = LCD\_FIFO\_CAPACITY + 1 }
- enum { **LCD\_TOP\_LINE** = (LCD\_Y\_MAX - 48) , **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\_BUFFER\_SIZE] = { 0 }
- static volatile `Fifo_t` **QRS\_Fifo** = 0
- static volatile `uint32_t` **QRS\_fifoBuffer** [QRS\_BUFFER\_SIZE] = { 0 }
- static volatile `bool` **QRS\_bufferIsFull** = false
- static volatile `Fifo_t` **LCD\_Fifo** = 0
- static volatile `uint32_t` **LCD\_fifoBuffer** [LCD\_BUFFER\_SIZE] = { 0 }
- static `float32_t` **QRS\_processingBuffer** [QRS\_BUFFER\_SIZE] = { 0 }
- static `uint16_t` **LCD\_prevSampleBuffer** [LCD\_X\_MAX] = { 0 }

### 7.29.1 Detailed Description

Main program file.

#### Author

Bryan McElvy

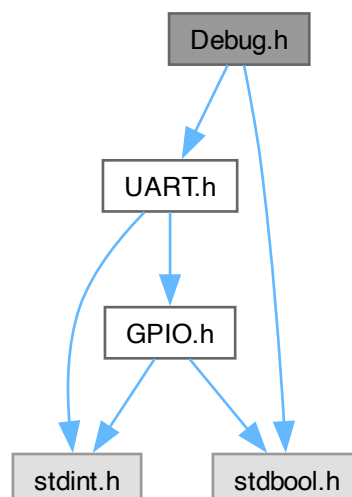
## 7.30 Debug.h File Reference

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

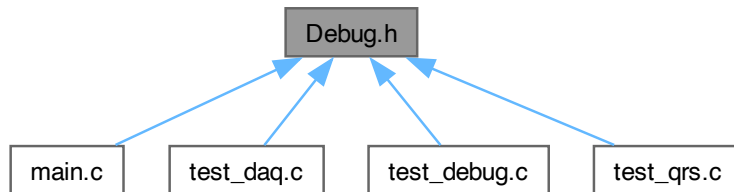
```
#include "UART.h"
```

```
#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](#) (void)  
*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.*

### 7.30.1 Detailed Description

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

#### Author

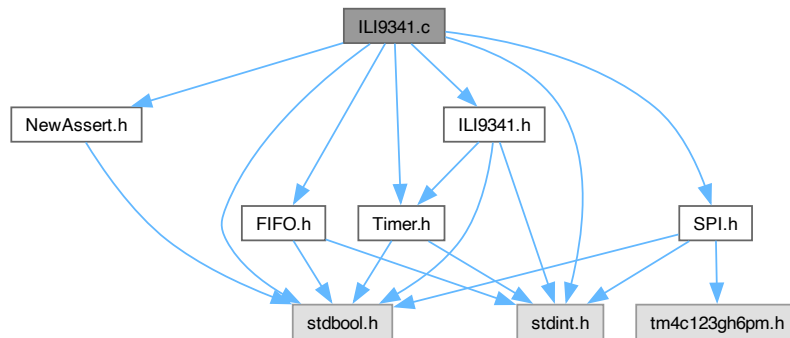
Bryan McElvy

## 7.31 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 }

### 7.31.1 Detailed Description

Source code for ILI9341 module.

#### Author

Bryan McElvy

## 7.32 ILI9341.h File Reference

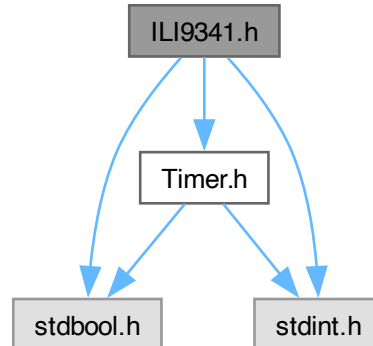
Driver module for interfacing with an ILI9341 LCD driver.

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

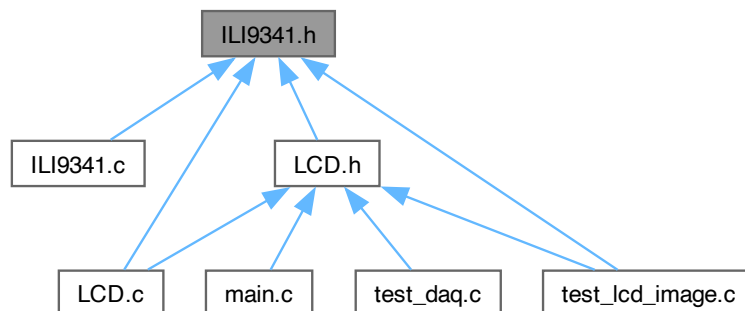


```
#include <stdint.h>
```

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 , **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 ,  
**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** = 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

- 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\_setDisplInversion** (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 areRowsAndColsSwitched, 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.*

### 7.32.1 Detailed Description

Driver module for interfacing with an ILI9341 LCD driver.

Author

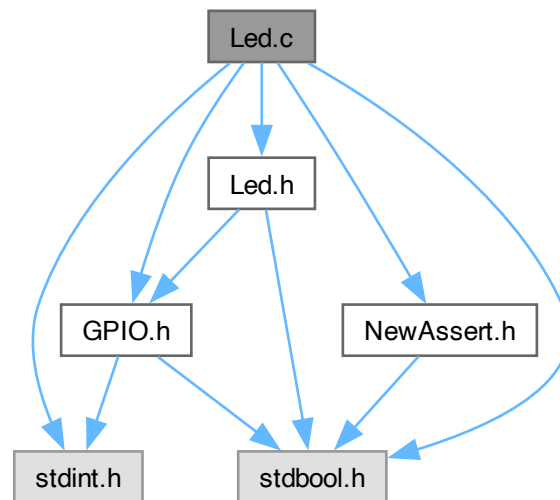
Bryan McElvy

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.

## 7.33 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](#) ([GPIO\\_Port\\_t](#) \*gpioPort, [GPIO\\_Pin\\_t](#) pin)  
*Initialize a light-emitting diode (LED) as an [Led\\_t](#).*
- bool [Led\\_isInit](#) ([Led\\_t](#) led)
- [GPIO\\_Port\\_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`

### 7.33.1 Detailed Description

Source code for LED module.

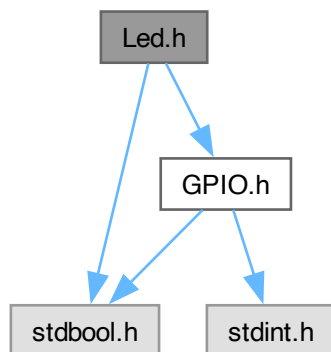
#### Author

Bryan McElvy

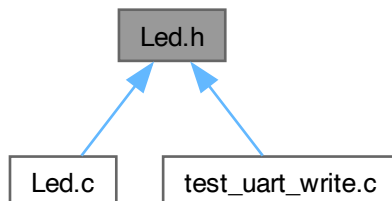
## 7.34 Led.h File Reference

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 (GPIO_Port_t *gpioPort, GPIO_Pin_t pin)`  
*Initialize a light-emitting diode (LED) as an `Led_t`.*
- `bool Led_isInit (Led_t led)`
- `GPIO_Port_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).*

### 7.34.1 Detailed Description

Interface for LED module.

Author

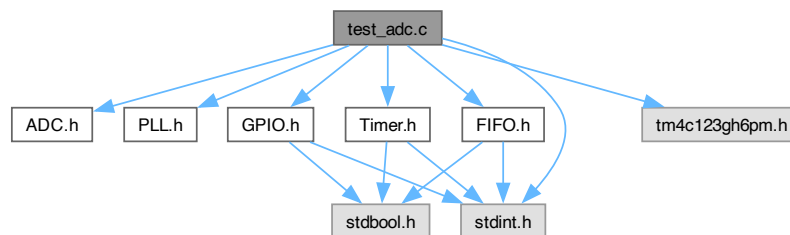
Bryan McElvy

## 7.35 test\_adc.c File Reference

Test script for analog-to-digital conversion (ADC) module.

```
#include "ADC.h"
#include "PLL.h"
#include "GPIO.h"
#include "Timer.h"
#include "FIFO.h"
#include "tm4c123gh6pm.h"
#include <stdint.h>
```

Include dependency graph for test\_adc.c:



## Macros

- `#define LED_PINS (GPIO_Pin_t)(GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3)`
- `#define SAMPLING_PERIOD_MS (uint32_t) 5`
- `#define NUM_SAMPLES (uint32_t) 1000`

## Functions

- `int main (void)`
- `void ADC0_SS3_Handler (void)`

## Variables

- volatile bool `buffer_is_full` = false
- volatile `Fifo_t fifo_ptr` = 0
- volatile `uint32_t fifo_buffer` [NUM\_SAMPLES]

### 7.35.1 Detailed Description

Test script for analog-to-digital conversion (ADC) module.

#### Author

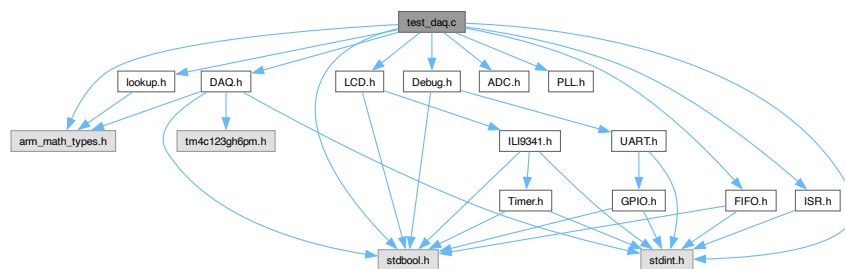
Bryan McElvy

### 7.36 test\_daq.c File Reference

Test script for the data acquisition (DAQ) module.

```
#include "DAQ.h"
#include "Debug.h"
#include "LCD.h"
#include "ADC.h"
#include "PLL.h"
#include "FIFO.h"
#include "ISR.h"
#include "lookup.h"
#include "arm_math_types.h"
#include <stdbool.h>
#include <stdint.h>
```

Include dependency graph for test\_daq.c:



## Macros

- `#define DAQ_BUFFER_SIZE 128`
- `#define LCD_TOP_LINE (LCD_Y_MAX - 48)`
- `#define LCD_NUM_Y_VALS 128`
- `#define LCD_X_AXIS_OFFSET 32`
- `#define LCD_Y_MIN (0 + LCD_X_AXIS_OFFSET)`
- `#define LCD_Y_MAX (LCD_NUM_Y_VALS + LCD_X_AXIS_OFFSET)`

## Functions

- void **LCD\_plotNewSample** (uint16\_t x, volatile const float32\_t sample)
- int **main** (void)
- void **ADC0\_SS3\_Handler** (void)

## Variables

- volatile Fifo\_t **inputFifo** = 0
- volatile uint32\_t **inputBuffer** [DAQ\_BUFFER\_SIZE] = { 0 }
- volatile bool **sampleReady** = false

### 7.36.1 Detailed Description

Test script for the data acquisition (DAQ) module.

#### Author

Bryan McElvy

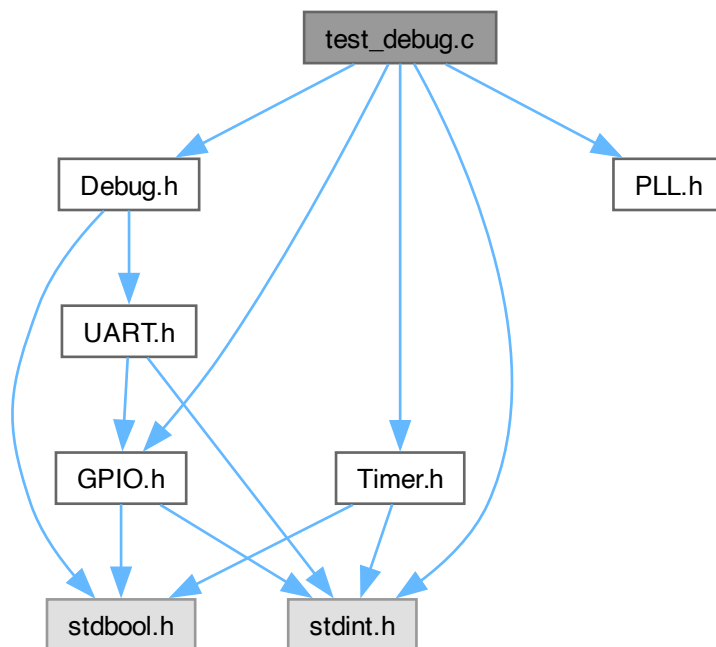
## 7.37 test\_debug.c File Reference

Test script for Debug module.

```
#include "Debug.h"
#include "GPIO.h"
#include "PLL.h"
#include "Timer.h"
```

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

Include dependency graph for test\_debug.c:



## Functions

- int **main** (void)

### 7.37.1 Detailed Description

Test script for Debug module.

#### Author

Bryan McElvy

## 7.38 test\_fifo.c File Reference

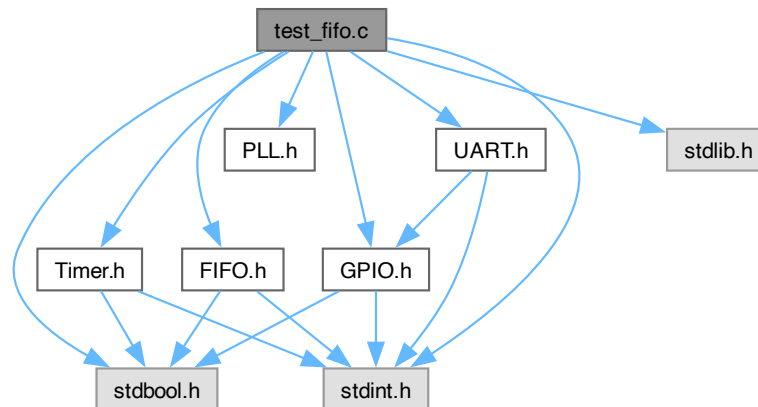
Test script for FIFO buffer.

```
#include "FIFO.h"
#include "PLL.h"
#include "UART.h"
#include "GPIO.h"
#include "Timer.h"
#include <stdbool.h>
```



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

Include dependency graph for test\_fifo.c:



## Macros

- `#define FIFO_LEN 10`
- `#define LED_PINS (GPIO_Pin_t)(GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3)`

## Functions

- `void FIFO_reportStatus (Fifo_t fifo_ptr)`
- `int main (void)`

## Variables

- `UART\_t * uart`

### 7.38.1 Detailed Description

Test script for FIFO buffer.

#### Author

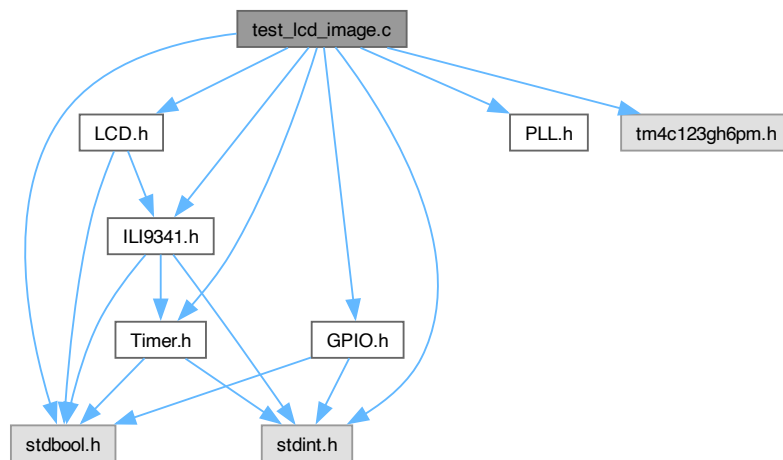
Bryan McElvy

### 7.39 test\_lcd\_image.c File Reference

Test script for writing images onto the display.

```
#include "LCD.h"
#include "GPIO.h"
#include "PLL.h"
#include "Timer.h"
#include "ILI9341.h"
#include "tm4c123gh6pm.h"
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for test\_lcd\_image.c:



#### Macros

- `#define X_OFFSET (uint16_t) 0`
- `#define SIZE (uint16_t) 4`
- `#define LED_PINS (GPIO_Pin_t)(GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3)`

#### Functions

- `int main (void)`

#### Variables

- `const uint8_t COLOR_ARR [6] = { LCD_RED, LCD_YELLOW, LCD_GREEN, LCD_CYAN, LCD_BLUE, LCD_PURPLE }`
- `uint8_t color_idx`

### 7.39.1 Detailed Description

Test script for writing images onto the display.

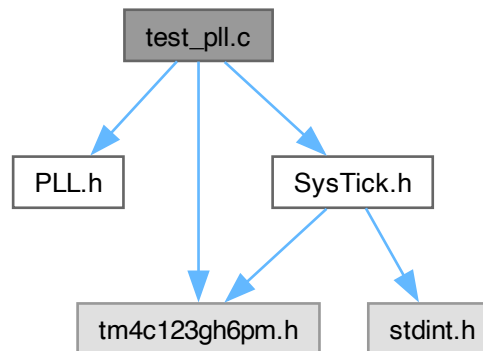
Author

Bryan McElvy

## 7.40 test\_pll.c File Reference

Test script for the PLL module.

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



### Macros

- `#define RED (uint8_t) 0x02`
- `#define BLUE (uint8_t) 0x04`
- `#define GREEN (uint8_t) 0x08`

### Functions

- `void GPIO_PortF_Init (void)`
- `int main ()`

### 7.40.1 Detailed Description

Test script for the PLL module.

Author

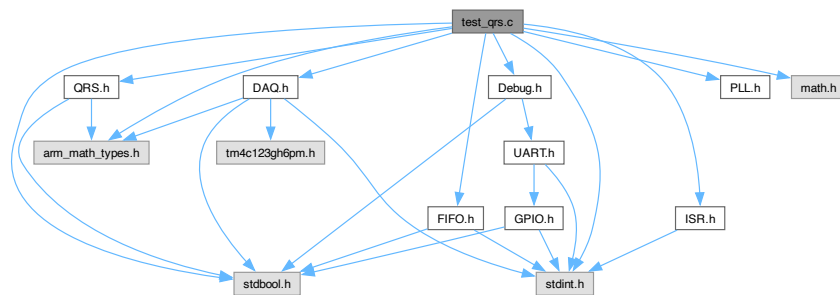
Bryan McElvy

## 7.41 test\_qrs.c File Reference

QRS detector test script.

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

Include dependency graph for test\_qrs.c:



### Enumerations

- enum { **ADC\_VECTOR\_NUM** = INT\_ADC0SS3 , **DAQ\_VECTOR\_NUM** = INT\_CAN0 }
- enum { **DAQ\_FIFO\_CAPACITY** = 8 , **DAQ\_BUFFER\_SIZE** = DAQ\_FIFO\_CAPACITY + 1 , **QRS\_BUFFER\_SIZE** = QRS\_NUM\_SAMP + 1 }

### Functions

- static void **ADC\_Handler** (void)
- static void **DAQ\_Handler** (void)
- int **main** (void)

### Variables

- static volatile Fifo\_t **DAQ\_Fifo** = 0
- static volatile uint32\_t **DAQ\_Buffer** [DAQ\_BUFFER\_SIZE] = { 0 }
- static volatile Fifo\_t **QRS\_Fifo** = 0
- static volatile uint32\_t **QRS\_FifoBuffer** [QRS\_BUFFER\_SIZE] = { 0 }
- static volatile bool **QRS\_bufferIsFull** = false
- volatile float32\_t **QRS\_InputBuffer** [QRS\_BUFFER\_SIZE] = { 0 }
- volatile float32\_t **QRS\_OutputBuffer** [QRS\_BUFFER\_SIZE] = { 0 }

### 7.41.1 Detailed Description

QRS detector test script.

Author

Bryan McElvy

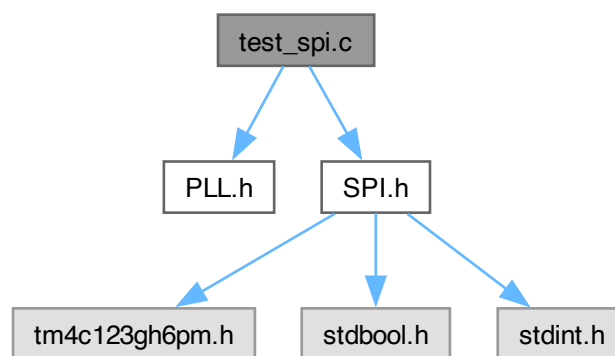
## 7.42 test\_spi.c File Reference

Test script for initializing SSI0 and writing data/commands via SPI.

```
#include "PLL.h"
```

```
#include "SPI.h"
```

Include dependency graph for test\_spi.c:



### Functions

- int **main** ()

### 7.42.1 Detailed Description

Test script for initializing SSI0 and writing data/commands via SPI.

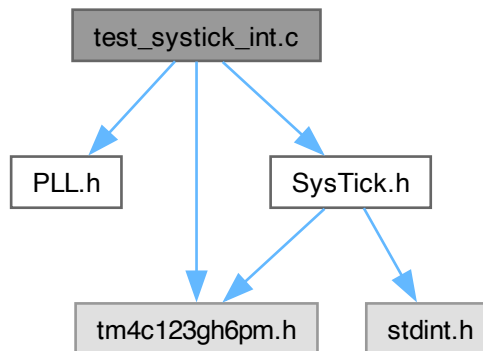
Author

Bryan McElvy

## 7.43 test\_systick\_int.c File Reference

Test script for SysTick interrupts.

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



### Functions

- void **GPIO\_PortF\_Init** (void)
- int **main** ()
- void **SysTick\_Handler** (void)

### Variables

- const uint8\_t **color\_table** [6] = { 0x02, 0x06, 0x04, 0x0C, 0x08, 0x0A }
- volatile uint8\_t **color\_idx** = 0
- volatile uint8\_t **led\_is\_on** = 0

### 7.43.1 Detailed Description

Test script for SysTick interrupts.

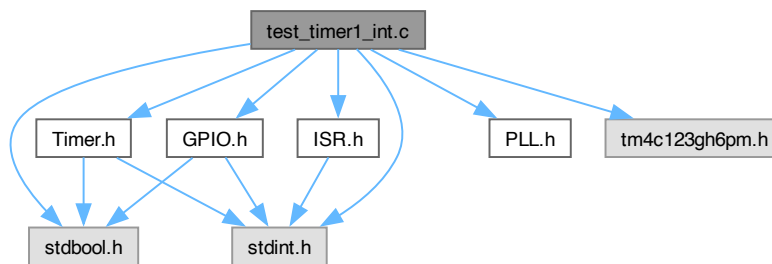
#### Author

Bryan McElvy

## 7.44 test\_timer1\_int.c File Reference

Test script for relocating the vector table to RAM.

```
#include "GPIO.h"
#include "PLL.h"
#include "Timer.h"
#include "ISR.h"
#include "tm4c123gh6pm.h"
#include <stdbool.h>
#include <stdint.h>
Include dependency graph for test_timer1_int.c:
```



### Macros

- `#define LED_PINS (GPIO_Pin_t)(GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3)`

### Functions

- `int main (void)`
- `void Timer1A_Handler (void)`

### Variables

- `GPIO_Port_t * portF = 0`
- `Timer_t timer1 = 0`
- `bool isLedOn = false`

#### 7.44.1 Detailed Description

Test script for relocating the vector table to RAM.

Test script for Timer1A interrupts.

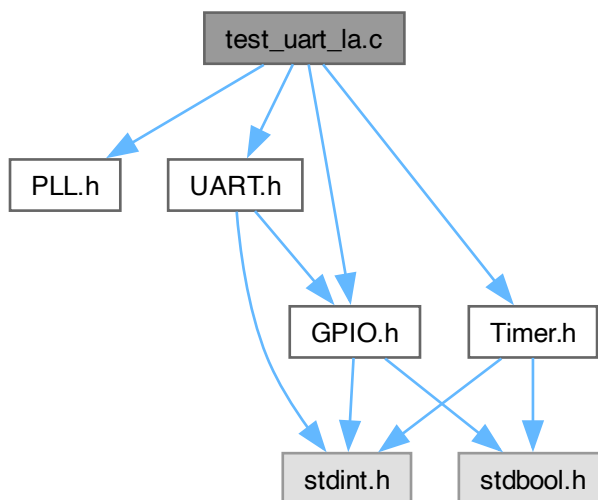
#### Author

Bryan McElvy

## 7.45 test\_uart\_la.c File Reference

Test script for using a USB logic analyzer to decode UART signals.

```
#include "PLL.h"
#include "GPIO.h"
#include "Timer.h"
#include "UART.h"
Include dependency graph for test_uart_la.c:
```



### Functions

- int **main** (void)

#### 7.45.1 Detailed Description

Test script for using a USB logic analyzer to decode UART signals.

#### Author

Bryan McElvy

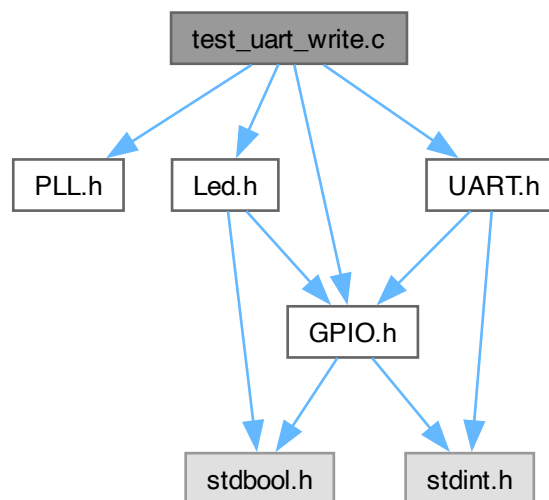


## 7.46 test\_uart\_write.c File Reference

Test script for writing to serial port via UART0.

```
#include "PLL.h"  
#include "GPIO.h"  
#include "Led.h"  
#include "UART.h"
```

Include dependency graph for test\_uart\_write.c:



### Functions

- `int main (void)`

### Variables

- volatile unsigned char `in_char`
- `uint32_t counter`

#### 7.46.1 Detailed Description

Test script for writing to serial port via UART0.

#### Author

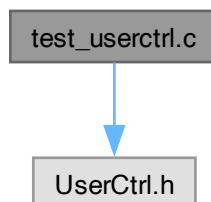
Bryan McElvy

## 7.47 test\_userctrl.c File Reference

Test file for GPIO/UserCtrl modules and GPIO interrupts.

```
#include "UserCtrl.h"
```

Include dependency graph for test\_userctrl.c:



### Functions

- int **main** ()

#### 7.47.1 Detailed Description

Test file for GPIO/UserCtrl modules and GPIO interrupts.

#### Author

Bryan McElvy

## Index

- ADC.c, [95](#)
- ADC.h, [96](#)
- ADC\_Init
  - Analog-to-Digital Conversion (ADC), [7](#)
- Analog-to-Digital Conversion (ADC), [6](#)
  - ADC\_Init, [7](#)
- Application Software, [43](#)
- Assert
  - Common, [65](#)
- bandpassFiltStruct
  - Data Acquisition (DAQ), [49](#)
- Bug List, [1](#)
- CASET
  - ILI9341, [32](#)
- Cmd\_t
  - ILI9341, [32](#)
- COEFF\_BANDPASS
  - QRS, [63](#)
- COEFF\_DERFILT
  - QRS, [64](#)
- COEFF\_MOVAVG
  - QRS, [64](#)
- COEFFS\_BANDPASS
  - Data Acquisition (DAQ), [48](#)
- COEFFS\_NOTCH
  - Data Acquisition (DAQ), [48](#)
- Common, [64](#)
  - Assert, [65](#)
- DAQ.c, [77](#)
- DAQ.h, [79](#)
- DAQ\_BandpassFilter
  - Data Acquisition (DAQ), [47](#)
- DAQ\_BUFFER\_SIZE
  - Main, [72](#)
- DAQ\_convertToMilliVolts
  - Data Acquisition (DAQ), [46](#)
- DAQ\_FIFO\_CAPACITY
  - Main, [72](#)
- DAQ\_Handler
  - Main, [73](#)
- DAQ\_Init
  - Data Acquisition (DAQ), [46](#)
- DAQ\_NotchFilter
  - Data Acquisition (DAQ), [47](#)
- DAQ\_readSample
  - Data Acquisition (DAQ), [46](#)
- Data Acquisition (DAQ), [44](#)
  - bandpassFiltStruct, [49](#)
  - COEFFS\_BANDPASS, [48](#)
  - COEFFS\_NOTCH, [48](#)
  - DAQ\_BandpassFilter, [47](#)
  - DAQ\_convertToMilliVolts, [46](#)
  - DAQ\_Init, [46](#)
  - DAQ\_NotchFilter, [47](#)
  - DAQ\_readSample, [46](#)
  - Lookup\_GetPtr, [48](#)
- Debug, [49](#)
  - Debug\_Assert, [52](#)
  - Debug\_Init, [50](#)
  - Debug\_SendFromList, [51](#)
  - Debug\_SendMsg, [50](#)
  - Debug\_WriteFloat, [51](#)
- Debug.h, [129](#)
- Debug\_Assert
  - Debug, [52](#)
- Debug\_Init
  - Debug, [50](#)
- Debug\_SendFromList
  - Debug, [51](#)
- Debug\_SendMsg
  - Debug, [50](#)
- Debug\_WriteFloat
  - Debug, [51](#)
- Device Drivers, [5](#)
- DINVOFF
  - ILI9341, [32](#)
- DINVON
  - ILI9341, [32](#)
- DISPOFF
  - ILI9341, [32](#)
- DISPON
  - ILI9341, [32](#)
- DOWN
  - Timer, [14](#)
- FIFO, [65](#)
  - FIFO\_Flush, [68](#)
  - FIFO\_Get, [68](#)
  - FIFO\_getCurrSize, [71](#)
  - FIFO\_Init, [67](#)
  - FIFO\_isEmpty, [70](#)
  - FIFO\_isFull, [70](#)
  - FIFO\_PeekAll, [70](#)
  - FIFO\_PeekOne, [69](#)
  - FIFO\_Put, [67](#)
  - FIFO\_Reset, [69](#)
  - FIFO\_TransferAll, [69](#)
  - FIFO\_TransferOne, [68](#)
- Fifo.c, [90](#)
- Fifo.h, [92](#)
- FIFO\_Flush
  - FIFO, [68](#)
- FIFO\_Get
  - FIFO, [68](#)
- FIFO\_getCurrSize
  - FIFO, [71](#)
- FIFO\_Init
  - FIFO, [67](#)

- FIFO\_isEmpty
  - FIFO, [70](#)
- FIFO\_isFull
  - FIFO, [70](#)
- FIFO\_PeekAll
  - FIFO, [70](#)
- FIFO\_PeekOne
  - FIFO, [69](#)
- FIFO\_Put
  - FIFO, [67](#)
- FIFO\_Reset
  - FIFO, [69](#)
- Fifo\_t, [75](#)
- FIFO\_TransferAll
  - FIFO, [69](#)
- FIFO\_TransferOne
  - FIFO, [68](#)
- FRMCTR1
  - ILI9341, [32](#)
- FRMCTR2
  - ILI9341, [32](#)
- FRMCTR3
  - ILI9341, [32](#)
- GPIO, [7](#)
- GPIO.c, [96](#)
  - GPIO\_ConfigAltMode, [104](#)
  - GPIO\_ConfigAnalog, [104](#)
  - GPIO\_ConfigDirInput, [100](#)
  - GPIO\_ConfigDirOutput, [99](#)
  - GPIO\_ConfigDriveStrength, [100](#)
  - GPIO\_ConfigInterrupts\_BothEdges, [102](#)
  - GPIO\_ConfigInterrupts\_Edge, [101](#)
  - GPIO\_ConfigInterrupts\_LevelTrig, [102](#)
  - GPIO\_ConfigNVIC, [102](#)
  - GPIO\_ConfigPortCtrl, [104](#)
  - GPIO\_ConfigPullDown, [100](#)
  - GPIO\_ConfigPullUp, [100](#)
  - GPIO\_DisableDigital, [101](#)
  - GPIO\_EnableDigital, [101](#)
  - GPIO\_InitPort, [99](#)
  - GPIO\_isPortInit, [99](#)
  - GPIO\_PTR\_ARR, [105](#)
  - GPIO\_ReadPins, [103](#)
  - GPIO\_Toggle, [103](#)
  - GPIO\_WriteHigh, [103](#)
  - GPIO\_WriteLow, [103](#)
- GPIO.h, [105](#)
  - GPIO\_ConfigAltMode, [112](#)
  - GPIO\_ConfigAnalog, [112](#)
  - GPIO\_ConfigDirInput, [108](#)
  - GPIO\_ConfigDirOutput, [108](#)
  - GPIO\_ConfigDriveStrength, [109](#)
  - GPIO\_ConfigInterrupts\_BothEdges, [110](#)
  - GPIO\_ConfigInterrupts\_Edge, [109](#)
  - GPIO\_ConfigInterrupts\_LevelTrig, [110](#)
  - GPIO\_ConfigNVIC, [110](#)
  - GPIO\_ConfigPortCtrl, [112](#)
  - GPIO\_ConfigPullDown, [108](#)
- GPIO\_ConfigPullUp, [108](#)
  - GPIO.c, [100](#)
  - GPIO.h, [108](#)
- GPIO\_DisableDigital
  - GPIO.c, [101](#)
  - GPIO.h, [109](#)
- GPIO\_EnableDigital
  - GPIO.c, [101](#)
  - GPIO.h, [109](#)
- GPIO\_InitPort
  - GPIO.c, [99](#)
  - GPIO.h, [107](#)
- GPIO\_isPortInit
  - GPIO.c, [99](#)
  - GPIO.h, [107](#)
- GPIO\_Port\_t, [76](#)
  - GPIO\_ConfigPullUp, [108](#)
  - GPIO\_DisableDigital, [109](#)
  - GPIO\_EnableDigital, [109](#)
  - GPIO\_InitPort, [107](#)
  - GPIO\_isPortInit, [107](#)
  - GPIO\_ReadPins, [111](#)
  - GPIO\_Toggle, [111](#)
  - GPIO\_WriteHigh, [111](#)
  - GPIO\_WriteLow, [111](#)

GPIO\_PTR\_ARR  
  GPIO.c, [105](#)  
GPIO\_ReadPins  
  GPIO.c, [103](#)  
  GPIO.h, [111](#)  
GPIO\_Toggle  
  GPIO.c, [103](#)  
  GPIO.h, [111](#)  
GPIO\_WriteHigh  
  GPIO.c, [103](#)  
  GPIO.h, [111](#)  
GPIO\_WriteLow  
  GPIO.c, [103](#)  
  GPIO.h, [111](#)  
  
IDMOFF  
  ILI9341, [32](#)  
IDMON  
  ILI9341, [32](#)  
IFCTL  
  ILI9341, [32](#)  
ILI9341, [29](#)  
  CASET, [32](#)  
  Cmd\_t, [32](#)  
  DINVOFF, [32](#)  
  DINVON, [32](#)  
  DISPOFF, [32](#)  
  DISPON, [32](#)  
  FRMCTR1, [32](#)  
  FRMCTR2, [32](#)  
  FRMCTR3, [32](#)  
  IDMOFF, [32](#)  
  IDMON, [32](#)  
  IFCTL, [32](#)  
  ILI9341\_Init, [33](#)  
  ILI9341\_NUM\_COLS, [31](#)  
  ILI9341\_NUM\_ROWS, [31](#)  
  ILI9341\_resetHard, [34](#)  
  ILI9341\_resetSoft, [34](#)  
  ILI9341\_sendParams, [33](#)  
  ILI9341\_setAddress, [32](#)  
  ILI9341\_setColAddress, [38](#)  
  ILI9341\_setColorDepth, [37](#)  
  ILI9341\_setColorExpression, [35](#)  
  ILI9341\_setDisplInversion, [36](#)  
  ILI9341\_setDisplayArea, [35](#)  
  ILI9341\_setDispOutput, [36](#)  
  ILI9341 setFrameRate, [38](#)  
  ILI9341\_setInterface, [33](#)  
  ILI9341\_setMemAccessCtrl, [37](#)  
  ILI9341\_setMode, [32](#)  
  ILI9341\_setPartialArea, [36](#)  
  ILI9341\_setRowAddress, [38](#)  
  ILI9341\_setSleepMode, [35](#)  
  ILI9341\_writeMemCmd, [39](#)  
  ILI9341\_writePixel, [39](#)  
  MADCTL, [32](#)  
  NOP, [32](#)  
  NORON, [32](#)  
  
  PASET, [32](#)  
  PIXSET, [32](#)  
  PLTAR, [32](#)  
  PRCTR, [32](#)  
  PTLON, [32](#)  
  RAMWR, [32](#)  
  SPLIN, [32](#)  
  SPLOUT, [32](#)  
  SWRESET, [32](#)  
  VSCRDEF, [32](#)  
  VSCRSAADD, [32](#)  
ILI9341.c, [131](#)  
ILI9341.h, [132](#)  
ILI9341\_Init  
  ILI9341, [33](#)  
ILI9341\_NUM\_COLS  
  ILI9341, [31](#)  
ILI9341\_NUM\_ROWS  
  ILI9341, [31](#)  
ILI9341\_resetHard  
  ILI9341, [34](#)  
ILI9341\_resetSoft  
  ILI9341, [34](#)  
ILI9341\_sendParams  
  ILI9341, [33](#)  
ILI9341\_setAddress  
  ILI9341, [32](#)  
ILI9341\_setColAddress  
  ILI9341, [38](#)  
ILI9341\_setColorDepth  
  ILI9341, [37](#)  
ILI9341\_setColorExpression  
  ILI9341, [35](#)  
ILI9341\_setDisplInversion  
  ILI9341, [36](#)  
ILI9341\_setDisplayArea  
  ILI9341, [35](#)  
ILI9341\_setDispOutput  
  ILI9341, [36](#)  
ILI9341 setFrameRate  
  ILI9341, [38](#)  
ILI9341\_setInterface  
  ILI9341, [33](#)  
ILI9341\_setMemAccessCtrl  
  ILI9341, [37](#)  
ILI9341\_setMode  
  ILI9341, [32](#)  
ILI9341\_setPartialArea  
  ILI9341, [36](#)  
ILI9341\_setRowAddress  
  ILI9341, [38](#)  
ILI9341\_setSleepMode  
  ILI9341, [35](#)  
ILI9341\_writeMemCmd  
  ILI9341, [39](#)  
ILI9341\_writePixel  
  ILI9341, [39](#)  
Interrupt Service Routines, [24](#)

- ISR\_addToIntTable, 26
- ISR\_Disable, 28
- ISR\_Enable, 27
- ISR\_GlobalDisable, 25
- ISR\_GlobalEnable, 25
- ISR\_InitNewTableInRam, 26
- ISR\_setPriority, 27
- ISR\_triggerInterrupt, 28
- ISR.c, 113
- ISR.h, 114
- ISR\_addToIntTable
  - Interrupt Service Routines, 26
- ISR\_Disable
  - Interrupt Service Routines, 28
- ISR\_Enable
  - Interrupt Service Routines, 27
- ISR\_GlobalDisable
  - Interrupt Service Routines, 25
- ISR\_GlobalEnable
  - Interrupt Service Routines, 25
- ISR\_InitNewTableInRam
  - Interrupt Service Routines, 26
- ISR\_setPriority
  - Interrupt Service Routines, 27
- ISR\_triggerInterrupt
  - Interrupt Service Routines, 28
- LCD, 52
  - LCD\_Draw, 55
  - LCD\_drawHoriLine, 56
  - LCD\_drawLine, 54
  - LCD\_drawRectangle, 57
  - LCD\_drawVertLine, 57
  - LCD\_Fill, 56
  - LCD\_Init, 54
  - LCD\_plotSample, 58
  - LCD\_setColor, 55
  - LCD\_setOutputMode, 54
  - LCD\_setX, 54
  - LCD\_setY, 55
- LCD.c, 81
- LCD.h, 83
- LCD\_BUFFER\_SIZE
  - Main, 72
- LCD\_Draw
  - LCD, 55
- LCD\_drawHoriLine
  - LCD, 56
- LCD\_drawLine
  - LCD, 54
- LCD\_drawRectangle
  - LCD, 57
- LCD\_drawVertLine
  - LCD, 57
- LCD\_FIFO\_CAPACITY
  - Main, 72
- LCD\_Fill
  - LCD, 56
- LCD\_Handler
  - Main, 73
- LCD\_Init
  - LCD, 54
- LCD\_plotSample
  - LCD, 58
- LCD\_setColor
  - LCD, 55
- LCD\_setOutputMode
  - LCD, 54
- LCD\_setX
  - LCD, 54
- LCD\_setY
  - LCD, 55
- LCD\_TOP\_LINE
  - Main, 72
- LCD\_WAVE\_NUM\_Y
  - Main, 72
- LED, 40
  - Led\_GetPin, 42
  - Led\_GetPort, 41
  - Led\_Init, 41
  - Led\_isOn, 42
  - Led\_Toggle, 43
  - Led\_TurnOff, 43
  - Led\_TurnOn, 42
- Led.c, 135
- Led.h, 136
- Led\_GetPin
  - LED, 42
- Led\_GetPort
  - LED, 41
- Led\_Init
  - LED, 41
- Led\_isOn
  - LED, 42
- Led\_t, 76
- Led\_Toggle
  - LED, 43
- Led\_TurnOff
  - LED, 43
- Led\_TurnOn
  - LED, 42
- lookup.c, 84
- lookup.h, 85
- Lookup\_GetPtr
  - Data Acquisition (DAQ), 48
- MADCTL
  - ILI9341, 32
- Main, 71
  - DAQ\_BUFFER\_SIZE, 72
  - DAQ\_FIFO\_CAPACITY, 72
  - DAQ\_Handler, 73
  - LCD\_BUFFER\_SIZE, 72
  - LCD\_FIFO\_CAPACITY, 72
  - LCD\_Handler, 73
  - LCD\_TOP\_LINE, 72
  - LCD\_WAVE\_NUM\_Y, 72
  - main, 74

- Processing\_Handler, 73
- main
  - Main, 74
- main.c, 128
- Middleware, 29
- NewAssert, 71
- NewAssert.c, 93
- NewAssert.h, 94
- NOP
  - ILI9341, 32
- NORON
  - ILI9341, 32
- ONESHOT
  - Timer, 14
- PASET
  - ILI9341, 32
- PERIODIC
  - Timer, 14
- Phase-Locked Loop (PLL), 7
  - PLL\_Init, 8
- PIXSET
  - ILI9341, 32
- PLL.c, 116
- PLL.h, 116
- PLL\_Init
  - Phase-Locked Loop (PLL), 8
- PLTAR
  - ILI9341, 32
- PRCTR
  - ILI9341, 32
- Processing\_Handler
  - Main, 73
- PTLON
  - ILI9341, 32
- QRS, 58
  - COEFF\_BANDPASS, 63
  - COEFF\_DERFILT, 64
  - COEFF\_MOVAVG, 64
  - QRS\_applyDecisionRules, 62
  - QRS\_findFiducialMarks, 60
  - QRS\_Init, 61
  - QRS\_initLevels, 60
  - QRS\_Preprocess, 62
  - QRS\_runDetection, 63
  - QRS\_updateLevel, 61
  - QRS\_updateThreshold, 61
- QRS.c, 87
- QRS.h, 89
- QRS\_applyDecisionRules
  - QRS, 62
- QRS\_findFiducialMarks
  - QRS, 60
- QRS\_Init
  - QRS, 61
- QRS\_initLevels
  - QRS, 60
- QRS\_Preprocess
  - QRS, 62
- QRS\_runDetection
  - QRS, 63
- QRS\_updateLevel
  - QRS, 61
- QRS\_updateThreshold
  - QRS, 61
- RAMWR
  - ILI9341, 32
- Serial Peripheral Interface (SPI), 8
  - SPI\_Init, 9
  - SPI\_Read, 10
  - SPI\_SET\_DC, 9
  - SPI\_WriteCmd, 10
  - SPI\_WriteData, 10
- SPI.c, 117
- SPI.h, 118
- SPI\_Init
  - Serial Peripheral Interface (SPI), 9
- SPI\_Read
  - Serial Peripheral Interface (SPI), 10
- SPI\_SET\_DC
  - Serial Peripheral Interface (SPI), 9
- SPI\_WriteCmd
  - Serial Peripheral Interface (SPI), 10
- SPI\_WriteData
  - Serial Peripheral Interface (SPI), 10
- SPLIN
  - ILI9341, 32
- SPLOUT
  - ILI9341, 32
- SWRESET
  - ILI9341, 32
- System Tick (SysTick), 11
  - SysTick\_Interrupt\_Init, 12
- SysTick.c, 119
- SysTick.h, 120
- SysTick\_Interrupt\_Init
  - System Tick (SysTick), 12
- test\_adc.c, 137
- test\_daq.c, 138
- test\_debug.c, 139
- test\_fifo.c, 140
- test\_lcd\_image.c, 142
- test\_pll.c, 143
- test\_qrs.c, 144
- test\_spi.c, 145
- test\_systick\_int.c, 146
- test\_timer1\_int.c, 147
- test\_uart\_la.c, 148
- test\_uart\_write.c, 149
- test\_userctrl.c, 150
- Timer, 12
  - DOWN, 14

- ONESHOT, [14](#)
- PERIODIC, [14](#)
- Timer\_clearInterruptFlag, [17](#)
- Timer\_disableAdcTrigger, [16](#)
- Timer\_disableInterruptOnTimeout, [17](#)
- Timer\_enableAdcTrigger, [15](#)
- Timer\_enableInterruptOnTimeout, [16](#)
- Timer\_getName, [14](#)
- Timer\_Init, [14](#)
- Timer\_isCounting, [19](#)
- Timer\_isInit, [15](#)
- TIMER\_POOL, [20](#)
- Timer\_setInterval\_ms, [18](#)
- Timer\_setMode, [15](#)
- Timer\_Start, [18](#)
- Timer\_Stop, [19](#)
- Timer\_Wait1ms, [19](#)
- timerDirection\_t, [14](#)
- timerMode\_t, [14](#)
- UP, [14](#)
- Timer.c, [121](#)
- Timer.h, [123](#)
- Timer\_clearInterruptFlag
  - Timer, [17](#)
- Timer\_disableAdcTrigger
  - Timer, [16](#)
- Timer\_disableInterruptOnTimeout
  - Timer, [17](#)
- Timer\_enableAdcTrigger
  - Timer, [15](#)
- Timer\_enableInterruptOnTimeout
  - Timer, [16](#)
- Timer\_getName
  - Timer, [14](#)
- Timer\_Init
  - Timer, [14](#)
- Timer\_isCounting
  - Timer, [19](#)
- Timer\_isInit
  - Timer, [15](#)
- TIMER\_POOL
  - Timer, [20](#)
- Timer\_setInterval\_ms
  - Timer, [18](#)
- Timer\_setMode
  - Timer, [15](#)
- Timer\_Start
  - Timer, [18](#)
- Timer\_Stop
  - Timer, [19](#)
- Timer\_t, [77](#)
- Timer\_Wait1ms
  - Timer, [19](#)
- timerDirection\_t
  - Timer, [14](#)
- timerMode\_t
  - Timer, [14](#)
- UART.c, [124](#)
- UART.h, [126](#)
- UART\_ARR
  - Universal Asynchronous Receiver/Transmitter (UART), [24](#)
- UART\_Init
  - Universal Asynchronous Receiver/Transmitter (UART), [22](#)
- UART\_ReadChar
  - Universal Asynchronous Receiver/Transmitter (UART), [22](#)
- UART\_t, [77](#)
- UART\_WriteChar
  - Universal Asynchronous Receiver/Transmitter (UART), [22](#)
- UART\_WriteFloat
  - Universal Asynchronous Receiver/Transmitter (UART), [23](#)
- UART\_WriteInt
  - Universal Asynchronous Receiver/Transmitter (UART), [23](#)
- UART\_WriteStr
  - Universal Asynchronous Receiver/Transmitter (UART), [23](#)
- Universal Asynchronous Receiver/Transmitter (UART), [20](#)
  - UART\_ARR, [24](#)
  - UART\_Init, [22](#)
  - UART\_ReadChar, [22](#)
  - UART\_WriteChar, [22](#)
  - UART\_WriteFloat, [23](#)
  - UART\_WriteInt, [23](#)
  - UART\_WriteStr, [23](#)
- UP
  - Timer, [14](#)
- VSCRDEF
  - ILI9341, [32](#)
- VSCRSADD
  - ILI9341, [32](#)