# ECG-HRM

Generated by Doxygen 1.9.8

# 1 Topic Index

## 1.1 Topics

Here is a list of all topics with brief descriptions:

# 2 Data Structure Index

## 2.1 Data Structures

Here are the data structures with brief descriptions:



# 3 File Index

## 3.1 File List

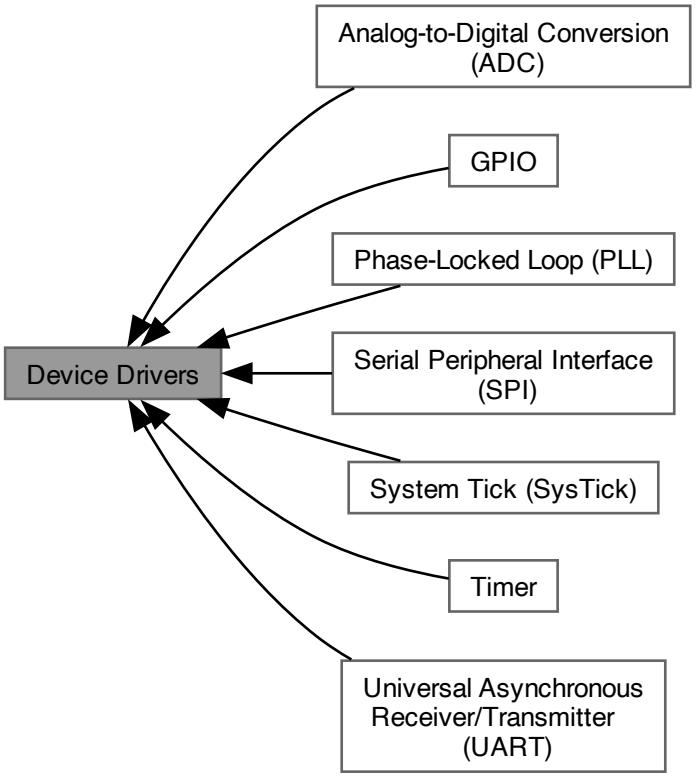
Here is a list of all documented files with brief descriptions:

# 4 Topic Documentation

## 4.1 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)

### 4.1.1 Detailed Description

Low level device driver modules.

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

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

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


#### Files

- file ADC.c

  *Source code for ADC module.*
- file ADC.h

  *Driver module for analog-to-digital conversion (ADC).*

#### Functions

- void **ADC_Init** (void)

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

  *Enable the ADC interrupt.*
- void **ADC_InterruptDisable** (void)

  *Disable the ADC interrupt.*
- float32_t ADC_ConvertToVolts (uint16_t raw_sample)

  *Convert a raw ADC sample to voltage in [mV].*

#### 4.1.2.1 Detailed Description

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

#### 4.1.2.2 Function Documentation

#### ADC_ConvertToVolts()

```
float32_t ADC_ConvertToVolts (
            uint16_t raw_sample )
```

Convert a raw ADC sample to voltage in [mV].

**Parameters**

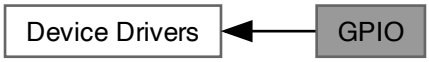
| | |
|---|---|
| *raw_sample* | 12-bit unsigned ADC value. `sample = [0, 0xFFF]` |

**Returns**

double Voltage value in range `[-5.5, 5.5) [mV]`.

### 4.1.3 GPIO

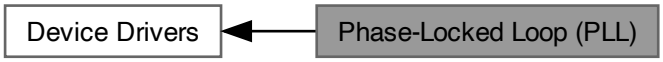
Collaboration diagram for GPIO:




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

### 4.1.4 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)

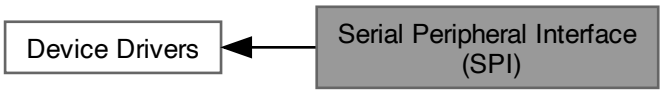
  *Initializes the phase-locked-loop (PLL), allowing a bus frequency of 80[MHz].*

#### 4.1.4.1 Detailed Description

Function for initializing the phase-locked loop.

### 4.1.5 Serial Peripheral Interface (SPI)

Collaboration diagram for Serial Peripheral Interface (SPI):




#### Files

- file SPI.c

  *Source code for SPI module.*
- file SPI.h

  *Driver module for using the serial peripheral interface (SPI) protocol.*

#### Macros

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

#### Enumerations

- enum {
  **SPI_CLK_PIN** = GPIO_PIN2 , **SPI_CS_PIN** = GPIO_PIN3 , **SPI_RX_PIN** = GPIO_PIN4 , **SPI_TX_PIN** = GPIO_PIN5 ,
  **SPI_DC_PIN** = GPIO_PIN6 , **SPI_RESET_PIN** = GPIO_PIN7 , **SPI_SSI0_PINS** = (SPI_CLK_PIN | SPI_↩ CS_PIN | SPI_RX_PIN | SPI_TX_PIN) , **SPI_GPIO_PINS** = (SPI_DC_PIN | SPI_RESET_PIN) ,
  **SPI_ALL_PINS** = (SPI_SSI0_PINS | SPI_GPIO_PINS) }

#### Functions

- void SPI_Init (void)

  *Initialize SSI0 to act as an SPI Controller (AKA Master) in mode 0.*
- uint8_t SPI_Read (void)

  *Read data from the peripheral.*
- void SPI_WriteCmd (uint8_t cmd)

  *Write an 8-bit command to the peripheral.*
- void SPI_WriteData (uint8_t data)

  *Write 8-bit data to the peripheral.*

#### 4.1.5.1 Detailed Description

Functions for SPI-based communication via SSI0 peripheral.

#### 4.1.5.2 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)

#### 4.1.5.3 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 peripheral.

**Returns**

uint8_t

#### SPI_WriteCmd()

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

Write an 8-bit command to the peripheral.

**Parameters**

| | |
|---|---|
| *cmd* | command for peripheral |

#### SPI_WriteData()

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

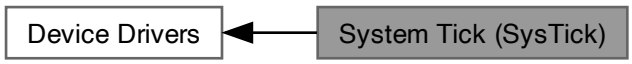
Write 8-bit data to the peripheral.

**Parameters**

| | |
|---|---|
| *data* | input data for peripheral |

### 4.1.6 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.*

#### 4.1.6.1 Detailed Description

Functions for timing and periodic interrupts via SysTick.

#### 4.1.6.2 Function Documentation

**SysTick_Interrupt_Init()**

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

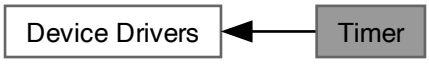
Initialize SysTick for interrupts.

**Parameters**

| | |
|---|---|
| *time_ms* | Time in [ms] between interrupts. Cannot be more than 200[ms]. |

### 4.1.7 Timer

Collaboration diagram for Timer:




**Files**

- file Timer.c

  *Implementation for timer module.*
- file Timer.h

  *Driver module for general-purpose timer modules.*

**Timer0A**

- void **Timer0A_Init** (void)

  *Initialize timer 0 as 32-bit, one-shot, countdown timer.*
- void Timer0A_Start (uint32_t time_ms)

  *Count down starting from the inputted value.*
- uint8_t Timer0A_isCounting (void)

  *Returns 1 if Timer0 is still counting and 0 if not.*
- void Timer0A_Wait1ms (uint32_t time_ms)

  *Wait for the specified amount of time in [ms].*

**Timer1A**

- void Timer1A_Init (uint32_t time_ms)

  *Initialize timer 1 as a 32-bit, periodic, countdown timer with interrupts.*

**Timer2A**

- void **Timer2A_Init** (void)

  *Initialize timer 2 as 32-bit, one-shot, countdown timer.*
- void Timer2A_Start (uint32_t time_ms)

  *Count down starting from the inputted value.*
- uint8_t Timer2A_isCounting (void)

  *Returns 1 if Timer2 is still counting and 0 if not.*
- void Timer2A_Wait1ms (uint32_t time_ms)

  *Wait for the specified amount of time in [ms].*
- void Timer3A_Init (uint32_t time_ms)

  *Initialize Timer3A as a 32-bit, periodic, countdown timer that triggers ADC sample capture.*

#### 4.1.7.1 Detailed Description

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

#### 4.1.7.2 Function Documentation

**Timer0A_isCounting()**

```
uint8_t Timer0A_isCounting (
            void  )
```

Returns 1 if Timer0 is still counting and 0 if not.

**Returns**

uint8_t status

**Timer0A_Start()**

```
void Timer0A_Start (
            uint32_t time_ms )
```

Count down starting from the inputted value.

**Parameters**

| | |
|---|---|
| *time_ms* | Time in [ms] to load into Timer 0. Must be <= 53 seconds. |

### Timer0A_Wait1ms()

```
void Timer0A_Wait1ms (
            uint32_t time_ms )
```

Wait for the specified amount of time in [ms].

**Parameters**

| | |
|---|---|
| *time_ms* | Time in [ms] to load into Timer 0. Must be <= 53 seconds. |

### Timer1A_Init()

```
void Timer1A_Init (
            uint32_t time_ms )
```

Initialize timer 1 as a 32-bit, periodic, countdown timer with interrupts.

**Parameters**

| | |
|---|---|
| *time_ms* | Time in [ms] between interrupts. Must be <= 53 seconds. |

### Timer2A_isCounting()

```
uint8_t Timer2A_isCounting (
            void  )
```

Returns 1 if Timer2 is still counting and 0 if not.

**Returns**

uint8_t status

### Timer2A_Start()

```
void Timer2A_Start (
            uint32_t time_ms )
```

Count down starting from the inputted value.

**Parameters**

| | |
|---|---|
| *time_ms* | Time in [ms] to load into Timer 2. Must be <= 53 seconds. |

#### Timer2A_Wait1ms()

```
void Timer2A_Wait1ms (
            uint32_t time_ms )
```

Wait for the specified amount of time in [ms].

**Parameters**

| | |
|---|---|
| *time_ms* | Time in [ms] to load into Timer 2. Must be <= 53 seconds. |

#### Timer3A_Init()

```
void Timer3A_Init (
            uint32_t time_ms )
```

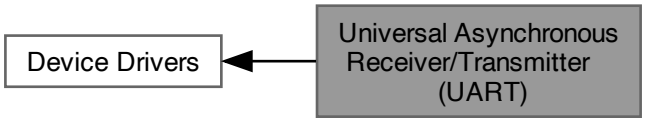
Initialize Timer3A as a 32-bit, periodic, countdown timer that triggers ADC sample capture.

**Parameters**

| | |
|---|---|
| *time_ms* | Time in [ms] to load into Timer3A. Must be <= 53 seconds. |

### 4.1.8 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) 0xFC8 }
- 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.*

#### 4.1.8.1 Detailed Description

Functions for UART-based communication.

#### 4.1.8.2 Function Documentation

### UART_Init()

```
UART_t * UART_Init (
            GPIO_Port_t * port,
            UART_Num_t uartNum )
```

Initialize the specified UART peripheral.

**Parameters**

| | | |
|---|---|---|
| in | *port* | GPIO port to use. |
| in | *uartNum* | UART number. Should be either one of the enumerated constants or an int in range [0, 7]. |
| out | *UART↩ _t** | (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 | *uart* | UART to read from. |
| out | *unsigned* | 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 | *uart* | UART to read from. |
| in | *input_char* | ASCII character to send. |

### UART_WriteFloat()

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

Write a floating-point number the UART.

**Parameters**

| | | |
|---|---|---|
| in | *uart* | UART to read from. |
| in | *n* | Floating-point number to be converted and transmitted. |
| in | *num_decimals* | Number of digits after the decimal point to include. |

### UART_WriteInt()

```
void UART_WriteInt (
            UART_t * uart,
            int32_t n )
```

Write a 32-bit unsigned integer the UART.

**Parameters**

| | | |
|---|---|---|
| in | *uart* | UART to read from. |
| in | *n* | Unsigned 32-bit `int` to be converted and transmitted. |

### UART_WriteStr()

```
void UART_WriteStr (
            UART_t * uart,
            void * input_str )
```

Write a C string to the UART.

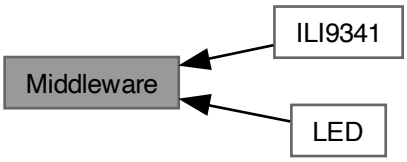
**Parameters**

| | | |
|---|---|---|
| in | *uart* | UART to read from. |
| in | *input_str* | Array of ASCII characters. |

## 4.2 Middleware

High-level device driver modules.

Collaboration diagram for Middleware:




**Modules**

- ILI9341
- LED

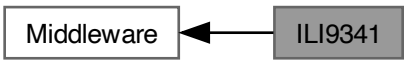
#### 4.2.1 Detailed Description

High-level device driver modules.

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

#### 4.2.2 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.*

**Macros**

- #define **NUM_COLS** (uint16_t) 240
- #define **NUM_ROWS** (uint16_t) 320

### Enumerations

- enum Cmd_t {
  **NOP** = 0x00 , SWRESET = 0x01 , SPLIN = 0x10 , SPLOUT = 0x11 ,
  PTLON = 0x12 , NORON = 0x13 , DINVOFF = 0x20 , DINVON = 0x21 ,
  CASET = 0x2A , PASET = 0x2B , RAMWR = 0x2C , DISPOFF = 0x28 ,
  DISPON = 0x29 , PLTAR = 0x30 , VSCRDEF = 0x33 , MADCTL = 0x36 ,
  VSCRSADD = 0x37 , IDMOFF = 0x38 , IDMON = 0x39 , PIXSET = 0x3A ,
  FRMCTR1 = 0xB1 , FRMCTR2 = 0xB2 , FRMCTR3 = 0xB3 , PRCTR = 0xB5 ,
  IFCTL = 0xF6 }

### Functions

- void **ILI9341_Init** (void)

  *Initialize the LCD driver, the SPI module, and Timer2A.*
- void ILI9341_resetHard (void)

  *Perform a hardware reset of the LCD driver.*
- void ILI9341_resetSoft (void)

  *Perform a software reset of the LCD driver.*
- void ILI9341_setSleepMode (bool isSleeping)

  *Enter or exit sleep mode. The LCD driver is in sleep mode by default upon powering on or either kind of reset.*
- void ILI9341_setDispMode (bool isNormal, bool isFullColors)

  *Set the display area and color expression.*
- void ILI9341_setPartialArea (uint16_t rowStart, uint16_t rowEnd)

  *Set the partial display area for partial mode. Call before activating partial mode via ILI9341_setDisplayMode().*
- void ILI9341_setDispInversion (bool is_ON)

  *Toggle display inversion. Turning `ON` causes colors to be inverted on the display.*
- void ILI9341_setDispOutput (bool is_ON)

  *Turn display output ON or OFF. This function clears the display and stops outputting to the display area, but does not affect frame memory or power.*
- void ILI9341_setScrollArea (uint16_t topFixedArea, uint16_t vertScrollArea, uint16_t bottFixedArea)

  *Set the vertical scrolling area of the display. The sum of the three parameters should be equal to the max number of rows `NUM_ROWS = 320`.*
- void ILI9341_setScrollStart (uint16_t startRow)

  *Set the start row for vertical scrolling.*
- 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 (bool is_16bit)

  *Set the pixel format to be 16-bit (65K colors) or 18-bit (262K colors).*
- void **ILI9341_NoOpCmd** (void)

  *Send the "No Operation" command (`NOP = 0x00`) to the LCD driver. Can be used to terminate the "Memory Write" (`RAMWR`) and "Memory Read" (`RAMRD`) commands, but does nothing otherwise.*
- void ILI9341_setFrameRateNorm (uint8_t divisionRatio, uint8_t clocksPerLine)

  *TODO: Write brief.*
- void ILI9341_setFrameRateIdle (uint8_t divisionRatio, uint8_t clocksPerLine)

  *TODO: Write brief.*
- 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.*
- void ILI9341_setRowAddress (uint16_t startRow, uint16_t endRow)

  *not using backlight, so these aren't necessary*

- void ILI9341_setColAddress (uint16_t startCol, uint16_t endCol)

  *Sets the start/end rows to be written to.*
- void ILI9341_writeMemCmd (void)

  *Sends the "Write Memory" (`RAMWR`) command to the LCD driver, signalling that incoming data should be written to memory.*
- void ILI9341_writePixel (uint8_t red, uint8_t green, uint8_t blue, bool is_16bit)

  *Write a single pixel to frame memory.*
- void **ILI9341_setBlankingPorch** (uint8_t vpf, uint8_t vbp, uint8_t hfp, uint8_t hbp)

  *TODO: Write.*

#### 4.2.2.1 Detailed Description

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

#### 4.2.2.2 Enumeration Type Documentation

**Cmd_t**

`enum Cmd_t`

**Enumerator**

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

#### 4.2.2.3 Function Documentation

**ILI9341_resetHard()**

```
void ILI9341_resetHard (
            void  )
```

Perform a hardware reset of the LCD driver.

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

**ILI9341_resetSoft()**

```
void ILI9341_resetSoft (
            void  )
```

Perform a software reset of the LCD driver.

the driver needs 5 [ms] before another command

**ILI9341_setColAddress()**

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

Sets the start/end rows to be written to.

```
Should be called along with `ILI9341_setRowAddress()` and
before `ILI9341_writeMemCmd()`.
```

**Parameters**

| | |
|---|---|
| *startCol* | 0 <= `startCol` <= `endCol` |
| *endCol* | `startCol` <= `endCol` < 240 |

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_setColorDepth()**

```
void ILI9341_setColorDepth (
            bool is_16bit )
```

Set the pixel format to be 16-bit (65K colors) or 18-bit (262K colors).

**Parameters**

| | |
|---|---|
| *is_16bit* | |

16-bit requires 2 transfers and allows for 65K colors. 18-bit requires 3 transfers and allows for 262K colors.

### ILI9341_setDispInversion()

```
void ILI9341_setDispInversion (
            bool is_ON )
```

Toggle display inversion. Turning `ON` causes colors to be inverted on the display.

**Parameters**

| | |
|---|---|
| *is_ON* | `true` to turn ON, `false` to turn OFF |

TODO: Write description

### ILI9341_setDispMode()

```
void ILI9341_setDispMode (
            bool isNormal,
            bool isFullColors )
```

Set the display area and color expression.

```
        Normal mode is the default and allows output to the full
        display area. Partial mode should be activated after calling
        `ILI9341_setPartialArea()`.

        Setting `isFullColors` to `false` restricts the color expression
        to 8 colors, determined by the MSB of the R/G/B values.
```

**Parameters**

| | |
|---|---|
| *isNormal* | `true` for normal mode, `false` for partial mode |
| *isFullColors* | `true` for full colors, `false` for 8 colors |

### ILI9341_setDispOutput()

```
void ILI9341_setDispOutput (
            bool is_ON )
```

Turn display output ON or OFF. This function clears the display and stops outputting to the display area, but does not affect frame memory or power.

**Parameters**

| | |
|---|---|
| *is_ON* | `true` to turn ON, `false` to turn OFF |

TODO: Write description

### ILI9341_setFrameRateIdle()

```
void ILI9341_setFrameRateIdle (
             uint8_t divisionRatio,
             uint8_t clocksPerLine )
```

TODO: Write brief.

TODO: Write description

### ILI9341_setFrameRateNorm()

```
void ILI9341_setFrameRateNorm (
             uint8_t divisionRatio,
             uint8_t clocksPerLine )
```

TODO: Write brief.

TODO: Write description

### 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_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 | *areRowsFlipped* | |
| in | *areColsFlipped* | |
| in | *areRowsAndColsSwitched* | |
| in | *isVertRefreshFlipped* | |
| in | *isColorOrderFlipped* | |
| in | *isHorRefreshFlipped* | |

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_setPartialArea()

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

Set the partial display area for partial mode. Call before activating partial mode via ILI9341_setDisplayMode().

**Parameters**

| | |
|---|---|
| *rowStart* | |
| *rowEnd* | |

### ILI9341_setRowAddress()

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

not using backlight, so these aren't necessary

Sets the start/end rows to be written to.

```
        Should be called along with `ILI9341_setColAddress()` and
        before `ILI9341_writeMemCmd()`.
```

**Parameters**

| | |
|---|---|
| *startRow* | 0 <= `startRow` <= `endRow` |
| *endRow* | `startRow` <= `endRow` < 320 |

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_setScrollArea()

```
void ILI9341_setScrollArea (
             uint16_t topFixedArea,
             uint16_t vertScrollArea,
             uint16_t bottFixedArea )
```

Set the vertical scrolling area of the display. The sum of the three parameters should be equal to the max number of rows `NUM_ROWS = 320`.

**Parameters**

| | |
|---|---|
| *topFixedArea* | Number of rows fixed at the top of the screen. |
| *vertScrollArea* | Number of rows that scroll. |
| *bottFixedArea* | Number of rows fixed at the bottom of the screen. |

### ILI9341_setScrollStart()

```
void ILI9341_setScrollStart (
             uint16_t startRow )
```

Set the start row for vertical scrolling.

**Parameters**

| | |
|---|---|
| *startRow* | Start row for scrolling. Should be >= `topFixedArea - 1` |

### ILI9341_setSleepMode()

```
void ILI9341_setSleepMode (
            bool isSleeping )
```

Enter or exit sleep mode. The LCD driver is in sleep mode by default upon powering on or either kind of reset.

**Parameters**

| | |
|---|---|
| *isSleeping* | `true` to enter sleep mode, `false` to exit |

This function turns sleep mode ON or OFF depending on the value of `is_sleeping`. Either way, the MCU must wait >= 5 [ms] before sending further commands.

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_writeMemCmd()

```
void ILI9341_writeMemCmd (
            void  )
```

Sends the "Write Memory" (`RAMWR`) command to the LCD driver, signalling that incoming data should be written to memory.

Should be called after setting the row (`ILI9341_setRowAddress()`) and/or and/or column (`ILI9341_setRowAddress()`) addresses, but before writing image data (`ILI9341_writePixel()`).

### ILI9341_writePixel()

```
void ILI9341_writePixel (
            uint8_t red,
            uint8_t green,
            uint8_t blue,
            bool is_16bit )
```

Write a single pixel to frame memory.

```
Call `ILI9341_writeMemCmd()` before this one.
```

**Parameters**

| | |
|---|---|
| *red* | 5 or 6-bit `R` value |
| *green* | 5 or 6-bit `G` value |
| *blue* | 5 or 6-bit `B` value |
| *is_16bit* | `true` for 16-bit (65K colors, 2 transfers) color depth, `false` for 18-bit (262K colors, 3 transfer) color depth NOTE: set color depth via `ILI9341_setColorDepth()` |

This function sends one pixel to the display. Because the serial interface (SPI) is used, each pixel requires 2

transfers in 16-bit mode and 3 transfers in 18-bit mode.

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

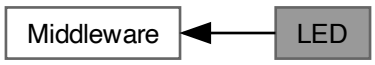
| Transfer | 1 | | | | | | | | 2 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit # | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Value | R4 | R3 | R2 | R1 | R0 | G5 | G4 | G3 | G2 | G1 | G0 | B4 | B3 | B2 | B1 | B0 |

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

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

### 4.2.3 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** 3

### Functions

- Led_t * Led_Init (GPIO_Port_t *gpioPort, GPIO_Pin_t pin)

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

#### 4.2.3.1 Detailed Description

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

#### 4.2.3.2 Function Documentation

##### Led_GetPin()

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

Get the GPIO pin associated with the LED.

**Parameters**

| | | |
|---|---|---|
| in | *led* | Pointer to LED data structure. |
| out | *GPIO_Pin_t* | GPIO pin associated with the LED. |

##### Led_GetPort()

```
GPIO_Port_t * Led_GetPort (
            Led_t * led )
```

Get the GPIO port associated with the LED.

**Parameters**

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

### 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 | *gpioPort* | Pointer to a `struct` representing a GPIO port. |
| in | *pin* | GPIO pin to use. |
| out | *Led_t*∗ | Pointer to LED data structure. |

### Led_isOn()

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

Check the LED's status.

**Parameters**

| | | |
|---|---|---|
| in | *led* | Pointer to LED data structure. |
| out | *true* | the LED is `ON`. |
| out | *false* | the LED is `OFF`. |

### Led_Toggle()

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

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

**Parameters**

| | | |
|---|---|---|
| in | *led* | Pointer to LED data structure. |

### Led_TurnOff()

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

Turn the LED `OFF`.

**Parameters**

| | | |
|---|---|---|
| in | *led* | Pointer to LED data structure. |

#### Led_TurnOn()

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

Turn the LED ON.

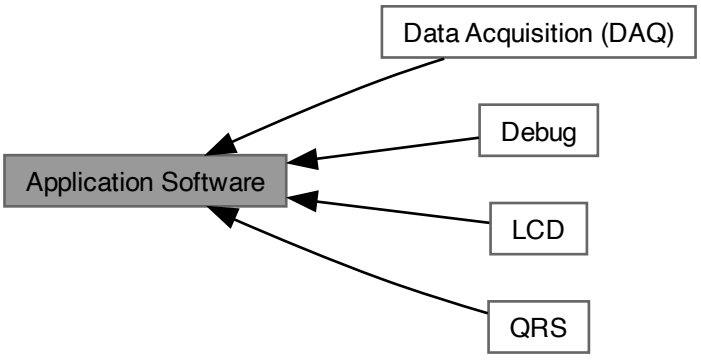
**Parameters**

| | | |
|---|---|---|
| in | *led* | Pointer to LED data structure. |

## 4.3 Application Software

Application-specific software modules.

Collaboration diagram for Application Software:




**Modules**

- Data Acquisition (DAQ)
- Debug
- LCD
- QRS

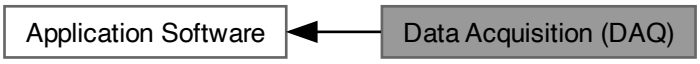
### 4.3.1 Detailed Description

Application-specific software modules.

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

### 4.3.2 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 acquision (DAQ) functions.*
- file lookup.c

  *Lookup table source code.*
- file lookup.h

  *Lookup table API.*

#### Macros

- #define **SAMPLING_PERIOD_MS** 5

  *sampling period in ms ( $T_s = 1/f_s$ )*
- #define **LOOKUP_ADC_MAX** (float32_t) 5.5
- #define **LOOKUP_ADC_MIN** (float32_t)(-5.5)

#### Typedefs

- typedef arm_biquad_casd_df1_inst_f32 **filt_t**

#### Enumerations

- enum { **NUM_FILT_STAGES** = 10 , **NUM_FILT_COEFFS** = NUM_FILT_STAGES $*$ 5 , **STATE_BUFF_SIZE** = NUM_FILT_STAGES $*$ 4 }

#### Functions

- void **DAQ_Init** (void)

  *Initialize the data acquisition module, including the input filter and timer interrupt-based analog-to-digital conversion (ADC) @ $f_s = 200[Hz]$.*
- float32_t DAQ_Filter (volatile float32_t inputSample)

  *Filter an input sample using a 40 [Hz] low pass filter and a 60 [Hz] notch filter.*
- const float32_t $*$ Lookup_GetPtr_ADC (void)

  *Return a pointer to the ADC lookup table.*

#### 4.3.2.1 Detailed Description

Module for managing data acquisition (DAQ) functions.

#### 4.3.2.2 Function Documentation

**DAQ_Filter()**

```
float32_t DAQ_Filter (
            volatile float32_t inputSample )
```

Filter an input sample using a 40 [Hz] low pass filter and a 60 [Hz] notch filter.

**Parameters**

| | | |
|---|---|---|
| in | *inputSample* | Raw input sample in range [-5.5, 5.5) [V]. |
| out | *float32_t* | Filtered output sample. |

**Lookup_GetPtr_ADC()**

```
const float32_t * Lookup_GetPtr_ADC (
            void  )
```

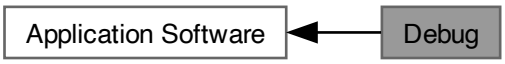
Return a pointer to the ADC lookup table.

**Returns**

const float32_t*

### 4.3.3 Debug

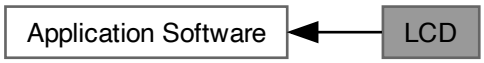
Collaboration diagram for Debug:




Module for debugging functions, including serial output and assertion.

### 4.3.4 LCD

Collaboration diagram for LCD:




#### Files

- file LCD.c

  *Source code for LCD module.*
- file LCD.h

  *Module for outputting the ECG waveform and HR to a liquid crystal display (LCD).*

#### Enumerations

- enum { **X_MAX** = NUM_ROWS , **Y_MAX** = NUM_COLS }

#### Color Setting Functions

- enum {
  **LCD_BLACK** = 0x00 , **LCD_RED** = 0x04 , **LCD_GREEN** = 0x02 , **LCD_BLUE** = 0x01 ,
  **LCD_YELLOW** = 0x06 , **LCD_CYAN** = 0x03 , **LCD_PURPLE** = 0x05 , **LCD_WHITE** = 0x07 ,
  **LCD_BLACK_INV** = LCD_WHITE , **LCD_RED_INV** = LCD_CYAN , **LCD_GREEN_INV** = LCD_PURPLE ,
  **LCD_BLUE_INV** = LCD_YELLOW ,
  **LCD_YELLOW_INV** = LCD_BLUE , **LCD_CYAN_INV** = LCD_RED , **LCD_PURPLE_INV** = LCD_GREEN ,
  **LCD_WHITE_INV** = LCD_BLACK }
- void LCD_setColor (uint8_t R_val, uint8_t G_val, uint8_t B_val)

  *Set the current color value for the display. Only the first 5-6 bits of each inputted value are used.*
- void LCD_setColor_3bit (uint8_t color_code)

  *Set the color value via a 3-bit code.*

#### Init./Config. Functions

- void **LCD_Init** (void)

  *Initialize the LCD driver and its internal independencies.*
- void LCD_setOutputMode (bool isOn)

  *Toggle display output `ON` or `OFF` (`OFF` by default). Turning output `OFF` stops the LCD driver chip from writing to the display, and also blanks out the display completely.*
- void LCD_toggleOutput (void)

  *Toggle display output `ON` or `OFF` (`OFF` by default).*
- void LCD_setColorInversionMode (bool isOn)

  *Turn color inversion `ON` or `OFF` (`OFF` by default).*
- void LCD_toggleColorInversion (void)

  *Toggle color inversion `ON` or `OFF` (`OFF` by default).*
- void LCD_setColorDepth (bool is_16bit)

  *Set the color depth to 16-bit or 18-bit. 16-bit color depth allows for only ∼65K colors, but only needs 2 data transfers. 18-bit color depth allows for ∼262K colors, but requires 3 transfers.*
- void LCD_toggleColorDepth (void)

  *Toggle 16-bit or 18-bit color depth (16-bit by default).*

**Drawing Area Definition Functions**

- void LCD_setArea (uint16_t x1_new, uint16_t x2_new, uint16_t y1_new, uint16_t y2_new)

  *Set the area of the display to be written to.* `0 <= x1 <= x2 < X_MAX 0 <= y1 <= y2 < Y_MAX`
- void LCD_setX (uint16_t x1_new, uint16_t x2_new)

  *Set only new x-coordinates to be written to.* `0 <= x1 <= x2 < X_MAX`
- void LCD_setY (uint16_t y1_new, uint16_t y2_new)

  *Set only new y-coordinates to be written to.* `0 <= y1 <= y2 < Y_MAX`

**Drawing Functions**

- void LCD_Draw (void)

  *Draw on the LCD display. Call this function after setting the drawable area via LCD_setArea(), or after individually calling LCD_setX() and/or LCD_setY().*
- 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)`.
- void LCD_graphSample (uint16_t x1, uint16_t dx, uint16_t y1, uint16_t dy, uint16_t y_min, uint16_t y_max, uint16_t color_code)

  *Draw a rectangle of size* `dx` *x* `dy` *and blank out all other pixels between* `y_min` *and* `y_max`.

#### 4.3.4.1 Detailed Description

Module for displaying graphs on an LCD via the ILI9341 module.

#### 4.3.4.2 Function Documentation

**LCD_Draw()**

```
void LCD_Draw (
            void  )
```

Draw on the LCD display. Call this function after setting the drawable area via LCD_setArea(), or after individually calling LCD_setX() and/or LCD_setY().

**LCD_drawHoriLine()**

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

Draw a horizontal line across the entire display.

**Parameters**

| | |
|---|---|
| *yCenter* | y-coordinate to center the line on |
| *lineWidth* | width of the line; should be a positive, odd number |

**See also**

LCD_drawVertLine, 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)`.

**Parameters**

| | |
|---|---|
| *x1* | lowest (left-most) x-coordinate |
| *dx* | length (horizontal distance) of the rectangle |
| *y1* | lowest (bottom-most) y-coordinate |
| *dy* | height (vertical distance) of the rectangle |
| *isFilled* | `true` to fill the rectangle, `false` to leave it unfilled |

### LCD_drawVertLine()

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

Draw a vertical line across the entire display.

**Parameters**

| | |
|---|---|
| *xCenter* | x-coordinate to center the line on |
| *lineWidth* | width of the line; should be a positive, odd number |

**See also**

LCD_drawHoriLine, LCD_drawRectangle()

### LCD_graphSample()

```
void LCD_graphSample (
```

```
            uint16_t x1,
            uint16_t dx,
            uint16_t y1,
            uint16_t dy,
            uint16_t y_min,
            uint16_t y_max,
            uint16_t color_code )
```

Draw a rectangle of size `dx` x `dy` and blank out all other pixels between `y_min` and `y_max`.

**Parameters**

| | |
|---|---|
| *x1* | lowest (left-most) x-coordinate |
| *dx* | length (horizontal distance) of the column |
| *y1* | y-coordinate of the pixel's bottom side |
| *dy* | height (vertical distance) of the pixel |
| *y_min* | lowest (bottom-most) y-coordinate |
| *y_max* | highest (top-most) y-coordinate |
| *color_code* | 3-bit color code |

TODO: Write description

### LCD_setArea()

```
void LCD_setArea (
            uint16_t x1_new,
            uint16_t x2_new,
            uint16_t y1_new,
            uint16_t y2_new )
```

Set the area of the display to be written to. `0 <= x1 <= x2 < X_MAX 0 <= y1 <= y2 < Y_MAX`

**Parameters**

| | |
|---|---|
| *x1_new* | left-most x-coordinate |
| *x2_new* | right-most x-coordinate |
| *y1_new* | lowest y-coordinate |
| *y2_new* | highest y-coordinate |

**See also**

LCD_setX(), LCD_setY()

### LCD_setColor()

```
void LCD_setColor (
            uint8_t R_val,
            uint8_t G_val,
            uint8_t B_val )
```

Set the current color value for the display. Only the first 5-6 bits of each inputted value are used.

**Parameters**

| | |
|---|---|
| *R_val* | 5-bit (`[0-31]`) R value; 6-bit (`[0-63]`) if color depth is 18-bit |
| *G_val* | 6-bit (`[0-63]`) G value |
| *B_val* | 5-bit (`[0-31]`) B value; 6-bit (`[0-63]`) if color depth is 18-bit |

**See also**

LCD_setColorDepth(), LCD_toggleColorDepth(), LCD_setColor_3bit()

### LCD_setColor_3bit()

```
void LCD_setColor_3bit (
            uint8_t color_code )
```

Set the color value via a 3-bit code.

**Parameters**

| | |
|---|---|
| *color_code* | 3-bit color value to use. Bits 2, 1, 0 correspond to R, G, and B values, respectively. |

**See also**

LCD_setColorDepth(), LCD_toggleColorDepth(), LCD_setColor()

This is simply a convenience function for setting the color using the enum values defined in the header file. The ones with the `_INV` suffix should be used when the display colors are inverted.

| hex | binary | macro |
|---|---|---|
| 0x00 | 000 | LCD_BLACK |
| 0x01 | 001 | LCD_BLUE |
| 0x02 | 010 | LCD_GREEN |
| 0x03 | 011 | LCD_CYAN |
| 0x04 | 100 | LCD_RED |
| 0x05 | 101 | LCD_PURPLE |
| 0x06 | 110 | LCD_YELLOW |
| 0x07 | 111 | LCD_WHITE |

### LCD_setColorDepth()

```
void LCD_setColorDepth (
            bool is_16bit )
```

Set the color depth to 16-bit or 18-bit. 16-bit color depth allows for only ~65K colors, but only needs 2 data transfers. 18-bit color depth allows for ~262K colors, but requires 3 transfers.

**Parameters**

| | | |
|---|---|---|
| in | *is_16bit* | `true` for 16-bit, `false` for 18b-bit |

**See also**

LCD_toggleColorDepth(), LCD_setColor(), LCD_setColor_3bit()

### LCD_setColorInversionMode()

```
void LCD_setColorInversionMode (
            bool isOn )
```

Turn color inversion `ON` or `OFF` (`OFF` by default).

**Parameters**

| | | |
|---|---|---|
| in | *isOn* | `true` to invert colors, `false` to use regular colors |

**See also**

LCD_toggleColorInversion(), LCD_setColor(), LCD_setColor_3bit()

### LCD_setOutputMode()

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

Toggle display output `ON` or `OFF` (`OFF` by default). Turning output `OFF` stops the LCD driver chip from writing to the display, and also blanks out the display completely.

**Parameters**

| | | |
|---|---|---|
| in | *isOn* | `true` to turn display output `ON`, `false` to turn `OFF` |

**See also**

LCD_toggleOutput()

### LCD_setX()

```
void LCD_setX (
            uint16_t x1_new,
            uint16_t x2_new )
```

Set only new x-coordinates to be written to. `0 <= x1 <= x2 < X_MAX`

**Parameters**

| | |
|---|---|
| *x1_new* | left-most x-coordinate |
| *x2_new* | right-most x-coordinate |

**See also**

LCD_setY(), LCD_setArea()

### LCD_setY()

```
void LCD_setY (
            uint16_t y1_new,
            uint16_t y2_new )
```

Set only new y-coordinates to be written to. `0 <= y1 <= y2 < Y_MAX`

**Parameters**

| | |
|---|---|
| *y1_new* | lowest y-coordinate |
| *y2_new* | highest y-coordinate |

**See also**

LCD_setX(), LCD_setArea()

### LCD_toggleColorDepth()

```
void LCD_toggleColorDepth (
            void  )
```

Toggle 16-bit or 18-bit color depth (16-bit by default).

**See also**

LCD_setColorDepth(), LCD_setColor(), LCD_setColor_3bit()

### LCD_toggleColorInversion()

```
void LCD_toggleColorInversion (
            void  )
```

Toggle color inversion `ON` or `OFF` (`OFF` by default).

**See also**

LCD_setColorInversionMode(), LCD_setColor(), LCD_setColor_3bit()

#### LCD_toggleOutput()

```
void LCD_toggleOutput (
            void  )
```

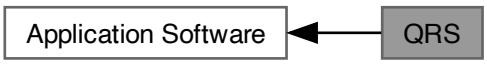
Toggle display output `ON` or `OFF` (`OFF` by default).

**See also**

LCD_setOutputMode()

### 4.3.5 QRS

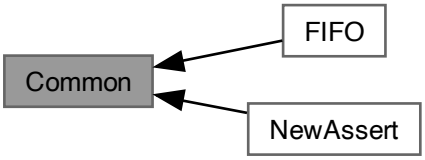
Collaboration diagram for QRS:




Module for analyzing ECG data to determine heart rate.

## 4.4 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`.*

## 4.4.1 Detailed Description

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

## 4.4.2 Function Documentation

### Assert()

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

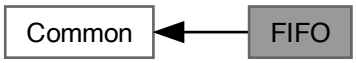
Custom `assert` implementation that is more lightweight than the one from `newlib`.

**Parameters**

| | | |
|---|---|---|
| in | *condition* | Conditional to test. Causes an infinite loop if `false`. |

## 4.4.3 FIFO

Collaboration diagram for FIFO:




### Files

- file FIFO.c

  *Source code for FIFO buffer module.*
- file FIFO.h

  *FIFO buffer data structure.*

### Data Structures

- struct FIFO_t

### Macros

- #define **FIFO_POOL_SIZE** 5

### Functions

- volatile 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.*

#### 4.4.3.1 Detailed Description

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

#### 4.4.3.2 Function Documentation

##### FIFO_Flush()

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

Empty the FIFO buffer's contents into an array.

**Parameters**

| | |
|---|---|
| *fifo* | Pointer to source FIFO buffer. |
| *outputBuffer* | Array to output values to. Should be the same length as the FIFO buffer. |

##### FIFO_Get()

```
uint32_t FIFO_Get (
            volatile FIFO_t * fifo )
```

Remove the first value of the buffer.

**Parameters**

| | |
|---|---|
| *fifo* | Pointer to FIFO object |

**Returns**

First sample in the FIFO.

##### FIFO_getCurrSize()

```
uint32_t FIFO_getCurrSize (
            volatile FIFO_t * fifo )
```

Get the current size of the FIFO buffer.

**Parameters**

| | |
|---|---|
| *fifo* | Pointer to the FIFO buffer. |

### FIFO_Init()

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

Initialize a FIFO buffer of length `N`.

#### Parameters

| | |
|---|---|
| *buffer* | Array of size `N` to be used as FIFO buffer |
| *N* | Length of `buffer`. Usable length is `N - 1`. |

#### Returns

pointer to the FIFO buffer

TODO: Add details

### FIFO_isEmpty()

```
bool FIFO_isEmpty (
            volatile FIFO_t * fifo )
```

Check if the FIFO buffer is empty.

#### Parameters

| | |
|---|---|
| *fifo* | Pointer to the FIFO buffer. |

#### Return values

| | |
|---|---|
| *true* | The buffer is empty. |
| *false* | The buffer is not empty. |

### FIFO_isFull()

```
bool FIFO_isFull (
            volatile FIFO_t * fifo )
```

Check if the FIFO buffer is full.

#### Parameters

| | |
|---|---|
| *fifo* | Pointer to the FIFO buffer. |

**Return values**

| | |
|---|---|
| *true* | The buffer is full. |
| *false* | The buffer is not full. |

### FIFO_PeekAll()

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

See the FIFO buffer's contents without removing them.

**Parameters**

| | |
|---|---|
| *fifo* | Pointer to FIFO object |
| *outputBuffer* | Array to output values to. Should be the same length as the FIFO buffer. |

### FIFO_PeekOne()

```
uint32_t FIFO_PeekOne (
            volatile FIFO_t * fifo )
```

See the first element in the FIFO without removing it.

**Parameters**

| | |
|---|---|
| *fifo* | Pointer to FIFO object |

**Returns**

First sample in the FIFO.

### FIFO_Put()

```
void FIFO_Put (
            volatile FIFO_t * fifo,
            const uint32_t val )
```

Add a value to the end of the buffer.

**Parameters**

| | |
|---|---|
| *fifo* | Pointer to FIFO object |
| *val* | last value in the buffer |

### FIFO_Reset()

```
void FIFO_Reset (
            volatile FIFO_t * fifo )
```

Reset the FIFO buffer.

**Parameters**

| | | |
|---|---|---|
| in | *fifo* | Pointer to FIFO buffer. |

### FIFO_TransferAll()

```
void FIFO_TransferAll (
            volatile FIFO_t * srcFifo,
            volatile FIFO_t * destFifo )
```

Transfer the contents of one FIFO buffer to another.

**Parameters**

| | |
|---|---|
| *srcFifo* | Pointer to source FIFO buffer. |
| *destFifo* | Pointer to destination FIFO buffer. |

### FIFO_TransferOne()

```
void FIFO_TransferOne (
            volatile FIFO_t * srcFifo,
            volatile FIFO_t * destFifo )
```

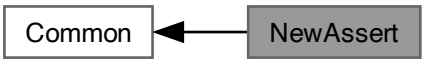
Transfer a value from one FIFO buffer to another.

**Parameters**

| | |
|---|---|
| *srcFifo* | Pointer to source FIFO buffer. |
| *destFifo* | Pointer to destination FIFO buffer. |

#### 4.4.4 NewAssert

Collaboration diagram for NewAssert:




Module for using a custom `assert` implementation.

# 5 Data Structure Documentation

## 5.1 FIFO_t Struct Reference

### Data Fields

- volatile uint32_t * **buffer**
  *(pointer to) array to use as FIFO buffer*
- volatile uint32_t **N**
  *length of* `buffer`
- volatile uint32_t **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

## 5.2 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

## 5.3 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*

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

- Led.c

## 5.4 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

# 6 File Documentation

## 6.1 DAQ.c File Reference

Source code for DAQ module.

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

### Macros

- #define **SAMPLING_PERIOD_MS** 5

  *sampling period in ms ( $T_s = 1/f_s$ )*

### Typedefs

- typedef arm_biquad_casd_df1_inst_f32 **filt_t**

### Enumerations

- enum { **NUM_FILT_STAGES** = 10 , **NUM_FILT_COEFFS** = NUM_FILT_STAGES $*$ 5 , **STATE_BUFF_SIZE** = NUM_FILT_STAGES $*$ 4 }

### Functions

- void **DAQ_Init** (void)

  *Initialize the data acquisition module, including the input filter and timer interrupt-based analog-to-digital conversion (ADC) @ $f_s = 200[Hz]$.*
- float32_t DAQ_Filter (volatile float32_t inputSample)

  *Filter an input sample using a 40 [Hz] low pass filter and a 60 [Hz] notch filter.*

#### 6.1.1 Detailed Description

Source code for DAQ module.

**Author**

Bryan McElvy

## 6.2 DAQ.h File Reference

Application software for handling data acquision (DAQ) functions.

```
#include "ADC.h"
#include "Timer.h"
#include "FIFO.h"
#include "arm_math_types.h"
#include "dsp/filtering_functions.h"
#include "tm4c123gh6pm.h"
#include <stdint.h>
```

### Functions

- void **DAQ_Init** (void)

  *Initialize the data acquisition module, including the input filter and timer interrupt-based analog-to-digital conversion (ADC) @ $f_s = 200[Hz]$.*
- float32_t DAQ_Filter (volatile float32_t inputSample)

  *Filter an input sample using a 40 [Hz] low pass filter and a 60 [Hz] notch filter.*

### 6.2.1 Detailed Description

Application software for handling data acquision (DAQ) functions.

**Author**

Bryan McElvy

## 6.3 Debug.h File Reference

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

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

### Enumerations

- enum **msg_t** {
  **START_MSG** , **DAQ_INIT** , **QRS_INIT** , **LCD_INIT** ,
  **ASSERT_FALSE** }

### Functions

- void **Debug_Init** (void)
  *Init. the Debug module and send a start message to the port.*
- 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.*
- void Debug_Assert (bool condition)
  *Stops program if `condition` is `true`. Useful for bug detection during debugging.*

### 6.3.1 Detailed Description

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

**Author**

Bryan McElvy

### 6.3.2 Function Documentation

#### Debug_Assert()

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

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

**Parameters**

| | |
|---|---|
| *condition* | |

### Debug_SendFromList()

```
void Debug_SendFromList (
            msg_t msg )
```

Send a message from the message list.

**Parameters**

| | | |
|---|---|---|
| in | *msg* | Message to send. |

### Debug_SendMsg()

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

Send a message to the serial port.

**Parameters**

| | |
|---|---|
| *message* | (Pointer to) array of ASCII characters. |

### Debug_WriteFloat()

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

Write a floating-point value to the serial port.

**Parameters**

| | | |
|---|---|---|
| in | *value* | Floating-point value. |

## 6.4 LCD.c File Reference

Source code for LCD module.

```
#include "LCD.h"
#include "ILI9341.h"
#include "SPI.h"
#include "Timer.h"
```

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

## Functions

### Init./Config. Functions

- void **LCD_Init** (void)
  *Initialize the LCD driver and its internal independencies.*
- void LCD_setOutputMode (bool isOn)
  *Toggle display output `ON` or `OFF` (`OFF` by default). Turning output `OFF` stops the LCD driver chip from writing to the display, and also blanks out the display completely.*
- void LCD_toggleOutput (void)
  *Toggle display output `ON` or `OFF` (`OFF` by default).*
- void LCD_setColorInversionMode (bool isOn)
  *Turn color inversion `ON` or `OFF` (`OFF` by default).*
- void LCD_toggleColorInversion (void)
  *Toggle color inversion `ON` or `OFF` (`OFF` by default).*
- void LCD_setColorDepth (bool is_16bit)
  *Set the color depth to 16-bit or 18-bit. 16-bit color depth allows for only ~65K colors, but only needs 2 data transfers. 18-bit color depth allows for ~262K colors, but requires 3 transfers.*
- void LCD_toggleColorDepth (void)
  *Toggle 16-bit or 18-bit color depth (16-bit by default).*

### Drawing Area Definition Functions

- void LCD_setArea (uint16_t x1_new, uint16_t x2_new, uint16_t y1_new, uint16_t y2_new)
  *Set the area of the display to be written to. `0 <= x1 <= x2 < X_MAX 0 <= y1 <= y2 < Y_MAX`*
- void LCD_setX (uint16_t x1_new, uint16_t x2_new)
  *Set only new x-coordinates to be written to. `0 <= x1 <= x2 < X_MAX`*
- void LCD_setY (uint16_t y1_new, uint16_t y2_new)
  *Set only new y-coordinates to be written to. `0 <= y1 <= y2 < Y_MAX`*

### Color Setting Functions

- void LCD_setColor (uint8_t R_val, uint8_t G_val, uint8_t B_val)
  *Set the current color value for the display. Only the first 5-6 bits of each inputted value are used.*
- void LCD_setColor_3bit (uint8_t color_code)
  *Set the color value via a 3-bit code.*

### Drawing Functions

- void LCD_Draw (void)
  *Draw on the LCD display. Call this function after setting the drawable area via LCD_setArea(), or after individually calling LCD_setX() and/or LCD_setY().*
- 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)`.*
- void LCD_graphSample (uint16_t x1, uint16_t dx, uint16_t y1, uint16_t dy, uint16_t y_min, uint16_t y_max, uint16_t color_code)
  *Draw a rectangle of size `dx` x `dy` and blank out all other pixels between `y_min` and `y_max`.*

### 6.4.1 Detailed Description

Source code for LCD module.

**Author**

Bryan McElvy

## 6.5 LCD.h File Reference

Module for outputting the ECG waveform and HR to a liquid crystal display (LCD).

```
#include "ILI9341.h"
#include "SPI.h"
#include "Timer.h"
#include "NewAssert.h"
#include "tm4c123gh6pm.h"
#include <stdbool.h>
#include <stdint.h>
```

### Enumerations

- enum { **X_MAX** = NUM_ROWS , **Y_MAX** = NUM_COLS }

### Functions

#### Init./Config. Functions

- void **LCD_Init** (void)
  *Initialize the LCD driver and its internal independencies.*
- void LCD_setOutputMode (bool isOn)
  *Toggle display output `ON` or `OFF` (`OFF` by default). Turning output `OFF` stops the LCD driver chip from writing to the display, and also blanks out the display completely.*
- void LCD_toggleOutput (void)
  *Toggle display output `ON` or `OFF` (`OFF` by default).*
- void LCD_setColorInversionMode (bool isOn)
  *Turn color inversion `ON` or `OFF` (`OFF` by default).*
- void LCD_toggleColorInversion (void)
  *Toggle color inversion `ON` or `OFF` (`OFF` by default).*
- void LCD_setColorDepth (bool is_16bit)
  *Set the color depth to 16-bit or 18-bit. 16-bit color depth allows for only ∼65K colors, but only needs 2 data transfers. 18-bit color depth allows for ∼262K colors, but requires 3 transfers.*
- void LCD_toggleColorDepth (void)
  *Toggle 16-bit or 18-bit color depth (16-bit by default).*

#### Drawing Area Definition Functions

- void LCD_setArea (uint16_t x1_new, uint16_t x2_new, uint16_t y1_new, uint16_t y2_new)
  *Set the area of the display to be written to. `0 <= x1 <= x2 < X_MAX 0 <= y1 <= y2 < Y_MAX`*
- void LCD_setX (uint16_t x1_new, uint16_t x2_new)
  *Set only new x-coordinates to be written to. `0 <= x1 <= x2 < X_MAX`*
- void LCD_setY (uint16_t y1_new, uint16_t y2_new)
  *Set only new y-coordinates to be written to. `0 <= y1 <= y2 < Y_MAX`*

### Drawing Functions

- void LCD_Draw (void)
  *Draw on the LCD display. Call this function after setting the drawable area via LCD_setArea(), or after individually calling LCD_setX() and/or LCD_setY().*
- 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)`.*
- void LCD_graphSample (uint16_t x1, uint16_t dx, uint16_t y1, uint16_t dy, uint16_t y_min, uint16_t y_max, uint16_t color_code)
  *Draw a rectangle of size `dx` x `dy` and blank out all other pixels between `y_min` and `y_max`.*

### Color Setting Functions

- enum {
  **LCD_BLACK** = 0x00 , **LCD_RED** = 0x04 , **LCD_GREEN** = 0x02 , **LCD_BLUE** = 0x01 ,
  **LCD_YELLOW** = 0x06 , **LCD_CYAN** = 0x03 , **LCD_PURPLE** = 0x05 , **LCD_WHITE** = 0x07 ,
  **LCD_BLACK_INV** = LCD_WHITE , **LCD_RED_INV** = LCD_CYAN , **LCD_GREEN_INV** = LCD_PURPLE ,
  **LCD_BLUE_INV** = LCD_YELLOW ,
  **LCD_YELLOW_INV** = LCD_BLUE , **LCD_CYAN_INV** = LCD_RED , **LCD_PURPLE_INV** = LCD_GREEN ,
  **LCD_WHITE_INV** = LCD_BLACK }
- void LCD_setColor (uint8_t R_val, uint8_t G_val, uint8_t B_val)
  *Set the current color value for the display. Only the first 5-6 bits of each inputted value are used.*
- void LCD_setColor_3bit (uint8_t color_code)
  *Set the color value via a 3-bit code.*

#### 6.5.1 Detailed Description

Module for outputting the ECG waveform and HR to a liquid crystal display (LCD).

**Author**

Bryan McElvy

## 6.6 QRS.h File Reference

QRS detection algorithm functions.

```
#include "dsp/filtering_functions_f16.h"
```

#### 6.6.1 Detailed Description

QRS detection algorithm functions.

**Author**

Bryan McElvy

This module contains functions for detecting heart rate (`HR`) using a simplified version of the Pan-Tompkins algorithm.

## 6.7 FIFO.c File Reference

Source code for FIFO buffer module.

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

### Data Structures

- struct FIFO_t

### Functions

- volatile 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.*

### 6.7.1 Detailed Description

Source code for FIFO buffer module.

**Author**

Bryan McElvy

## 6.8 FIFO.h File Reference

FIFO buffer data structure.

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

### Macros

- #define **FIFO_POOL_SIZE** 5

### Functions

- volatile 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.*

### 6.8.1 Detailed Description

FIFO buffer data structure.

**Author**

Bryan McElvy

## 6.9 lookup.c File Reference

Lookup table source code.

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

### Functions

- const float32_t * Lookup_GetPtr_ADC (void)
  *Return a pointer to the ADC lookup table.*

### 6.9.1 Detailed Description

Lookup table source code.

**Author**

Bryan McElvy

## 6.10 lookup.h File Reference

Lookup table API.

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

### Macros

- #define **LOOKUP_ADC_MAX** (float32_t) 5.5
- #define **LOOKUP_ADC_MIN** (float32_t)(-5.5)

### Functions

- const float32_t * Lookup_GetPtr_ADC (void)
  *Return a pointer to the ADC lookup table.*

### 6.10.1 Detailed Description

Lookup table API.

**Author**

Bryan McElvy

## 6.11 NewAssert.c File Reference

Source code for custom `assert` implementation.

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

### Functions

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

### 6.11.1 Detailed Description

Source code for custom `assert` implementation.

**Author**

Bryan McElvy

## 6.12 NewAssert.h File Reference

Header file for custom `assert` implementation.

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

### Functions

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

### 6.12.1 Detailed Description

Header file for custom `assert` implementation.

**Author**

Bryan McElvy

## 6.13 ADC.c File Reference

Source code for ADC module.

```
#include "ADC.h"
#include "GPIO.h"
#include "Timer.h"
#include "lookup.h"
#include "arm_math_types.h"
#include "tm4c123gh6pm.h"
#include <stdint.h>
```

### Functions

- void **ADC_Init** (void)

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

  *Enable the ADC interrupt.*
- void **ADC_InterruptDisable** (void)

  *Disable the ADC interrupt.*
- float32_t ADC_ConvertToVolts (uint16_t raw_sample)

  *Convert a raw ADC sample to voltage in [mV].*

#### 6.13.1 Detailed Description

Source code for ADC module.

**Author**

Bryan McElvy

## 6.14 ADC.h File Reference

Driver module for analog-to-digital conversion (ADC).

```
#include "GPIO.h"
#include "Timer.h"
#include "lookup.h"
#include "arm_math_types.h"
#include "tm4c123gh6pm.h"
#include <stdint.h>
```

### Functions

- void **ADC_Init** (void)

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

  *Enable the ADC interrupt.*
- void **ADC_InterruptDisable** (void)

  *Disable the ADC interrupt.*
- float32_t ADC_ConvertToVolts (uint16_t raw_sample)

  *Convert a raw ADC sample to voltage in [mV].*

### 6.14.1 Detailed Description

Driver module for analog-to-digital conversion (ADC).

**Author**

Bryan McElvy

## 6.15 GPIO.c File Reference

Source code for GPIO module.

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

**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.*

### 6.15.1 Detailed Description

Source code for GPIO module.

**Author**

Bryan McElvy

### 6.15.2 Function Documentation

#### GPIO_ConfigAltMode()

```
void GPIO_ConfigAltMode (
            GPIO_Port_t * gpioPort,
            GPIO_Pin_t pinMask )
```

Activate the alternate mode for the specified pins.

**Parameters**

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

#### GPIO_ConfigAnalog()

```
void GPIO_ConfigAnalog (
            GPIO_Port_t * gpioPort,
            GPIO_Pin_t pinMask )
```

Activate analog mode for the specified GPIO pins.

**Parameters**

| | | |
|---|---|---|
| in | *gpioPort* | Pointer to the specified GPIO port. |
| in | *pinMask* | Bit mask corresponding to the intended 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 | *gpioPort* | Pointer to the specified GPIO port. |
| in | *bitMask* | Bit mask corresponding to the intended `INPUT` pin(s). |

#### 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 | *gpioPort* | Pointer to the specified GPIO port. |
| in | *bitMask* | Bit mask corresponding to the intended `OUTPUT` 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 | *gpioPort* | Pointer to the specified GPIO port. |
| in | *pinMask* | Bit mask corresponding to the intended pin(s). |
| in | *drive_mA* | Drive strength in [mA]. Should be 2, 4, or 8 [mA]. |

### 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 | *gpioPort* | Pointer to the specified GPIO port. |
| in | *pinMask* | 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 | *gpioPort* | Pointer to the specified GPIO port. |
| in | *pinMask* | Bit mask corresponding to the intended pin(s). |
| in | *risingEdge* | `true` for rising edge, `false` for falling edge |

### 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 | *gpioPort* | Pointer to the specified GPIO port. |
| in | *pinMask* | Bit mask corresponding to the intended pin(s). |
| in | *highLevel* | `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 | *gpioPort* | Pointer to the specified GPIO port. |
| in | *priority* | Priority number between 0 (highest) and 7 (lowest). |

### 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 | *gpioPort* | Pointer to the specified GPIO port. |
| in | *pinMask* | Bit mask corresponding to the intended pin(s). |
| in | *fieldEncoding* | Number corresponding to intended alternate mode. |

### GPIO_ConfigPullDown()

```
void GPIO_ConfigPullDown (
            GPIO_Port_t * gpioPort,
            GPIO_Pin_t pinMask )
```

Activate the specified pins' internal pull-down resistors.

**Parameters**

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

### GPIO_EnableDigital()

```
void GPIO_EnableDigital (
            GPIO_Port_t * gpioPort,
            GPIO_Pin_t pinMask )
```

Enable digital I/O for the specified pins.

**Parameters**

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

### GPIO_InitPort()

```
GPIO_Port_t * GPIO_InitPort (
            GPIO_PortName_t portName )
```

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

**Parameters**

| | | |
|---|---|---|
| in | *portName* | 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 | *gpioPort* | Pointer to the specified GPIO port. |
| out | *true* | The GPIO port is initialized. |
| out | *false* | The GPIO port has not been initialized. |

### GPIO_ReadPins()

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

Read from the specified GPIO pin.

**Parameters**

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

## 6.16 GPIO.h File Reference

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

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

### Enumerations

- 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) }
- enum **GPIO_PortName_t** {
  **A** , **B** , **C** , **D** ,
  **E** , **F** }

### 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.*

### 6.16.1 Detailed Description

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

**Author**

Bryan McElvy

### 6.16.2 Function Documentation

#### GPIO_ConfigAltMode()

```
void GPIO_ConfigAltMode (
            GPIO_Port_t * gpioPort,
            GPIO_Pin_t pinMask )
```

Activate the alternate mode for the specified pins.

**Parameters**

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

#### GPIO_ConfigAnalog()

```
void GPIO_ConfigAnalog (
            GPIO_Port_t * gpioPort,
            GPIO_Pin_t pinMask )
```

Activate analog mode for the specified GPIO pins.

**Parameters**

| | | |
|---|---|---|
| in | *gpioPort* | Pointer to the specified GPIO port. |
| in | *pinMask* | Bit mask corresponding to the intended 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 | *gpioPort* | Pointer to the specified GPIO port. |
| in | *bitMask* | Bit mask corresponding to the intended `INPUT` pin(s). |

### 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 | *gpioPort* | Pointer to the specified GPIO port. |
| in | *bitMask* | Bit mask corresponding to the intended `OUTPUT` 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 | *gpioPort* | Pointer to the specified GPIO port. |
| in | *pinMask* | Bit mask corresponding to the intended pin(s). |
| in | *drive_mA* | Drive strength in [mA]. Should be 2, 4, or 8 [mA]. |

### 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 | *gpioPort* | Pointer to the specified GPIO port. |
| in | *pinMask* | 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 | *gpioPort* | Pointer to the specified GPIO port. |
| in | *pinMask* | Bit mask corresponding to the intended pin(s). |
| in | *risingEdge* | `true` for rising edge, `false` for falling edge |

### 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 | *gpioPort* | Pointer to the specified GPIO port. |
| in | *pinMask* | Bit mask corresponding to the intended pin(s). |
| in | *highLevel* | `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 | *gpioPort* | Pointer to the specified GPIO port. |
| in | *priority* | Priority number between 0 (highest) and 7 (lowest). |

### 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 | *gpioPort* | Pointer to the specified GPIO port. |
| in | *pinMask* | Bit mask corresponding to the intended pin(s). |
| in | *fieldEncoding* | Number corresponding to intended alternate mode. |

### GPIO_ConfigPullDown()

```
void GPIO_ConfigPullDown (
            GPIO_Port_t * gpioPort,
            GPIO_Pin_t pinMask )
```

Activate the specified pins' internal pull-down resistors.

**Parameters**

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

### GPIO_EnableDigital()

```
void GPIO_EnableDigital (
            GPIO_Port_t * gpioPort,
            GPIO_Pin_t pinMask )
```

Enable digital I/O for the specified pins.

**Parameters**

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

### GPIO_InitPort()

```
GPIO_Port_t * GPIO_InitPort (
            GPIO_PortName_t portName )
```

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

**Parameters**

| | | |
|---|---|---|
| in | *portName* | 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 | *gpioPort* | Pointer to the specified GPIO port. |
| out | *true* | The GPIO port is initialized. |
| out | *false* | The GPIO port has not been initialized. |

### GPIO_ReadPins()

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

Read from the specified GPIO pin.

**Parameters**

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

## 6.17 PLL.c File Reference

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

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

### Functions

- void **PLL_Init** (void)

  *Initializes the phase-locked-loop (PLL), allowing a bus frequency of 80[MHz].*

### 6.17.1 Detailed Description

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

**Author**

Bryan McElvy

## 6.18 PLL.h File Reference

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

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

### Functions

- void **PLL_Init** (void)

  *Initializes the phase-locked-loop (PLL), allowing a bus frequency of 80[MHz].*

### 6.18.1 Detailed Description

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

**Author**

Bryan McElvy

## 6.19 SPI.c File Reference

Source code for SPI module.

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

### Macros

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

### Enumerations

- enum {
  **SPI_CLK_PIN** = GPIO_PIN2 , **SPI_CS_PIN** = GPIO_PIN3 , **SPI_RX_PIN** = GPIO_PIN4 , **SPI_TX_PIN** = GPIO_PIN5 ,
  **SPI_DC_PIN** = GPIO_PIN6 , **SPI_RESET_PIN** = GPIO_PIN7 , **SPI_SSI0_PINS** = (SPI_CLK_PIN | SPI_↩ CS_PIN | SPI_RX_PIN | SPI_TX_PIN) , **SPI_GPIO_PINS** = (SPI_DC_PIN | SPI_RESET_PIN) ,
  **SPI_ALL_PINS** = (SPI_SSI0_PINS | SPI_GPIO_PINS) }

### Functions

- void SPI_Init (void)

  *Initialize SSI0 to act as an SPI Controller (AKA Master) in mode 0.*
- uint8_t SPI_Read (void)

  *Read data from the peripheral.*
- void SPI_WriteCmd (uint8_t cmd)

  *Write an 8-bit command to the peripheral.*
- void SPI_WriteData (uint8_t data)

  *Write 8-bit data to the peripheral.*

### 6.19.1 Detailed Description

Source code for SPI module.

**Author**

Bryan McElvy

## 6.20 SPI.h File Reference

Driver module for using the serial peripheral interface (SPI) protocol.

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

### 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 peripheral.*
- void SPI_WriteCmd (uint8_t cmd)

  *Write an 8-bit command to the peripheral.*
- void SPI_WriteData (uint8_t data)

  *Write 8-bit data to the peripheral.*

### 6.20.1 Detailed Description

Driver module for using the serial peripheral interface (SPI) protocol.

**Author**

Bryan McElvy

## 6.21 SysTick.c File Reference

Implementation details for SysTick functions.

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

### 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.*

#### 6.21.1 Detailed Description

Implementation details for SysTick functions.

**Author**

Bryan McElvy

## 6.22 SysTick.h File Reference

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

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

### 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.*

#### 6.22.1 Detailed Description

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

**Author**

Bryan McElvy

## 6.23 Timer.c File Reference

Implementation for timer module.

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

### Functions

#### Timer0A

- void **Timer0A_Init** (void)
  *Initialize timer 0 as 32-bit, one-shot, countdown timer.*
- void Timer0A_Start (uint32_t time_ms)
  *Count down starting from the inputted value.*
- uint8_t Timer0A_isCounting (void)
  *Returns 1 if Timer0 is still counting and 0 if not.*
- void Timer0A_Wait1ms (uint32_t time_ms)
  *Wait for the specified amount of time in [ms].*

#### Timer1A

- void Timer1A_Init (uint32_t time_ms)
  *Initialize timer 1 as a 32-bit, periodic, countdown timer with interrupts.*

#### Timer2A

- void **Timer2A_Init** (void)
  *Initialize timer 2 as 32-bit, one-shot, countdown timer.*
- void Timer2A_Start (uint32_t time_ms)
  *Count down starting from the inputted value.*
- uint8_t Timer2A_isCounting (void)
  *Returns 1 if Timer2 is still counting and 0 if not.*
- void Timer2A_Wait1ms (uint32_t time_ms)
  *Wait for the specified amount of time in [ms].*
- void Timer3A_Init (uint32_t time_ms)
  *Initialize Timer3A as a 32-bit, periodic, countdown timer that triggers ADC sample capture.*

### 6.23.1 Detailed Description

Implementation for timer module.

**Author**

Bryan McElvy

## 6.24 Timer.h File Reference

Driver module for general-purpose timer modules.

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

### Functions

#### Timer0A

- void **Timer0A_Init** (void)
  *Initialize timer 0 as 32-bit, one-shot, countdown timer.*
- void Timer0A_Start (uint32_t time_ms)
  *Count down starting from the inputted value.*
- uint8_t Timer0A_isCounting (void)
  *Returns 1 if Timer0 is still counting and 0 if not.*
- void Timer0A_Wait1ms (uint32_t time_ms)
  *Wait for the specified amount of time in [ms].*

#### Timer1A

- void Timer1A_Init (uint32_t time_ms)
  *Initialize timer 1 as a 32-bit, periodic, countdown timer with interrupts.*

#### Timer2A

- void **Timer2A_Init** (void)
  *Initialize timer 2 as 32-bit, one-shot, countdown timer.*
- void Timer2A_Start (uint32_t time_ms)
  *Count down starting from the inputted value.*
- uint8_t Timer2A_isCounting (void)
  *Returns 1 if Timer2 is still counting and 0 if not.*
- void Timer2A_Wait1ms (uint32_t time_ms)
  *Wait for the specified amount of time in [ms].*
- void Timer3A_Init (uint32_t time_ms)
  *Initialize Timer3A as a 32-bit, periodic, countdown timer that triggers ADC sample capture.*

### 6.24.1 Detailed Description

Driver module for general-purpose timer modules.

**Author**

Bryan McElvy

```
Timer | Function
---------------------
0A      Debouncing
1A      LCD Interrupts
2A      ILI9341 Resets
3A      ADC Interrupts
```

## 6.25 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>
```

### 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.*

### 6.25.1 Detailed Description

Source code for UART module.

**Author**

Bryan McElvy

## 6.26 UART.h File Reference

Driver module for serial communication via UART0 and UART 1.

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

### 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.*

### 6.26.1 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.
```

## 6.27 main.c File Reference

Main program file for ECG-HRM.

```
#include "DAQ.h"
#include "Debug.h"
#include "LCD.h"
#include "QRS.h"
#include "PLL.h"
```

### Functions

- int **main** (void)
- void **ADC0_SS3_Handler** (void)

  *Interrupt service routine (ISR) for collecting ADC samples.*
- void **Timer1A_Handler** (void)

  *Interrupt service routine (ISR) for outputting data to the LCD.*

#### 6.27.1 Detailed Description

Main program file for ECG-HRM.

**Author**

Bryan McElvy

## 6.28 ILI9341.c File Reference

Source code for ILI9341 module.

```
#include "ILI9341.h"
#include "SPI.h"
#include "Timer.h"
#include "FIFO.h"
#include "tm4c123gh6pm.h"
#include <stdbool.h>
#include <stdint.h>
```

### Enumerations

- enum Cmd_t {
  **NOP** = 0x00 , SWRESET = 0x01 , SPLIN = 0x10 , SPLOUT = 0x11 ,
  PTLON = 0x12 , NORON = 0x13 , DINVOFF = 0x20 , DINVON = 0x21 ,
  CASET = 0x2A , PASET = 0x2B , RAMWR = 0x2C , DISPOFF = 0x28 ,
  DISPON = 0x29 , PLTAR = 0x30 , VSCRDEF = 0x33 , MADCTL = 0x36 ,
  VSCRSADD = 0x37 , IDMOFF = 0x38 , IDMON = 0x39 , PIXSET = 0x3A ,
  FRMCTR1 = 0xB1 , FRMCTR2 = 0xB2 , FRMCTR3 = 0xB3 , PRCTR = 0xB5 ,
  IFCTL = 0xF6 }

### Functions

- void **ILI9341_Init** (void)

  *Initialize the LCD driver, the SPI module, and Timer2A.*
- void ILI9341_resetHard (void)

  *Perform a hardware reset of the LCD driver.*
- void ILI9341_resetSoft (void)

  *Perform a software reset of the LCD driver.*
- void ILI9341_setSleepMode (bool isSleeping)

  *Enter or exit sleep mode. The LCD driver is in sleep mode by default upon powering on or either kind of reset.*

- void ILI9341_setDispMode (bool isNormal, bool isFullColors)

  *Set the display area and color expression.*
- void ILI9341_setPartialArea (uint16_t rowStart, uint16_t rowEnd)

  *Set the partial display area for partial mode. Call before activating partial mode via ILI9341_setDisplayMode().*
- void ILI9341_setDispInversion (bool is_ON)

  *Toggle display inversion. Turning `ON` causes colors to be inverted on the display.*
- void ILI9341_setDispOutput (bool is_ON)

  *Turn display output ON or OFF. This function clears the display and stops outputting to the display area, but does not affect frame memory or power.*
- void ILI9341_setScrollArea (uint16_t topFixedArea, uint16_t vertScrollArea, uint16_t bottFixedArea)

  *Set the vertical scrolling area of the display. The sum of the three parameters should be equal to the max number of rows `NUM_ROWS = 320`.*
- void ILI9341_setScrollStart (uint16_t startRow)

  *Set the start row for vertical scrolling.*
- 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 (bool is_16bit)

  *Set the pixel format to be 16-bit (65K colors) or 18-bit (262K colors).*
- void **ILI9341_NoOpCmd** (void)

  *Send the "No Operation" command (`NOP = 0x00`) to the LCD driver. Can be used to terminate the "Memory Write" (`RAMWR`) and "Memory Read" (`RAMRD`) commands, but does nothing otherwise.*
- void ILI9341_setFrameRateNorm (uint8_t divisionRatio, uint8_t clocksPerLine)

  *TODO: Write brief.*
- void ILI9341_setFrameRateIdle (uint8_t divisionRatio, uint8_t clocksPerLine)

  *TODO: Write brief.*
- 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.*
- void ILI9341_setRowAddress (uint16_t startRow, uint16_t endRow)

  *not using backlight, so these aren't necessary*
- void ILI9341_setColAddress (uint16_t startCol, uint16_t endCol)

  *Sets the start/end rows to be written to.*
- void ILI9341_writeMemCmd (void)

  *Sends the "Write Memory" (`RAMWR`) command to the LCD driver, signalling that incoming data should be written to memory.*
- void ILI9341_writePixel (uint8_t red, uint8_t green, uint8_t blue, bool is_16bit)

  *Write a single pixel to frame memory.*

### 6.28.1 Detailed Description

Source code for ILI9341 module.

**Author**

Bryan McElvy

## 6.29 ILI9341.h File Reference

Driver module for interfacing with an ILI9341 LCD driver.

```
#include "SPI.h"
#include "Timer.h"
#include "FIFO.h"
#include "tm4c123gh6pm.h"
#include <stdbool.h>
#include <stdint.h>
```

### Macros

- #define **NUM_COLS** (uint16_t) 240
- #define **NUM_ROWS** (uint16_t) 320

### Functions

- void **ILI9341_Init** (void)

  *Initialize the LCD driver, the SPI module, and Timer2A.*
- void ILI9341_resetHard (void)

  *Perform a hardware reset of the LCD driver.*
- void ILI9341_resetSoft (void)

  *Perform a software reset of the LCD driver.*
- void ILI9341_setSleepMode (bool isSleeping)

  *Enter or exit sleep mode. The LCD driver is in sleep mode by default upon powering on or either kind of reset.*
- void ILI9341_setDispMode (bool isNormal, bool isFullColors)

  *Set the display area and color expression.*
- void ILI9341_setPartialArea (uint16_t rowStart, uint16_t rowEnd)

  *Set the partial display area for partial mode. Call before activating partial mode via ILI9341_setDisplayMode().*
- void ILI9341_setDispInversion (bool is_ON)

  *Toggle display inversion. Turning `ON` causes colors to be inverted on the display.*
- void ILI9341_setDispOutput (bool is_ON)

  *Turn display output ON or OFF. This function clears the display and stops outputting to the display area, but does not affect frame memory or power.*
- void ILI9341_setScrollArea (uint16_t topFixedArea, uint16_t vertScrollArea, uint16_t bottFixedArea)

  *Set the vertical scrolling area of the display. The sum of the three parameters should be equal to the max number of rows `NUM_ROWS = 320`.*
- void ILI9341_setScrollStart (uint16_t startRow)

  *Set the start row for vertical scrolling.*
- 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 (bool is_16bit)

  *Set the pixel format to be 16-bit (65K colors) or 18-bit (262K colors).*
- void **ILI9341_NoOpCmd** (void)

  *Send the "No Operation" command (`NOP = 0x00`) to the LCD driver. Can be used to terminate the "Memory Write" (`RAMWR`) and "Memory Read" (`RAMRD`) commands, but does nothing otherwise.*
- void ILI9341_setFrameRateNorm (uint8_t divisionRatio, uint8_t clocksPerLine)

  *TODO: Write brief.*

- void ILI9341_setFrameRateIdle (uint8_t divisionRatio, uint8_t clocksPerLine)

  *TODO: Write brief.*
- void **ILI9341_setBlankingPorch** (uint8_t vpf, uint8_t vbp, uint8_t hfp, uint8_t hbp)

  *TODO: Write.*
- 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.*
- void ILI9341_setRowAddress (uint16_t startRow, uint16_t endRow)

  *not using backlight, so these aren't necessary*
- void ILI9341_setColAddress (uint16_t startCol, uint16_t endCol)

  *Sets the start/end rows to be written to.*
- void ILI9341_writeMemCmd (void)

  *Sends the "Write Memory" (RAMWR) command to the LCD driver, signalling that incoming data should be written to memory.*
- void ILI9341_writePixel (uint8_t red, uint8_t green, uint8_t blue, bool is_16bit)

  *Write a single pixel to frame memory.*

### 6.29.1 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 SPI (serial peripheral interface) protocol.

## 6.30 Led.c File Reference

Source code for LED module.

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

**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`.*
- 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`).*

#### 6.30.1 Detailed Description

Source code for LED module.

**Author**

Bryan McElvy

## 6.31 Led.h File Reference

Interface for LED module.

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

### Macros

- #define **LED_POOL_SIZE** 3

### Functions

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

### 6.31.1 Detailed Description

Interface for LED module.

**Author**

Bryan McElvy

## 6.32 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>
```

**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]

### 6.32.1 Detailed Description

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

**Author**

Bryan McElvy

## 6.33 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 "lookup.h"
#include "arm_math_types.h"
#include <stdbool.h>
#include <stdint.h>
```

### Macros

- #define **DAQ_BUFFER_SIZE** 128
- #define **LCD_TOP_LINE** (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 * **input_fifo_ptr** = 0
- volatile uint32_t **input_buffer** [DAQ_BUFFER_SIZE] = { 0 }
- volatile bool **sampleReady** = false

#### 6.33.1 Detailed Description

Test script for the data acquisition (DAQ) module.

**Author**

Bryan McElvy

## 6.34 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>
```

### Functions

- int **main** (void)

#### 6.34.1 Detailed Description

Test script for Debug module.

**Author**

Bryan McElvy

## 6.35 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>
```

### 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**

#### 6.35.1 Detailed Description

Test script for FIFO buffer.

**Author**

Bryan McElvy

## 6.36 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>
```

### 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**

#### 6.36.1 Detailed Description

Test script for writing images onto the display.

**Author**

Bryan McElvy

## 6.37 test_lcd_scroll.c File Reference

Test script for writing different colors on the LCD.

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

### Macros

- #define **LED_PINS** (GPIO_Pin_t)(GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3)
- #define **TOP_LINE_OFFSET** (uint16_t) 180
- #define **TOP_LINE_THICKNESS** (uint16_t) 5
- #define **DX** (uint16_t) 5
- #define **DY** (uint16_t) 10
- #define **COL_Y_MIN** (uint16_t) 0
- #define **COL_Y_MAX** (uint16_t) 177

### Functions

- int **main** (void)

#### 6.37.1 Detailed Description

Test script for writing different colors on the LCD.

**Author**

Bryan McElvy

## 6.38 test_pll.c File Reference

Test script for the PLL module.

```
#include "PLL.h"
#include "SysTick.h"
#include "tm4c123gh6pm.h"
```

### Macros

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

### Functions

- void **GPIO_PortF_Init** (void)
- int **main** ()

#### 6.38.1 Detailed Description

Test script for the PLL module.

**Author**

Bryan McElvy

## 6.39 test_spi.c File Reference

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

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

### Functions

- int **main** ()

#### 6.39.1 Detailed Description

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

**Author**

Bryan McElvy

## 6.40 test_systick_int.c File Reference

Test script for SysTick interrupts.

```
#include "PLL.h"
#include "SysTick.h"
#include "tm4c123gh6pm.h"
```

### 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

#### 6.40.1 Detailed Description

Test script for SysTick interrupts.

**Author**

Bryan McElvy

## 6.41 test_timer1_int.c File Reference

Test script for Timer1A interrupts.

```
#include "GPIO.h"
#include "PLL.h"
#include "Timer.h"
#include "tm4c123gh6pm.h"
```

### Functions

- int **main** (void)
- void **Timer1A_Handler** (void)

### Variables

- uint8_t **is_led_on** = 0

### 6.41.1 Detailed Description

Test script for Timer1A interrupts.

**Author**

Bryan McElvy

## 6.42 test_uart_interrupt.c File Reference

(**DISABLED**) Test script for writing to serial port via UART0

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

### Functions

- int **main** (void)

### Variables

- const uint8_t COLOR_LIST [8]
- const char * COLOR_NAMES [8]

### 6.42.1 Detailed Description

(**DISABLED**) Test script for writing to serial port via UART0

**Author**

Bryan McElvy

### 6.42.2 Variable Documentation

#### COLOR_LIST

```
const uint8_t COLOR_LIST[8]
```

**Initial value:**

```
= { 0,        LED_RED,  LED_YELLOW, LED_GREEN,
                            LED_CYAN, LED_BLUE, LED_PURPLE, LED_WHITE }
```

#### COLOR_NAMES

```
const char* COLOR_NAMES[8]
```

**Initial value:**

```
= { "BLACK\n", "RED\n",  "YELLOW\n", "GREEN\n",
                            "CYAN\n",  "BLUE\n", "PURPLE\n", "WHITE\n" }
```

## 6.43 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"
```

### Functions

- int **main** (void)

### 6.43.1 Detailed Description

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

**Author**

Bryan McElvy

## 6.44 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 "Timer.h"
#include "UART.h"
```

### Functions

- int **main** (void)

### Variables

- volatile unsigned char **in_char**
- uint32_t **counter**

#### 6.44.1 Detailed Description

Test script for writing to serial port via UART0.

**Author**

Bryan McElvy

## 6.45 test_userctrl.c File Reference

Test file for GPIO/UserCtrl modules and GPIO interrupts.

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

### Functions

- int **main** ()

#### 6.45.1 Detailed Description

Test file for GPIO/UserCtrl modules and GPIO interrupts.

**Author**

Bryan McElvy

# Index