uHeartMonitor: An ECG-based Heart Rate Monitor

Generated by Doxygen 1.9.8

1 uHeartMonitor: An ECG-based Heart Rate Monitor

option HTML option LaTeX

1.1 Introduction

1.1.1 Background

Electrocardiography (or ***ECG***) is a diagnostic technique in which the electrical activity of a patient's heart is captured as time series data (AKA the ECG signal) and analyzed to assess cardiovascular health. Specifically, the ECG signal can be analyzed to detect biomarkers for cardiovascular diseases like arrhythmia, myocardial infarction, etc. which manifest as abnormalities in the ECG waveform. In clinical environments, ECG is performed using machines that implement the required hardware and software to acquire, process, and analyze the ECG signal. This must be done in such a way that preserves the important information within the signal (specifically the shape of the ECG waveform) while also maintaining the safety of the patient [1].

The ECG waveform consists of 5 smaller "waves" – the P, Q, R, S, and T waves – that each give information on a patient's cardiac health both individually and collectively. The term ***QRS complex*** refers to the part of the ECG waveform that is generally taken to be the heart "beat". Thus, ECG-based heart rate monitors commonly use a category of algorithms called ***QRS detectors*** to determine the locations of the R-peaks within a block of ECG signal data and calculate the time period between each adjacent peak (i.e. the ***RR interval***) [2]. The RR interval is related to the heart rate by this equation:

$RR = \frac{60}{HR}$

...where RR is the time in $[s]$ between two adjacent R peaks, and HR is the heart rate in $[bpm]$ (beats per minute).

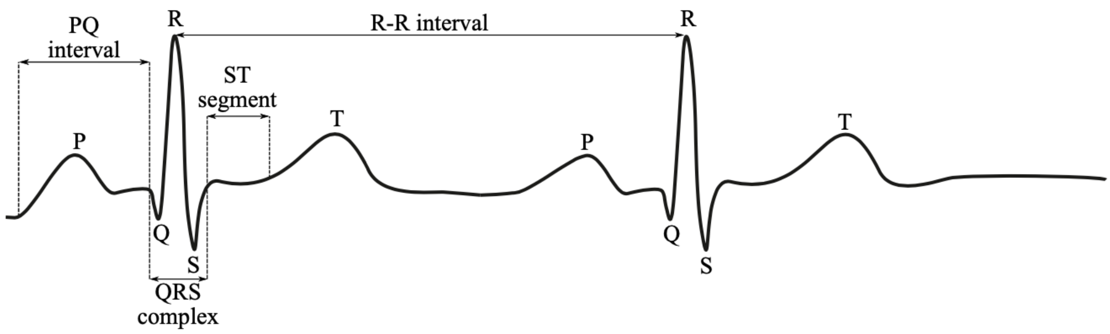


Figure 3. Sample ECG curve.

The ***HeartMonitor*** is an embedded system that implements the Pan-Tompkins algorithm for QRS detection. The system consists of both hardware and software that cooperate to achieve this task while also visually outputting the ECG waveform and heart rate to a liquid crystal display (LCD). The text below and the contents of this repository reflect the current progress made, but the end goal is to have the full system mounted on 1-2 printed circuit boards (PCBs) situated inside an insulated enclosure.

1.1.2 Motivation

My primary motivations for doing this project are:

- Learning more about and gaining exposure to the many different concepts, tools, and challenges involved in embedded systems engineering
- Applying the skills and knowledge I gained from previous coursework, including but not limited to:
  - BIOE 4315: Bioinstrumentation
  - BIOE 4342: Biomedical Signal Processing
  - COSC 2306: Data Programming
  - `Embedded Systems - Shape the World`
- Showing tangible proof of qualification for junior-level embedded software engineering roles to potential employers

I also hope that anyone interested in any of the fields of knowledge relevant to this project (biomedical/electrical/computer/software engineering) will find this helpful to look at or even use in their own projects.

1.1.3 Disclaimer

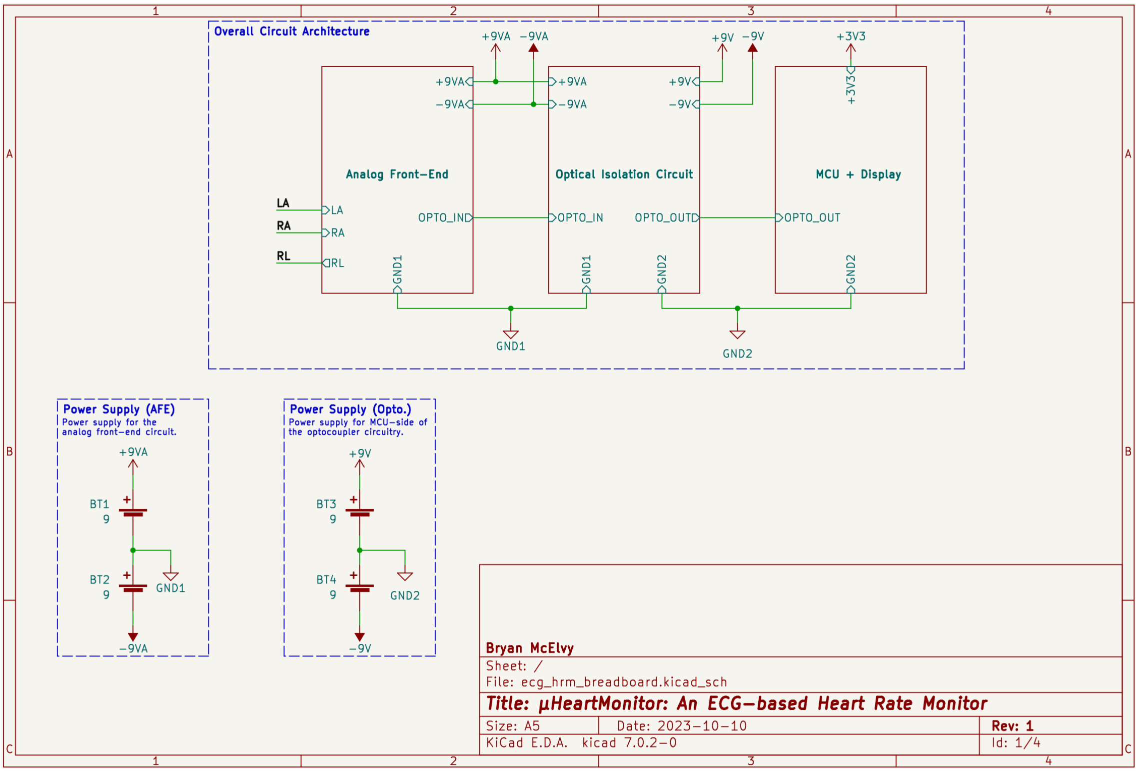
This project is neither a product nor a medical device (by any legal definition, anyway), and is not intended to be either or both of things now or in the future. It is simply a passion project.

1.1.4 Key Terms

- Electrocardiogram/Electrocardiography (ECG)
- Heart rate
- Heart rate monitor
- QRS complex
- QRS detector
- RR interval

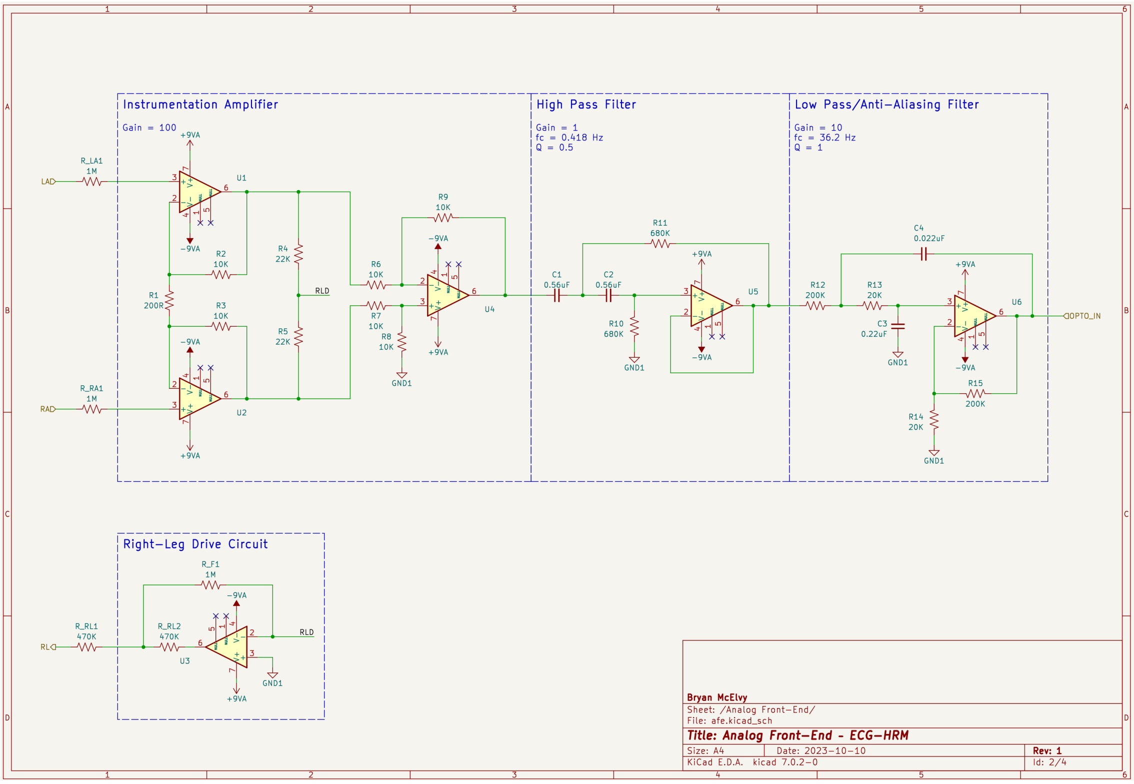
1.2 Materials & Methods

1.2.1 Hardware Design



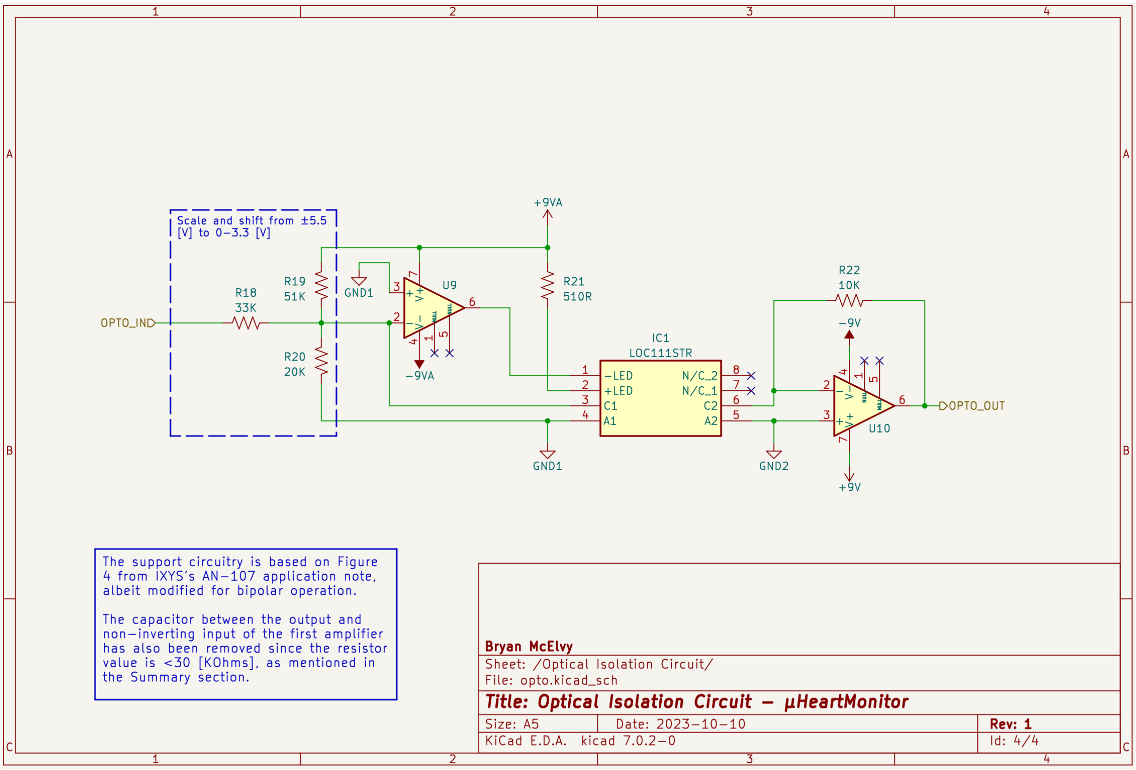
The hardware is divided into three modules: the analog-front end (AFE), the optical isolation circuit, and the micro-controller/display circuit.

Analog-Front End



The AFE consists of an instrumentation amplifier with a gain of 100; a 2nd-order Sallen-Key high-pass filter with a gain of 1 and a cutoff frequency of $\sim 0.5\ Hz$; and a 2nd-order Sallen-Key low-pass filter with a passband gain of 11 and a cutoff frequency of $\sim 40\ Hz$. The overall gain is 1100

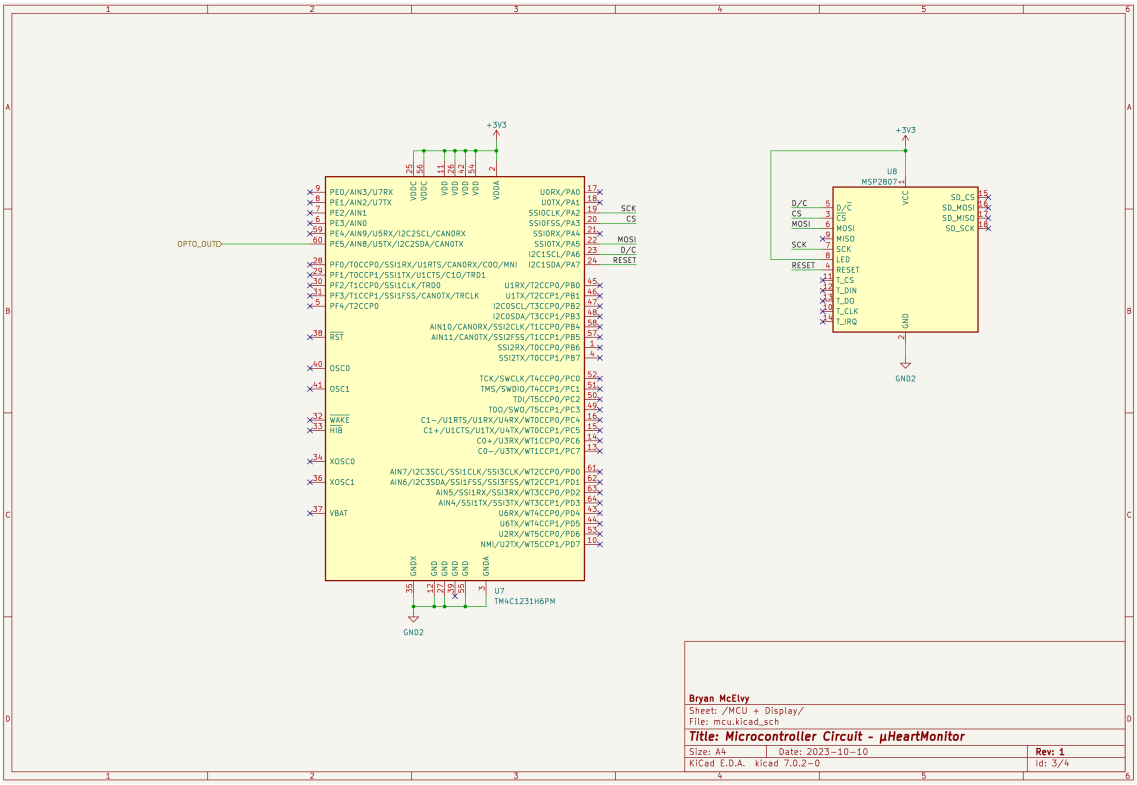
Optical Isolation Circuitry



The optical isolation circuit uses a linear optocoupler to transmit the ECG signal from the analog-front end circuit to the microcontroller circuit. This circuitry serves as a safety measure against power surges and other potential hazards that can occur as a result of connecting someone directly to mains power (for example, death).

It also has three resistors on the AFE-side that effectively shift the signal from the projected output range of $\pm\ 5.5\ V$ to the range $[0, 3.5)\ V$, which is necessary for both the optocoupler and the microcontroller's built-in analog-to-digital converter (ADC) circuitry.

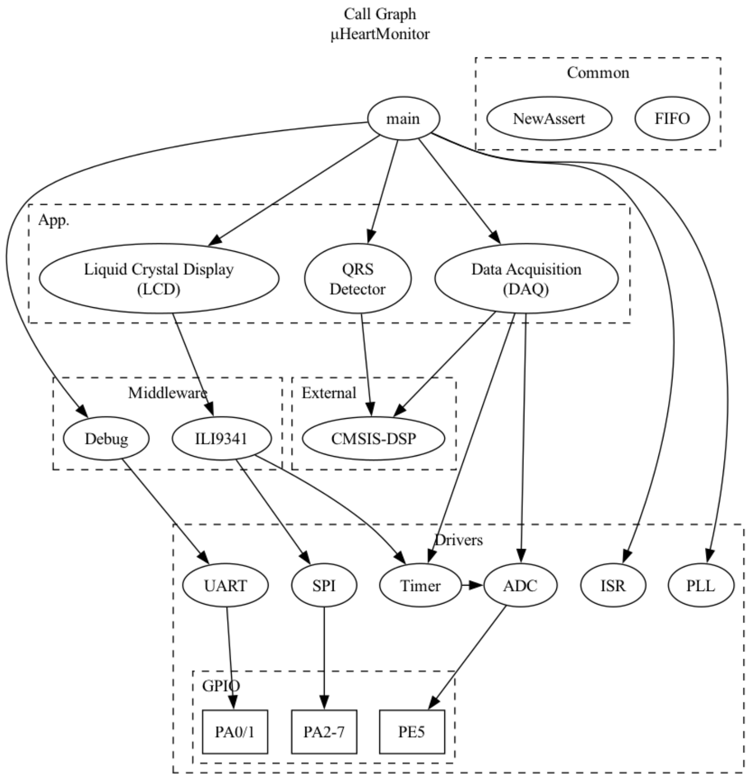
Microcontroller Circuit



The microcontroller circuit currently consists of a TM4C123 microcontroller mounted on a LaunchPad evaluation kit, and an MSP2807 liquid crystal display (LCD).

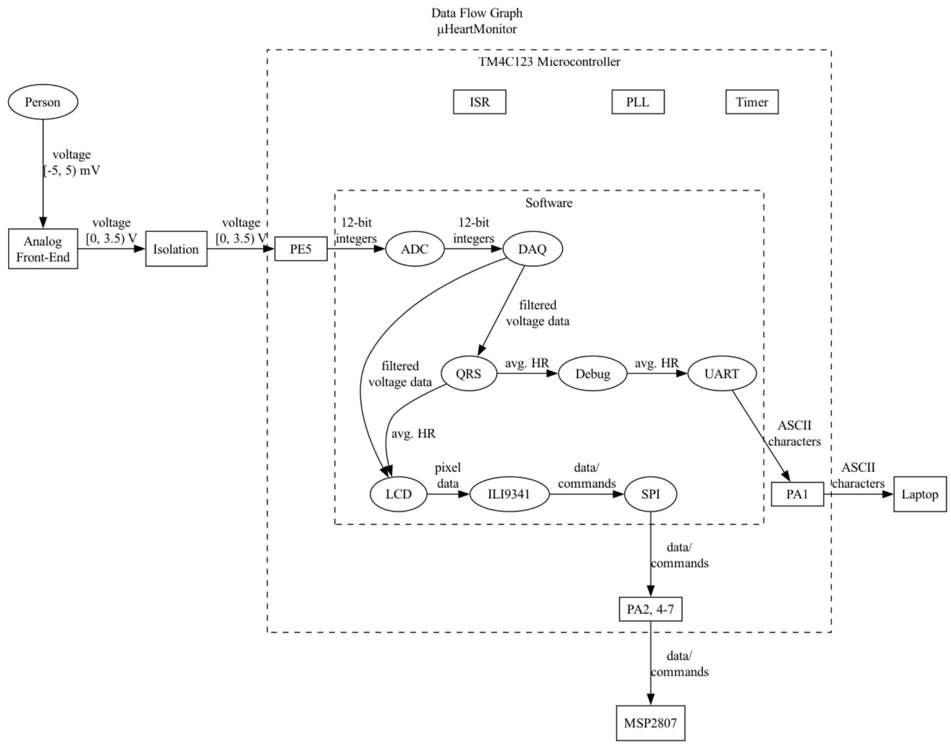
1.2.2 Software Architecture

The software has a total of 14 modules, 11 of which are (somewhat loosely) divided into three layers: application-specific software, middleware, and device drivers. The call graph and data flow graph visually represent the software architecture.



This graph shows which modules communicate with (or "call") each other. Each arrow points from the "caller" to the "callee".

It also somewhat doubles as an `#include` dependency graph.



This graph shows the flow of information from the patient to the LCD (and also the laptop).

Device Drivers

The device driver layer consists of software modules that interface directly with the microcontroller's built-in peripheral devices.

See also

Device Drivers

Middleware

The middleware layer consists of higher-level device drivers that interface with some hardware connected to one of the built-in peripherals (i.e. the Debug module connects to UART and the ILI9341 module primarily uses SPI).

See also

Middleware

Application Software

The application software layer has modules that are at least partially, if not completely built for this project. This layer includes the data acquisition module, whose functions handle receiving raw input samples and denoising them; the QRS detector, which analyzes the filtered signal to determine the average heart rate; and the LCD module, which plots the ECG waveform and displays the heart rate.

See also

Application Software

External

This "layer" includes modules/libraries/files that were not written (or at least heavily altered) by me. It currently only contains portions of ARM's CMSIS-Core and CMSIS-DSP libraries.

Common

The "common" modules are general-purpose modules that don't necessarily fit into the above categories/layers. This category includes the "Fifo" module, which contains a ring buffer-based implementation of the FIFO buffer (AKA "queue") data structure; and "NewAssert", which is essentially just an implementation of the `assert` macro that causes a breakpoint (and also doesn't use up as much RAM as the standard implementation does).

See also

Common

1.3 Current Results

Video Demonstration: `YouTube Link`

The project is currently implemented using 2 breadboards and a Tiva C LaunchPad development board. The manual tests I've been running use a clone of the JDS6600 signal generator, which I loaded a sample ECG waveform from the MIT-BIH arrhythmia database onto using scripts in the corresponding folder in the `/tools` directory. As can be seen in the video demonstration, the calculated heart rate isn't 100% correct at the moment, but still gets relatively close.

1.4 To-do

1.4.1 Hardware

- Design a custom PCB
  - Replace most of the AFE circuitry with an AFE IC (e.g. AD8232)
  - Add electrostatic discharge (ESD) protection
  - Add decoupling capacitors

1.4.2 Software

- Rework the structure of/relationship between the LCD and ILI9341 modules
- Refactor ADC module to be more general
- Refactor SPI module to be more general
- Remove statically-allocated data structures for unused Timers and GPIO ports
- Add remaining parts of the Pan-Tompkins algorithm
  - Thresholding procedure for bandpass-filtered signal (not just integrated signal)
  - Search-back procedure
  - T-wave discrimination
- Add heart rate variability (HRV) calculation
- Move CMSIS-DSP filters from DAQ and QRS modules to their own module
- Expand the automated test suite

1.5 Build Instructions

1.5.1 Hardware

WIP

1.5.2 Software

WIP

1.6 References

[1] J. Pan and W. J. Tompkins, “A Real-Time QRS Detection Algorithm,” IEEE Trans. Biomed. Eng., vol. BME-32, no. 3, pp. 230–236, Mar. 1985, doi: 10.1109/TBME.1985.325532.

[2] R. Martinek et al., “Advanced Bioelectrical Signal Processing Methods: Past, Present and Future Approach—↩ Part I: Cardiac Signals,” Sensors, vol. 21, no. 15, p. 5186, Jul. 2021, doi: 10.3390/s21155186.

[3] C. Ünsalan, M. E. Yücel, and H. D. Gürhan, Digital Signal Processing using Arm Cortex-M based Microcontrollers: Theory and Practice. Cambridge: ARM Education Media, 2018.

[4] B. B. Winter and J. G. Webster, “Driven-right-leg circuit design,” IEEE Trans Biomed Eng, vol. 30, no. 1, pp. 62–66, Jan. 1983, doi: 10.1109/tbme.1983.325168.

[5] J. Valvano, Embedded Systems: Introduction to ARM Cortex-M Microcontrollers, 5th edition. Jonathan Valvano, 2013.

[6] S. W. Smith, The Scientist and Engineer’s Guide to Digital Signal Processing, 2nd edition. San Diego, Calif: California technical Publishin, 1999.

2 Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

4 File Index

4.1 File List

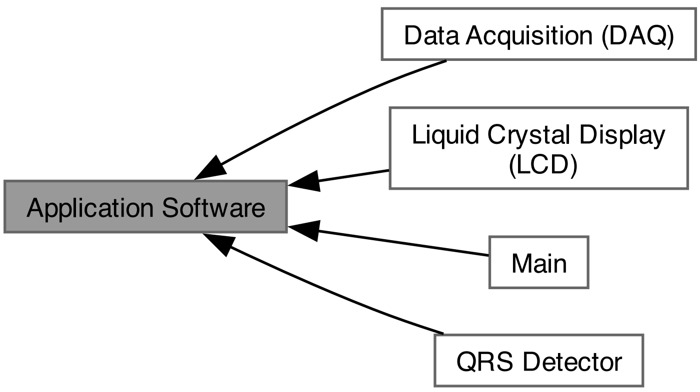
Here is a list of all documented files with brief descriptions:

5 Topic Documentation

5.1 Application Software

Application-specific software modules.

Collaboration diagram for Application Software:



Modules

- Data Acquisition (DAQ)

  Module for managing data acquisition (DAQ) functions.
- Liquid Crystal Display (LCD)

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

  Module for analyzing ECG data to determine heart rate.
- Main

  Main program file.

5.1.1 Detailed Description

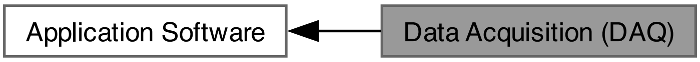
Application-specific software modules.

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

5.1.2 Data Acquisition (DAQ)

Module for managing data acquisition (DAQ) functions.

Collaboration diagram for Data Acquisition (DAQ):



Files

- file DAQ.c

  Source code for DAQ module.
- file DAQ.h

  Application software for handling data acquision (DAQ) functions.
- file DAQ_lookup.c

  Source code for DAQ module's lookup table.

Macros

- #define **SAMPLING_PERIOD_MS** 5

  sampling period in ms ($T_s = \frac{1}{f_s}$)
- #define **DAQ_LOOKUP_MAX** ((float32_t) 5.5f)

  maximum lookup table value
- #define **DAQ_LOOKUP_MIN** ((float32_t) (-5.5f))

  minimum lookup table value

Variables

- static const float32_t **DAQ_LOOKUP_TABLE** [4096]

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

Digital Filters

- enum {
  NUM_STAGES_NOTCH = 6 , **NUM_COEFFS_NOTCH** = NUM_STAGES_NOTCH ∗ 5 , **STATE_BUFF_SIZE_NOTCH** = NUM_STAGES_NOTCH ∗ 4 , **NUM_STAGES_BANDPASS** = 4 ,
  NUM_COEFFS_DAQ_BANDPASS = NUM_STAGES_BANDPASS ∗ 5 , **STATE_BUFF_SIZE_BANDPASS** = NUM_STAGES_BANDPASS ∗ 4 }
- typedef arm_biquad_casd_df1_inst_f32 **Filter_t**
- static const float32_t COEFFS_NOTCH [NUM_COEFFS_NOTCH]
- static const float32_t COEFFS_BANDPASS [NUM_COEFFS_DAQ_BANDPASS]
- static float32_t **stateBuffer_Notch** [STATE_BUFF_SIZE_NOTCH]
- static const Filter_t **notchFiltStruct** = { NUM_STAGES_NOTCH, stateBuffer_Notch, COEFFS_NOTCH }
- static const Filter_t ∗const **notchFilter** = &notchFiltStruct
- static float32_t **stateBuffer_Bandpass** [STATE_BUFF_SIZE_BANDPASS]
- static const Filter_t bandpassFiltStruct
- static const Filter_t ∗const **bandpassFilter** = &bandpassFiltStruct

Initialization

- void DAQ_Init (void)

  Initialize the data acquisition (DAQ) module.

Reading Input Data

- uint16_t DAQ_readSample (void)

  Read a sample from the ADC.
- void DAQ_acknowledgeInterrupt (void)

  Acknowledge the ADC interrupt.
- float32_t DAQ_convertToMilliVolts (uint16_t sample)

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

Digital Filtering Functions

- float32_t DAQ_NotchFilter (volatile float32_t xn)

  Apply a 60 [Hz] notch filter to an input sample.
- float32_t DAQ_BandpassFilter (volatile float32_t xn)

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

5.1.2.1 Detailed Description

Module for managing data acquisition (DAQ) functions.

5.1.2.2 Function Documentation

DAQ_Init()

```
void DAQ_Init (
            void )
```

Initialize the data acquisition (DAQ) module.

Postcondition

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

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

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

DAQ_readSample()

```
uint16_t DAQ_readSample (
            void )
```

Read a sample from the ADC.

Precondition

Initialize the DAQ module.

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

Parameters

out	*sample*	12-bit sample in range `[0x000, 0xFFF]`

Postcondition

The sample can now be converted to millivolts.

See also

DAQ_convertToMilliVolts()

DAQ_acknowledgeInterrupt()

```
void DAQ_acknowledgeInterrupt (
            void )
```

Acknowledge the ADC interrupt.

Precondition

This should be used within an interrupt handler.

DAQ_NotchFilter()

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

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

Precondition

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

Parameters

in	*xn*	Raw input sample
out	*yn*	Filtered output sample

Postcondition

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

See also

DAQ_BandpassFilter()

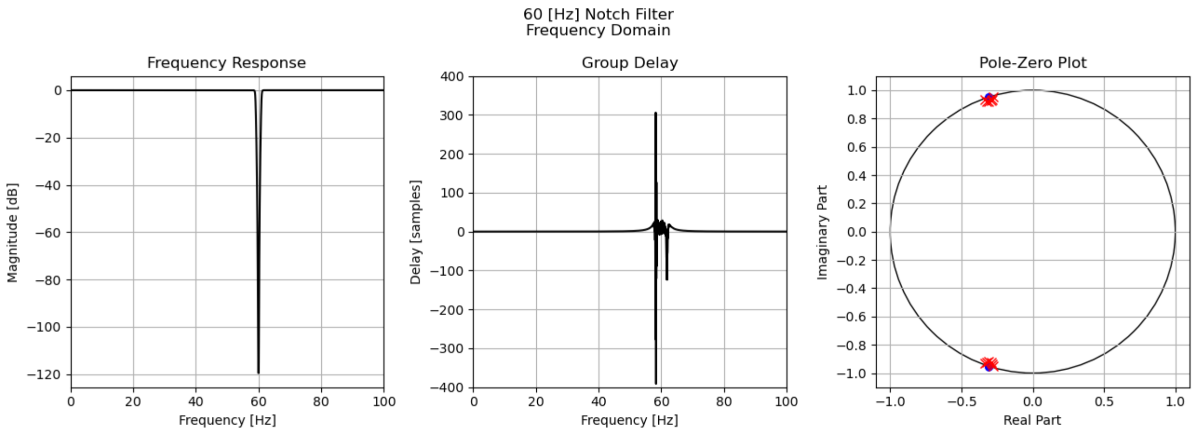


Figure 1 Frequency domain parameters for the notch filter.

DAQ_BandpassFilter()

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

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

Precondition

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

Parameters

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

Postcondition

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

See also

DAQ_NotchFilter()

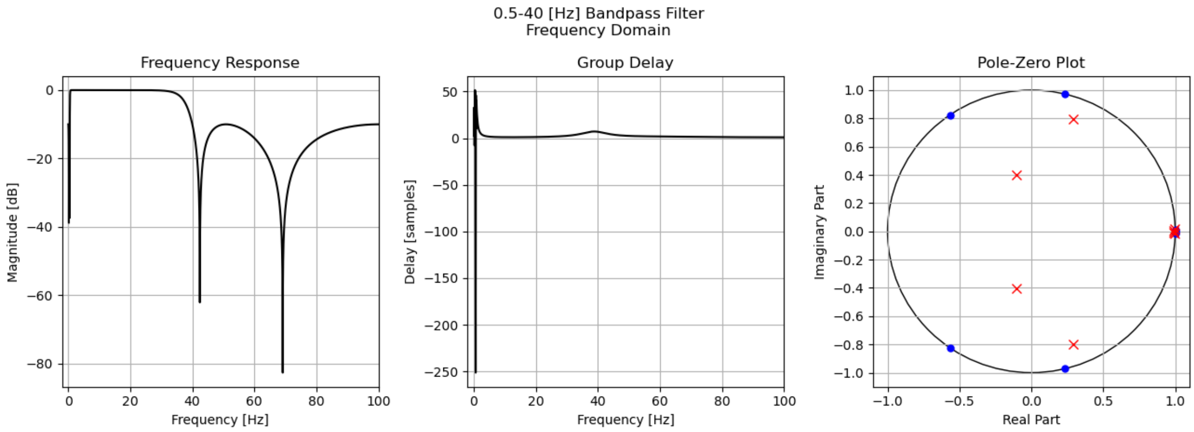


Figure 2 Frequency domain parameters for the bandpass filter.

DAQ_convertToMilliVolts()

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

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

Precondition

Read a sample from the ADC.

Parameters

`in`	*sample*	12-bit sample in range `[0x000, 0xFFF]`
`out`	*xn*	Voltage value in range $[-5.5, 5.5)[mV]$

Postcondition

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

See also

DAQ_readSample()

Note

Defined in DAQ_lookup.c rather than DAQ.c.

5.1.2.3 Variable Documentation

COEFFS_NOTCH

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

Initial value:

```
= {

    0.8856732845306396f, 0.5476464033126831f, 0.8856732845306396f,
    -0.5850160717964172f, -0.9409302473068237f,

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

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

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

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

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

COEFFS_BANDPASS

```
const float32_t COEFFS_BANDPASS[NUM_COEFFS_DAQ_BANDPASS]  [static]
```

Initial value:

```
= {

    0.3240305185317993f, 0.3665695786476135f, 0.3240305185317993f,
    -0.20968256890773773f, -0.1729172021150589f,

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

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

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

bandpassFiltStruct

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

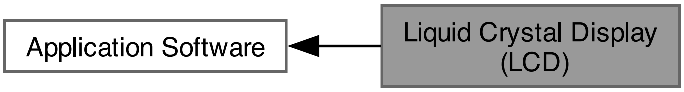
Initial value:

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

5.1.3 Liquid Crystal Display (LCD)

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

Collaboration diagram for Liquid Crystal Display (LCD):



Files

- file Font.c
  Contains bitmaps for a selection of ASCII characters.
- file LCD.c
  Source code for LCD module.
- file LCD.h
  Header file for LCD module.

Macros

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

Functions

- static void LCD_plotSample (uint16_t x, uint16_t y, uint8_t color)
  Plot a sample at coordinates `(x, y)`.

Variables

- const uint8_t ∗const **FONT_ARRAY** [128]
- struct {
  uint16_t **x1**
  starting x-value in range [0, x2]
  uint16_t **x2**
  ending x-value in range [0, NUM_ROWS)
  uint16_t **y1**
  starting y-value in range [0, y2]
  uint16_t **y2**
  ending x-value in range [0, NUM_COLS)
  uint16_t **lineNum**
  line number for text; in range `[0, NUM_LINES)`
  uint16_t **colNum**
  column number for text; in range `[0, NUM_COLS)`
  uint8_t **color**
  bool **isInit**
  if `true`*, LCD has been initialized*
  } **lcd** = { 0 }
- const uint8_t ∗const **FONT_ARRAY** [128]

Initialization & Configuration

- enum **LCD_PLOT_INFO** { **LCD_X_MAX** = ILI9341_NUM_ROWS - 1 , **LCD_Y_MAX** = ILI9341_NUM_COLS - 1 }
- enum **LCD_COLORS** {
  LCD_BLACK = 0x00 ^ 0x07 , **LCD_RED** = 0x04 ^ 0x07 , **LCD_GREEN** = 0x02 ^ 0x07 , **LCD_BLUE** = 0x01 ^ 0x07 ,
  LCD_YELLOW = 0x06 ^ 0x07 , **LCD_CYAN** = 0x03 ^ 0x07 , **LCD_PURPLE** = 0x05 ^ 0x07 , **LCD_WHITE** = 0x07 ^ 0x07 }
- void LCD_Init (void)

  Initialize the LCD.
- void LCD_setOutputMode (bool isOn)

  Toggle display output ON or OFF (OFF by default).
- void LCD_setX (uint16_t x1, uint16_t x2)

  Set new x-coordinates to be written to. $0 <= x1 <= x2 <= X_{MAX}$.
- void LCD_setY (uint16_t y1, uint16_t y2)

  Set new y-coordinates to be written to. $0 <= y1 <= y2 <= Y_{MAX}$.
- void LCD_setColor (uint8_t color)

  Set the color value.

Writing

- enum **LCD_WRITING_INFO** { **HEIGHT_CHAR** = 8 , **LEN_CHAR** = 5 , **NUM_LINES** = 30 , **NUM_COLS** = 64 }
- void LCD_setCursor (uint16_t lineNum, uint16_t colNum)

  Set the cursor to line x, column y.
- void **LCD_writeChar** (unsigned char inputChar)
- void **LCD_writeStr** (void $*$asciiString)
- void **LCD_writeInt** (int32_t num)
- void **LCD_writeFloat** (float num)

ASCII Characters (Punctuation)

- static const uint8_t FONT_SPACE [8]
- static const uint8_t FONT_PERIOD [8]
- static const uint8_t FONT_COLON [8]

ASCII Characters (Numbers)

- static const uint8_t FONT_0 [8]
- static const uint8_t FONT_1 [8]
- static const uint8_t FONT_2 [8]
- static const uint8_t FONT_3 [8]
- static const uint8_t FONT_4 [8]
- static const uint8_t FONT_5 [8]
- static const uint8_t FONT_6 [8]
- static const uint8_t FONT_7 [8]
- static const uint8_t FONT_8 [8]
- static const uint8_t FONT_9 [8]

ASCII Characters (Uppercase Letters)

- static const uint8_t FONT_UPPER_A [8]
- static const uint8_t FONT_UPPER_B [8]
- static const uint8_t FONT_UPPER_C [8]
- static const uint8_t FONT_UPPER_D [8]
- static const uint8_t FONT_UPPER_E [8]
- static const uint8_t FONT_UPPER_F [8]
- static const uint8_t FONT_UPPER_G [8]
- static const uint8_t FONT_UPPER_H [8]
- static const uint8_t FONT_UPPER_I [8]
- static const uint8_t FONT_UPPER_J [8]
- static const uint8_t FONT_UPPER_K [8]
- static const uint8_t FONT_UPPER_L [8]
- static const uint8_t FONT_UPPER_M [8]
- static const uint8_t FONT_UPPER_N [8]
- static const uint8_t FONT_UPPER_O [8]
- static const uint8_t FONT_UPPER_P [8]
- static const uint8_t FONT_UPPER_Q [8]
- static const uint8_t FONT_UPPER_R [8]
- static const uint8_t FONT_UPPER_S [8]
- static const uint8_t FONT_UPPER_T [8]
- static const uint8_t FONT_UPPER_U [8]
- static const uint8_t FONT_UPPER_V [8]
- static const uint8_t FONT_UPPER_W [8]
- static const uint8_t FONT_UPPER_X [8]
- static const uint8_t FONT_UPPER_Y [8]
- static const uint8_t FONT_UPPER_Z [8]

ASCII Characters (Lowercase Letters)

- static const uint8_t FONT_LOWER_A [8]
- static const uint8_t FONT_LOWER_B [8]
- static const uint8_t FONT_LOWER_C [8]
- static const uint8_t FONT_LOWER_D [8]
- static const uint8_t FONT_LOWER_E [8]
- static const uint8_t FONT_LOWER_F [8]
- static const uint8_t FONT_LOWER_G [8]
- static const uint8_t FONT_LOWER_H [8]
- static const uint8_t FONT_LOWER_I [8]
- static const uint8_t FONT_LOWER_J [8]
- static const uint8_t FONT_LOWER_K [8]
- static const uint8_t FONT_LOWER_L [8]
- static const uint8_t FONT_LOWER_M [8]
- static const uint8_t FONT_LOWER_N [8]
- static const uint8_t FONT_LOWER_O [8]
- static const uint8_t FONT_LOWER_P [8]
- static const uint8_t FONT_LOWER_Q [8]
- static const uint8_t FONT_LOWER_R [8]
- static const uint8_t FONT_LOWER_S [8]
- static const uint8_t FONT_LOWER_T [8]
- static const uint8_t FONT_LOWER_U [8]
- static const uint8_t FONT_LOWER_V [8]
- static const uint8_t FONT_LOWER_W [8]
- static const uint8_t FONT_LOWER_X [8]
- static const uint8_t FONT_LOWER_Y [8]
- static const uint8_t FONT_LOWER_Z [8]

Helper Functions

- static void LCD_drawLine (uint16_t center, uint16_t lineWidth, bool is_horizontal)

  Helper function for drawing straight lines.
- static void **LCD_updateCursor** (void)

  Update the cursor for after writing text on the display.

Drawing

- void LCD_Draw (void)

  Draw on the LCD.
- void LCD_Fill (void)

  Fill the display with a single color.
- void LCD_drawHoriLine (uint16_t yCenter, uint16_t lineWidth)

  Draw a horizontal line across the entire display.
- void LCD_drawVertLine (uint16_t xCenter, uint16_t lineWidth)

  Draw a vertical line across the entire display.
- void LCD_drawRectangle (uint16_t x1, uint16_t dx, uint16_t y1, uint16_t dy)

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

5.1.3.1 Detailed Description

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

5.1.3.2 Function Documentation

LCD_drawLine()

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

Helper function for drawing straight lines.

Parameters

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

LCD_Init()

```
void LCD_Init (
            void )
```

Initialize the LCD.

Postcondition

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

LCD_setOutputMode()

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

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

Parameters

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

Postcondition

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

LCD_setX()

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

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

Parameters

in	*x1*	left-most x-coordinate
in	*x2*	right-most x-coordinate

See also

LCD_setY()

LCD_setY()

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

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

Parameters

in	*y1*	lowest y-coordinate
in	*y2*	highest y-coordinate

See also

LCD_setX()

LCD_setColor()

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

Set the color value.

Parameters

in	*color*	Color to use.

Postcondition

Outgoing pixel data will use the selected color.

LCD_Draw()

```
void LCD_Draw (
            void  )
```

Draw on the LCD.

Precondition

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

Postcondition

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

See also

LCD_setX(), LCD_setY(), LCD_setColor()

References ILI9341_writeMemCmd(), and ILI9341_writePixel().

LCD_Fill()

```
void LCD_Fill (
            void  )
```

Fill the display with a single color.

Precondition

Select the desired color to fill the display with.

See also

LCD_setColor()

LCD_drawHoriLine()

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

Draw a horizontal line across the entire display.

Precondition

Select the desired color to use for the line.

Parameters

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

See also

LCD_drawVertLine, LCD_drawRectangle()

LCD_drawVertLine()

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

Draw a vertical line across the entire display.

Precondition

Select the desired color to use for the line.

Parameters

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

See also

LCD_drawHoriLine, LCD_drawRectangle()

LCD_drawRectangle()

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

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

Precondition

Select the desired color to use for the rectangle.

Parameters

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

See also

LCD_Draw(), LCD_Fill(), LCD_drawHoriLine(), LCD_drawVertLine()

LCD_plotSample()

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

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

Parameters

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

See also

LCD_setX(), LCD_setY(), LCD_setColor(), LCD_Draw()

LCD_setCursor()

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

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

Parameters

in	*lineNum*	Line number to place characters. Should be in range `[0, 30)`.
in	*colNum*	Column number to place characters. Should be in range `[0, 64)`.

5.1.3.3 Variable Documentation

FONT_SPACE

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

Initial value:

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

FONT_PERIOD

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

Initial value:

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

FONT_COLON

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

Initial value:

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

FONT_0

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

Initial value:

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

FONT_1

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

Initial value:

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

FONT_2

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

Initial value:

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

FONT_3

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

Initial value:

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

FONT_4

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

Initial value:

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

FONT_5

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

Initial value:

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

FONT_6

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

Initial value:

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

FONT_7

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

Initial value:

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

FONT_8

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

Initial value:

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

FONT_9

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

Initial value:

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

FONT_UPPER_A

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

Initial value:

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

FONT_UPPER_B

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

Initial value:

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

FONT_UPPER_C

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

Initial value:

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

FONT_UPPER_D

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

Initial value:
```
= {
    0x1E,
    0x11,
    0x11,
    0x11,
    0x11,
    0x11,
    0x11,
    0x1E
}
```

FONT_UPPER_E

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

Initial value:
```
= {
    0x1F,
    0x10,
    0x10,
    0x1E,
    0x10,
    0x10,
    0x10,
    0x1F
}
```

FONT_UPPER_F

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

Initial value:
```
= {
    0x1F,
    0x10,
    0x10,
    0x1E,
    0x10,
    0x10,
    0x10,
    0x10
}
```

FONT_UPPER_G

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

Initial value:
```
= {
    0x0E,
    0x11,
    0x10,
    0x10,
    0x17,
    0x11,
    0x11,
    0x0E
}
```

FONT_UPPER_H

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

Initial value:

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

FONT_UPPER_I

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

Initial value:

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

FONT_UPPER_J

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

Initial value:

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

FONT_UPPER_K

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

Initial value:

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

FONT_UPPER_L

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

Initial value:

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

FONT_UPPER_M

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

Initial value:

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

FONT_UPPER_N

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

Initial value:

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

FONT_UPPER_O

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

Initial value:

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

FONT_UPPER_P

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

Initial value:

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

FONT_UPPER_Q

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

Initial value:

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

FONT_UPPER_R

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

Initial value:

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

FONT_UPPER_S

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

Initial value:

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

FONT_UPPER_T

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

Initial value:
```
= {
    0x1F,
    0x04,
    0x04,
    0x04,
    0x04,
    0x04,
    0x04,
    0x04
}
```

FONT_UPPER_U

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

Initial value:
```
= {
    0x11,
    0x11,
    0x11,
    0x11,
    0x11,
    0x11,
    0x11,
    0x0E
}
```

FONT_UPPER_V

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

Initial value:
```
= {
    0x11,
    0x11,
    0x11,
    0x11,
    0x11,
    0x0A,
    0x0A,
    0x04
}
```

FONT_UPPER_W

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

Initial value:
```
= {
    0x11,
    0x11,
    0x11,
    0x15,
    0x15,
    0x1B,
    0x11,
    0x11
}
```

FONT_UPPER_X

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

Initial value:

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

FONT_UPPER_Y

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

Initial value:

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

FONT_UPPER_Z

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

Initial value:

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

FONT_LOWER_A

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

Initial value:

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

FONT_LOWER_B

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

Initial value:
```
= {
    0x10,
    0x10,
    0x1E,
    0x11,
    0x11,
    0x11,
    0x1E,
    0x00
}
```

FONT_LOWER_C

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

Initial value:
```
= {
    0x00,
    0x00,
    0x0E,
    0x10,
    0x10,
    0x11,
    0x0E,
    0x00
}
```

FONT_LOWER_D

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

Initial value:
```
= {
    0x01,
    0x01,
    0x0F,
    0x11,
    0x11,
    0x11,
    0x0F,
    0x00
}
```

FONT_LOWER_E

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

Initial value:
```
= {
    0x00,
    0x00,
    0x0E,
    0x11,
    0x1F,
    0x10,
    0x0E,
    0x00
}
```

FONT_LOWER_F

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

Initial value:

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

FONT_LOWER_G

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

Initial value:

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

FONT_LOWER_H

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

Initial value:

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

FONT_LOWER_I

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

Initial value:

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

FONT_LOWER_J

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

Initial value:

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

FONT_LOWER_K

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

Initial value:

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

FONT_LOWER_L

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

Initial value:

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

FONT_LOWER_M

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

Initial value:

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

FONT_LOWER_N

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

Initial value:

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

FONT_LOWER_O

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

Initial value:

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

FONT_LOWER_P

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

Initial value:

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

FONT_LOWER_Q

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

Initial value:

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

FONT_LOWER_R

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

Initial value:

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

FONT_LOWER_S

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

Initial value:

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

FONT_LOWER_T

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

Initial value:

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

FONT_LOWER_U

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

Initial value:

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

FONT_LOWER_V

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

Initial value:

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

FONT_LOWER_W

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

Initial value:

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

FONT_LOWER_X

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

Initial value:

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

FONT_LOWER_Y

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

Initial value:

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

FONT_LOWER_Z

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

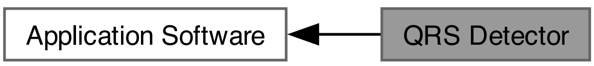
Initial value:

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

5.1.4 QRS Detector

Module for analyzing ECG data to determine heart rate.

Collaboration diagram for QRS Detector:



Files

- file QRS.c

  Source code for QRS detection module.
- file QRS.h

  Header file for QRS detection module.

Macros

- #define **QRS_NUM_FID_MARKS** 40
- #define **FLOAT_COMPARE_TOLERANCE** (float32_t)(1E-5f)
- #define **IS_GREATER**(X, Y) (bool) ((X - Y) > FLOAT_COMPARE_TOLERANCE)
- #define **IS_PEAK**(X_MINUS_1, X, X_PLUS_1) (bool) (IS_GREATER(X, X_MINUS_1) && IS_GREATER(X, X_PLUS_1))
- #define **QRS_SAMP_FREQ** ((uint32_t) 200)
- #define **QRS_SAMP_PERIOD_SEC** ((float32_t) 0.005f)
- #define **QRS_NUM_SAMP** ((uint16_t) (1 << 11))

Variables

- struct {
  bool **isCalibrated**
  float32_t **signalLevel**
  estimated signal level
  float32_t **noiseLevel**
  estimated noise level
  float32_t **threshold**
  amplitude threshold
  uint16_t **fidMarkArray** [QRS_NUM_FID_MARKS]
  float32_t **utilityBuffer1** [QRS_NUM_FID_MARKS]
  array to hold fidMark indices
  float32_t **utilityBuffer2** [QRS_NUM_FID_MARKS]
  } **Detector** = { false, 0.0f, 0.0f, 0.0f, { 0 }, { 0 }, { 0 } }

Digital Filters

- enum {
  NUM_STAGES_BANDPASS = 4 , **NUM_COEFF_HIGHPASS** = NUM_STAGES_BANDPASS ∗ 5 , **STATE_BUFF_SIZE_BANDPASS** = NUM_STAGES_BANDPASS ∗ 4 , **NUM_COEFF_DERFILT** = 5 ,
  BLOCK_SIZE_DERFILT = (1 << 8) , **STATE_BUFF_SIZE_DERFILT** = NUM_COEFF_DERFILT + BLOCK_SIZE_DERFILT - 1 , **NUM_COEFF_MOVAVG** = 10 , **BLOCK_SIZE_MOVAVG** = BLOCK_SIZE_DERFILT ,
  STATE_BUFF_SIZE_MOVAVG = NUM_COEFF_MOVAVG + BLOCK_SIZE_MOVAVG - 1 }
- typedef arm_biquad_casd_df1_inst_f32 **IIR_Filt_t**
- typedef arm_fir_instance_f32 **FIR_Filt_t**
- static const float32_t COEFF_BANDPASS [NUM_COEFF_HIGHPASS]
- static const float32_t COEFF_DERFILT [NUM_COEFF_DERFILT]
- static const float32_t COEFF_MOVAVG [NUM_COEFF_MOVAVG]
- static float32_t **stateBuffer_bandPass** [STATE_BUFF_SIZE_BANDPASS] = { 0 }
- static const IIR_Filt_t **bandpassFiltStruct** = { NUM_STAGES_BANDPASS, stateBuffer_bandPass, COEFF_BANDPASS }
- static const IIR_Filt_t ∗const **bandpassFilter** = &bandpassFiltStruct
- static float32_t **stateBuffer_DerFilt** [STATE_BUFF_SIZE_DERFILT] = { 0 }
- static const FIR_Filt_t **derivativeFiltStruct** = { NUM_COEFF_DERFILT, stateBuffer_DerFilt, COEFF_DERFILT }
- static const FIR_Filt_t ∗const **derivativeFilter** = &derivativeFiltStruct
- static float32_t **stateBuffer_MovingAvg** [STATE_BUFF_SIZE_MOVAVG] = { 0 }
- static const FIR_Filt_t **movingAvgFiltStruct** = { NUM_COEFF_MOVAVG, stateBuffer_MovingAvg, COEFF_MOVAVG }
- static const FIR_Filt_t ∗const **movingAverageFilter** = &movingAvgFiltStruct

Implementation

- static uint8_t QRS_findFiducialMarks (const float32_t yn[ ], uint16_t fidMarkArray[ ])
  Mark local peaks in the input signal y as potential candidates for QRS complexes (AKA "fiducial marks").
- static void QRS_initLevels (const float32_t yn[ ], float32_t ∗sigLvlPtr, float32_t ∗noiseLvlPtr)
  Initialize the signal and noise levels for the QRS detector using the initial block of input signal data.
- static float32_t QRS_updateLevel (const float32_t peakAmplitude, float32_t level)
  Update the signal level (if a fiducial mark is a confirmed peak) or the noise level (if a fiducial mark is rejected).
- static float32_t QRS_updateThreshold (const float32_t signalLevel, const float32_t noiseLevel)
  Update the amplitude threshold used to identify peaks based on the signal and noise levels.

Interface Functions

- void QRS_Init (void)

  Initialize the QRS detector.
- void QRS_Preprocess (const float32_t xn[ ], float32_t yn[ ])

  Preprocess the ECG data to remove noise and/or exaggerate the signal characteristic(s) of interest.
- float32_t QRS_applyDecisionRules (const float32_t yn[ ])

  Calculate the average heart rate (HR) using predetermined decision rules.
- float32_t QRS_runDetection (const float32_t xn[ ], float32_t yn[ ])

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

5.1.4.1 Detailed Description

Module for analyzing ECG data to determine heart rate.

5.1.4.2 Function Documentation

QRS_findFiducialMarks()

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

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

Parameters

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

Postcondition

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

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

QRS_initLevels()

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

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

Parameters

in	*yn*	Array containing the preprocessed ECG signal $y[n]$
in	*sigLvlPtr*	Pointer to variable holding the signal level value.
in	*noiseLvlPtr*	Pointer to variable holding the noise level value.

Postcondition

The signal and noise levels are initialized.

QRS_updateLevel()

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

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

Parameters

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

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

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

QRS_updateThreshold()

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

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

Parameters

in	*signalLevel*	Current signal level.
in	*noiseLevel*	Current noise level.
out	*threshold*	New threshold to use for next comparison.

See also

QRS_updateLevel(), QRS_applyDecisionRules

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

QRS_Init()

```
void QRS_Init (
            void )
```

Initialize the QRS detector.

Note

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

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

QRS_Preprocess()

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

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

Precondition

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

Parameters

in	*xn*	Array of raw ECG signal values.
in	*yn*	Array used to store preprocessed ECG signal values.

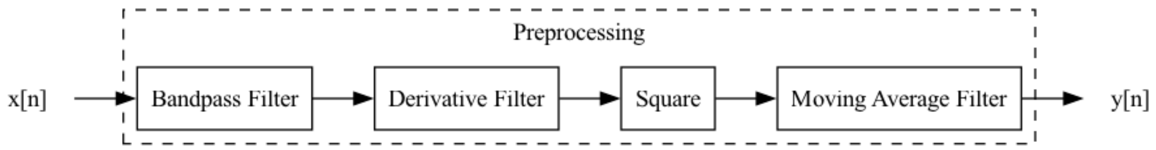
Postcondition

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

See also

QRS_applyDecisionRules()

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



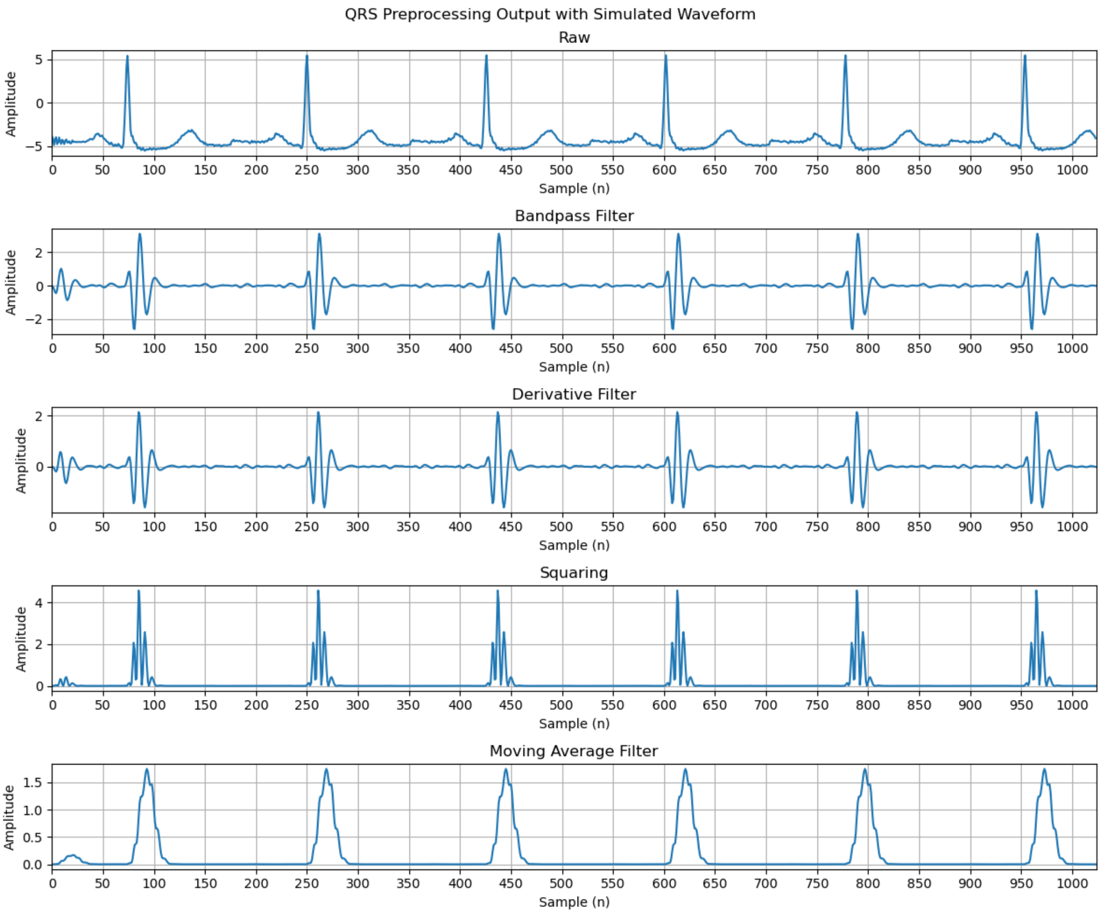


Figure 3 Output of each preprocessing step.

Note

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

QRS_applyDecisionRules()

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

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

Precondition

Preprocess the raw ECG data.

Parameters

in	*yn*	Array of preprocessed ECG signal values.
out	*heartRate*	Average heart rate in [bpm].

Postcondition

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

Warning

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

See also

QRS_Preprocess()

QRS_runDetection()

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

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

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

Parameters

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

Postcondition

`yn` will contain the preprocessed data.

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

See also

QRS_Preprocess(), QRS_applyDecisionRules()

5.1.4.3 Variable Documentation

COEFF_BANDPASS

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

Initial value:

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

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

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

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

COEFF_DERFILT

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

Initial value:

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

COEFF_MOVAVG

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

Initial value:

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

5.1.5 Main

Main program file.

Collaboration diagram for Main:

Files

- file main.c

  Main program file.

Enumerations

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

Functions

- static void DAQ_Handler (void)

  ISR for the data acquisition system.
- static void Processing_Handler (void)

  ISR for intermediate processing of the input data.
- static void LCD_Handler (void)

  ISR for plotting the waveform and outputting the heart rate to the LCD.
- int main (void)

  Main function for the project.

Variables

- static volatile Fifo_t **DAQ_Fifo** = 0
- static volatile uint32_t **DAQ_fifoBuffer** [DAQ_ARRAY_LEN] = { 0 }
- static volatile Fifo_t **QRS_Fifo** = 0
- static volatile uint32_t **QRS_fifoBuffer** [QRS_ARRAY_LEN] = { 0 }
- static volatile Fifo_t **LCD_Fifo1** = 0
- static volatile uint32_t **LCD_fifoBuffer1** [LCD_ARRAY_1_LEN] = { 0 }
- static volatile Fifo_t **LCD_Fifo2** = 0
- static volatile uint32_t **LCD_fifoBuffer2** [LCD_ARRAY_2_LEN] = { 0 }
- static volatile bool **qrsBufferIsFull** = false

  flag for QRS detection to start
- static volatile bool **heartRateIsReady** = false

  flag for LCD to output heart rate
- static float32_t **QRS_processingBuffer** [QRS_ARRAY_LEN] = { 0 }
- static uint16_t **LCD_prevSampleBuffer** [LCD_X_MAX] = { 0 }

5.1.5.1 Detailed Description

Main program file.

5.1.5.2 Enumeration Type Documentation

ISR_VECTOR_NUMS

```
enum ISR_VECTOR_NUMS
```

Enumerator

DAQ_VECTOR_NUM	vector number for the DAQ_Handler()
PROC_VECTOR_NUM	vector number for the Processing_Handler()
LCD_VECTOR_NUM	vector number for the LCD_Handler()

FIFO_INFO

```
enum FIFO_INFO
```

Enumerator

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

LCD_INFO

```
enum LCD_INFO
```

Enumerator

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

5.1.5.3 Function Documentation

DAQ_Handler()

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

ISR for the data acquisition system.

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

Precondition

Initialize the DAQ module.

Postcondition

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

See also

DAQ_Init(), Processing_Handler()

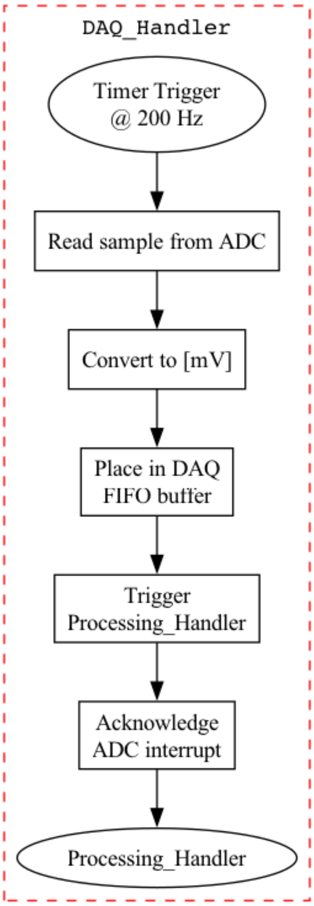


Figure 4 Flowchart for the DAQ handler.

Processing_Handler()

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

ISR for intermediate processing of the input data.

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

Postcondition

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

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

See also

DAQ_Handler(), main(), LCD_Handler()

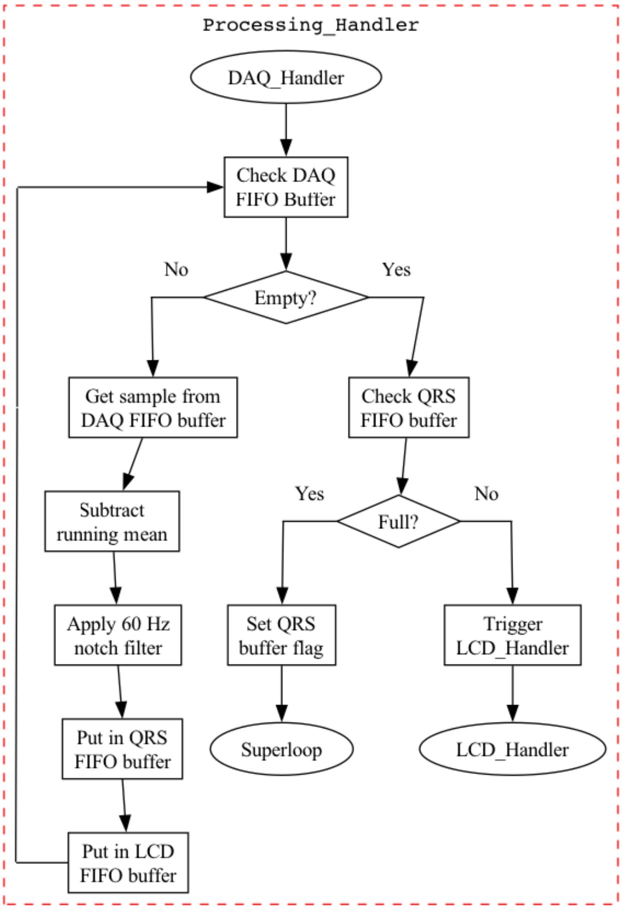


Figure 5 Flowchart for the processing handler.

LCD_Handler()

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

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

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

Precondition

Initialize the LCD module.

Postcondition

The bandpass-filtered sample is plotted to the LCD.

The heart rate is updated after each block is analyzed.

See also

LCD_Init(), Processing_Handler(), main()

main()

```
int main (
        void )
```

Main function for the project.

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

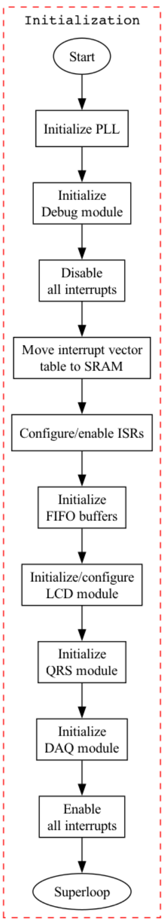


Figure 6 Flowchart for the initialization phase.

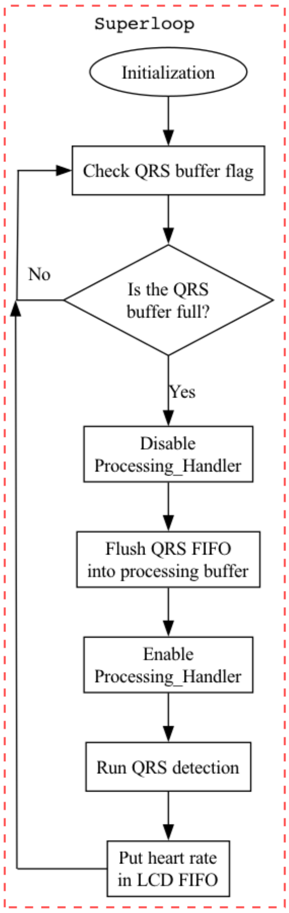
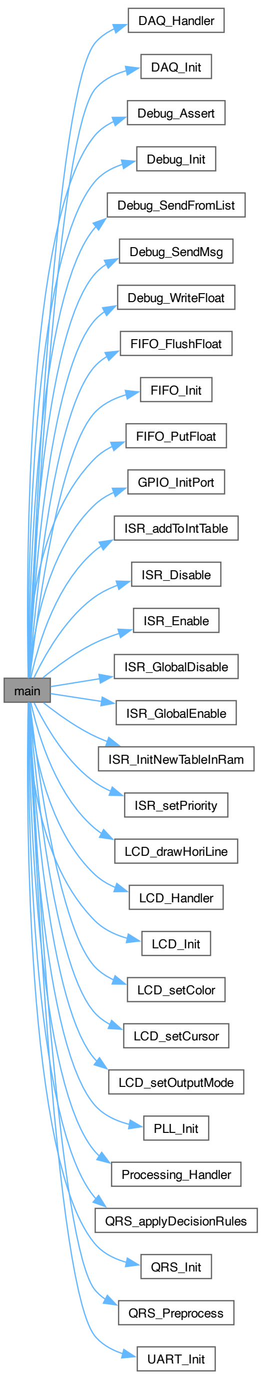


Figure 7 Flowchart for the superloop.

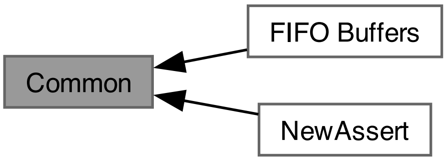
Here is the call graph for this function:



5.2 Common

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

Collaboration diagram for Common:



Modules

- FIFO Buffers

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

  Module for using a custom `assert` implementation.

Files

- file NewAssert.c

  Source code for custom `assert` implementation.
- file NewAssert.h

  Header file for custom `assert` implementation.

Functions

- void Assert (bool condition)

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

5.2.1 Detailed Description

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

5.2.2 Function Documentation

Assert()

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

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

Parameters

in	*condition*	Conditional to test.

Postcondition

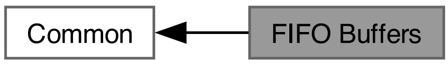
If `condition == true`, the function simply returns.

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

5.2.3 FIFO Buffers

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

Collaboration diagram for FIFO Buffers:



Files

- file Fifo.c

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

  Header file for FIFO buffer implementation.

Data Structures

- struct Fifo_t

Macros

- #define **FIFO_POOL_SIZE** 5

Functions

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

  Initialize a FIFO buffer of length N*.*
- void FIFO_Reset (volatile Fifo_t fifo)

  Reset the FIFO buffer.

Variables

- static FifoStruct_t **fifoPool** [FIFO_POOL_SIZE] = { 0 }

  pre-allocated pool
- static uint8_t **numFreeFifos** = FIFO_POOL_SIZE

Basic Operations

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

  Add a value to the end of the buffer.
- uint32_t FIFO_Get (volatile Fifo_t fifo)

  Remove the first value of the buffer.
- void FIFO_Flush (volatile Fifo_t fifo, uint32_t outputBuffer[ ])

  Empty the FIFO buffer's contents into an array.
- void FIFO_PutFloat (volatile Fifo_t fifo, const float val)

  Add a floating-point value to the end of the buffer.
- float FIFO_GetFloat (volatile Fifo_t fifo)

  Remove the first value of the buffer, and cast it to `float`.
- void FIFO_FlushFloat (volatile Fifo_t fifo, float outputBuffer[ ])

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

Peeking

- uint32_t FIFO_PeekOne (volatile Fifo_t fifo)

  See the first element in the FIFO without removing it.
- void FIFO_PeekAll (volatile Fifo_t fifo, uint32_t outputBuffer[ ])

  See the FIFO buffer's contents without removing them.

Status Checks

- bool FIFO_isFull (volatile Fifo_t fifo)

  Check if the FIFO buffer is full.
- bool FIFO_isEmpty (volatile Fifo_t fifo)

  Check if the FIFO buffer is empty.
- uint32_t FIFO_getCurrSize (volatile Fifo_t fifo)

  Get the current size of the FIFO buffer.

5.2.3.1 Detailed Description

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

5.2.3.2 Function Documentation

FIFO_Init()

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

Initialize a FIFO buffer of length `N`.

Parameters

in	*buffer*	Array of size `N` to be used as FIFO buffer
in	*N*	Length of `buffer`. Usable length is `N - 1`.
out	*fifo*	pointer to the FIFO buffer

Postcondition

The number of available FIFO buffers is reduced by 1.

TODO: Add details

FIFO_Reset()

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

Reset the FIFO buffer.

Parameters

in	*fifo*	Pointer to FIFO buffer.

Postcondition

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

FIFO_Put()

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

Add a value to the end of the buffer.

Parameters

in	*fifo*	Pointer to FIFO object
in	*val*	Value to add to the buffer.

Postcondition

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

See also

FIFO_PutFloat()

FIFO_Get()

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

Remove the first value of the buffer.

Parameters

in	*fifo*	Pointer to FIFO object
out	*val*	First sample in the FIFO.

Postcondition

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

See also

FIFO_GetFloat()

FIFO_Flush()

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

Empty the FIFO buffer's contents into an array.

Parameters

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

Postcondition

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

See also

FIFO_FlushFloat()

FIFO_PutFloat()

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

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

Parameters

in	*fifo*	Pointer to FIFO object
in	*val*	Value to add to the buffer.

Postcondition

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

Note

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

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

See also

FIFO_Put()

Remarks

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

FIFO_GetFloat()

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

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

Parameters

in	*fifo*	Pointer to FIFO object
out	*val*	First sample in the FIFO.

Postcondition

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

Note

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

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

See also

FIFO_Get()

Remarks

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

FIFO_FlushFloat()

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

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

Parameters

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

Postcondition

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

Note

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

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

See also

FIFO_Flush()

FIFO_PeekOne()

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

See the first element in the FIFO without removing it.

Parameters

in	*fifo*	Pointer to FIFO object
out	*val*	First sample in the FIFO.

FIFO_PeekAll()

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

See the FIFO buffer's contents without removing them.

Parameters

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

Postcondition

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

FIFO_isFull()

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

Check if the FIFO buffer is full.

Parameters

in	*fifo*	Pointer to the FIFO buffer.
out	*true*	The FIFO buffer is full.
out	*false*	The FIFO buffer is not full.

FIFO_isEmpty()

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

Check if the FIFO buffer is empty.

Parameters

in	*fifo*	Pointer to the FIFO buffer.
out	*true*	The FIFO buffer is empty.
out	*false*	The FIFO buffer is not empty.

FIFO_getCurrSize()

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

Get the current size of the FIFO buffer.

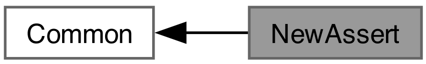
Parameters

in	*fifo*	Pointer to the FIFO buffer.
out	*size*	Current number of values in the FIFO buffer.

5.2.4 NewAssert

Module for using a custom `assert` implementation.

Collaboration diagram for NewAssert:

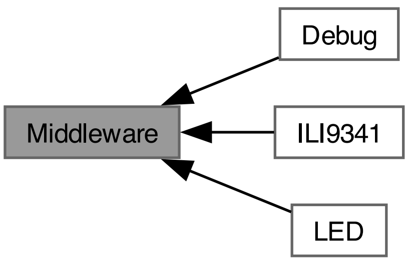


Module for using a custom `assert` implementation.

5.3 Middleware

High-level device driver modules.

Collaboration diagram for Middleware:



Modules

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

5.3.1 Detailed Description

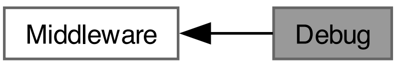
High-level device driver modules.

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

5.3.2 Debug

Module for debugging functions, including serial output and assertions.

Collaboration diagram for Debug:



Files

- file Debug.h
  Functions to output debugging information to a serial port via UART.

Serial Output

- enum **Msg_t** { **DEBUG_DAQ_INIT** , **DEBUG_QRS_INIT** , **DEBUG_LCD_INIT** , **DEBUG_QRS_START** }
- void Debug_SendMsg (void ∗message)
  Send a message to the serial port.
- void Debug_SendFromList (Msg_t msg)
  Send a message from the message list.
- void Debug_WriteFloat (double value)
  Write a floating-point value to the serial port.

Initialization

- void Debug_Init (Uart_t uart)
  Initialize the Debug module.

Assertions

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

5.3.2.1 Detailed Description

Module for debugging functions, including serial output and assertions.

5.3.2.2 Function Documentation

Debug_Init()

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

Initialize the Debug module.

Parameters

in	*uart*	UART to use for serial output.

Postcondition

An initialization message is sent to the serial port.

Debug_SendMsg()

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

Send a message to the serial port.

Precondition

Initialize the Debug module.

Parameters

message	(Pointer to) array of ASCII characters.

Postcondition

A floating point value is written to the serial port.

See also

Debug_SendMsg()

Debug_SendFromList()

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

Send a message from the message list.

Precondition

Initialize the Debug module.

Parameters

in	*msg*	An entry from the enumeration.

Postcondition

The corresponding message is sent to the serial port.

See also

Debug_SendMsg()

Debug_WriteFloat()

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

Write a floating-point value to the serial port.

Precondition

Initialize the Debug module.

Parameters

in	*value*	Floating-point value.

Postcondition

A floating point value is written to the serial port.

See also

Debug_SendMsg()

Debug_Assert()

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

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

Precondition

Initialize the Debug module.

Parameters

in	*condition*	Conditional statement to evaluate.

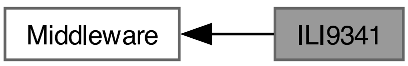
Postcondition

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

5.3.3 ILI9341

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

Collaboration diagram for ILI9341:



Files

- file ILI9341.c

  Source code for ILI9341 module.
- file ILI9341.h

  Driver module for interfacing with an ILI9341 LCD driver.

Enumerations

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

Functions

- static void ILI9341_setMode (uint8_t param)
- static void ILI9341_setAddress (uint16_t start_address, uint16_t end_address, bool is_row)
- static void ILI9341_sendParams (Cmd_t cmd)

  Send a command and/or the data within the FIFO buffer. A command is only sent when `cmd != NOP` *(where* `NOP = 0`*). Data is only sent if the FIFO buffer is not empty.*
- void ILI9341_Init (Timer_t timer)

  Initialize the LCD driver and the SPI module.
- void ILI9341_setInterface (void)

  Sets the interface for the ILI9341.
- void ILI9341_resetHard (Timer_t timer)

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

  Perform a software reset of the LCD driver.
- void ILI9341_setSleepMode (sleepMode_t sleepMode, Timer_t timer)

  Enter or exit sleep mode (`ON` *by default).*
- void ILI9341_setDisplayArea (displayArea_t displayArea)

  Set the display area.
- void ILI9341_setColorExpression (colorExpr_t colorExpr)

  Set the color expression (`FULL_COLORS` *by default).*
- void ILI9341_setPartialArea (uint16_t rowStart, uint16_t rowEnd)

  Set the display area for partial mode. Call before activating partial mode.
- void ILI9341_setDispInversion (invertMode_t invertMode)

  Toggle display inversion (`OFF` *by default).*
- void ILI9341_setDispOutput (outputMode_t outputMode)

  Change whether the IC is outputting to the display for not.
- void ILI9341_setMemAccessCtrl (bool areRowsFlipped, bool areColsFlipped, bool areRowsAndCols↩ Switched, bool isVertRefreshFlipped, bool isColorOrderFlipped, bool isHorRefreshFlipped)

  Set how data is converted from memory to display.
- void ILI9341_setColorDepth (colorDepth_t colorDepth)

  Set the color depth for the display.
- void ILI9341_setFrameRate (uint8_t divisionRatio, uint8_t clocksPerLine)

  TODO: Write brief.

- void ILI9341_setRowAddress (uint16_t startRow, uint16_t endRow)

  Sets the start/end rows to be written to.
- void ILI9341_setColAddress (uint16_t startCol, uint16_t endCol)

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

  Signal to the driver that pixel data is incoming and should be written to memory.
- void ILI9341_writePixel (uint8_t red, uint8_t green, uint8_t blue)

  Write a single pixel to frame memory.

Variables

- static uint32_t **ILI9341_Buffer** [8]
- static Fifo_t **ILI9341_Fifo**
- struct {
  sleepMode_t **sleepMode**
  displayArea_t **displayArea**
  colorExpr_t **colorExpression**
  invertMode_t **invertMode**
  outputMode_t **outputMode**
  colorDepth_t **colorDepth**
  bool **isInit**
  } **ili9341** = { SLEEP_ON, NORMAL_AREA, FULL_COLORS, INVERT_OFF, OUTPUT_ON, COLORDEPTH_16BIT, false }

5.3.3.1 Detailed Description

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

5.3.3.2 Enumeration Type Documentation

anonymous enum

`anonymous enum`

Enumerator

ILI9341_NUM_COLS	**5.3.3.3 of columns available on the display**
ILI9341_NUM_ROWS	**5.3.3.4 of rows available on the display**

Cmd_t

`enum Cmd_t`

Enumerator

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

5.3.3.5 Function Documentation

ILI9341_setMode()

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

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

ILI9341_setAddress()

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

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

The input parameters represent the first and last addresses to be written to when `ILI9341_writePixel()` is called.

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

ILI9341_sendParams()

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

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

Parameters

in	*cmd*	Command to send.

ILI9341_Init()

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

Initialize the LCD driver and the SPI module.

Parameters

in	*timer*	Hardware timer to use during initialization.

ILI9341_setInterface()

```
void ILI9341_setInterface (
            void  )
```

Sets the interface for the ILI9341.

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

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

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

The first param's bits are cleared so that the corresponding MADCTL bits (ILI9341_setMemoryAccessCtrl()) are unaffected and overflowing pixel data is ignored. The EPF bits are cleared so that the LSB of the R and B values is copied from the MSB when using 16-bit color depth. The TM4C123 sends the MSB first, so the ENDIAN bit is cleared. The other bits are cleared and/or irrelevant since the RGB and VSYNC interfaces aren't used.

ILI9341_resetHard()

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

Perform a hardware reset of the LCD driver.

Parameters

in	*timer*	Hardware timer to use during reset.

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

ILI9341_resetSoft()

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

Perform a software reset of the LCD driver.

Parameters

in	*timer*	Hardware timer to use during reset.

the driver needs 5 [ms] before another command

ILI9341_setSleepMode()

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

Enter or exit sleep mode (`ON` by default).

Parameters

in	*sleepMode*	`SLEEP_ON` or `SLEEP_OFF`
in	*timer*	Hardware timer to use for a slight delay after the mode change.

Postcondition

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

The MCU must wait >= 5 [ms] before sending further commands regardless of the selected mode.

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

ILI9341_setDisplayArea()

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

Set the display area.

Precondition

If using partial mode, set the partial area first.

Parameters

in	*displayArea*	`NORMAL_AREA` or `PARTIAL_AREA`

See also

ILI9341_setPartialArea()

ILI9341_setColorExpression()

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

Set the color expression (`FULL_COLORS` by default).

Parameters

in	*colorExpr*	`FULL_COLORS` or `PARTIAL_COLORS`

Postcondition

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

ILI9341_setPartialArea()

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

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

Parameters

in	*rowStart*	
in	*rowEnd*	

See also

ILI9341_setDisplayArea()

ILI9341_setDispInversion()

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

Toggle display inversion (`OFF` by default).

Parameters

in	*invertMode*	`INVERT_ON` or `INVERT_OFF`

Postcondition

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

ILI9341_setDispOutput()

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

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

Parameters

in	*outputMode*	`OUTPUT_ON` or `OUTPUT_OFF`

Postcondition

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

TODO: Write description

ILI9341_setMemAccessCtrl()

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

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

Set how data is converted from memory to display.

Parameters

in	*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_setColorDepth()

```
void ILI9341_setColorDepth (
            colorDepth_t colorDepth )
```

Set the color depth for the display.

Parameters

in	*colorDepth*	`COLORDEPTH_16BIT` or `COLORDEPTH_18BIT`

Postcondition

`16BIT` mode allows for ∼65K (2^16) colors and requires 2 transfers. `18BIT` mode allows for ∼262K (2^18) colors but requires 3 transfers.

ILI9341_setFrameRate()

```
void ILI9341_setFrameRate (
```

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

TODO: Write brief.

TODO: Write description

ILI9341_setRowAddress()

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

Sets the start/end rows to be written to.

Parameters

in		

0 <= `startRow` <= `endRow`

Parameters

in		

startRow<=endRow` < 240

See also

ILI9341_setRowAddress, ILI9341_writePixel()

This function is simply an interface to ILI9341_setAddress(). To work correctly, `start_row` must be no greater than `end_row`, and `end_row` cannot be greater than the max row number (default 320).

ILI9341_setColAddress()

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

Sets the start/end columns to be written to.

Parameters

in		

0 <= `startCol` <= `endCol`

Parameters

in			

startCol<=endCol` < 240

See also

ILI9341_setColAddress, ILI9341_writePixel()

This function is simply an interface to ILI9341_setAddress(). To work correctly, `start_col` must be no greater than `end_col`, and `end_col` cannot be greater than the max column number (default 240).

ILI9341_writeMemCmd()

```
void ILI9341_writeMemCmd (
            void  )
```

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

Precondition

Set the row and/or column addresses.

Postcondition

The LCD driver is ready to accept pixel data.

See also

ILI9341_setRowAddress, ILI9341_setColAddress(), ILI9341_writePixel()

ILI9341_writePixel()

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

Write a single pixel to frame memory.

Precondition

Send the "Write Memory" command.

Set the desired color depth for the display.

Parameters

in	*red*	5 or 6-bit R value
in	*green*	5 or 6-bit G value
in	*blue*	5 or 6-bit B value

See also

ILI9341_setColorDepth, ILI9341_writeMemCmd(), ILI9341_writePixel()

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

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

Transfer	**1**	**2**

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

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

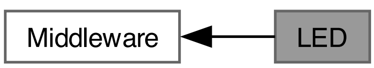
Transfer	**1**	**2**

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

5.3.4 LED

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

Collaboration diagram for LED:



Files

- file Led.c

  Source code for LED module.
- file Led.h

  Interface for LED module.

Data Structures

- struct Led_t

Macros

- #define **LED_POOL_SIZE** 1

Variables

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

Initialization & Configuration

- Led_t Led_Init (GpioPort_t gpioPort, GpioPin_t pin)
  Initialize a light-emitting diode (LED) as an `Led_t`.
- GpioPort_t Led_GetPort (Led_t led)
  Get the GPIO port associated with the LED.
- GpioPin_t Led_GetPin (Led_t led)
  Get the GPIO pin associated with the LED.

Status Checking

- bool Led_isInit (Led_t led)
  Check if an LED is initialized.
- bool Led_isOn (Led_t led)
  Check the LED's status.

Operations

- void Led_TurnOn (Led_t led)
  Turn an LED `ON`.
- void Led_TurnOff (Led_t led)
  Turn an LED `OFF`.
- void Led_Toggle (Led_t led)
  Toggle an LED.

5.3.4.1 Detailed Description

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

5.3.4.2 Function Documentation

Led_Init()

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

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

Parameters

in	*gpioPort*	Pointer to a `struct` representing a GPIO port.
in	*pin*	GPIO pin to use.
out	*led*	Pointer to LED data structure.

Led_GetPort()

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

Get the GPIO port associated with the LED.

Precondition

Initialize the LED.

Parameters

in	*led*	Pointer to LED data structure.
out	*gpioPort*	Pointer to a GPIO port data structure.

See also

Led_Init(), Led_GetPin()

Led_GetPin()

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

Get the GPIO pin associated with the LED.

Precondition

Initialize the LED.

Parameters

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

See also

Led_Init(), Led_GetPort()

Led_isInit()

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

Check if an LED is initialized.

Parameters

in	*led*	Pointer to LED data structure.
out	*true*	The LED is initialized.
out	*false*	The LED is not initialized.

See also

Led_Init()

Led_isOn()

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

Check the LED's status.

Precondition

Initialize the LED.

Parameters

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

See also

Led_TurnOn(), Led_TurnOff(), Led_Toggle()

Led_TurnOn()

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

Turn an LED `ON`.

Precondition

Initialize the LED.

Parameters

in	*led*	Pointer to LED data structure.

Postcondition

The LED is turned ON.

See also

Led_TurnOff(), Led_Toggle()

Led_TurnOff()

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

Turn an LED `OFF`.

Precondition

Initialize the LED.

Parameters

in	*led*	Pointer to LED data structure.

Postcondition

The LED is turned OFF.

See also

Led_TurnOn(), Led_Toggle()

Led_Toggle()

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

Toggle an LED.

Precondition

Initialize the LED.

Parameters

in	*led*	Pointer to LED data structure.

Postcondition

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

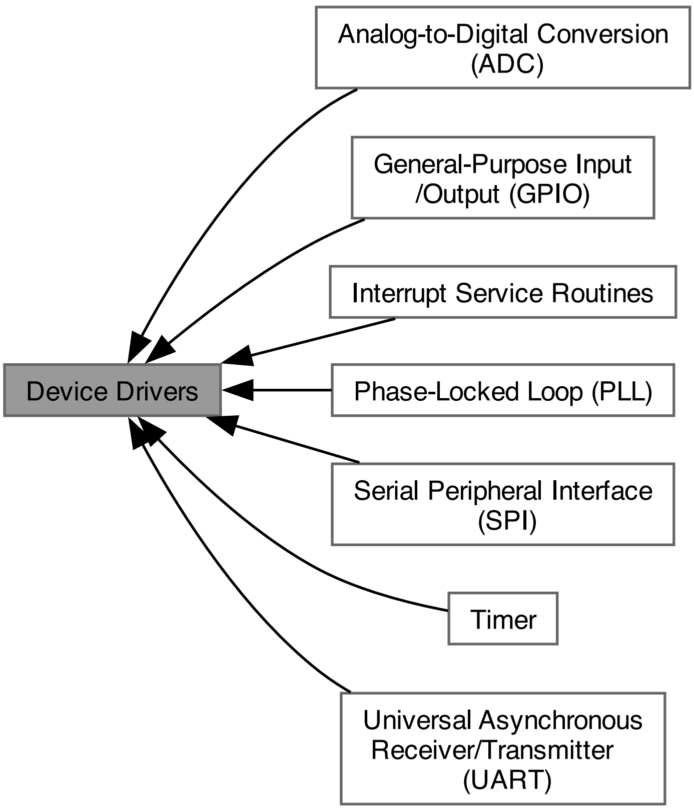
See also

Led_TurnOn(), Led_TurnOff()

5.4 Device Drivers

Low level device driver modules.

Collaboration diagram for Device Drivers:



Modules

- Analog-to-Digital Conversion (ADC)

    Functions for analog-to-digital conversion.
- General-Purpose Input/Output (GPIO)

    Functions for using GPIO ports.
- Phase-Locked Loop (PLL)

    Function for initializing the phase-locked loop.
- Serial Peripheral Interface (SPI)

    Functions for SPI-based communication via the SSI peripheral.
- Timer

    Functions for using hardware timers.
- Universal Asynchronous Receiver/Transmitter (UART)

    Functions for serial communication via the UART peripheral.
- Interrupt Service Routines

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

5.4.1 Detailed Description

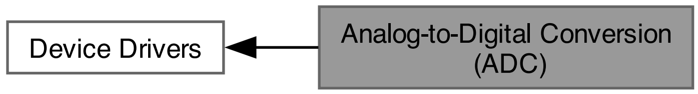
Low level device driver modules.

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

5.4.2 Analog-to-Digital Conversion (ADC)

Functions for analog-to-digital conversion.

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



Files

- file ADC.c

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

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

Functions

- void ADC_Init (void)

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

5.4.2.1 Detailed Description

Functions for analog-to-digital conversion.

5.4.2.2 Function Documentation

ADC_Init()

```
void ADC_Init (
            void )
```

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

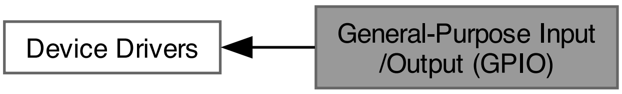
Postcondition

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

5.4.3 General-Purpose Input/Output (GPIO)

Functions for using GPIO ports.

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



Files

- file GPIO.c

  Source code for GPIO module.
- file GPIO.h

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

Data Structures

- struct GpioPort_t

Macros

- #define **GPIO_NUM_PORTS** 6

Enumerations

- enum {
  GPIO_PORTA_BASE_ADDRESS = (uint32_t) 0x40004000 , **GPIO_PORTB_BASE_ADDRESS** = (uint32_t) 0x40005000 , **GPIO_PORTC_BASE_ADDRESS** = (uint32_t) 0x40006000 , **GPIO_PORTD_BASE_ADDRESS** = (uint32_t) 0x40007000 ,
  GPIO_PORTE_BASE_ADDRESS = (uint32_t) 0x40024000 , **GPIO_PORTF_BASE_ADDRESS** = (uint32_t) 0x40025000 }
- enum {
  GPIO_DATA_R_OFFSET = (uint32_t) 0x03FC , **GPIO_DIR_R_OFFSET** = (uint32_t) 0x0400 , **GPIO_IS_R_OFFSET** = (uint32_t) 0x0404 , **GPIO_IBE_R_OFFSET** = (uint32_t) 0x0408 ,
  GPIO_IEV_R_OFFSET = (uint32_t) 0x040C , **GPIO_IM_R_OFFSET** = (uint32_t) 0x0410 , **GPIO_ICR_R_OFFSET** = (uint32_t) 0x041C , **GPIO_AFSEL_R_OFFSET** = (uint32_t) 0x0420 ,
  GPIO_DR2R_R_OFFSET = (uint32_t) 0x0500 , **GPIO_DR4R_R_OFFSET** = (uint32_t) 0x0504 , **GPIO_DR8R_R_OFFSET** = (uint32_t) 0x0508 , **GPIO_PUR_R_OFFSET** = (uint32_t) 0x0510 ,
  GPIO_PDR_R_OFFSET = (uint32_t) 0x0518 , **GPIO_DEN_R_OFFSET** = (uint32_t) 0x051C , **GPIO_LOCK_R_OFFSET** = (uint32_t) 0x0520 , **GPIO_COMMIT_R_OFFSET** = (uint32_t) 0x0524 ,
  GPIO_AMSEL_R_OFFSET = (uint32_t) 0x0528 , **GPIO_PCTL_R_OFFSET** = (uint32_t) 0x052C }
- enum **GPIO_PortName_t** {
  GPIO_PORT_A , **GPIO_PORT_B** , **GPIO_PORT_C** , **GPIO_PORT_D** ,
  GPIO_PORT_E , **GPIO_PORT_F** , **A** = GPIO_PORT_A , **B** = GPIO_PORT_B ,
  C = GPIO_PORT_C , **D** = GPIO_PORT_D , **E** = GPIO_PORT_E , **F** = GPIO_PORT_F }
- enum **GpioPin_t** {
  GPIO_PIN0 = ((uint8_t) 1) , **GPIO_PIN1** = ((uint8_t) (1 << 1)) , **GPIO_PIN2** = ((uint8_t) (1 << 2)) , **GPIO_PIN3** = ((uint8_t) (1 << 3)) ,
  GPIO_PIN4 = ((uint8_t) (1 << 4)) , **GPIO_PIN5** = ((uint8_t) (1 << 5)) , **GPIO_PIN6** = ((uint8_t) (1 << 6)) , **GPIO_PIN7** = ((uint8_t) (1 << 7)) ,
  GPIO_ALL_PINS = ((uint8_t) (0xFF)) }
- enum GPIO_LAUNCHPAD_LEDS {
  LED_RED = GPIO_PIN1 , LED_GREEN = GPIO_PIN3 , LED_BLUE = GPIO_PIN2 , **LED_YELLOW** = (LED_RED + LED_GREEN) ,
  LED_CYAN = (LED_BLUE + LED_GREEN) , **LED_PURPLE** = (LED_RED + LED_BLUE) , **LED_WHITE** = (LED_RED + LED_BLUE + LED_GREEN) }

Functions

- GpioPort_t GPIO_InitPort (GPIO_PortName_t portName)

  Initialize a GPIO Port and return a pointer to its `struct`*.*
- bool GPIO_isPortInit (GpioPort_t gpioPort)

  Check if the GPIO port is initialized.
- uint32_t GPIO_getBaseAddr (GpioPort_t gpioPort)

  Get the base address of a GPIO port.
- void GPIO_ConfigDirOutput (GpioPort_t gpioPort, GpioPin_t pinMask)

  Configure the direction of the specified GPIO pins. All pins are configured to `INPUT` *by default, so this function should only be called to specify* `OUTPUT` *pins.*
- void GPIO_ConfigDirInput (GpioPort_t gpioPort, GpioPin_t pinMask)

  Configure the specified GPIO pins as `INPUT` *pins. All pins are configured to* `INPUT` *by default, so this function is technically unnecessary, but useful for code readability.*
- void GPIO_ConfigPullUp (GpioPort_t gpioPort, GpioPin_t pinMask)

  Activate the specified pins' internal pull-up resistors.
- void GPIO_ConfigPullDown (GpioPort_t gpioPort, GpioPin_t pinMask)

  Activate the specified pins' internal pull-down resistors.
- void GPIO_ConfigDriveStrength (GpioPort_t gpioPort, GpioPin_t pinMask, uint8_t drive_mA)

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

- void GPIO_EnableDigital (GpioPort_t gpioPort, GpioPin_t pinMask)

    Enable digital I/O for the specified pins.
- void GPIO_DisableDigital (GpioPort_t gpioPort, GpioPin_t pinMask)

    Disable digital I/O for the specified pins.
- void GPIO_ConfigInterrupts_Edge (GpioPort_t gpioPort, GpioPin_t pinMask, bool risingEdge)

    Configure the specified GPIO pins to trigger an interrupt on the rising or falling edge of an input.
- void GPIO_ConfigInterrupts_BothEdges (GpioPort_t gpioPort, GpioPin_t pinMask)

    Configure the specified GPIO pins to trigger an interrupt on both edges of an input.
- void GPIO_ConfigInterrupts_LevelTrig (GpioPort_t gpioPort, GpioPin_t pinMask, bool highLevel)

    Configure the specified GPIO pins to trigger an interrupt on a high level or low level pulse.
- void GPIO_ConfigNVIC (GpioPort_t gpioPort, uint8_t priority)

    Configure interrupts for the selected port in the NVIC.
- volatile uint32_t * GPIO_getDataRegister (GpioPort_t gpioPort)

    Get the address of a GPIO port's data register.
- uint8_t GPIO_ReadPins (GpioPort_t gpioPort, GpioPin_t pinMask)

    Read from the specified GPIO pin.
- void GPIO_WriteHigh (GpioPort_t gpioPort, GpioPin_t pinMask)

    Write a `1` to the specified GPIO pins.
- void GPIO_WriteLow (GpioPort_t gpioPort, GpioPin_t pinMask)

    Write a `0` to the specified GPIO pins.
- void GPIO_Toggle (GpioPort_t gpioPort, GpioPin_t pinMask)

    Toggle the specified GPIO pins.
- void GPIO_ConfigAltMode (GpioPort_t gpioPort, GpioPin_t pinMask)

    Activate the alternate mode for the specified pins.
- void GPIO_ConfigPortCtrl (GpioPort_t gpioPort, GpioPin_t pinMask, uint8_t fieldEncoding)

    Specify the alternate mode to use for the specified pins.
- void GPIO_ConfigAnalog (GpioPort_t gpioPort, GpioPin_t pinMask)

    Activate analog mode for the specified GPIO pins.

Variables

- static GpioPortStruct_t GPIO_PTR_ARR [6]

5.4.3.1 Detailed Description

Functions for using GPIO ports.

5.4.3.2 Enumeration Type Documentation

GPIO_LAUNCHPAD_LEDS

`enum GPIO_LAUNCHPAD_LEDS`

Enumerator

LED_RED	PF1.
LED_GREEN	PF3.
LED_BLUE	PF2.

5.4.3.3 Function Documentation

GPIO_InitPort()

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

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

Parameters

`in`	*portName*	Name of the chosen port.
`out`	*gpioPort*	Pointer to the specified GPIO port.

GPIO_isPortInit()

```
bool GPIO_isPortInit (
            GpioPort_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_getBaseAddr()

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

Get the base address of a GPIO port.

Parameters

`in`	*gpioPort*	Pointer to the specified GPIO port.
`out`	*baseAddress*	Base address of the GPIO port.

GPIO_ConfigDirOutput()

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

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

Parameters

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

GPIO_ConfigDirInput()

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

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

Parameters

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

GPIO_ConfigPullUp()

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

Activate the specified pins' internal pull-up resistors.

Parameters

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

GPIO_ConfigPullDown()

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

Activate the specified pins' internal pull-down resistors.

Parameters

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

GPIO_ConfigDriveStrength()

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

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

Parameters

in	*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_EnableDigital()

```
void GPIO_EnableDigital (
            GpioPort_t gpioPort,
            GpioPin_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_DisableDigital()

```
void GPIO_DisableDigital (
            GpioPort_t gpioPort,
            GpioPin_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_ConfigInterrupts_Edge()

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

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

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

Parameters

in	*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_BothEdges()

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

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

Parameters

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

GPIO_ConfigInterrupts_LevelTrig()

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

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

Parameters

in	*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 (
            GpioPort_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_getDataRegister()

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

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

Parameters

in	*gpioPort*	Pointer to the specified GPIO port.
out	*dataRegister*	Address of the GPIO port's data register.

GPIO_ReadPins()

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

Read from the specified GPIO pin.

Parameters

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

GPIO_WriteHigh()

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

Write a `1` to the specified GPIO pins.

Parameters

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

GPIO_WriteLow()

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

Write a `0` to the specified GPIO pins.

Parameters

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

GPIO_Toggle()

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

Toggle the specified GPIO pins.

Parameters

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

GPIO_ConfigAltMode()

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

Activate the alternate mode for the specified pins.

Parameters

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

GPIO_ConfigPortCtrl()

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

Specify the alternate mode to use for the specified pins.

Parameters

in	*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_ConfigAnalog()

```
void GPIO_ConfigAnalog (
            GpioPort_t gpioPort,
            GpioPin_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).

5.4.3.4 Variable Documentation

GPIO_PTR_ARR

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

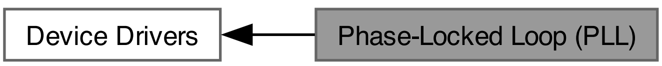
Initial value:

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

5.4.4 Phase-Locked Loop (PLL)

Function for initializing the phase-locked loop.

Collaboration diagram for Phase-Locked Loop (PLL):



Files

- file PLL.c

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

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

Functions

- void PLL_Init (void)

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

5.4.4.1 Detailed Description

Function for initializing the phase-locked loop.

5.4.4.2 Function Documentation

PLL_Init()

```
void PLL_Init (
            void  )
```

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

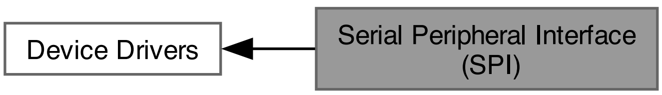
Postcondition

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

5.4.5 Serial Peripheral Interface (SPI)

Functions for SPI-based communication via the SSI peripheral.

Collaboration diagram for Serial Peripheral Interface (SPI):



Files

- file SPI.c

  Source code for serial peripheral interface (SPI) module.
- file SPI.h

  Header file for serial peripheral interface (SPI) module.

Macros

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

Enumerations

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

Functions

- void SPI_Init (void)

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

  Read data from the serial port.
- void SPI_WriteCmd (uint8_t cmd)

  Write a command to the serial port.
- void SPI_WriteData (uint8_t data)

  Write data to the serial port.

Variables

- static register_t **gpioPortReg** = 0

5.4.5.1 Detailed Description

Functions for SPI-based communication via the SSI peripheral.

5.4.5.2 Function Documentation

SPI_Init()

```
void SPI_Init (
            void )
```

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

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

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

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

SPI_Read()

```
uint8_t SPI_Read (
            void )
```

Read data from the serial port.

Precondition

Initialize the SPI module.

Parameters

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

SPI_WriteCmd()

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

Write a command to the serial port.

Precondition

Initialize the SPI module.

Parameters

in	*cmd*	8-bit command to write.

Postcondition

The D/C pin is cleared.

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

SPI_WriteData()

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

Write data to the serial port.

Precondition

Initialize the SPI module.

Parameters

in	*data*	8-bit data to write.

Postcondition

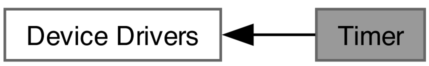
The D/C pin is set.

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

5.4.6 Timer

Functions for using hardware timers.

Collaboration diagram for Timer:



Files

- file Timer.c

  Source code for Timer module.
- file Timer.h

  Device driver for general-purpose timer modules.

Data Structures

- struct Timer_t

Enumerations

- enum {
  TIMER0_BASE = 0x40030000 , **TIMER1_BASE** = 0x40031000 , **TIMER2_BASE** = 0x40032000 , **TIMER3↩
  _BASE** = 0x40033000 ,
  TIMER4_BASE = 0x40034000 , **TIMER5_BASE** = 0x40035000 }
- enum **REGISTER_OFFSETS** {
  CONFIG = 0x00 , **MODE** = 0x04 , **CTRL** = 0x0C , **INT_MASK** = 0x18 ,
  INT_CLEAR = 0x24 , **INTERVAL** = 0x28 , **VALUE** = 0x054 }
- enum **timerName_t** {
  TIMER0 , **TIMER1** , **TIMER2** , **TIMER3** ,
  TIMER4 , **TIMER5** }
- enum timerMode_t { ONESHOT , PERIODIC }
- enum timerDirection_t { UP , DOWN }

Functions

- Timer_t Timer_Init (timerName_t timerName)

  Initialize a hardware timer.
- void Timer_Deinit (Timer_t timer)

  De-initialize a hardware timer.
- timerName_t Timer_getName (Timer_t timer)

  Get the name of a timer object.
- bool Timer_isInit (Timer_t timer)

  Check if a timer object is initialized.
- void Timer_setMode (Timer_t timer, timerMode_t timerMode, timerDirection_t timerDirection)

  Set the mode for the timer.
- void Timer_enableAdcTrigger (Timer_t timer)

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

  Disable ADC sample capture on timeout.
- void Timer_enableInterruptOnTimeout (Timer_t timer)

  Set the timer to trigger an interrupt on timeout.
- void Timer_disableInterruptOnTimeout (Timer_t timer)

  Stop the timer from triggering interrupts on timeout.
- void Timer_clearInterruptFlag (Timer_t timer)

  Clear the timer's interrupt flag to acknowledge the interrupt.
- void Timer_setInterval_ms (Timer_t timer, uint32_t time_ms)

  Set the interval to use.
- uint32_t **Timer_getCurrentValue** (Timer_t timer)
- void Timer_Start (Timer_t timer)

  Start the timer.
- void Timer_Stop (Timer_t timer)

  Stop the timer.
- bool Timer_isCounting (Timer_t timer)

  Check if the timer is currently counting.
- void Timer_Wait1ms (Timer_t timer, uint32_t time_ms)

  Initiate a time delay.

Variables

- static TimerStruct_t TIMER_POOL [6]

5.4.6.1 Detailed Description

Functions for using hardware timers.

5.4.6.2 Enumeration Type Documentation

timerMode_t

```
enum timerMode_t
```

Enumerator

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

timerDirection_t

```
enum timerDirection_t
```

Enumerator

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

5.4.6.3 Function Documentation

Timer_Init()

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

Initialize a hardware timer.

Parameters

`in`	*timerName*	Name of the hardware timer to use.
`out`	*timer*	Pointer to timer object.

Postcondition

The timer is ready to be configured and used.

See also

Timer_isInit(), Timer_Deinit()

Timer_Deinit()

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

De-initialize a hardware timer.

Parameters

in	*timerName*	Name of the hardware timer to use.

Postcondition

The hardware timer is no longer initialized or receiving power.

See also

Timer_Init(), Timer_isInit()

Timer_getName()

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

Get the name of a timer object.

Parameters

in	*timer*	Pointer to timer object.
out	*timer←Name_t*	Name of the hardware timer being used.

Timer_isInit()

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

Check if a timer object is initialized.

Parameters

in	*timer*	Pointer to timer object.
out	*true*	The timer is initialized.
out	*false*	The timer is not initialized.

See also

Timer_Init(), Timer_Deinit()

Timer_setMode()

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

Set the mode for the timer.

Parameters

in	*timer*	Pointer to timer object.
in	*timerMode*	Mode for hardware timer to use.
in	*timerDirection*	Direction to count towards.

Timer_enableAdcTrigger()

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

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

Precondition

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

Parameters

in	*timer*	Pointer to timer object.

Postcondition

A timeout event triggers ADC sample capture.

See also

Timer_disableAdcTrigger()

Timer_disableAdcTrigger()

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

Disable ADC sample capture on timeout.

Precondition

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

Parameters

in	*timer*	Pointer to timer object.

Postcondition

A timeout event no longer triggers ADC sample capture.

See also

Timer_enableAdcTrigger()

Timer_enableInterruptOnTimeout()

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

Set the timer to trigger an interrupt on timeout.

Precondition

Configure the interrupt service routine using the ISR module.

Parameters

in	*timer*	Pointer to timer object.

Postcondition

Upon timeout, an interrupt is triggered.

See also

Timer_disableInterruptOnTimeout()

Timer_disableInterruptOnTimeout()

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

Stop the timer from triggering interrupts on timeout.

Parameters

in	*timer*	Pointer to timer object.

Postcondition

Timeout no longer triggers ADC sample capture.

See also

Timer_enableInterruptOnTimeout()

Timer_clearInterruptFlag()

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

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

Precondition

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

Parameters

in	*timer*	Pointer to timer object.

Timer_setInterval_ms()

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

Set the interval to use.

Precondition

Initialize and configure the timer.

Parameters

in	*timer*	Pointer to timer object.
in	*time_ms*	Time in [ms].

Postcondition

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

See also

Timer_Init(), Timer_setMode()

Timer_Start()

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

Start the timer.

Precondition

Initialize and configure the timer.

Parameters

in	*timer*	Pointer to timer object.

Postcondition

The timer is counting.

See also

Timer_Stop(), Timer_isCounting()

Timer_Stop()

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

Stop the timer.

Precondition

Start the timer.

Parameters

in	*timer*	Pointer to timer object.

Postcondition

The timer is no longer counting.

See also

Timer_Start(), Timer_isCounting()

Timer_isCounting()

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

Check if the timer is currently counting.

Parameters

in	*timer*	Pointer to timer object.
out	*true*	The timer is counting.
out	*false*	The timer is not counting.

See also

Timer_Start(), Timer_Stop()

Timer_Wait1ms()

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

Initiate a time delay.

Precondition

Initialize and configure the timer.

Parameters

in	*timer*	Pointer to timer object.
in	*time_ms*	Time in [ms] to wait for.

Postcondition

The program is delayed for the desired time.

5.4.6.4 Variable Documentation

TIMER_POOL

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

Initial value:

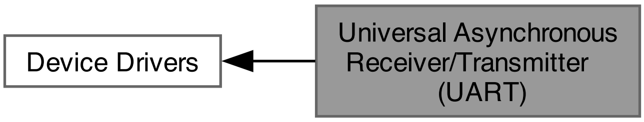
```
= {
    { TIMER0, TIMER0_BASE, (register_t) (TIMER0_BASE + CTRL), (register_t) (TIMER0_BASE + INTERVAL),
      (register_t) (TIMER0_BASE + INT_CLEAR), false },
    { TIMER1, TIMER1_BASE, (register_t) (TIMER1_BASE + CTRL), (register_t) (TIMER1_BASE + INTERVAL),
      (register_t) (TIMER1_BASE + INT_CLEAR),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

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

5.4.7 Universal Asynchronous Receiver/Transmitter (UART)

Functions for serial communication via the UART peripheral.

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



Files

- file UART.c

  Source code for UART module.
- file UART.h

  Driver module for serial communication via UART0 and UART 1.

Data Structures

- struct Uart_t

Macros

- #define **ASCII_CONVERSION** 0x30

Enumerations

- enum **GPIO_BASE_ADDRESSES** {
  GPIO_PORTA_BASE = (uint32_t) 0x40004000 , **GPIO_PORTB_BASE** = (uint32_t) 0x40005000 , **GPIO_PORTC_BASE** = (uint32_t) 0x40006000 , **GPIO_PORTD_BASE** = (uint32_t) 0x40007000 ,
  GPIO_PORTE_BASE = (uint32_t) 0x40024000 , **GPIO_PORTF_BASE** = (uint32_t) 0x40025000 }
- enum **UART_BASE_ADDRESSES** {
  UART0_BASE = (uint32_t) 0x4000C000 , **UART1_BASE** = (uint32_t) 0x4000D000 , **UART2_BASE** = (uint32_t) 0x4000E000 , **UART3_BASE** = (uint32_t) 0x4000F000 ,
  UART4_BASE = (uint32_t) 0x40010000 , **UART5_BASE** = (uint32_t) 0x40011000 , **UART6_BASE** = (uint32_t) 0x40012000 , **UART7_BASE** = (uint32_t) 0x40013000 }
- enum **UART_REG_OFFSETS** {
  UART_FR_R_OFFSET = (uint32_t) 0x18 , **IBRD_R_OFFSET** = (uint32_t) 0x24 , **FBRD_R_OFFSET** = (uint32_t) 0x28 , **LCRH_R_OFFSET** = (uint32_t) 0x2C ,
  CTL_R_OFFSET = (uint32_t) 0x30 , **CC_R_OFFSET** = (uint32_t) 0xFC8 }
- enum **uartNum_t** {
  UART0 , **UART1** , **UART2** , **UART3** ,
  UART4 , **UART5** , **UART6** , **UART7** }

Functions

- Uart_t UART_Init (GpioPort_t port, uartNum_t uartNum)

    Initialize the specified UART peripheral.
- bool UART_isInit (Uart_t uart)

    Check if the UART object is initialized.
- unsigned char UART_ReadChar (Uart_t uart)

    Read a single ASCII character from the UART.
- void UART_WriteChar (Uart_t uart, unsigned char inputChar)

    Write a single character to the UART.
- void UART_WriteStr (Uart_t uart, void ∗inputStr)

    Write a C string to the UART.
- void UART_WriteInt (Uart_t uart, int32_t n)

    Write a 32-bit unsigned integer the UART.
- void UART_WriteFloat (Uart_t uart, double n, uint8_t numDecimals)

    Write a floating-point number the UART.

Variables

- static UartStruct_t UART_ARR [8]

5.4.7.1 Detailed Description

Functions for serial communication via the UART peripheral.

5.4.7.2 Function Documentation

UART_Init()

```
Uart_t UART_Init (
            GpioPort_t port,
            uartNum_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*	(Pointer to) initialized UART peripheral.

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

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

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

UART_isInit()

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

Check if the UART object is initialized.

Parameters

in	*uart*	UART to check.
out	*true*	The UART object is initialized.
out	*false*	The UART object is not initialized.

UART_ReadChar()

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

Read a single ASCII character from the UART.

Parameters

in	*uart*	UART to read from.
out	*unsigned*	char ASCII character from sender.

UART_WriteChar()

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

Write a single character to the UART.

Parameters

in	*uart*	UART to write to.
in	*input_char*	ASCII character to send.

UART_WriteStr()

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

Write a C string to the UART.

Parameters

in	*uart*	UART to write to.
in	*input_str*	Array of ASCII characters.

UART_WriteInt()

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

Write a 32-bit unsigned integer the UART.

Parameters

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

UART_WriteFloat()

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

Write a floating-point number the UART.

Parameters

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

5.4.7.3 Variable Documentation

UART_ARR

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

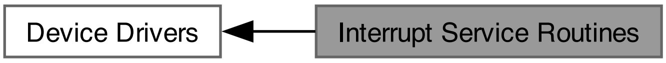
Initial value:

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

5.4.8 Interrupt Service Routines

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

Collaboration diagram for Interrupt Service Routines:



Files

- file ISR.c

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

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

Macros

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

Typedefs

- typedef void(∗ **ISR_t**) (void)

  Type definition for function pointers representing ISRs.

Functions

- static void **ISR_setStatus** (const uint8_t vectorNum, const bool isEnabled)
- void ISR_GlobalDisable (void)

  Disable all interrupts globally.
- void ISR_GlobalEnable (void)

  Enable all interrupts globally.
- static ISR_t newVectorTable[VECTOR_TABLE_SIZE] **__attribute__** ((aligned(VECTOR_TABLE_↩ ALIGNMENT)))
- void ISR_InitNewTableInRam (void)

  Relocate the vector table to RAM.
- void ISR_addToIntTable (ISR_t isr, const uint8_t vectorNum)

Add an ISR to the interrupt table.

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

  Set the priority for an interrupt.
- void ISR_Enable (const uint8_t vectorNum)

  Enable an interrupt in the NVIC.
- void ISR_Disable (const uint8_t vectorNum)

  Disable an interrupt in the NVIC.
- void ISR_triggerInterrupt (const uint8_t vectorNum)

  Generate a software-generated interrupt (SGI).

Variables

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

5.4.8.1 Detailed Description

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

5.4.8.2 Function Documentation

ISR_GlobalDisable()

```
void ISR_GlobalDisable (
            void )
```

Disable all interrupts globally.

See also

ISR_GlobalEnable()

ISR_GlobalEnable()

```
void ISR_GlobalEnable (
            void )
```

Enable all interrupts globally.

See also

ISR_GlobalDisable()

ISR_InitNewTableInRam()

```
void ISR_InitNewTableInRam (
            void )
```

Relocate the vector table to RAM.

Precondition

Disable interrupts globally before calling this.

Postcondition

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

See also

ISR_GlobalDisable(), ISR_addToIntTable()

ISR_addToIntTable()

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

Add an ISR to the interrupt table.

Precondition

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

Parameters

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

Postcondition

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

See also

ISR_InitNewTableInRam()

ISR_setPriority()

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

Set the priority for an interrupt.

Precondition

Disable the interrupt before adjusting its priority.

Parameters

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

Postcondition

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

See also

ISR_Disable()

ISR_Enable()

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

Enable an interrupt in the NVIC.

Precondition

If needed, add the interrupt to the vector table.

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

Parameters

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

Postcondition

The interrupt is now enabled in the NVIC.

See also

ISR_addToIntTable(), ISR_setPriority(), ISR_Disable()

ISR_Disable()

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

Disable an interrupt in the NVIC.

Parameters

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

Postcondition

The interrupt is now disabled in the NVIC.

See also

ISR_Enable()

ISR_triggerInterrupt()

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

Generate a software-generated interrupt (SGI).

Precondition

Enable the ISR (and set priority as needed).

Enable all interrupts.

Parameters

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

Postcondition

The ISR should trigger once any higher priority ISRs return.

See also

ISR_clearPending()

6 Data Structure Documentation

6.1 Fifo_t Struct Reference

Data Fields

- volatile uint32_t * **buffer**

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

  length of `buffer`

- volatile uint32_t **frontIdx**

    idx of front of FIFO
- volatile uint32_t **backIdx**

    idx of back of FIFO

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

- Fifo.c

6.2 GpioPort_t Struct Reference

Data Fields

- const uint32_t **BASE_ADDRESS**
- const uint32_t **DATA_REGISTER**
- bool **isInit**

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

- GPIO.c

6.3 Led_t Struct Reference

Data Fields

- GpioPort_t **GPIO_PORT_PTR**

    pointer to GPIO port data structure
- GpioPin_t **GPIO_PIN**

    GPIO pin number.
- volatile uint32_t ∗ **gpioDataRegister**
- bool **isOn**

    state indicator
- bool **isInit**

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

- Led.c

6.4 Timer_t Struct Reference

Data Fields

- const timerName_t **NAME**
- const uint32_t **BASE_ADDR**
- register_t **controlRegister**
- register_t **intervalLoadRegister**
- register_t **interruptClearRegister**
- bool **isInit**

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

- Timer.c

6.5 Uart_t Struct Reference

Data Fields

- const uint32_t **BASE_ADDRESS**
- register_t **FLAG_R_ADDRESS**
- GpioPort_t **GPIO_PORT**

  pointer to GPIO port data structure
- GpioPin_t **RX_PIN_NUM**

  GPIO pin number.
- GpioPin_t **TX_PIN_NUM**

  GPIO pin number.
- bool **isInit**

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

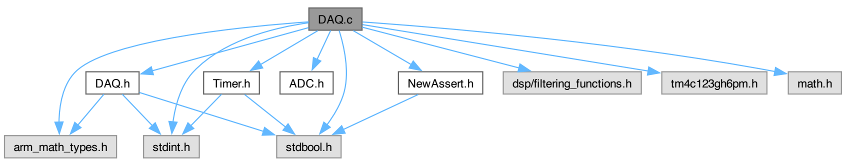
- UART.c

7 File Documentation

7.1 DAQ.c File Reference

Source code for DAQ module.

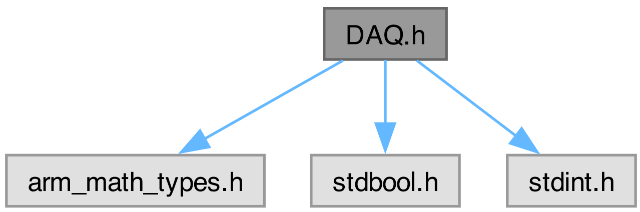
Include dependency graph for DAQ.c:



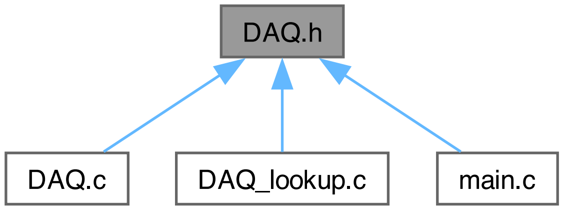
7.2 DAQ.h File Reference

Application software for handling data acquision (DAQ) functions.

Include dependency graph for DAQ.h:



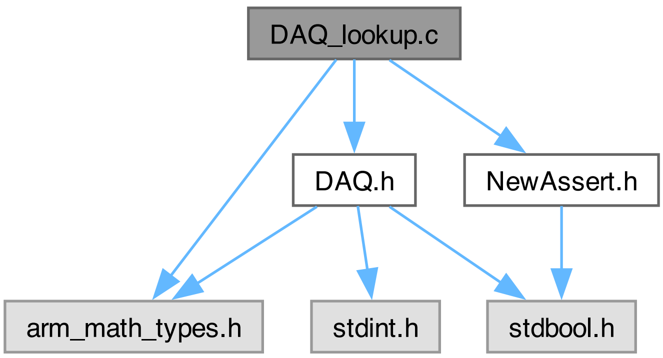
This graph shows which files directly or indirectly include this file:



7.3 DAQ_lookup.c File Reference

Source code for DAQ module's lookup table.

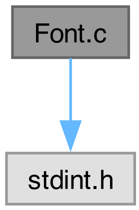
Include dependency graph for DAQ_lookup.c:



7.4 Font.c File Reference

Contains bitmaps for a selection of ASCII characters.

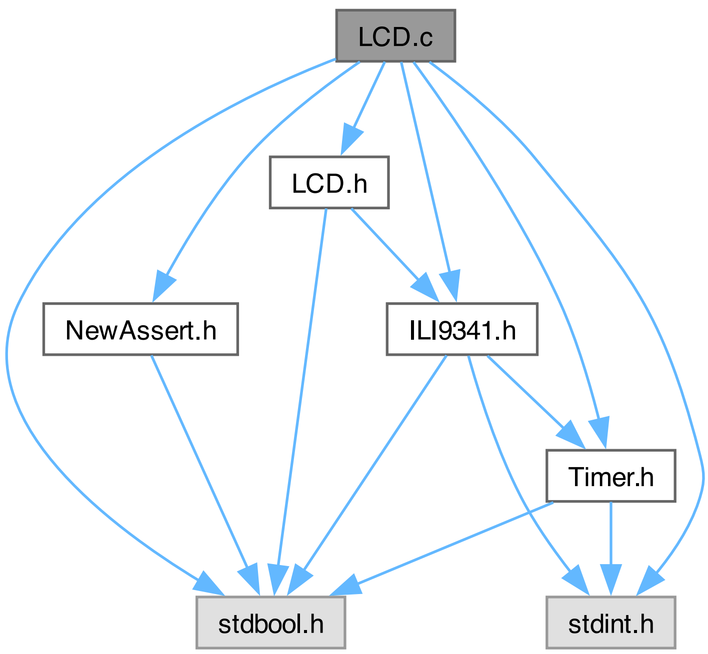
Include dependency graph for Font.c:



7.5 LCD.c File Reference

Source code for LCD module.

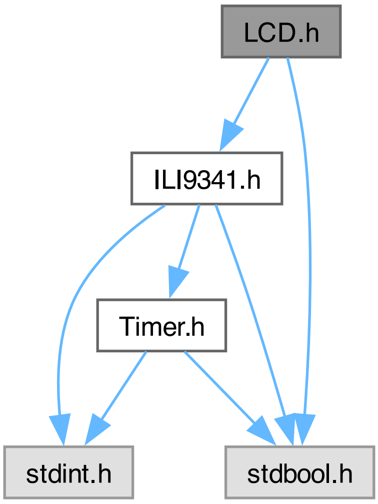
Include dependency graph for LCD.c:



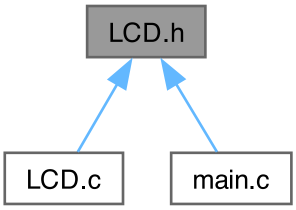
7.6 LCD.h File Reference

Header file for LCD module.

Include dependency graph for LCD.h:



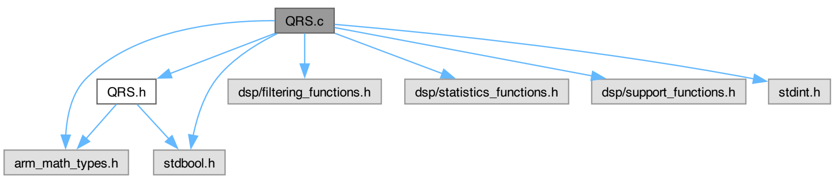
This graph shows which files directly or indirectly include this file:



7.7 QRS.c File Reference

Source code for QRS detection module.

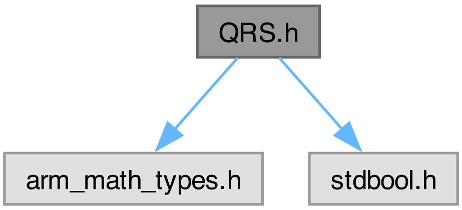
Include dependency graph for QRS.c:



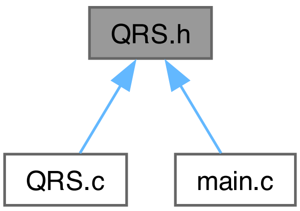
7.8 QRS.h File Reference

Header file for QRS detection module.

Include dependency graph for QRS.h:



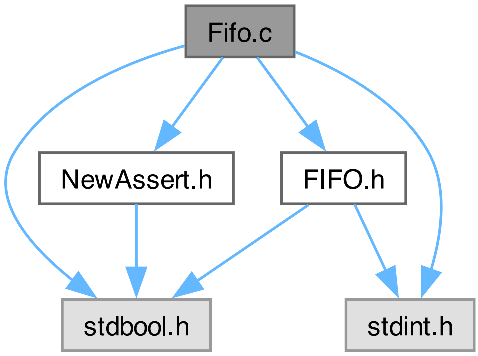
This graph shows which files directly or indirectly include this file:



7.9 Fifo.c File Reference

Source code for FIFO buffer module.

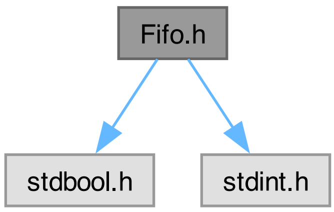
Include dependency graph for Fifo.c:



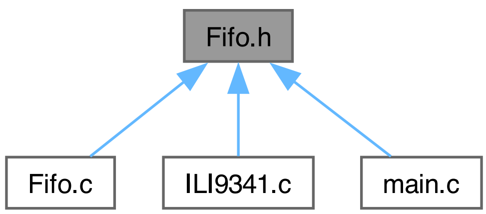
7.10 Fifo.h File Reference

Header file for FIFO buffer implementation.

Include dependency graph for Fifo.h:



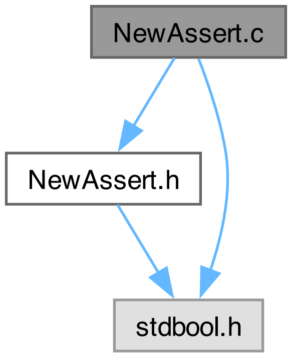
This graph shows which files directly or indirectly include this file:



7.11 NewAssert.c File Reference

Source code for custom `assert` implementation.

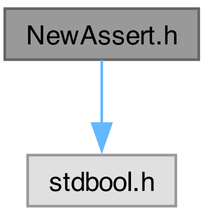
Include dependency graph for NewAssert.c:



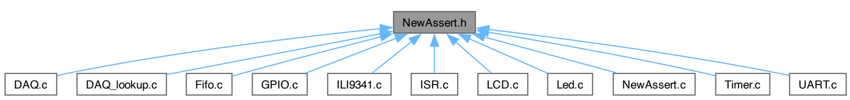
7.12 NewAssert.h File Reference

Header file for custom `assert` implementation.

Include dependency graph for NewAssert.h:



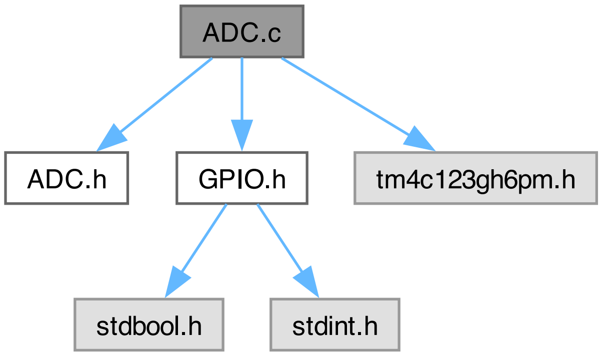
This graph shows which files directly or indirectly include this file:



7.13 ADC.c File Reference

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

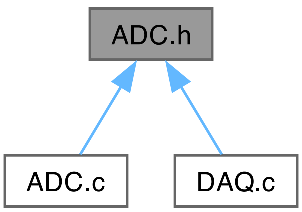
Include dependency graph for ADC.c:



7.14 ADC.h File Reference

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

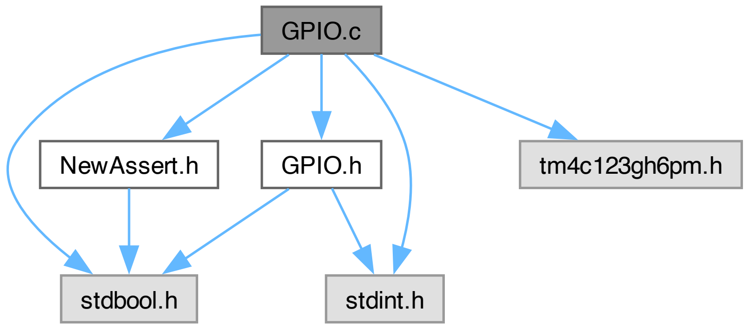
This graph shows which files directly or indirectly include this file:



7.15 GPIO.c File Reference

Source code for GPIO module.

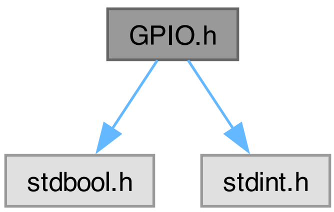
Include dependency graph for GPIO.c:



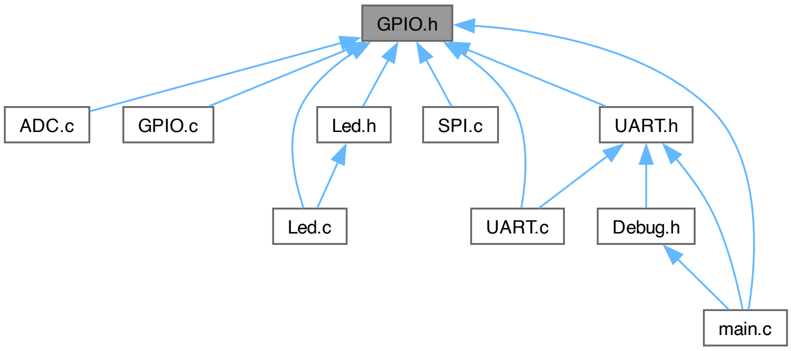
7.16 GPIO.h File Reference

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

Include dependency graph for GPIO.h:



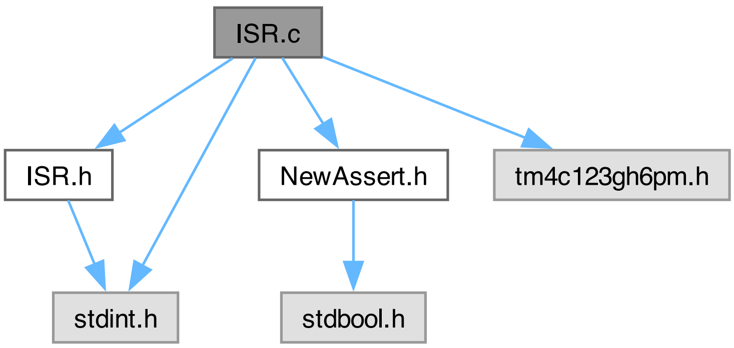
This graph shows which files directly or indirectly include this file:



7.17 ISR.c File Reference

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

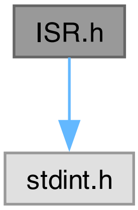
Include dependency graph for ISR.c:



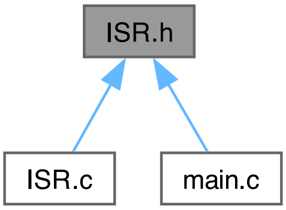
7.18 ISR.h File Reference

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

Include dependency graph for ISR.h:



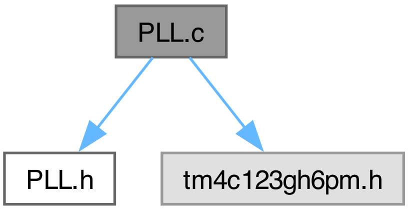
This graph shows which files directly or indirectly include this file:



7.19 PLL.c File Reference

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

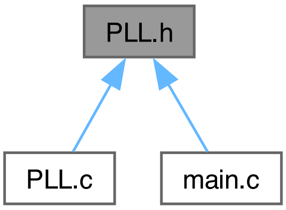
Include dependency graph for PLL.c:



7.20 PLL.h File Reference

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

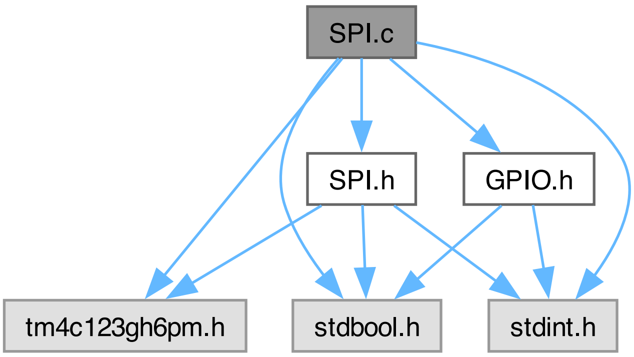
This graph shows which files directly or indirectly include this file:



7.21 SPI.c File Reference

Source code for serial peripheral interface (SPI) module.

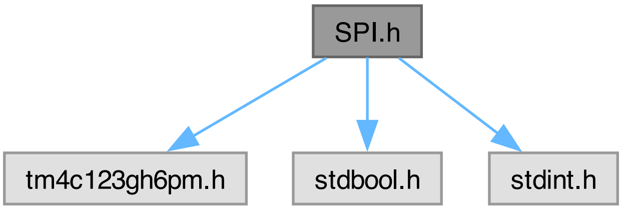
Include dependency graph for SPI.c:



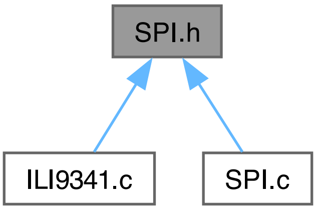
7.22 SPI.h File Reference

Header file for serial peripheral interface (SPI) module.

Include dependency graph for SPI.h:



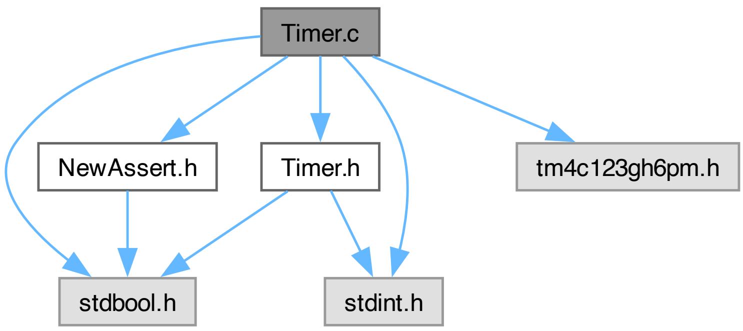
This graph shows which files directly or indirectly include this file:



7.23 Timer.c File Reference

Source code for Timer module.

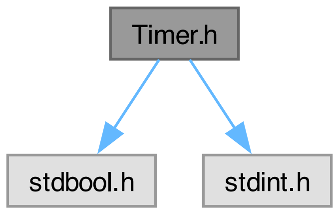
Include dependency graph for Timer.c:



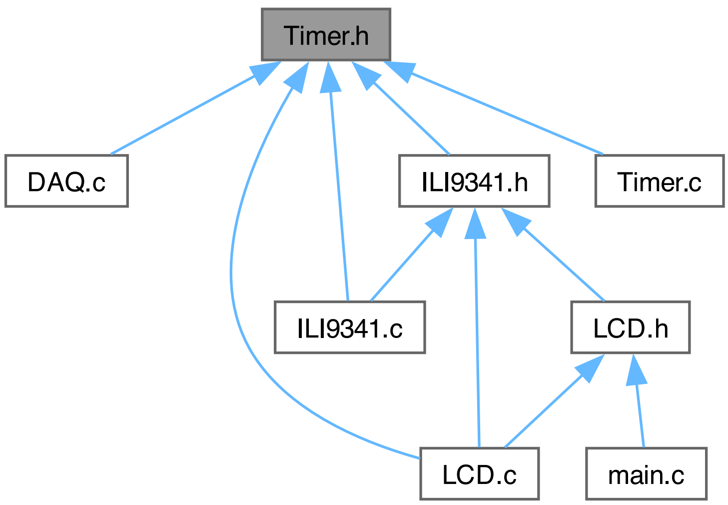
7.24 Timer.h File Reference

Device driver for general-purpose timer modules.

Include dependency graph for Timer.h:



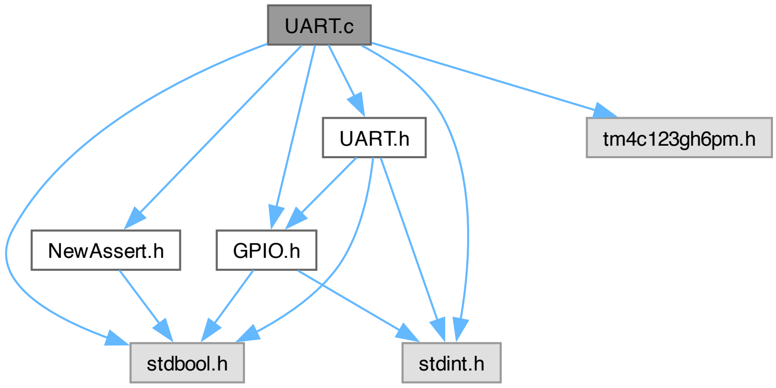
This graph shows which files directly or indirectly include this file:



7.25 UART.c File Reference

Source code for UART module.

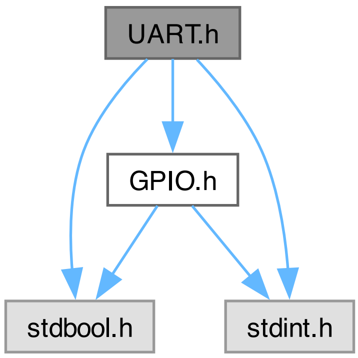
Include dependency graph for UART.c:



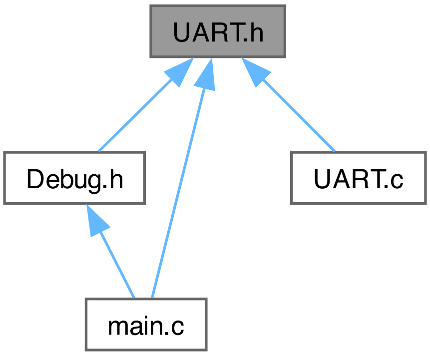
7.26 UART.h File Reference

Driver module for serial communication via UART0 and UART 1.

Include dependency graph for UART.h:



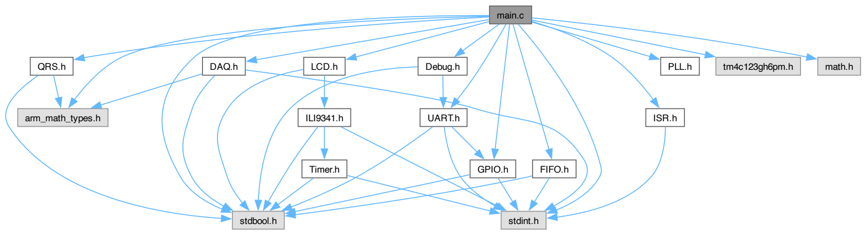
This graph shows which files directly or indirectly include this file:



7.27 main.c File Reference

Main program file.

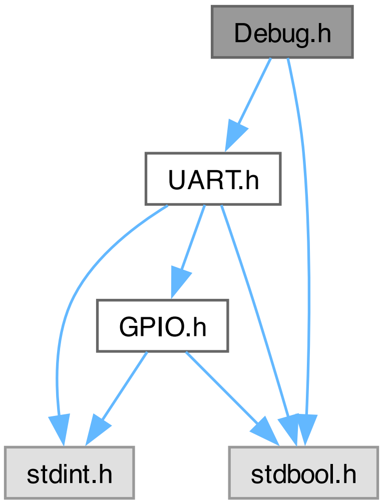
Include dependency graph for main.c:



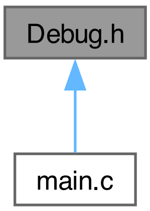
7.28 Debug.h File Reference

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

Include dependency graph for Debug.h:



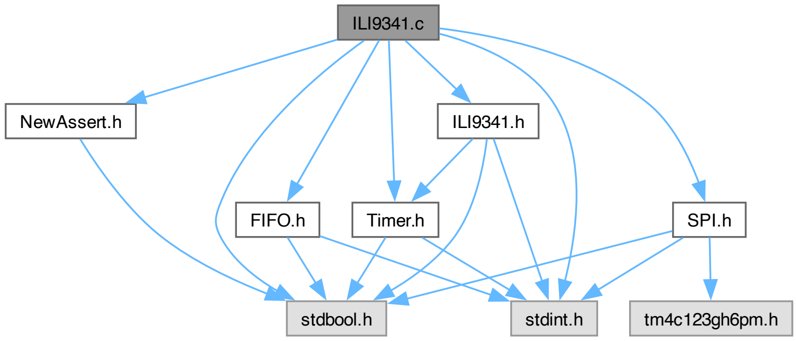
This graph shows which files directly or indirectly include this file:



7.29 ILI9341.c File Reference

Source code for ILI9341 module.

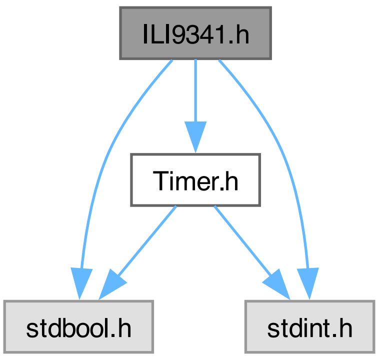
Include dependency graph for ILI9341.c:



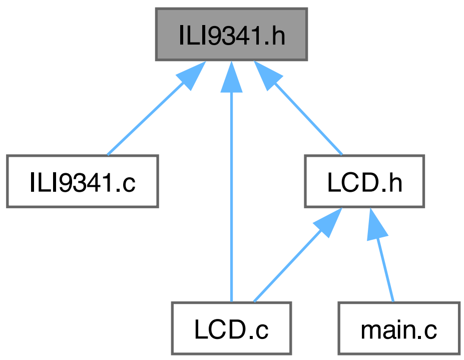
7.30 ILI9341.h File Reference

Driver module for interfacing with an ILI9341 LCD driver.

Include dependency graph for ILI9341.h:



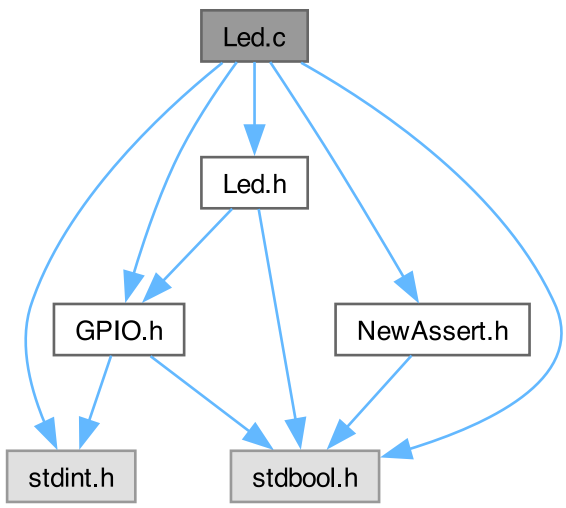
This graph shows which files directly or indirectly include this file:



7.31 Led.c File Reference

Source code for LED module.

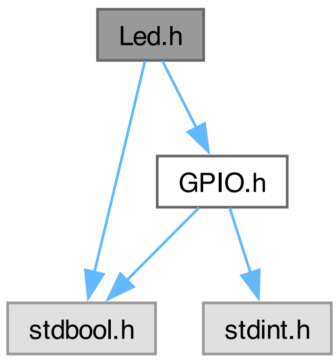
Include dependency graph for Led.c:



7.32 Led.h File Reference

Interface for LED module.

Include dependency graph for Led.h:



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