uHeartMonitor: An ECG-based Heart Rate Monitor

Generated by Doxygen 1.9.8

1 Overview

HeartMonitor is a personal project that I made to increase my experience in embedded software engineering and apply my previous coursework in biomedical engineering. Essentially, it's a fully-functional, ECG-based heart rate monitor that runs on the popular Tiva LaunchPad evaluation kit for the TM4C123 microcontroller.

Github Repository Link: `https://github.com/bryanmcelvy/microHeartMonitor`

Introduction: Link

2 Introduction

2.1 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).

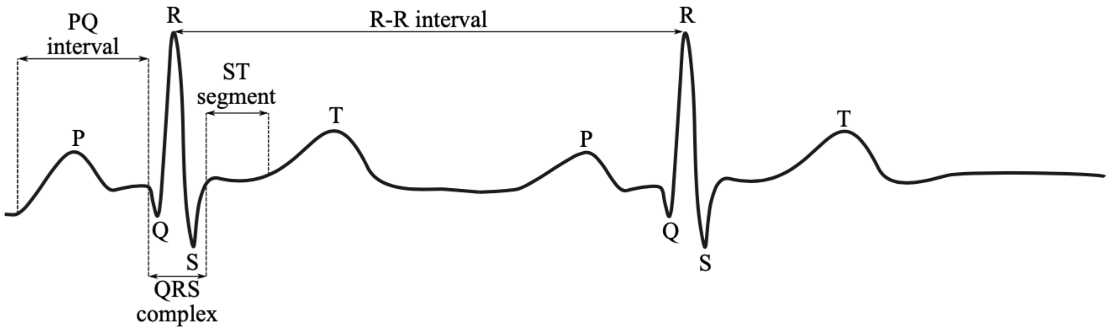


Figure 3. Sample ECG curve.

The uHeartMonitor 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.

2.2 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.

2.3 Disclaimer

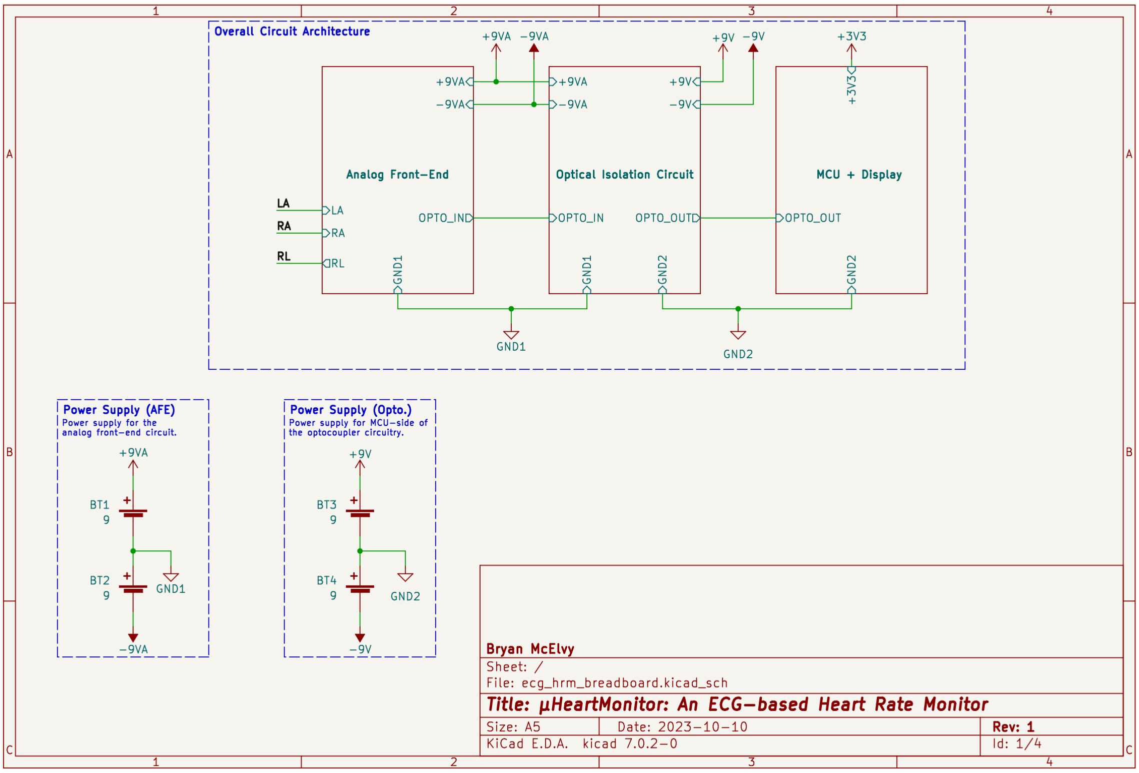
This project is neither a product nor a medical device (by any legal definition, anyway), and is not intended to be either or both of things now or in the future. It is simply a passion project.

2.4 Key Terms

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

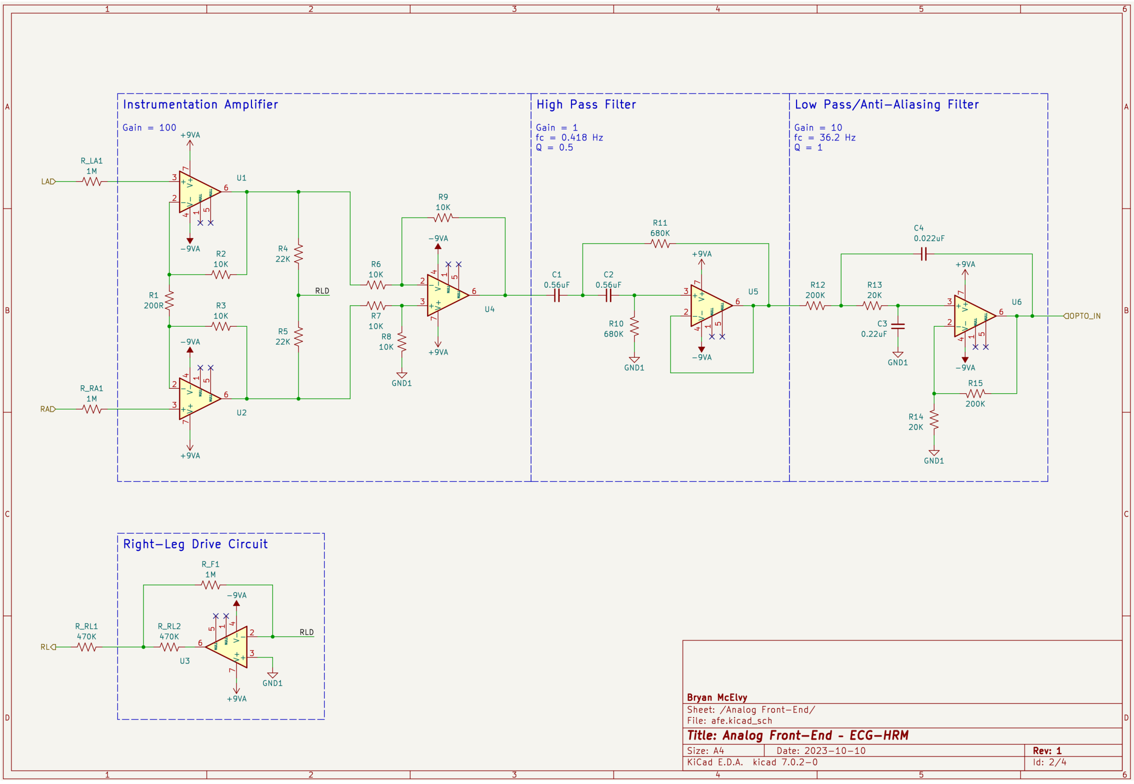
3 Materials & Methods

3.1 Hardware Design


Overall Circuit Architecture
+9VA −9VA
+9V −9V
+3V3
Analog Front−End
Optical Isolation Circuit
MCU + Display
+9VA
−9VA
+9VA
−9VA
+9V
−9V
+3V3
LA
RA
RL
OPTO_IND
OPTO_IN
OPTO_OUT
OPTO_OUT
GND1
GND1
GND2
GND2
GND1
GND2
Power Supply (AFE)
Power supply for the analog front−end circuit.
+9VA
BT1 9
BT2 9
GND1
−9VA
Power Supply (Opto.)
Power supply for MCU−side of the optocoupler circuitry.
+9V
BT3 9
BT4 9
GND2
−9V
Bryan McElvy
Sheet: /
File: ecg_hrm_breadboard.kicad_sch
Title: µHeartMonitor: An ECG−based Heart Rate Monitor
Size: A5 Date: 2023−10−10
Rev: 1
KiCad E.D.A. kicad 7.0.2−0
Id: 1/4
1 2 3 4
A B C

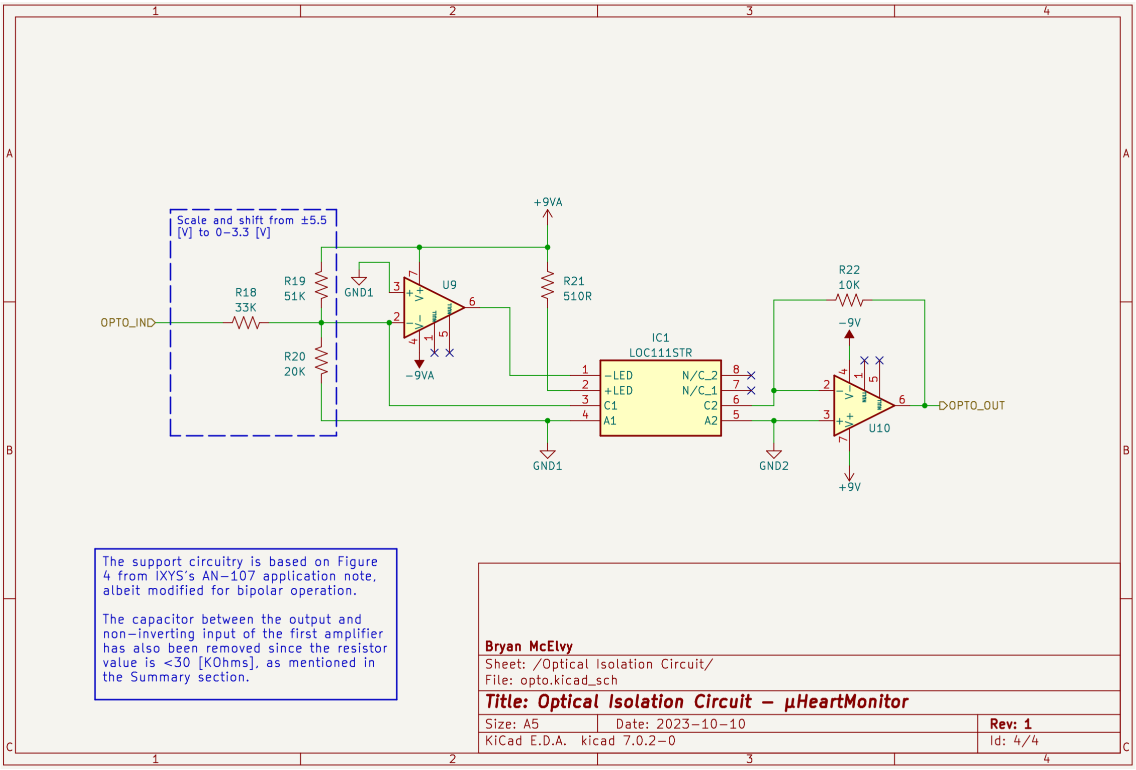
The hardware is divided into three modules: the analog-front end (AFE), the optical isolation circuit, and the micro-controller/display circuit.

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

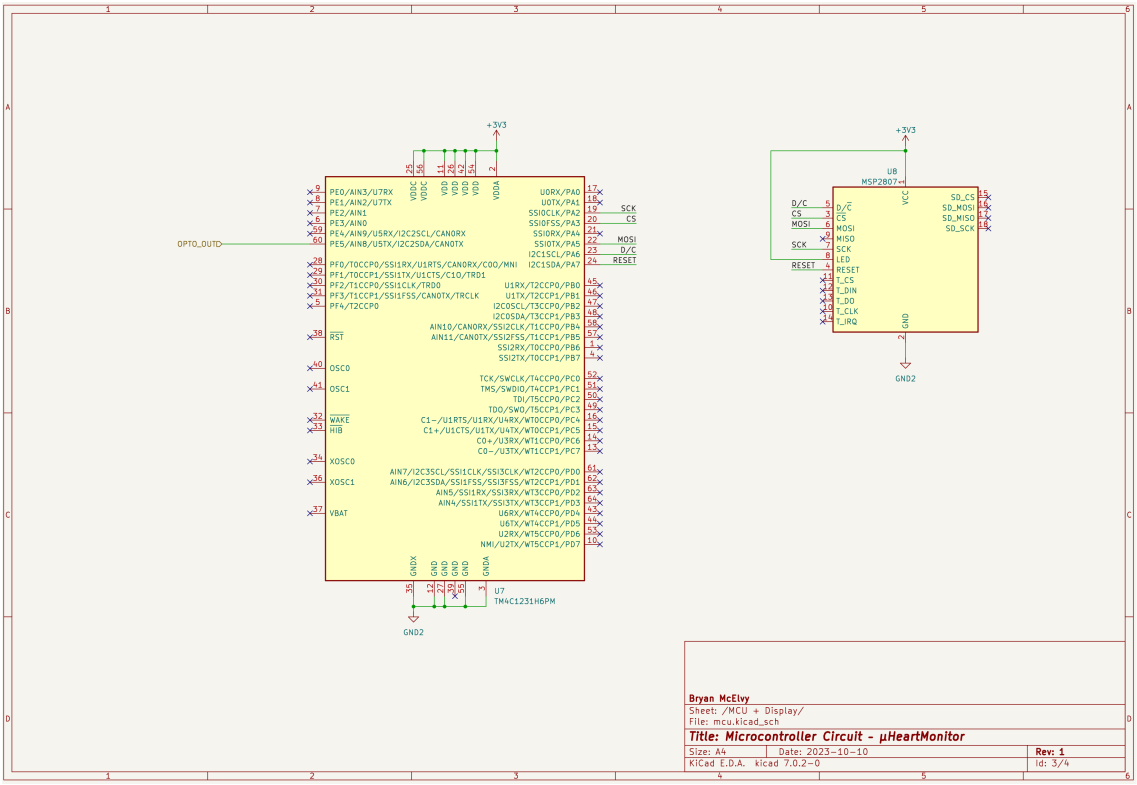
Optical Isolation Circuitry


Scale and shift from ±5.5 [V] to 0–3.3 [V]
OPTO_IN
R18
33K
R19
51K
R20
20K
GND1
U9
−9VA
+9VA
R21
510R
IC1
LOC111STR
−LED
+LED
C1
A1
N/C_2
N/C_1
C2
A2
GND1
GND2
R22
10K
−9V
+9V
U10
OPTO_OUT
The support circuitry is based on Figure 4 from IXYS's AN–107 application note, albeit modified for bipolar operation.
The capacitor between the output and non–inverting input of the first amplifier has also been removed since the resistor value is <30 [KOhms], as mentioned in the Summary section.
Bryan McElvy
Sheet: /Optical Isolation Circuit/
File: opto.kicad_sch
Title: Optical Isolation Circuit — µHeartMonitor
Size: A5
Date: 2023–10–10
Rev: 1
KiCad E.D.A. kicad 7.0.2–0
Id: 4/4

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 ± $5.5\ V$ 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.

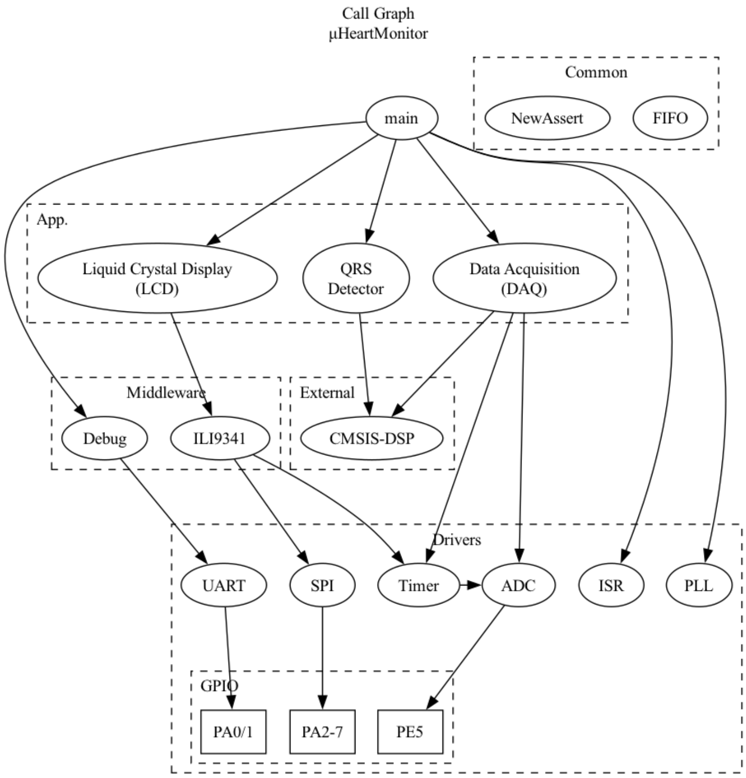
Microcontroller Circuit



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

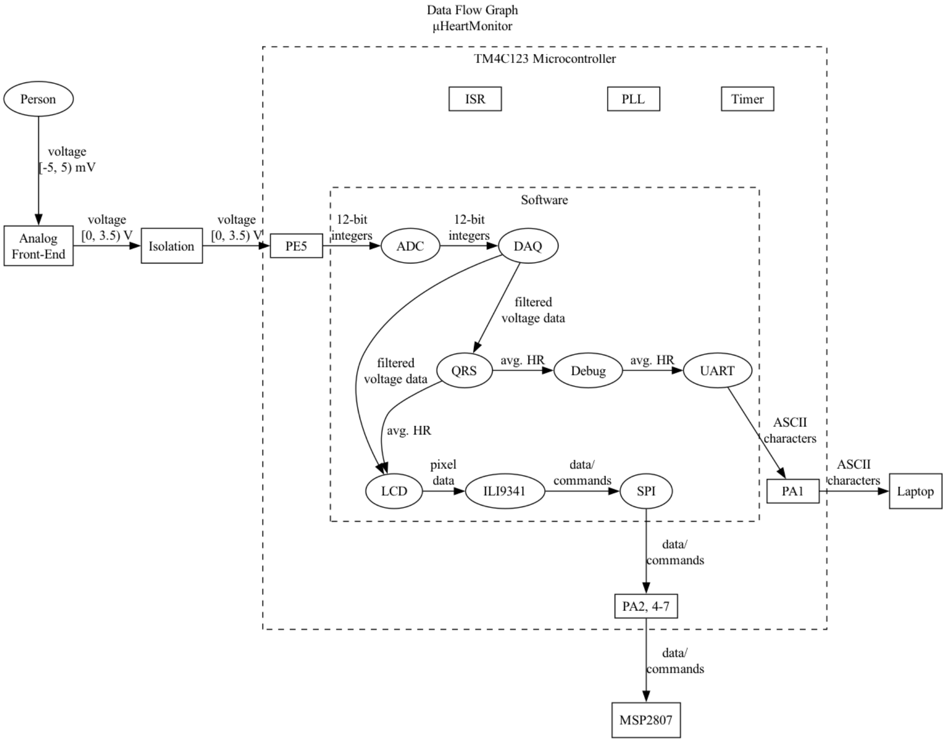
3.2 Software Architecture

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

3.3 Build Instructions

3.3.1 Hardware

WIP

3.3.2 Software

WIP

4 Results

4.1 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.

4.2 To-do

4.2.1 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

4.2.2 Software

- Expand the automated test suite

  Note

  See the other Todo List section for other software-related todos.

5 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.

6 Todo List

Module adc

Refactor to be more general.

Module qrs

Add heart rate variability (HRV) calculation.

File QRS.c

Add thresholding for bandpass filtered signal.

Add searchback procedure via RR intervals.

Add T-wave discrimination.

Global QRS_applyDecisionRules (const float32_t yn[])

Write implementation explanation

Module spi

Remove statically-allocated data structures for unused SSIs.

7 Bug List

Global QRS_applyDecisionRules (const float32_t yn[])

The current implementation processes one block of data at a time and discards the entire block immediately after. As a result, QRS complexes that are cutoff between one block and another are not being counted.

8 Topic Index

8.1 Topics

Here is a list of all topics with brief descriptions:

9 Data Structure Index

9.1 Data Structures

Here are the data structures with brief descriptions:

10 File Index

10.1 File List

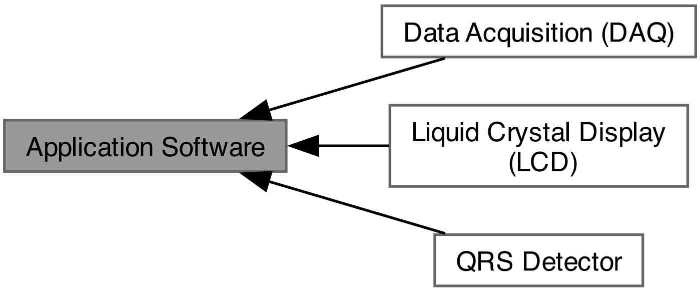
Here is a list of all documented files with brief descriptions:

11 Topic Documentation

11.1 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.

11.1.1 Detailed Description

Application-specific software modules.

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

11.1.2 Data Acquisition (DAQ)

Module for managing data acquisition (DAQ) functions.

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]

  Coefficients of the 60 [Hz] notch filter in biquad (AKA second-order section, or "sos") form.
- static const float32_t COEFFS_BANDPASS [NUM_COEFFS_DAQ_BANDPASS]

  Coefficients of the bandpass filter in biquad (AKA second-order section, or "sos") form.
- 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.

11.1.2.1 Detailed Description

Module for managing data acquisition (DAQ) functions.

11.1.2.2 Enumeration Type Documentation

anonymous enum

```
anonymous 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
};
```

11.1.2.3 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 $f_s = 200Hz$).

```
                    {
    ADC_Init();

    Timer_t DAQ_Timer = Timer_Init(TIMER3);
    Timer_setMode(DAQ_Timer, PERIODIC, UP);
    Timer_enableAdcTrigger(DAQ_Timer);
    Timer_setInterval_ms(DAQ_Timer, SAMPLING_PERIOD_MS);
    Timer_Start(DAQ_Timer);

    return;
}
```

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()

```
                              {
    return (uint16_t) (ADC0_SSFIFO3_R & 0xFFF);
}
```

DAQ_acknowledgeInterrupt()

```
void DAQ_acknowledgeInterrupt (
             void )
```

Acknowledge the ADC interrupt.

Precondition

This should be used within an interrupt handler.

```
                                    {
    ADC0_ISC_R |= 0x08;
    return;
}
```

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()

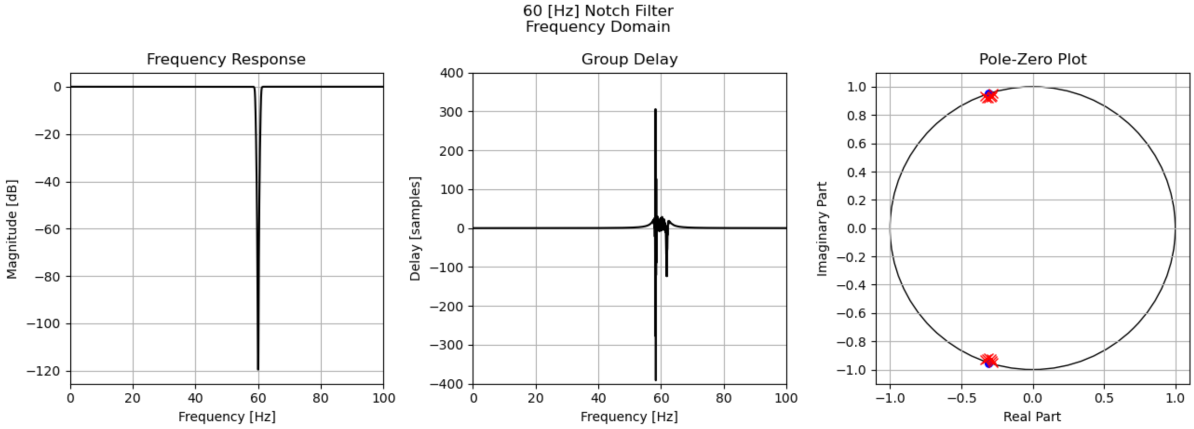


Figure 1 Frequency domain parameters for the notch filter.

```
                                                       {
    float32_t outputSample = 0;

    arm_biquad_cascade_df1_f32(notchFilter, (const float32_t *) &inputSample, &outputSample, 1);
    assert(isfinite(outputSample));

    return outputSample;
}
```

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()

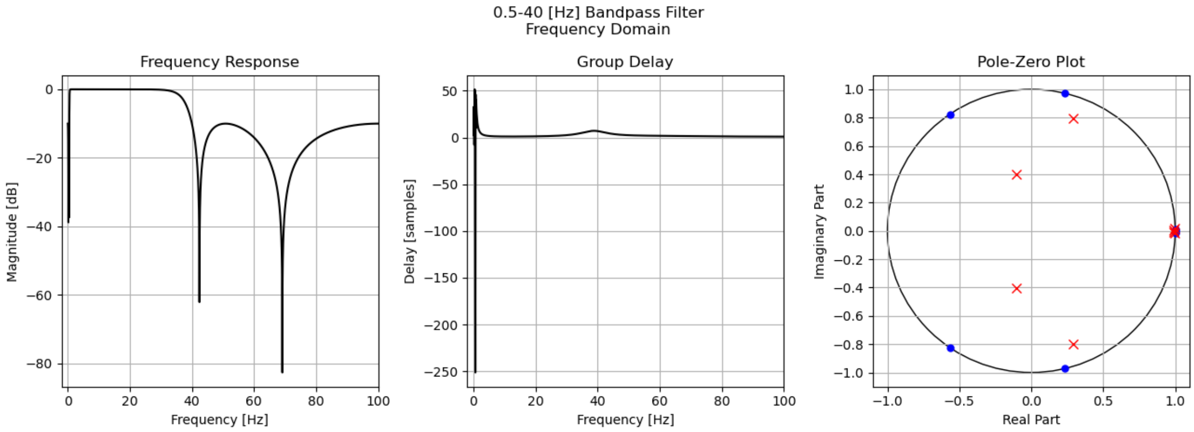


Figure 2 Frequency domain parameters for the bandpass filter.

```
                                                              {
    float32_t outputSample = 0;

    arm_biquad_cascade_df1_f32(bandpassFilter, (const float32_t *) &inputSample, &outputSample, 1);
    assert(isfinite(outputSample));

    return outputSample;
}
```

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.

```
                                                    {
    assert(sample < (1 « 12));
    return DAQ_LOOKUP_TABLE[sample];
}
```

11.1.2.4 Variable Documentation

COEFFS_NOTCH

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

Initial value:

```
= {

    0.88567328453063 96f, 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,
}
```

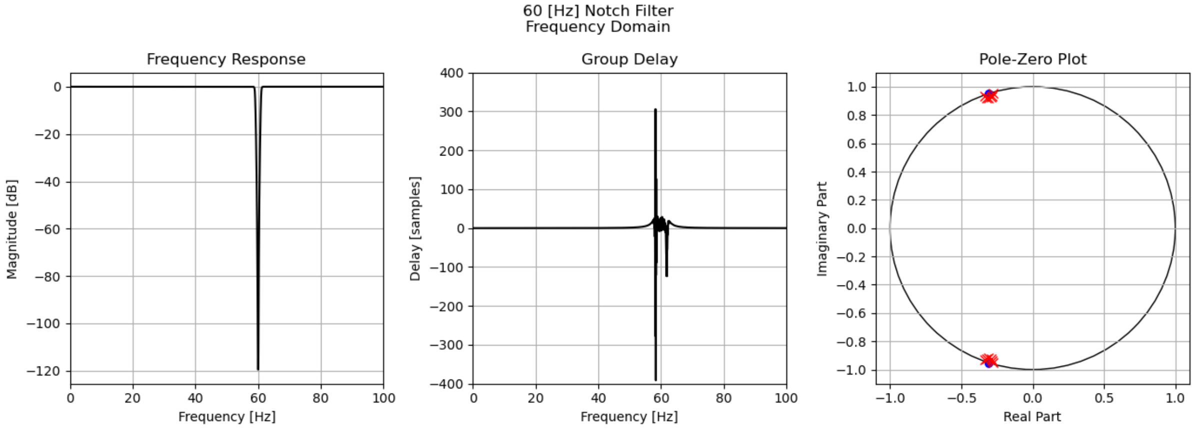
Coefficients of the 60 [Hz] notch filter in biquad (AKA second-order section, or "sos") form.

These coefficients were generated with the following Python code:

```
import numpy as np
from scipy import signal

fs = 200

sos_notch = signal.iirfilter(N=6, Wn=[59, 61], btype='bandstop', output='sos', fs=fs)
```

Note

CMSIS-DSP and Scipy use different formats for biquad filters. To convert the output to CMSIS-DSP format, the a_0 coefficients were removed from each section, and the other denominator coefficients were negated.



```
                                                        {
    // Section 1
    0.8856732845306396f, 0.5476464033126831f, 0.8856732845306396f,
    -0.5850160717964172f, -0.9409302473068237f,
    // Section 2
    1.0f, 0.6183391213417053f, 1.0f,
    -0.615153431892395f, -0.9412328004837036f,
    // Section 3
    1.0f, 0.6183391213417053f, 1.0f,
    -0.5631667971611023f, -0.9562366008758545f,
    // Section 4
    1.0f, 0.6183391213417053f, 1.0f,
    -0.6460562348365784f, -0.9568508863449097f,
    // Section 5
    1.0f, 0.6183391213417053f, 1.0f,
    -0.5554963946342468f, -0.9837208390235901f,
    // Section 6
    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,
}
```

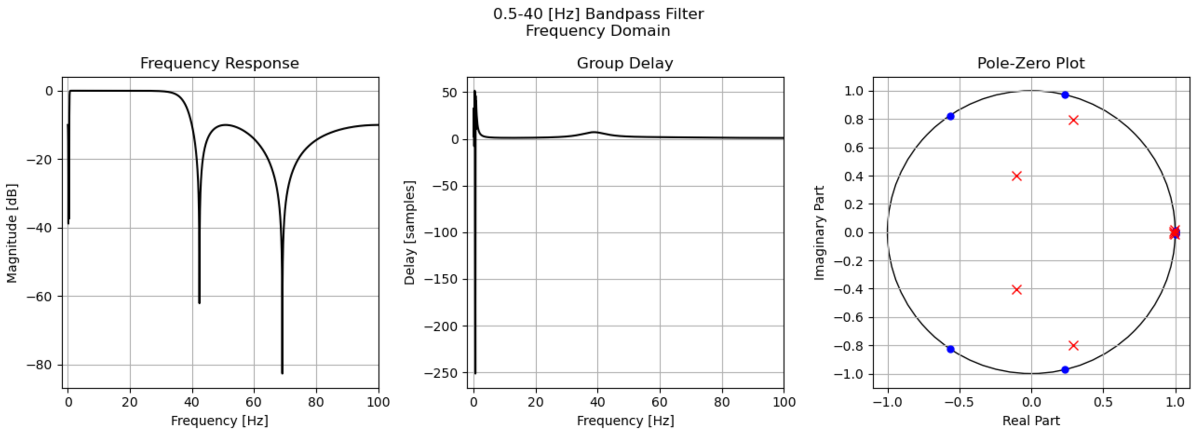
Coefficients of the bandpass filter in biquad (AKA second-order section, or "sos") form.

These coefficients were generated with the following Python code:

```
import numpy as np
from scipy import signal

fs = 200

sos_high = signal.iirfilter(N=4, Wn=0.5, btype="highpass", rs=10, ftype=’cheby2’, fs=fs, output=’sos’)
z_high, p_high, k_high = signal.sos2zpk(sos_high)

sos_low = signal.iirfilter(N=4, Wn=40, btype="lowpass", rs=10, ftype=’cheby2’, fs=fs, output=’sos’)
z_low, p_low, k_low = signal.sos2zpk(sos_low)

z_bandpass = np.concatenate([z_high, z_low])
p_bandpass = np.concatenate([p_high, p_low])
k_bandpass = k_high * k_low

sos_bandpass = signal.zpk2sos(z_bandpass, p_bandpass, k_bandpass)
```

Note

CMSIS-DSP and Scipy use different formats for biquad filters. To convert the output to CMSIS-DSP format, the a_0 coefficients were removed from each section, and the other denominator coefficients were negated.



```
                                                                  {
    // Section 1
    0.3240305185317993f, 0.3665695786476135f, 0.3240305185317993f,
    -0.20968256890773773f, -0.1729172021150589f,
    // Section 2
    1.0f, -0.4715292155742645f, 1.0f,
    0.5868059992790222f, -0.7193671464920044f,
    // Section 3
    1.0f, -1.9999638795852661f, 1.0f,
    1.9863483905792236f, -0.986438512802124f,
    // Section 4
    1.0f, -1.9997893571853638f, 1.0f,
    1.994096040725708f, -0.9943605065345764f,
};                                         /* clang-format on */
```

notchFiltStruct

```
const Filter_t notchFiltStruct = { NUM_STAGES_NOTCH, stateBuffer_Notch, COEFFS_NOTCH }  [static]
```

```
{ NUM_STAGES_NOTCH, stateBuffer_Notch, COEFFS_NOTCH };
```

bandpassFiltStruct

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

Initial value:

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

```
                                           { NUM_STAGES_BANDPASS, stateBuffer_Bandpass,
                                             COEFFS_BANDPASS };
```

DAQ_LOOKUP_TABLE

```
const float32_t DAQ_LOOKUP_TABLE[4096]  [static]
```

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

```
                                                     {
    -5.499999523162842f, -5.497313499450684f, -5.494627475738525f, -5.491940975189209f,
    -5.489254951477051f, -5.486568927764893f, -5.483882427215576f, -5.481196403503418f,
    -5.47851037979126f, -5.475823879241943f, -5.473137378692627f, -5.470451354980469f,
    -5.4677653312683105f, -5.465078830718994f, -5.462392807006836f, -5.459706783294678f,
    -5.457020282745361f, -5.454334259033203f, -5.451648235321045f, -5.4489617347717285f,
    -5.44627571105957f, -5.443589687347412f, -5.440903186798096f, -5.4382171630859375f,
    -5.435531139373779f, -5.432844638824463f, -5.430158615112305f, -5.4274725914001465f,
    -5.42478609085083f, -5.422100067138672f, -5.4194135665893555f, -5.416727066040039f,
    -5.414041042327881f, -5.411355018615723f, -5.408668518066406f, -5.405982494354248f,
    -5.40329647064209f, -5.400609970092773f, -5.397923946380615f, -5.395237922668457f,
    -5.392551422119141f, -5.389865398406982f, -5.387179374694824f, -5.384492874145508f,
    -5.38180685043335f, -5.379120826721191f, -5.376434326171875f, -5.373748302459717f,
    -5.371062278747559f, -5.368375778198242f, -5.365689277648926f, -5.363003253936768f,
    -5.360317230224609f, -5.357630729675293f, -5.354944705963135f, -5.352258682250977f,
    -5.34957218170166f, -5.346886157989502f, -5.344200134277344f, -5.341513633728027f,
    -5.338827610015869f, -5.336141586303711f, -5.3334550857543945f, -5.330769062042236f,
    -5.328083038330078f, -5.325396537780762f, -5.3227105140686035f, -5.320024490356445f,
    -5.317337989807129f, -5.314651966094971f, -5.311965465545654f, -5.309278964996338f,
    -5.30659294128418f, -5.3039069175720215f, -5.301220417022705f, -5.298534393310547f,
    -5.295848369598389f, -5.293161869049072f, -5.290475845336914f, -5.287789821624756f,
    -5.2851033210754395f, -5.282417297363281f, -5.279731273651123f, -5.277044773101807f,
    -5.274358749389648f, -5.27167272567749f, -5.268986225128174f, -5.266300201416016f,
    -5.263614177703857f, -5.260927677154541f, -5.258241176605225f, -5.255555152893066f,
    -5.252869129180908f, -5.250182628631592f, -5.247496604919434f, -5.244810581207275f,
    -5.242124080657959f, -5.239438056945801f, -5.236752033233643f, -5.234065532684326f,
    -5.231379508972168f, -5.22869348526001f, -5.226006984710693f, -5.223320960998535f,
    -5.220634937286377f, -5.2179484367370605f, -5.215262413024902f, -5.212576389312744f,
    -5.209889888763428f, -5.2072038650512695f, -5.204517364501953f, -5.201830863952637f,
    -5.1991448402404785f, -5.19645881652832f, -5.193772315979004f, -5.191086292266846f,
    -5.1884002685546875f, -5.185713768005371f, -5.183027744293213f, -5.180341720581055f,
    -5.177655220031738f, -5.17496919631958f, -5.172283172607422f, -5.1695966720581055f,
    -5.166910648345947f, -5.164224624633789f, -5.161538124084473f, -5.1588521003723145f,
    -5.156166076660156f, -5.15347957611084f, -5.150793075561523f, -5.148107051849365f,
    -5.145421028137207f, -5.142734527587891f, -5.140048503875732f, -5.137362480163574f,
    -5.134675979614258f, -5.1319899559021f, -5.129303932189941f, -5.126617431640625f,
    -5.123931407928467f, -5.121245384216309f, -5.118558883666992f, -5.115872859954834f,
    -5.113186836242676f, -5.110500335693359f, -5.107814311981201f, -5.105128288269043f,
    -5.102441787719727f, -5.099755764007568f, -5.097069263458252f, -5.0943827629089355f,
    -5.091696739196777f, -5.089010715484619f, -5.086324214935303f, -5.0836381912231445f,
    -5.080952167510986f, -5.07826566696167f, -5.075579643249512f, -5.0728936195373535f,
    -5.070207118988037f, -5.067521095275879f, -5.064835071563721f, -5.062148571014404f,
    -5.059462547302246f, -5.056776523590088f, -5.0540900230407715f, -5.051403999328613f,
    -5.048717498779297f, -5.046031475067139f, -5.043344974517822f, -5.040658950805664f,
    -5.037972927093506f, -5.0352864265441895f, -5.032600402832031f, -5.029914379119873f,
    -5.027227878570557f, -5.024541854858398f, -5.02185583114624f, -5.019169330596924f,
    -5.016483306884766f, -5.013797283172607f, -5.011110782623291f, -5.008424758911133f,
    -5.005738735198975f, -5.003052234649658f, -5.0003662109375f, -4.997680187225342f,
    -4.994993686676025f, -4.992307662963867f, -4.989621162414551f, -4.986934661865234f,
    -4.984248638153076f, -4.981562614440918f, -4.978876113891602f, -4.976190090179443f,
    -4.973504066467285f, -4.970817565917969f, -4.9681315422058105f, -4.965445518493652f,
    -4.962759017944336f, -4.960072994232178f, -4.9573869705200195f, -4.954700469970703f,
    -4.952014446258545f, -4.949328422546387f, -4.94664192199707f, -4.943955898284912f,
    -4.941269397735596f, -4.9385833740234375f, -4.935896873474121f, -4.933210849761963f,
    -4.930524826049805f, -4.927838325500488f, -4.92515230178833f, -4.922466278076172f,
    -4.9197797775268555f, -4.917093753814697f, -4.914407730102539f, -4.911721229553223f,
    -4.9090352058410645f, -4.906349182128906f, -4.90366268157959f, -4.900976657867432f,
    -4.898290634155273f, -4.895604133605957f, -4.892918109893799f, -4.890232086181641f,
    -4.887545585632324f, -4.884859561920166f, -4.88217306137085f, -4.879486560821533f,
    -4.8768005371093750f, -4.874114513397217f, -4.8714280128479f, -4.868741989135742f,
    -4.866055965423584f, -4.863369464874268f, -4.860683441162109f, -4.857997417449951f,
    -4.855310916900635f, -4.852624893188477f, -4.849938869476318f, -4.847252368927002f,
    -4.844566345214844f, -4.8418803215026855f, -4.839193820953369f, -4.836507320404053f,
    -4.8338212966918945f, -4.831135272979736f, -4.82844877243042f, -4.825762748718262f,
    -4.8230767250061035f, -4.820390224456787f, -4.817704200744629f, -4.815018177032471f,
    -4.812331676483154f, -4.809645652770996f, -4.806959629058838f, -4.8042731285095215f,
    -4.801587104797363f, -4.798901081085205f, -4.796214580535889f, -4.7935285568237305f,
    -4.790842533111572f, -4.788156032562256f, -4.785470008850098f, -4.7827839851379395f,
    -4.780097484588623f, -4.777411460876465f, -4.774724960327148f, -4.772038459777832f,
    -4.769352436065674f, -4.766666412353516f, -4.763979911804199f, -4.761293888092041f,
    -4.758607864379883f, -4.755921363830566f, -4.753235340118408f, -4.75054931640625f,
    -4.747862815856934f, -4.745176792144775f, -4.742490768432617f, -4.739804267883301f,
    -4.737118244171143f, -4.734432220458984f, -4.731745719909668f, -4.729059219360352f,
    -4.726373195648193f, -4.723687171936035f, -4.721000671386719f, -4.7183146476745605f,
    -4.715628623962402f, -4.712942123413086f, -4.710256099700928f, -4.7075700759887695f,
    -4.704883575439453f, -4.702197551727295f, -4.699511528015137f, -4.69682502746582f,
    -4.694139003753662f, -4.691452980041504f, -4.6887664794921875f, -4.686080455780029f,
    -4.683394432067871f, -4.680707931518555f, -4.6780219078063965f, -4.675335884094238f,
    -4.672649383544922f, -4.6699628829956055f, -4.667276859283447f, -4.664590358734131f,
    -4.661904335021973f, -4.6592183113098145f, -4.656531810760498f, -4.65384578704834f,
    -4.651159763336182f, -4.648473262786865f, -4.645787239074707f, -4.643101215362549f,
    -4.640414714813232f, -4.637728691101074f, -4.635042667388916f, -4.6323561668396f,
    -4.629670143127441f, -4.626983642578125f, -4.624297618865967f, -4.62161111831665f,
    -4.618925094604492f, -4.616239070892334f, -4.613552570343018f, -4.610866546630859f,
    -4.608180522918701f, -4.605494022369385f, -4.602807998657227f, -4.600121974945068f,
    -4.597435474395752f, -4.594749450683594f, -4.5920634269714355f, -4.589376926422119f,
    -4.586690902709961f, -4.584004878997803f, -4.581318378448486f, -4.578632354736328f,
    -4.57594633102417f, -4.5732598304748535f, -4.570573806762695f, -4.567887783050537f,
    -4.565201282501221f, -4.5625152587890625f, -4.559828758239746f, -4.557142257690430f,
    -4.5544562339782715f, -4.551770210266113f, -4.549083709716797f, -4.546397686004639f,
    -4.5437116622924805f, -4.541025161743164f, -4.538339138031006f, -4.535653114318848f,
    -4.532966613769531f, -4.530280590057373f, -4.527594566345215f, -4.524908065795898f,
    -4.52222204208374f, -4.519535541534424f, -4.516849517822266f, -4.514163017272949f,
    -4.511476993560791f, -4.508790969848633f, -4.506104469299316f, -4.503418445587158f,
    -4.500732421875f, -4.498045921325684f, -4.495359897613525f, -4.492673873901367f,
    -4.489987373352051f, -4.487301349639893f, -4.484615325927734f, -4.481928825378418f,
    -4.47924280166626f, -4.476556777954102f, -4.473870277404785f, -4.471184253692627f,
    -4.468498229980469f, -4.465811729431152f, -4.463125705718994f, -4.460439682006836f,
    -4.4577531814575195f, -4.455067157745361f, -4.452380657196045f, -4.4496941566467285f,
    -4.44700813293457f, -4.444322109222412f, -4.441635608673096f, -4.4389495849609375f,
    -4.436263561248779f, -4.433577060699463f, -4.430891036987305f, -4.4282050132751465f,
    -4.42551851272583f, -4.422832489013672f, -4.420146465301514f, -4.417459964752197f,
    -4.414773941040039f, -4.412087440490723f, -4.4094014167785645f, -4.406714916229248f,
    -4.40402889251709f, -4.401342868804932f, -4.398656368255615f, -4.395970344543457f,
    -4.393284320831299f, -4.390597820281982f, -4.387911796569824f, -4.385225772857666f,
    -4.38253927230835f, -4.379853248596191f, -4.377167224884033f, -4.374480724334717f,
    -4.371794700622559f, -4.3691086769104f, -4.366422176361084f, -4.363736152648926f,
    -4.361050128936768f, -4.358363628387451f, -4.355677604675293f, -4.352991580963135f,
    -4.350305080413818f, -4.34761905670166f, -4.344932556152344f, -4.342246055603027f,
    -4.339560031890869f, -4.336874008178711f, -4.3341875076293945f, -4.331501483917236f,
    -4.328815460205078f, -4.326128959655762f, -4.3234429359436035f, -4.320756912231445f,
    -4.318070411682129f, -4.315384387969971f, -4.3126983642578125f, -4.310011863708496f,
    -4.30732536315918f, -4.3046393394470215f, -4.301953315734863f, -4.299266815185547f,
    -4.296580791473389f, -4.2938947677612305f, -4.291208267211914f, -4.288522243499756f,
    -4.285836219787598f, -4.283149719238281f, -4.280463695526123f, -4.277777671813965f,
    -4.275091171264648f, -4.27240514755249f, -4.269719123840332f, -4.267032623291016f,
    -4.264346599578857f, -4.261660575866699f, -4.258974075317383f, -4.256288051605225f,
    -4.253602027893066f, -4.25091552734375f, -4.248229503631592f, -4.245543479919434f,
    -4.242856979370117f, -4.240170478820801f, -4.237484455108643f, -4.234797954559326f,
    -4.232111930847168f, -4.22942590713501f, -4.226739406585693f, -4.224053382873535f,
    -4.221367359161377f, -4.2186808586120605f, -4.215994834899902f, -4.213308811187744f,
    -4.210622310638428f, -4.2079362869262695f, -4.205250263214111f, -4.202563762664795f,
    -4.1998772621154785f, -4.19719123840332f, -4.194505214691162f, -4.191818714141846f,
    -4.1891326904296875f, -4.186446666717529f, -4.183760166168213f, -4.181074142456055f,
    -4.1783881187438965f, -4.17570161819458f, -4.173015594482422f, -4.170329570770264f,
    -4.167643070220947f, -4.164957046508789f, -4.162271022796631f, -4.1595845222473145f,
    -4.156898498535156f, -4.154212474822998f, -4.151525974273682f, -4.148839950561523f,
    -4.146153926849365f, -4.143467426300049f, -4.140781402587891f, -4.138095378875732f,
    -4.135408878326416f, -4.1327223777771f, -4.130036354064941f, -4.127349853515625f,
    -4.124663829803467f, -4.121977806091309f, -4.119291305541992f, -4.116605281829834f,
    -4.113919258117676f, -4.111232757568359f, -4.108546733856201f, -4.105860710144043f,
    -4.103174209594727f, -4.100488185882568f, -4.09780216217041f, -4.095115661621094f,
    -4.092429161071777f, -4.089743137359619f, -4.087057113647461f, -4.0843706130981445f,
    -4.081684589385986f, -4.078998565673828f, -4.076312065124512f, -4.0736260414123535f,
    -4.070940017700195f, -4.068253517150879f, -4.065567493438721f, -4.0628814697265625f,
    -4.060194969177246f, -4.057508945465088f, -4.05482292175293f, -4.052136421203613f,
    -4.049450397491455f, -4.046764373779297f, -4.0440778732299805f, -4.041391849517822f,
    -4.038705825805664f, -4.036019325256348f, -4.0333333015441895f, -4.030647277832031f,
    -4.027960777282715f, -4.025274276733398f, -4.02258825302124f, -4.019901752471924f,
    -4.017215728759766f, -4.014529705047607f, -4.011843204498291f, -4.009157180786133f,
    -4.006471157073975f, -4.003784656524658f, -4.0010986328125f, -3.998412609100342f,
    -3.9957263469696045f, -3.993040084838867f, -3.990354061126709f, -3.9876673221588135f,
    -3.9849812984466553f, -3.982295036315918f, -3.9796087741851807f, -3.9769227504730225f,
    -3.974236488342285f, -3.971550226211548f, -3.9688642024993896f, -3.9661779403686523f,
    -3.963491678237915f, -3.960805654525757f, -3.9581193923950195f, -3.9554331302642822f,
    -3.952747106552124f, -3.9500608444213867f, -3.9473745822906494f, -3.944688558578491f,
    -3.942002296447754f, -3.9393160343170166f, -3.9366300106048584f, -3.933943748474121f,
    -3.931257486343384f, -3.9285714626312256f, -3.9258852005004883f, -3.923198938369751f,
    -3.9205124378204346f, -3.9178261756896973f, -3.915140151977539f, -3.9124538898468018f,
    -3.9097676277160645f, -3.9070816040039062f, -3.904395341873169f, -3.9017090797424316f,
    -3.8990230560302734f, -3.896336793899536f, -3.893650531768799f, -3.8909645080566406f,
    -3.8882782459259033f, -3.885591983795166f, -3.882905960083008f, -3.8802192211151123f,
    -3.877533197402954f, -3.874846935272217f, -3.8721606731414795f, -3.8694746494293213f,
    -3.866788387298584f, -3.8641021251678467f, -3.8614161014556885f, -3.858729839324951f,
    -3.856043577194214f, -3.8533575534820557f, -3.8506712913513184f, -3.847985029220581f,
    -3.845299005508423f, -3.8426127433776855f, -3.8399264812469482f, -3.83724045753479f,
    -3.8345541954040527f, -3.8318679332733154f, -3.8291819095611572f, -3.82649564743042f,
    -3.8238093852996826f, -3.8211233615875244f, -3.818437099456787f, -3.81575083732605f,
    -3.8130648136138916f, -3.8103785514831543f, -3.807692050933838f, -3.8050057888031006f,
    -3.8023195266723633f, -3.799633502960205f, -3.7969472408294678f, -3.7942609786987305f,
    -3.7915749549865723f, -3.788888692855835f, -3.7862024307250977f, -3.7835164070129395f,
    -3.780829668045044f, -3.7781436443328857f, -3.7754573822021484f, -3.772771120071411f,
    -3.770085096359253f, -3.7673988342285156f, -3.7647125720977783f, -3.76202654838562f,
    -3.759340286254883f, -3.7566540241241455f, -3.7539680004119873f, -3.75128173828125f,
    -3.7485954761505127f, -3.7459094524383545f, -3.743223190307617f, -3.74053692817688f,
    -3.7378509044647217f, -3.7351646423339844f, -3.732478380203247f, -3.729792356491089f,
    -3.7271060943603516f, -3.7244198322296143f, -3.721733808517456f, -3.7190475463867188f,
    -3.7163612842559814f, -3.713675260543823f, -3.710988998413086f, -3.7083027362823486f,
    -3.7056167125701904f, -3.702930450439453f, -3.7002439498901367f, -3.6975576877593994f,
    -3.694871425628662f, -3.692185401916504f, -3.6894991397857666f, -3.6868128776550293f,
    -3.684126853942871f, -3.681440591812134f, -3.6787543296813965f, -3.6760683059692383f,
    -3.673381567001342 8f, -3.6706955432891846f, -3.6680092811584473f, -3.66532301902771f,
    -3.6626369953155518f, -3.6599507331848145f, -3.657264471054077f, -3.654578447341919f,
    -3.6518921852111816f, -3.6492059230804443f, -3.646519899368286f, -3.643833637237549f,
    -3.6411473751068115f, -3.6384613513946533f, -3.635775089263916f, -3.6330888271331787f,
    -3.6304028034210205f, -3.627716541290283f, -3.625030279159546f, -3.6223442554473877f,
    -3.6196579933166504f, -3.616971731185913f, -3.614285707473755f, -3.6115994453430176f,
    -3.6089131832122803f, -3.606227159500122f, -3.6035408973693848f, -3.6008546352386475f,
    -3.5981686115264893f, -3.595482349395752f, -3.5927958488464355f, -3.5901095867156982f,
    -3.587423324584961f, -3.5847373008728027f, -3.5820510387420654f, -3.579364776611328f,
    -3.57667875289917f, -3.5739924907684326f, -3.5713062286376953f, -3.568620204925537f,
    -3.5659334659576416f, -3.5632474422454834f, -3.560561180114746f, -3.557874917984009f,
    -3.5551888942718506f, -3.5525026321411133f, -3.549816370010376f, -3.5471303462982178f,
    -3.5444440841674805f, -3.541757822036743f, -3.539071798324585f, -3.5363855361938477f,
    -3.5336992740631104f, -3.531013250350952f, -3.528326988220215f, -3.5256407260894775f,
    -3.5229547023773193f, -3.520268440246582f, -3.5175821781158447f, -3.5148961544036865f,
    -3.512209892272949f, -3.509523630142212f, -3.5068376064300537f, -3.5041513442993164f,
    -3.501465082168579f, -3.498779058456421f, -3.4960927963256836f, -3.4934065341949463f,
    -3.490720510482788f, -3.4880337715148926f, -3.4853477478027344f, -3.482661485671997f,
    -3.4799752235412598f, -3.4772891998291016f, -3.4746029376983643f, -3.471916675567627f,
    -3.4692306518554688f, -3.4665443897247314f, -3.463858127593994f, -3.461171627044677f,
    -3.4584853649139404f, -3.4557993412017822f, -3.453113079071045f, -3.4504268169403076f,
    -3.4477407932281494f, -3.445054531097412f, -3.442368268966675f, -3.4396822452545166f,
    -3.4369959831237793f, -3.434309720993042f, -3.431623697280884f, -3.4289374351501465f,
    -3.426251173019409f, -3.423565149307251f, -3.4208788871765137f, -3.4181926250457764f,
    -3.415506601333618f, -3.412820339202881f, -3.4101340770721436f, -3.4074480533599854f,
    -3.404761791229248f, -3.4020755290985107f, -3.3993895053863525f, -3.3967032432556152f,
    -3.394016981124878f, -3.3913309574127197f, -3.3886446952819824f, -3.385958433151245f,
    -3.383272409439087f, -3.3805856704711914f, -3.377899646759033f, -3.375213384628296f,
    -3.3725271224975586f, -3.3698410987854004f, -3.367154836654663f, -3.364468574523926f,
    -3.3617825508117676f, -3.3590962886810303f, -3.356410026550293f, -3.3537235260009766f,
    -3.3510372638702393f, -3.348351240158081f, -3.3456649780273438f, -3.3429787158966064f,
    -3.3402926921844482f, -3.337606430053711f, -3.3349201679229736f, -3.3322343826293945f,
    -3.3295481204986572f, -3.326861619949341f, -3.3241755962371826f, -3.3214893341064453f,
    -3.318803071975708f, -3.31611704826355f, -3.3134307861328125f, -3.310744524002075f,
    -3.308058500289917f, -3.3053722381591797f, -3.3026859760284424f, -3.299999713897705f,
    -3.2973134517669678f, -3.2946274280548096f, -3.2919411659240723f, -3.289254903793335f,
    -3.2865688800811768f, -3.2838826179504395f, -3.281196355819702f, -3.278510332107544f,
    -3.2758240699768066f, -3.2731375694274902f, -3.270451545715332f, -3.2677652835845947f,
    -3.2650790214538574f, -3.262392997741699f, -3.259706735610962f, -3.2570204734802246f,
    -3.2543344497680664f, -3.251648187637329f, -3.248961925506592f, -3.2462756633758545f,
    -3.243589401245117f, -3.240903377532959f, -3.2382171154022217f, -3.2355308532714844f,
    -3.232844829559326f, -3.230158567428589f, -3.2274723052978516f, -3.2247862815856934f,
    -3.222100019454956f, -3.2194135189056396f, -3.2167274951934814f, -3.214041233062744f,
    -3.211354970932007f, -3.2086689472198486f, -3.2059826850891113f, -3.203296422958374f,
    -3.200610399246216f, -3.1979241371154785f, -3.195237874984741f, -3.192551612854004f,
    -3.1898653507232666f, -3.1871793270111084f, -3.184493064880371f, -3.181806802749634f,
    -3.1791207790374756f, -3.1764345169067383f, -3.173748254776001f, -3.1710622310638428f,
    -3.1683759689331055f, -3.165689468383789f, -3.163003444671631f, -3.1603171825408936f,
    -3.1576309204101562f, -3.154944896697998f, -3.1522586345672607f, -3.1495723724365234f,
    -3.1468863487243652f, -3.144200086593628f, -3.1415138244628906f, -3.1388275623321533f,
    -3.136141300201416f, -3.133455276489258f, -3.1307690143585205f, -3.128082752227783f,
    -3.125396728515625f, -3.1227104663848877f, -3.1200242042541504f, -3.117338180541992f,
    -3.114651679992676f, -3.1119654178619385f, -3.1092793941497803f, -3.106593132019043f,
    -3.1039068698883057f, -3.1012208461761475f, -3.09853458404541f, -3.095848321914673f,
    -3.0931622982025146f, -3.0904760360717773f, -3.08778977394104f, -3.0851035118103027f,
    -3.0824172496795654f, -3.0797312259674072f, -3.07704496383667f, -3.0743587017059326f,
    -3.0716726779937744f, -3.068986415863037f, -3.0663001537323f, -3.0636141300201416f,
    -3.060927629470825f, -3.058241367340088f, -3.0555553436279297f, -3.0528690814971924f,
    -3.050182819366455f, -3.047496795654297f, -3.0448105335235596f, -3.0421242713928223f,
    -3.039438247680664f, -3.0367519855499268f, -3.0340657234191895f, -3.031379461288452f,
    -3.028693199157715f, -3.0260071754455566f, -3.0233209133148193f, -3.020634651184082f,
    -3.017948627471924f, -3.0152623653411865f, -3.012576103210449f, -3.009890079498291f,
    -3.0072035789489746f, -3.0045173168182373f, -3.001831293106079f, -2.999145030975342f,
    -2.9964590072631836f, -2.9937727451324463f, -2.991086483001709f, -2.988400459289551f,
    -2.9857141971588135f, -2.983027935028076f, -2.980341672897339f, -2.9776554107666016f,
    -2.9749691486358643f, -2.972283124923706f, -2.9695968627929688f, -2.9669106006622314f,
    -2.9642245769500732f, -2.961538314819336f, -2.9588520526885986f, -2.9561660289764404f,
    -2.953479528427124f, -2.950793504714966f, -2.9481072425842285f, -2.945420980453491f,
    -2.942734956741333f, -2.9400486946105957f, -2.9373624324798584f, -2.9346764087677f,
    -2.931990146636963f, -2.9293038845062256f, -2.9266176223754883f, -2.923931360244751f,
    -2.9212450981140137f, -2.9185590744018555f, -2.915872812271118f, -2.913186550140381f,
    -2.9105005264282227f, -2.9078142642974854f, -2.905128002166748f, -2.90244197845459f,
    -2.8997554779052734f, -2.8970694541931152f, -2.894383192062378f, -2.8916969299316406f,
    -2.8890109062194824f, -2.886324644088745f, -2.883638381958008f, -2.8809523582458496f,
    -2.8782660961151123f, -2.875579833984375f, -2.8728935718536377f, -2.8702073097229004f,
    -2.867521047592163f, -2.864835023880005f, -2.8621487617492676f, -2.8594624996185303f,
    -2.856776475906372f, -2.8540902137756348f, -2.8514039516448975f, -2.8487179279327393f,
    -2.846031427383423f, -2.8433454036712646f, -2.8406591415405273f, -2.83797287940979f,
    -2.835286855697632f, -2.8326005935668945f, -2.8299143314361572f, -2.827228307723999f,
    -2.8245420455932617f, -2.8218555450439453f, -2.819169521331787f, -2.81648325920105f,
    -2.8137969970703125f, -2.8111109733581543f, -2.808424711227417f, -2.8057384490966797f,
    -2.8030524253845215f, -2.800366163253784f, -2.797679901123047f, -2.7949936389923096f,
    -2.7923073768615723f, -2.789621353149414f, -2.786935091018677f, -2.7842488288879395f,
    -2.7815628051757812f, -2.778876543045044f, -2.7761902809143066f, -2.7735042572021484f,
    -2.770817995071411f, -2.7681314945220947f, -2.7654454708099365f, -2.762759208679199f,
    -2.760072946548462f, -2.7573869228363037f, -2.7547006607055664f, -2.752014398574829f,
    -2.749328374862671f, -2.7466421127319336f, -2.7439558506011963f, -2.741269588470459f,
    -2.7385833263397217f, -2.7358973026275635f, -2.733211040496826f, -2.730524778366089f,
    -2.7278387546539307f, -2.7251524925231934f, -2.722466230392456f, -2.719780206680298f,
    -2.7170939445495605f, -2.714407444000244f, -2.711721420288086f, -2.7090351581573486f,
    -2.7063488960266113f, -2.703662872314453f, -2.700976610183716f, -2.6982903480529785f,
    -2.6956043243408203f, -2.692918062210083f, -2.6902318000793457f, -2.6875455379486084f,
    -2.684859275817871f, -2.682173252105713f, -2.6794869899749756f, -2.6768007278442383f,
    -2.67411470413208f, -2.6714284420013428f, -2.6687421798706055f, -2.6660561561584473f,
    -2.66336989402771f, -2.6606833934783936f, -2.6579973697662354f, -2.655311107635498f,
    -2.6526248455047607f, -2.6499388217926025f, -2.6472525596618652f, -2.644566297531128f,
    -2.6418802738189697f, -2.6391940116882324f, -2.636507749557495f, -2.633821487426758f,
    -2.6311352252960205f, -2.6284492015838623f, -2.625762939453125f, -2.6230766773223877f,
    -2.6203906536102295f, -2.617704391479492f, -2.615018129348755f, -2.6123321056365967f,
    -2.6096458435058594f, -2.606959342956543f, -2.6042733192443848f, -2.6015870571136475f,
    -2.59890079498291f, -2.596214771270752f, -2.5935285091400146f, -2.5908422470092773f,
    -2.588156223297119f, -2.585469961166382f, -2.5827836990356445f, -2.5800974369049072f,
    -2.57741117477417f, -2.5747251510620117f, -2.5720388889312744f, -2.569352626800537f,
    -2.566666603088379f, -2.5639803409576416f, -2.5612940788269043f, -2.558608055114746f,
    -2.555921792984009f, -2.5532352924346924f, -2.550549268722534f, -2.547863006591797f,
    -2.5451767444610596f, -2.5424907207489014f, -2.539804458618164f, -2.5371181964874268f,
    -2.5344321727752686f, -2.5317459106445312f, -2.529059648513794f, -2.5263733863830566f,
    -2.5236871242523193f, -2.521001100540161f, -2.518314838409424f, -2.5156285762786865f,
    -2.5129425525665283f, -2.510256290435791f, -2.5075700283050537f, -2.5048840045928955f,
    -2.502197742462158f, -2.499511241912842f, -2.4968252182006836f, -2.4941389560699463f,
    -2.491452693939209f, -2.488766670227051f, -2.4860804080963135f, -2.483394145965576f,
    -2.480708122253418f, -2.4780218601226807f, -2.4753355979919434f, -2.472649335861206f,
    -2.4699630737304688f, -2.4672770500183105f, -2.4645907878875732f, -2.461904525756836f,
    -2.4592185020446777f, -2.4565322399139404f, -2.453845977783203f, -2.451159954071045f,
    -2.4484734535217285f, -2.445787191390991f, -2.443101167678833f, -2.4404149055480957f,
    -2.4377286434173584f, -2.4350426197052f, -2.432356357574463f, -2.4296700954437256f,
    -2.4269840717315674f, -2.42429780960083f, -2.4216115474700928f, -2.4189252853393555f,
    -2.416239023208618f, -2.41355299949646f, -2.4108667373657227f, -2.4081804752349854f,
    -2.405494451522827f, -2.40280818939209f, -2.4001219272613525f, -2.3974359035491943f,
    -2.394749402999878f, -2.3920631408691406f, -2.3893771171569824f, -2.386690855026245f,
    -2.384004592895508f, -2.3813185691833496f, -2.3786323070526123f, -2.375946044921875f,
    -2.373260021209717f, -2.3705737590789795f, -2.367887496948242f, -2.365201234817505f,
    -2.3625149726867676f, -2.3598289489746094f, -2.357142686843872f, -2.3544564247131348f,
    -2.3517704010009766f, -2.3490841388702393f, -2.346397876739502f, -2.3437118530273438f,
    -2.3410253524780273f, -2.33833909034729f, -2.335653066635132f, -2.3329668045043945f,
    -2.3302805423736572f, -2.327594518661499f, -2.3249082565307617f, -2.3222219944000244f,
    -2.319535970687866f, -2.316849708557129f, -2.3141634464263916f, -2.3114771842956543f,
    -2.308790922164917f, -2.306104898452759f, -2.3034186363220215f, -2.300732374191284f,
    -2.298046350479126f, -2.2953600883483887f, -2.2926738262176514f, -2.289987802505493f,
    -2.287301301956177 68f, -2.2846150398254395f, -2.2819290161132812f, -2.279242753982544f,
    -2.2765564918518066f, -2.2738704681396484f, -2.271184206008911f, -2.268497943878174f,
    -2.2658119201660156f, -2.2631256580352783f, -2.260439395904541f, -2.2577531337738037f,
    -2.2550668716430664f, -2.252380847930908f, -2.249694585800171f, -2.2470083236694336f,
    -2.2443222999572754f, -2.241636037826538f, -2.238949775695801f, -2.2362637519836426f,
    -2.233577251434326f, -2.230890989303589f, -2.2282049655914307f, -2.2255187034606934f,
    -2.222832441329956f, -2.220146417617798f, -2.2174601554870605f, -2.2147738933563232f,
    -2.212087869644165f, -2.2094016075134277f, -2.2067153453826904f, -2.204029083251953f,
    -2.201342821121216f, -2.1986567974090576f, -2.1959705352783203f, -2.193284273147583f,
    -2.190598249435425f, -2.1879119873046875f, -2.18522572517395f, -2.182539701461792f,
    -2.1798532009124756f, -2.1771669387817383f, -2.17448091506958f, -2.1717946529388428f,
    -2.1691083908081055f, -2.166422128677368f, -2.163735866546631f, -2.1610498428344727f,
    -2.1583635807037354f, -2.155677318572998f, -2.15299129486084f, -2.1503050327301025f,
    -2.1476187705993652f, -2.144932746887207f, -2.1422464847564697f, -2.1395602226257324f,
    -2.136874198913574f, -2.134187936782837f, -2.1315016746520996f, -2.1288156509399414f,
    -2.126129388809204f, -2.123443126678467f, -2.1207571029663086f, -2.1180708408355713f,
    -2.115384340286255f, -2.1126980781555176f, -2.1100118160247803f, -2.107325792312622f,
    -2.1046395301818848f, -2.1019532680511475f, -2.0992672443389893f, -2.096580982208252f,
    -2.0938947200775146f, -2.0912086963653564f, -2.088522434234619f, -2.085836172103882f,
    -2.0831501483917236f, -2.0804638862609863f, -2.077777624130249f, -2.075091600418091f,
    -2.0724053382873535f, -2.069719076156616f, -2.067033052444458f, -2.0643467903137207f,
    -2.0616602897644043f, -2.058974027633667f, -2.0562877655029297f, -2.0536017417907715f,
    -2.050915479660034f, -2.048229217529297f, -2.0455431938171387f, -2.0428569316864014f,
    -2.040170669555664f, -2.037484645843506f, -2.0347983837127686f, -2.0321121215820312f,
    -2.029426097869873f, -2.0267398357391357f, -2.0240535736083984f, -2.0213675498962402f,
    -2.018681287765503f, -2.0159950256347656f, -2.0133090019226074f, -2.01062273979187f,
    -2.0079362392425537f, -2.0052499771118164f, -2.002563714981079f, -1.9998775720596313f,
    -1.9971914291381836f, -1.9945052862167358f, -1.9918190240859985f, -1.9891328811645508f,
    -1.986446738243103f, -1.9837604761123657f, -1.981074333190918f, -1.9783881902694702f,
    -1.975701928138733f, -1.9730157852172852f, -1.9703296422958374f, -1.9676433801651f,
    -1.9649572372436523f, -1.9622710943222046f, -1.9595848321914673f, -1.9568986892700195f,
    -1.9542120695114136f, -1.9515259265899658f, -1.948839783668518f, -1.9461535215377808f,
    -1.943467378616333f, -1.9407812356948853f, -1.938094973564148f, -1.9354088306427002f,
    -1.9327226877212524f, -1.9300364255905151f, -1.9273502826690674f, -1.9246641397476196f,
    -1.9219778776168823f, -1.9192917346954346f, -1.9166055917739868f, -1.9139193296432495f,
    -1.9112331867218018f, -1.908547043800354f, -1.9058607816696167f, -1.903174638748169f,
    -1.900488018989563f, -1.8978018760681152f, -1.8951157331466675f, -1.8924294710159302f,
    -1.8897433280944824f, -1.8870571851730347f, -1.8843709230422974f, -1.8816847801208496f,
    -1.8789986371994019f, -1.8763123750686646f, -1.8736262321472168f, -1.870940089225769f,
    -1.8682538270950317f, -1.865567684173584f, -1.8628815412521362f, -1.860195279121399f,
    -1.8575091361999512f, -1.8548229932785034f, -1.8521367311477661f, -1.8494505882263184f,
    -1.8467639684677124f, -1.8440778255462646f, -1.841391682624817f, -1.8387054204940796f,
    -1.8360192775726318f, -1.833333134651184f, -1.8306468725204468f, -1.827960729598999f,
    -1.8252745866775513f, -1.822588324546814f, -1.8199021816253662f, -1.8172160387039185f,
    -1.8145297765731812f, -1.8118436336517334f, -1.8091574907302856f, -1.8064712285995483f,
    -1.8037850856781006f, -1.8010989427566528f, -1.7984126806259155f, -1.7957261800765991f,
    -1.7930399179458618f, -1.790353775024414f, -1.7876676321029663f, -1.784981369972229f,
    -1.7822952270507812f, -1.7796090841293335f, -1.7769228219985962f, -1.7742366790771484f,
    -1.7715505361557007f, -1.7688642740249634f, -1.7661781311035156f, -1.7634919881820679f,
    -1.7608057260513306f, -1.7581195831298828f, -1.755433440208435f, -1.7527471780776978f,
    -1.75006103515625f, -1.7473748922348022f, -1.744688630104065f, -1.7420021295547485f,
    -1.7393158674240112f, -1.7366297245025635f, -1.7339435815811157f, -1.7312573194503784f,
    -1.7285711765289307f, -1.725885033607483f, -1.7231987714767456f, -1.7205126285552979f,
    -1.71782648563385f, -1.7151402235031128f, -1.712454080581665f, -1.7097679376602173f,
    -1.70708167552948f, -1.7043955326080322f, -1.7017093896865845f, -1.6990231275558472f,
    -1.6963369846343994f, -1.6936508417129517f, -1.6909645795822144f, -1.688278079032898f,
    -1.6855918169021606f, -1.682905673980713f, -1.6802195310592651f, -1.6775332689285278f,
    -1.67484712600708f, -1.6721609830856323f, -1.669474720954895f, -1.6667885780334473f,
    -1.6641024351119995f, -1.6614161729812622f, -1.6587300300598145f, -1.6560438871383667f,
    -1.6533576250076294f, -1.6506714820861816f, -1.6479853391647339f, -1.6452990770339966f,
    -1.6426129341125488f, -1.639926791191101f, -1.6372405290603638f, -1.6345540285110474f,
    -1.63186776638031f, -1.6291816234588623f, -1.6264954805374146f, -1.6238092184066772f,
    -1.6211230754852295f, -1.6184369325637817f, -1.6157506704330444f, -1.6130645275115967f,
    -1.610378384590149f, -1.6076921224594116f, -1.6050059795379639f, -1.602319836616516lf,
    -1.5996335744857788f, -1.596947431564331f, -1.5942612886428833f, -1.591575026512146f,
    -1.5888888835906982f, -1.5862027406692505f, -1.5835164785385132f, -1.5808299779891968f,
    -1.5781437158584595f, -1.5754575729370117f, -1.572771430015564f, -1.5700851678848267f,
    -1.567399024963379f, -1.5647128820419312f, -1.5620266199111938f, -1.559340476989746f,
    -1.5566543340682983f, -1.553968071937561f, -1.5512819290161133f, -1.5485957860946655f,
    -1.5459095239639282f, -1.5432233810424805f, -1.5405372381210327f, -1.5378509759902954f,
    -1.5351648330688477f, -1.5324786901474f, -1.5297924280166626f, -1.5271059274673462f,
    -1.5244196653366089f, -1.5217335224151611f, -1.5190473794937134f, -1.516361117362976f,
    -1.5136749744415283f, -1.5109888315200806f, -1.5083025693893433f, -1.5056164264678955f,
    -1.5029302835464478f, -1.5002440214157104f, -1.4975578784942627f, -1.494871735572815f,
    -1.4921855926513672f, -1.4894993305206299f, -1.4868131875991821f, -1.4841270446777344f,
    -1.481440782546997f, -1.4787546396255493f, -1.4760680198669434f, -1.4733818769454956f,
    -1.4706957340240479f, -1.4680094718933105f, -1.4653233289718628f, -1.462637186050415f,
    -1.4599509239196777f, -1.45726478099823f, -1.4545786380767822f, -1.451892375946045f,
    -1.4492062330245972f, -1.4465200901031494f, -1.443833827972412f, -1.4411476850509644f,
    -1.4384615421295166f, -1.4357752799987793f, -1.4330891370773315f, -1.4304029941558838f,
    -1.4277167320251465f, -1.4250305891036987f, -1.4223439693450928f, -1.419657826423645f,
    -1.4169716835021973f, -1.41428542137146f, -1.4115992784500122f, -1.4089131355285645f,
    -1.4062268733978271f, -1.4035407304763794f, -1.4008545875549316f, -1.3981683254241943f,
    -1.3954821825027466f, -1.3927960395812988f, -1.3901097774505615f, -1.3874236345291138f,
    -1.384737491607666f, -1.3820512294769287f, -1.379365086555481f, -1.3766789436340332f,
    -1.373992681503296f, -1.3713065385818481f, -1.3686199188232422f, -1.3659337759017944f,
    -1.3632476329803467f, -1.3605613708496094f, -1.3578752279281616f, -1.3551890850067139f,
    -1.3525028228759766f, -1.3498166799545288f, -1.347130537033081f, -1.3444442749023438f,
    -1.341758131980896f, -1.3390719890594482f, -1.336385726928711f, -1.3336995840072632f,
    -1.3310134410858154f, -1.3283271789550781f, -1.3256410360336304f, -1.3229548931121826f,
    -1.3202686309814453f, -1.3175824880599976f, -1.3148958683013916f, -1.3122097253799438f,
    -1.3095235824584961f, -1.3068373203277588f, -1.304151177406311f, -1.3014650344848633f,
    -1.298778772354126f, -1.2960926294326782f, -1.2934064865112305f, -1.2907202243804932f,
    -1.2880340814590454f, -1.2853479385375977f, -1.2826616764068604f, -1.2799755334854126f,
    -1.2772893905639648f, -1.2746031284332275f, -1.2719169855117798f, -1.269230842590332f,
    -1.2665445804595947f, -1.263858437538147f, -1.261171817779541f, -1.2584856748580933f,
    -1.2557995319366455f, -1.2531132698059082f, -1.2504271268844604f, -1.2477409839630127f,
    -1.2450547218322754f, -1.2423685789108276f, -1.2396824359893799f, -1.2369961738586426f,
    -1.2343100309371948f, -1.231623888015747f, -1.2289376258850098f, -1.226251482963562f,
    -1.2235653400421143f, -1.220879077911377f, -1.2181929349899292f, -1.2155067920684814f,
    -1.2128205299377441f, -1.2101343870162964f, -1.2074477672576904f, -1.2047616243362427f,
    -1.202075481414795f, -1.1993892192840576f, -1.1967030763626099f, -1.194016933441162f,
    -1.1913306713104248f, -1.188644528388977f, -1.1859583854675293f, -1.183272123336792f,
    -1.1805859804153442f, -1.1778998374938965f, -1.1752135753631592f, -1.1725274324417114f,
    -1.1698412895202637f, -1.1671550273895264f, -1.1644688844680786f, -1.1617827415466309f,
    -1.1590964794158936f, -1.1564099788665771f, -1.1537237167358398f, -1.151037573814392f,
    -1.1483514308929443f, -1.145665168762207f, -1.1429790258407593f, -1.1402928829193115f,
    -1.1376066207885742f, -1.1349204778671265f, -1.1322343349456787f, -1.1295480728149414f,
    -1.1268619298934937f, -1.124175786972046f, -1.1214895248413086f, -1.1188033819198608f,
    -1.116117238998413f, -1.1134309768676758f, -1.110744833946228f, -1.1080586910247803f,
    -1.105372428894043f, -1.1026859283447266f, -1.0999996662139893f, -1.0973135232925415f,
    -1.0946273803710938f, -1.0919411182403564f, -1.0892549753189087f, -1.086568832397461f,
    -1.0838825702667236f, -1.0811964273452759f, -1.0785102844238281f, -1.0758240222930908f,
    -1.073137879371643f, -1.0704517364501953f, -1.067765474319458f, -1.0650793313980103f,
    -1.0623931884765625f, -1.0597069263458252f, -1.0570207834243774f, -1.0543346405029297f,
    -1.0516483783721924f, -1.048961877822876f, -1.0462756156921387f, -1.043589472770691f,
    -1.0409033298492432f, -1.0382170677185059f, -1.035530924797058f, -1.0328447818756104f,
    -1.030158519744873f, -1.0274723768234253f, -1.0247862339019775f, -1.0220999717712402f,
    -1.0194138288497925f, -1.0167276859283447f, -1.0140414237976074f, -1.0113552808761597f,
    -1.008669137954712f, -1.0059828758239746f, -1.0032967329025269f, -1.000610589981079f,
    -0.9979243874549866f, -0.9952377676963806f, -0.9925516247749329f, -0.9898654222488403f,
    -0.9871792197227478f, -0.9844930768013f, -0.9818068742752075f, -0.979120671749115f,
    -0.9764345288276672f, -0.9737483263015747f, -0.9710621237754822f, -0.9683759808540344f,
    -0.9656897783279419f, -0.9630035758018494f, -0.9603174328804016f, -0.9576312303543091f,
    -0.9549450278282166f, -0.9522588849067688f, -0.9495726823806763f, -0.9468864798545837f,
    -0.944200336933136f, -0.94151371717453f, -0.9388275742530823f, -0.9361413717269897f,
    -0.9334551692008972f, -0.9307690262794495f, -0.9280828237533569f, -0.9253966212272644f,
    -0.9227104783058167f, -0.9200242757797241f, -0.9173380732536316f, -0.9146519303321838f,
    -0.9119657278060913f, -0.9092795252799988f, -0.906593382358551f, -0.9039071798324585f,
    -0.901220977306366f, -0.8985348343849182f, -0.8958486318588257f, -0.8931624293327332f,
    -0.8904762864112854f, -0.8877896666526794f, -0.8851035237312317f, -0.8824173212051392f,
    -0.8797311186790466f, -0.8770449757575989f, -0.8743587732315063f, -0.8716725707054138f,
    -0.8689864277839661f, -0.8663002252578735f, -0.863614022731781f, -0.8609278798103333f,
    -0.8582416772842407f, -0.8555554747581482f, -0.8528693318367004f, -0.8501831293106079f,
    -0.8474969267845154f, -0.8448107838630676f, -0.8421245813369751f, -0.8394383788108826f,
    -0.8367522358894348f, -0.8340656161308289f, -0.8313794732093811f, -0.8286932706832886f,
    -0.826007068157196f, -0.8233209252357483f, -0.8206347227096558f, -0.8179485201835632f,
    -0.8152623772621155f, -0.812576174736023f, -0.8098899722099304f, -0.8072038292884827f,
    -0.8045176267623901f, -0.8018314242362976f, -0.7991452813148499f, -0.7964590787887573f,
    -0.7937728762626648f, -0.791086733341217f, -0.7884005308151245f, -0.785714328289032f,
    -0.7830277681350708f, -0.7803415656089783f, -0.7776554226875305f, -0.774969220161438f,
    -0.7722830176353455f, -0.7695968747138977f, -0.7669106721878052f, -0.7642244696617126f,
    -0.7615383267402649f, -0.7588521242141724f, -0.7561659216880798f, -0.7534797787666321f,
    -0.7507935762405396f, -0.7481074333190918f, -0.7454212307929993f, -0.7427350282669067f,
    -0.740048885345459f, -0.7373626828193665f, -0.7346764802932739f, -0.7319903373718262f,
    -0.7293037176132202f, -0.7266175746917725f, -0.7239313721656799f, -0.7212451696395874f,
    -0.7185590267181396f, -0.7158728241920471f, -0.7131866216659546f, -0.7105004787445068f,
    -0.7078142762184143f, -0.7051280736923218f, -0.702441930770874f, -0.6997557282447815f,
    -0.697069525718689f, -0.6943833827972412f, -0.6916971802711487f, -0.6890109777450562f,
    -0.6863248348236084f, -0.6836386322975159f, -0.6809524297714233f, -0.6782662868499756f,
    -0.6755796670913696f, -0.6728935241699219f, -0.6702073216438293f, -0.6675211191177368f,
    -0.6648349761962891f, -0.6621487736701965f, -0.659462571144104f, -0.6567764282226562f,
    -0.6540902256965637f, -0.6514040231704712f, -0.6487178802490234f, -0.6460316777229309f,
    -0.6433454751968384f, -0.6406593322753906f, -0.6379731297492981f, -0.6352869272232056f,
    -0.6326007843017578f, -0.6299145817756653f, -0.6272283792495728f, -0.624542236328125f,
    -0.621855616569519f, -0.6191694736480713f, -0.6164832711219788f, -0.6137970685958862f,
    -0.6111109256744385f, -0.608424723148346f, -0.6057385206222534f, -0.6030523777008057f,
    -0.6003661751747131f, -0.5976799726486206f, -0.5949938297271729f, -0.5923076272010803f,
    -0.5896214246749878f, -0.58693528175354f, -0.5842490792274475f, -0.581562876701355f,
    -0.5788767337799072f, -0.5761905312538147f, -0.5735043287277222f, -0.5708181858062744f,
    -0.5681315660476685f, -0.5654454231262207f, -0.5627592206001282f, -0.5600730180740356f,
    -0.5573868751525879f, -0.5547006726264954f, -0.5520144701004028f, -0.5493283271789551f,
    -0.5466421246528625f, -0.54395592212677f, -0.5412697792053223f, -0.5385835766792297f,
    -0.5358973741531372f, -0.5332112312316895f, -0.5305250287055969f, -0.5278388261795044f,
    -0.5251526832580566f, -0.5224664807319641f, -0.5197802782058716f, -0.5170941352844238f,
    -0.5144075155258179f, -0.5117213726043701f, -0.5090351700782776f, -0.5063489675521851f,
    -0.5036628246307373f, -0.5009766221046448f, -0.49829044938087463f, -0.4956042468547821f,
    -0.49291807413101196f, -0.4902319014072418f, -0.4875456988811493f, -0.48485952615737915f,
    -0.482173353433609f, -0.4794871509075165f, -0.47680097818374634f, -0.4741148054599762f,
    -0.47142860293388367f, -0.4687424302101135f, -0.4660562574863434f, -0.4633696675300598f,
    -0.4606834948062897f, -0.45799729228019714f, -0.455311119556427f, -0.45262494683265686f,
    -0.44993874430656433f, -0.4472525715827942f, -0.44456639885902405f, -0.4418801963329315f,
    -0.4391940236091614f, -0.43650785088539124f, -0.4338216483592987f, -0.43113547563552856f,
    -0.4284493029117584f, -0.4257631003856659f, -0.42307692766189575f, -0.4203907549381256f,
    -0.417704552412033lf, -0.41501837968826294f, -0.412332206964492 8f, -0.40964561700820923f,
    -0.4069594442844391f, -0.40427324175834656f, -0.4015870690345764f, -0.3989008963108063f,
    -0.39621469378471375f, -0.3935285210609436f, -0.39084234833717346f, -0.38815614581108093f,
    -0.3854699730873108f, -0.38278380036354065f, -0.3800975978374481f, -0.377411425113678f,
    -0.37472525238990784f, -0.3720390796661377f, -0.36935287714004517f, -0.366666704416275f,
    -0.3639805316925049f, -0.36129432916641235f, -0.3586081564426422f, -0.35592156648635864f,
    -0.3532353937625885f, -0.35054922103881836f, -0.34786301851272583f, -0.3451768457889557f,
    -0.34249067306518555f, -0.339804470539093f, -0.3371182978153229f, -0.33443212509155273f,
    -0.3317459225654602f, -0.32905974984169006f, -0.3263735771179199f, -0.3236873745918274f,
    -0.32100120186805725f, -0.3183150291442871f, -0.3156288266181946f, -0.31294265389442444f,
    -0.3102564811706543f, -0.30757027864456177f, -0.3048841059207916f, -0.30219751596450806f,
    -0.2995113432407379f, -0.2968251705169678f, -0.29413896799087524f, -0.2914527952671051f,
    -0.28876662254333496f, -0.28608042001724243f, -0.2833942472934723f, -0.28070807456970215f,
    -0.2780218720436096f, -0.2753356993198395f, -0.27264952659606934f, -0.2699633240699768f,
    -0.26727715134620667f, -0.2645909786224365f, -0.261904776096344f, -0.25921860337257385f,
    -0.2565324306488037f, -0.2538462281227112f, -0.25116005539894104f, -0.24847348034381866f,
    -0.24578729271888733f, -0.243101105093956f, -0.24041493237018585f, -0.23772874474525452f,
    -0.23504255712032318f, -0.23235638439655304f, -0.2296701967716217f, -0.22698400914669037f,
    -0.22429783642292023f, -0.2216116487979889f, -0.21892546117305756f, -0.21623928844928741f,
    -0.21355310082435608f, -0.21086691319942474f, -0.2081807404756546f, -0.20549455285072327f,
    -0.20280836522579193f, -0.2001221925020218f, -0.19743600487709045f, -0.19474942982196808f,
    -0.19206324219703674f, -0.1893770545721054f, -0.18669088184833527f, -0.18400469422340393f,
    -0.1813185214996338f, -0.17863233387470245f, -0.17594614624977112f, -0.17325997352600098f,
    -0.17057378590106964f, -0.1678875982761383f, -0.16520142555236816f, -0.16251523792743683f,
    -0.1598290503025055f, -0.15714287757873535f, -0.15445668995380402f, -0.15177050232887268f,
    -0.14908432960510254f, -0.1463981419801712f, -0.14371156692504883f, -0.1410253793001175f,
    -0.13833919167518616f, -0.13565301895141602f, -0.13296683132648468f, -0.13028064370155334f,
    -0.1275944709777832f, -0.12490828335285187f, -0.12222210317850113f, -0.1195359155535698f,
    -0.11684973537921906f, -0.11416355520486832f, -0.11147736757993698f, -0.10879118740558624f,
    -0.1061050072312355f, -0.10341881960630417f, -0.10073263943195343f, -0.09804645925760269f,
    -0.09536027163267136f, -0.09267409145832062f, -0.08998751640319824f, -0.0873013287782669f,
    -0.08461514860391617f, -0.08192896842956543f, -0.0792427808046341f, -0.07655660063028336f,
    -0.07387042045593262f, -0.07118423283100128f, -0.06849805265665054f, -0.0658118724822998f,
    -0.06312568485736847f, -0.06043950468301773f, -0.057753320783376694f, -0.055067140609025955f,
    -0.05238095670938492f, -0.04969477280974388f, -0.04700859263539314f, -0.044322408735752106f,
    -0.04163622856140137f, -0.03895004466176033f, -0.036263465881347656f, -0.03357728198170662f,
    -0.03089109994471073f, -0.028204916045069695f, -0.025518734008073807f, -0.02283255197107792f,
    -0.02014636993408203f, -0.017460186034440994f, -0.014774003997445107f, -0.012087821029126644f,
    -0.009401638992130756f, -0.006715456489473581f, -0.004029273986816406f, -0.0013430912513285875f,
    0.0013430912513285875f, 0.004029273986816406f, 0.006715456489473581f, 0.009401638992130756f,
    0.012087821029126644f, 0.014774003997445107f, 0.01746058464050293f, 0.020146766677498817f,
    0.022832948714494705f, 0.025519130751490593f, 0.02820531465113163f, 0.030891496688127518f,
    0.033577680587768555f, 0.03626386076211929f, 0.03895004466176033f, 0.04163622856140137f,
    0.044322408735752106f, 0.04700859263539314f, 0.04969477280974388f, 0.05238095670938492f,
    0.055067140609025955f, 0.057753320783376694f, 0.06043950468301773f, 0.06312568485736847f,
    0.0658118724822998f, 0.06849805265665054f, 0.07118463516235352f, 0.07387081533670425f,
    0.07655699551105499f, 0.07924318313598633f, 0.08192936331033707f, 0.0846155434846878f,
    0.08730173110961914f, 0.08998791128396988f, 0.09267409145832062f, 0.09536027163267136f,
    0.09804645925760269f, 0.10073263943195343f, 0.10341881960630417f, 0.1061050072312355f,
    0.10879118740558624f, 0.11147736757993698f, 0.11416355520486832f, 0.11684973537921906f,
    0.1195359155535698f, 0.12222210317850113f, 0.1249086782336235f, 0.12759485840797424f,
    0.13028104603290558f, 0.13296723365783691f, 0.13565340638160706f, 0.1383395940065384f,
    0.14102578163146973f, 0.14371195435523987f, 0.1463981419801712f, 0.14908432960510254f,
    0.15177050232887268f, 0.15445668995380402f, 0.15714287757873535f, 0.1598290503025055f,
    0.16251523792743683f, 0.16520142555236816f, 0.1678875982761383f, 0.17057378590106964f,
    0.17325997352600098f, 0.17594654858112335f, 0.1786327362060547f, 0.18131890892982483f,
    0.18400509655475616f, 0.1866912841796875f, 0.18937745690345764f, 0.19206364452838898f,
    0.19474981725215912f, 0.19743600487709045f, 0.2001221925020218f, 0.20280836522579193f,
    0.20549455285072327f, 0.2081807404756546f, 0.21086691319942474f, 0.21355310082435608f,
    0.21623928844928741f, 0.21892546117305756f, 0.2216116487979889f, 0.22429783642292023f,
    0.22698400914669037f, 0.22967059910297394f, 0.23235677182674408f, 0.23504295945167542f,
    0.23772914707660675f, 0.2404153198003769f, 0.24310150742530823f, 0.24578769505023956f,
    0.2484738677740097f, 0.25116005539894104f, 0.2538462281227112f, 0.2565324306488037f,
    0.25921860337257385f, 0.261904776096344f, 0.2645909786224365f, 0.26727715134620667f,
    0.2699633240699768f, 0.27264952659606934f, 0.2753356993198395f, 0.2780218720436096f,
    0.28070807456970215f, 0.2833946347236633f, 0.28608083724975586f, 0.288767009973526f,
    0.29145318269729614f, 0.29413938522338867f, 0.2968255579471588f, 0.29951173067092896f,
    0.3021979331970215f, 0.3048841059207916f, 0.30757027864456177f, 0.3102564811706543f,
    0.31294265389442444f, 0.3156288266181946f, 0.3183150291442871f, 0.32100120186805725f,
    0.3236873745918274f, 0.3263735771179199f, 0.32905974984169006f, 0.3317459225654602f,
    0.33443212509155273f, 0.3371186852455139f, 0.33980488777160645f, 0.3424910604953766f,
    0.34517723321914673f, 0.34786343574523926f, 0.3505496084690094f, 0.35323578119277954f,
    0.35592198371887207f, 0.3586081564426422f, 0.36129432916641235f, 0.3639805316925049f,
    0.366666704416275f, 0.36935287714004517f, 0.3720390796661377f, 0.37472525238990784f,
    0.377411425113678f, 0.3800975978374481f, 0.38278380036354065f, 0.3854699730873108f,
    0.38815614581108093f, 0.3908427357673645f, 0.39352890849113464f, 0.3962151110172272f,
    0.3989012837409973f, 0.40158745646476746f, 0.40427365899086f, 0.4069598317146301f,
    0.40964600443840027f, 0.4123322069644928f, 0.41501837968826294f, 0.4177045524120331f,
    0.4203907549381256f, 0.42307692766189575f, 0.4257631003856659f, 0.4284493029117584f,
    0.43113547563552856f, 0.4338216483592987f, 0.43650785088539124f, 0.4391940236091614f,
    0.4418801963329315f, 0.4445667862892151f, 0.44725295901298523f, 0.44993916153907776f,
    0.4526253342628479f, 0.45531150698661804f, 0.45799770951271057f, 0.4606838822364807f,
    0.46337005496025085f, 0.4660562574863434f, 0.4687424302101135f, 0.47142860293388367f,
    0.4741148054599762f, 0.47680097818374634f, 0.4794871509075165f, 0.482173353433609f,
    0.48485952615737915f, 0.4875456988811493f, 0.4902319014072418f, 0.49291807413101196f,
    0.4956042468547821f, 0.4982908368110657f, 0.5009770393371582f, 0.503663182258606f,
    0.5063493847846985f, 0.509035587310791f, 0.5117217302322388f, 0.5144079327583313f,
    0.5170941352844238f, 0.5197802782058716f, 0.5224664807319641f, 0.5251526832580566f,
    0.5278388261795044f, 0.5305250287055969f, 0.5332112312316895f, 0.5358973741531372f,
    0.5385835766792297f, 0.5412697792053223f, 0.54395592212677f, 0.5466421246528625f,
    0.5493286848068237f, 0.5520148873329163f, 0.5547010898590088f, 0.5573872327804565f,
    0.5600734353065491f, 0.5627596378326416f, 0.5654457807540894f, 0.5681319832801819f,
    0.5708181858062744f, 0.5735043287277222f, 0.5761905312538147f, 0.5788767337799072f,
    0.581562876701355f, 0.5842490792274475f, 0.58693528175354f, 0.5896214246749878f,
    0.5923076272010803f, 0.5949938297271729f, 0.5976799726486206f, 0.6003661751747131f,
    0.6030527353286743f, 0.6057389378547668f, 0.6084251403808594f, 0.6111112833023071f,
    0.6137974858283997f, 0.6164836883544922f, 0.6191698312759399f, 0.6218560338020325f,
    0.624542236328125f, 0.6272283792495728f, 0.6299145817756653f, 0.6326007843017578f,
    0.6352869272232056f, 0.6379731297492981f, 0.6406593322753906f, 0.6433454751968384f,
    0.6460316777229309f, 0.6487178802490234f, 0.6514040231704712f, 0.6540902256965637f,
    0.6567767858505249f, 0.6594629883766174f, 0.66214919090271f, 0.6648353338241577f,
    0.6675215363502502f, 0.6702077388763428f, 0.6728938817977905f, 0.6755800843238831f,
    0.6782662868499756f, 0.6809524297714233f, 0.6836386322975159f, 0.6863248348236084f,
    0.6890109777450562f, 0.6916971802711487f, 0.6943833827972412f, 0.697069525718689f,
    0.6997557282447815f, 0.702441930770874f, 0.7051280736923218f, 0.7078142762184143f,
    0.7105008363723755f, 0.713187038898468f, 0.7158732414245605f, 0.7185593843460083f,
    0.7212455868721008f, 0.7239317893981934f, 0.7266179323196411f, 0.7293041348457336f,
    0.7319903373718262f, 0.7346764802932739f, 0.7373626828193665f, 0.740048885345459f,
    0.7427350282669067f, 0.7454212307929993f, 0.7481074333190918f, 0.7507935762405396f,
    0.7534797787666321f, 0.7561659216880798f, 0.7588521242141724f, 0.7615383267402649f,
    0.7642248868942261f, 0.7669110894203186f, 0.7695972323417664f, 0.7722834348678589f,
    0.7749696373939514f, 0.7776557803153992f, 0.7803419828414917f, 0.7830281853675842f,
    0.785714328289032f, 0.7884005308151245f, 0.791086733341217f, 0.7937728762626648f,
    0.7964590787887573f, 0.7991452813148499f, 0.8018314242362976f, 0.8045176267623901f,
    0.8072038292884827f, 0.8098899722099304f, 0.812576174736023f, 0.8152623772621155f,
    0.8179489374160767f, 0.8206351399421692f, 0.8233212828636169f, 0.8260074853897095f,
    0.828693687915802f, 0.8313798308372498f, 0.8340660333633423f, 0.8367522358894348f,
    0.8394383788108826f, 0.8421245813369751f, 0.8448107838630676f, 0.8474969267845154f,
    0.8501831293106079f, 0.8528693318367004f, 0.8555554747581482f, 0.8582416772842407f,
    0.8609278798103333f, 0.863614022731781f, 0.8663002252578735f, 0.8689867854118347f,
    0.8716729879379272f, 0.8743591904640198f, 0.8770453333854675f, 0.8797315359115601f,
    0.8824177384376526f, 0.8851038813591003f, 0.8877900838851929f, 0.8904762864112854f,
    0.8931624293327332f, 0.8958486318588257f, 0.8985348343849182f, 0.901220977306366f,
    0.9039071798324585f, 0.906593382358551f, 0.9092795252799988f, 0.9119657278060913f,
    0.9146519303321838f, 0.9173380732536316f, 0.9200242757797241f, 0.9227108359336853f,
    0.9253970384597778f, 0.9280832409858704f, 0.9307693839073181f, 0.9334555864334106f,
    0.9361417889595032f, 0.9388279318809509f, 0.9415141344070435f, 0.944200336933136f,
    0.9468864798545837f, 0.9495726823806763f, 0.9522588849067688f, 0.9549450278282166f,
    0.9576312303543091f, 0.9603174328804016f, 0.9630035758018494f, 0.9656897783279419f,
    0.9683759808540344f, 0.9710621237754822f, 0.9737483263015747f, 0.9764348864555359f,
    0.9791210889816284f, 0.981807291507721f, 0.9844934344291687f, 0.9871796369552612f,
    0.9898658394813538f, 0.9925519824028015f, 0.995238184928894f, 0.9979243874549866f,
    1.000610589981079f, 1.0032967329025269f, 1.0059828758239746f, 1.008669137954712f,
    1.0113552808761597f, 1.0140414237976074f, 1.0167276859283447f, 1.0194138288497925f,
    1.0220999717712402f, 1.0247862339019775f, 1.0274723768234253f, 1.030158996582031 2f,
    1.032845139503479f, 1.0355312824249268f, 1.038217544555664f, 1.0409036874771118f,
    1.0435898303985596f, 1.0462760925292969f, 1.0489622354507446f, 1.0516483783721924f,
    1.0543346405029297f, 1.0570207834243774f, 1.0597069263458252f, 1.0623931884765625f,
    1.0650793313980103f, 1.067765474319458f, 1.0704517364501953f, 1.073137879371643f,
    1.0758240222930908f, 1.0785102844238281f, 1.0811964273452759f, 1.0838830471038818f,
    1.0865691900253296f, 1.0892553329467773f, 1.0919415950775146f, 1.0946277379989624f,
    1.0973138809204102f, 1.1000001430511475f, 1.1026862859725952f, 1.105372428894043f,
    1.1080586910247803f, 1.110744833946228f, 1.1134309768676758f, 1.116117238998413f,
    1.1188033819198608f, 1.1214895248413086f, 1.124175786972046f, 1.1268619298934937f,
    1.1295480728149414f, 1.1322343349456787f, 1.1349204778671265f, 1.1376070976257324f,
    1.1402932405471802f, 1.142979383468628f, 1.1456656455993652f, 1.148351788520813f,
    1.1510379314422607f, 1.153724193572998f, 1.1564103364944458f, 1.1590964794158936f,
    1.1617827415466309f, 1.1644688844680786f, 1.1671555042266846f, 1.1698416471481323f,
    1.17252779006958f, 1.1752140522003174f, 1.1779001951217651f, 1.180586338043213f,
    1.1832726001739502f, 1.185958743095398f, 1.1886448860168457f, 1.191331148147583f,
    1.1940172910690308f, 1.1967034339904785f, 1.1993896961212158f, 1.2020758390426636f,
    1.2047619819641113f, 1.2074482440948486f, 1.2101343870162964f, 1.2128205299377441f,
    1.2155067920684814f, 1.2181929349899292f, 1.220879077911377f, 1.2235653400421143f,
    1.226251482963562f, 1.2289376258850098f, 1.231623888015747f, 1.2343100309371948f,
    1.2369961738586426f, 1.2396824359893799f, 1.2423685789108276f, 1.2450547218322754f,
    1.2477409839630127f, 1.2504271268844604f, 1.2531132698059082f, 1.2557995319366455f,
    1.2584856748580933f, 1.261171817779541f, 1.2638580799102783f, 1.266544222831726f,
    1.2692312002182007f, 1.2719173431396484f, 1.2746036052703857f, 1.2772897481918335f,
    1.2799758911132812f, 1.2826621532440186f, 1.2853482961654663f, 1.288034439086914f,
    1.2907207012176514f, 1.2934068441390991f, 1.2960929870605469f, 1.2987792491912842f,
    1.301465392112732f, 1.3041515350341797f, 1.306837797164917f, 1.3095239400863647f,
    1.3122100830078125f, 1.3148963451385498f, 1.3175824880599976f, 1.3202686309814453f,
    1.3229548931121826f, 1.3256410360336304f, 1.3283271789550781f, 1.3310134410858154f,
    1.3336995840072632f, 1.336385726928711f, 1.3390719890594482f, 1.341758131980896f,
    1.3444442749023438f, 1.347130537033081f, 1.3498166799545288f, 1.3525028228759766f,
    1.3551890850067139f, 1.3578752279281616f, 1.3605613708496094f, 1.3632476329803467f,
    1.3659337759017944f, 1.3686199188232422f, 1.3713061809539795f, 1.3739923238754272f,
    1.3766793012619019f, 1.3793654441833496f, 1.382051706314087f, 1.3847378492355347f,
    1.3874239921569824f, 1.3901102542877197f, 1.3927963972091675f, 1.3954825401306152f,
    1.3981688022613525f, 1.4008549451828003f, 1.403541088104248f, 1.4062273502349854f,
    1.408913493156433f, 1.4115996360778809f, 1.4142858982086182f, 1.416972041130066f,
    1.4196581840515137f, 1.422344446182251f, 1.4250305891036987f, 1.4277167320251465f,
    1.4304029941558838f, 1.4330891370773315f, 1.4357752799987793f, 1.4384615421295166f,
    1.4411476850509644f, 1.443833827972412f, 1.4465200901031494f, 1.4492062330245972f,
    1.451892375946045f, 1.4545786380767822f, 1.45726478099823f, 1.4599509239196777f,
    1.462637186050415f, 1.4653233289718628f, 1.4680094718933105f, 1.4706957340240479f,
    1.4733818769454956f, 1.4760680198669434f, 1.4787542819976807f, 1.4814404249191284f,
    1.484127402305603f, 1.4868135452270508f, 1.489499807357788f, 1.4921859502792358f,
    1.4948720932006836f, 1.497558355331421f, 1.5002444982528687f, 1.5029306411743164f,
    1.5056167840957642f, 1.5083030462265015f, 1.5109891891479492f, 1.513675332069397f,
    1.5163615942001343f, 1.519047737121582f, 1.5217338800430298f, 1.524420142173767f,
    1.5271062850952148f, 1.5297924280166626f, 1.5324786901474f, 1.5351648330688477f,
    1.5378509759902954f, 1.5405372381210327f, 1.5432233810424805f, 1.5459095239639282f,
    1.5485957860946655f, 1.5512819290161133f, 1.553968071937561f, 1.5566543340682983f,
    1.559340476989746f, 1.5620266199111938f, 1.5647128820419312f, 1.567399024963379f,
    1.5700851678848267f, 1.572771430015564f, 1.5754575729370117f, 1.5781437158584595f,
    1.5808299779891968f, 1.5835161209106445f, 1.5862022638320923f, 1.588889241218567f,
    1.5915755033493042f, 1.594261646270752f, 1.5969477891921997f, 1.599634051322937f,
    1.6023201942443848f, 1.6050063371658325f, 1.6076925992965698f, 1.6103787422180176f,
    1.6130648851394653f, 1.6157511472702026f, 1.6184372901916504f, 1.6211234331130981f,
    1.6238096952438354f, 1.6264958381652832f, 1.629181981086731f, 1.6318682432174683f,
    1.634554386138916f, 1.6372405290603638f, 1.639926791191101f, 1.6426129341125488f,
    1.6452990770339966f, 1.6479853391647339f, 1.6506714820861816f, 1.6533576250076294f,
    1.6560438871383667f, 1.6587300300598145f, 1.6614161729812622f, 1.6641024351119995f,
    1.6667885780334473f, 1.669474720954895f, 1.6721609830856323f, 1.67484712600708f,
    1.6775332689285278f, 1.6802195310592651f, 1.682905673980713f, 1.6855918169021606f,
    1.688278079032898f, 1.6909642219543457f, 1.6936503648757935f, 1.696337342262268f,
    1.6990236043930054f, 1.7017097473144531f, 1.7043958902359009f, 1.7070821523666382f,
    1.709768295288086f, 1.7124544382095337f, 1.715140700340271f, 1.7178268432617188f,
    1.7205129861831665f, 1.7231992483139038f, 1.7258853912353516f, 1.7285715341567993f,
    1.7312577962875366f, 1.7339439392089844f, 1.7366300821304321f, 1.7393163442611694f,
    1.7420024871826172f, 1.744688630104065f, 1.7473748922348022f, 1.75006103515625f,
    1.7527471780776978f, 1.755433440208435f, 1.7581195831298828f, 1.7608057260513306f,
    1.7634919881820679f, 1.7661781311035156f, 1.7688642740249634f, 1.7715505361557007f,
    1.7742366790771484f, 1.7769228219985962f, 1.7796090841293335f, 1.7822952270507812f,
    1.784981369972229f, 1.7876676321029663f, 1.790353775024414f, 1.7930399179458618f,
    1.7957261800765991f, 1.7984123229980469f, 1.8010984659194946f, 1.8037854433059692f,
    1.8064717054367065f, 1.8091578483581543f, 1.811843991279602f, 1.8145302534103394f,
    1.817216396331787f, 1.8199025392532349f, 1.8225888013839722f, 1.82527494430542f,
    1.8279610872268677f, 1.830647349357605f, 1.8333334922790527f, 1.8360196352005005f,
    1.8387058973312378f, 1.8413920402526855f, 1.8440781831741333f, 1.8467644453048706f,
    1.8494505882263184f, 1.8521367311477661f, 1.8548229932785034f, 1.8575091361999512f,
    1.860195279121399f, 1.8628815412521362f, 1.865567684173584f, 1.8682538270950317f,
    1.870940089225769f, 1.8736262321472168f, 1.8763123750686646f, 1.8789986371994019f,
    1.8816847801208496f, 1.8843709230422974f, 1.8870571851730347f, 1.8897433280944824f,
    1.8924294710159302f, 1.8951157331466675f, 1.8978018760681152f, 1.900488018989563f,
    1.9031742811203003f, 1.905860424041748f, 1.9085474014282227f, 1.9112335443496704f,
    1.9139198064804077f, 1.9166059494018555f, 1.9192920923233032f, 1.9219783544540405f,
    1.9246644973754883f, 1.927350640296936f, 1.9300369024276733f, 1.932723045349121f,
    1.9354091882705688f, 1.9380954504013062f, 1.940781593322754f, 1.9434677362442017f,
    1.946153998374939f, 1.9488401412963867f, 1.9515262842178345f, 1.9542125463485718f,
    1.9568986892700195f, 1.9595848321914673f, 1.9622710943222046f, 1.9649572372436523f,
    1.9676433801651f, 1.9703296422958374f, 1.9730157852172852f, 1.975701928138733f,
    1.9783881902694702f, 1.981074333190918f, 1.9837604761123657f, 1.986446738243103f,
    1.9891328811645508f, 1.9918190240859985f, 1.9945052862167358f, 1.9971914291381836f,
    1.9998775720596313f, 2.002563714981079f, 2.0052499771118164f, 2.0079362392425537f,
    2.010622262954712f, 2.013308525085449f, 2.015995502471924f, 2.018681764602661f,
    2.0213677883148193f, 2.0240540504455566f, 2.026740312576294f, 2.029426336288452f,
    2.0321125984191895f, 2.0347988605499268f, 2.037484884262085f, 2.0401711463928223f,
    2.0428574085235596f, 2.0455434322357178f, 2.048229694366455f, 2.0509159564971924f,
    2.0536019802093506f, 2.056288242340088f, 2.058974504470825f, 2.0616605281829834f,
    2.0643467903137207f, 2.067033052444458f, 2.069719076156616f, 2.0724053382873535f,
    2.075091600418091f, 2.077777624130249f, 2.0804638862609863f, 2.0831501483917236f,
    2.085836172103882f, 2.088522434234619f, 2.0912086963653564f, 2.0938947200775146f,
    2.096580982208252f, 2.0992672443389893f, 2.1019532680511475f, 2.1046395301818848f,
    2.107325792312622f, 2.1100118160247803f, 2.1126980781555176f, 2.115384340286255f,
    2.118070363998413f, 2.1207566261291504f, 2.123443603515625f, 2.1261298656463623f,
    2.1288158893585205f, 2.131502151489258f, 2.134188413619995f, 2.1368744373321533f,
    2.1395606994628906f, 2.142246961593628f, 2.144932985305786f, 2.1476192474365234f,
    2.1503055095672607f, 2.152991533279419f, 2.1556777954101562f, 2.1583640575408936f,
    2.1610500812530518f, 2.163736343383789f, 2.1664226055145264f, 2.1691086292266846f,
    2.171794891357422f, 2.174481153488159f, 2.1771671772003174f, 2.1798534393310547f,
    2.182539701461792f, 2.18522572517395f, 2.1879119873046875f, 2.190598249435425f,
    2.193284273147583f, 2.1959705352783203f, 2.1986567974090576f, 2.201342821121216f,
    2.204029083251953f, 2.2067153453826904f, 2.2094013690948486f, 2.212087631225586f,
    2.2147738933563232f, 2.2174599170684814f, 2.2201461791992188f, 2.222832441329956f,
    2.2255184650421143f, 2.2282054424285889f, 2.230891704559326f, 2.2335779666900635f,
    2.2362639904022217f, 2.238950252532959f, 2.2416365146636963f, 2.2443225383758545f,
    2.247008800506592f, 2.249695062637329f, 2.2523810863494873f, 2.2550673484802246f,
    2.257753610610962f, 2.26043963432312f, 2.2631258964538574f, 2.2658121585845947f,
    2.268498182296753f, 2.2711844444274902f, 2.2738707065582275f, 2.2765567302703857f,
    2.279242992401123f, 2.2819292545318604f, 2.2846152782440186f, 2.287301540374756f,
    2.289987802505493f, 2.2926738262176514f, 2.2953600883483887f, 2.298046350479126f,
    2.300732374191284f, 2.3034186363220215f, 2.306104898452759f, 2.308790922164917f,
    2.3114771842956543f, 2.3141634464263916f, 2.31684947013855f, 2.319535732269287f,
    2.3222219944000244f, 2.3249080181121826f, 2.32759428024292f, 2.3302805423736572f,
    2.3329665660858154f, 2.33565354347229f, 2.3383398056030273f, 2.3410260677337646f,
    2.343712091445923f, 2.34639835357666f, 2.3490846157073975f, 2.3517706394195557f,
    2.354456901550293f, 2.3571431636810303f, 2.3598291873931885f, 2.362515449523926f,
    2.365201711654663f, 2.3678877353668213f, 2.3705739974975586f, 2.373260259628296f,
    2.375946283340454f, 2.3786325454711914f, 2.3813188076019287f, 2.384004831314087f,
    2.386691093444824f, 2.3893773555755615f, 2.3920633792877197f, 2.394749641418457f,
    2.3974359035491943f, 2.4001219272613525f, 2.40280818939209f, 2.405494451522827f,
    2.4081804752349854f, 2.4108667373657227f, 2.41355299949646f, 2.416239023208618f,
    2.4189252853393555f, 2.4216115474700928f, 2.424297571182251f, 2.4269838333129883f,
    2.4296700954437256f, 2.432356119155884f, 2.435042381286621f, 2.4377286434173584f,
    2.4404146671295166f, 2.443101644515991f, 2.4457879066467285f, 2.448474168777466f,
    2.451160192489624f, 2.4538464546203613f, 2.4565327167510986f, 2.459218740463257f,
    2.461905002593994f, 2.4645912647247314f, 2.4672772884368896f, 2.469963550567627f,
    2.4726498126983643f, 2.4753358364105225f, 2.4780220985412598f, 2.480708360671997f,
    2.4833943843841553f, 2.4860806465148926f, 2.48876690864563f, 2.491452932357788f,
    2.4941391944885254f, 2.4968254566192627f, 2.499511480331421f, 2.502197742462158f,
    2.5048840045928955f, 2.5075700283050537f, 2.510256290435791f, 2.5129425525665283f,
    2.5156285762786865f, 2.518314838409424f, 2.521001100540161f, 2.5236871242523193f,
    2.5263733863830566f, 2.529059648513794f, 2.531745672225952f, 2.5344319343566895f,
    2.5371181964874268f, 2.539804220199585f, 2.5424904823303223f, 2.5451767444610596f,
    2.547863721847534f, 2.5505497455596924f, 2.5532360076904297f, 2.555922269821167f,
    2.558608293533325f, 2.5612945556640625f, 2.5639808177948f, 2.566666841506958f,
    2.5693531036376953f, 2.5720393657684326f, 2.574725389480591f, 2.577411651611328f,
    2.5800979137420654f, 2.5827839374542236f, 2.585470199584961f, 2.5881564617156982f,
    2.5908424854278564f, 2.5935287475585938f, 2.596215009689331f, 2.5989010334014893f,
    2.6015872955322266f, 2.604273557662964f, 2.606959581375122f, 2.6096458435058594f,
    2.6123321056365967f, 2.615018129348755f, 2.617704391479492f, 2.6203906536102295f,
    2.6230766773223877f, 2.625762939453125f, 2.6284492015838623f, 2.6311352252960205f,
    2.633821487426758f, 2.636507749557495f, 2.6391937732696533f, 2.6418800354003906f,
    2.644566297531128f, 2.647252321243286f, 2.6499385833740234f, 2.6526248455047607f,
    2.6553118228912354f, 2.6579978466033936f, 2.660684108734131f, 2.663370370864868f,
    2.6660563945770264f, 2.6687426567077637f, 2.671428918838501f, 2.674114942550659f,
    2.6768012046813965f, 2.679487466812134f, 2.682173490524292f, 2.6848597526550293f,
    2.6875460147857666f, 2.690232038497925f, 2.692918300628662f, 2.6956045627593994f,
    2.6982905864715576f, 2.700976848602295f, 2.7036631107330322f, 2.7063491344451904f,
    2.7090353965759277f, 2.711721658706665f, 2.7144076824188232f, 2.7170939445495605f,
    2.719780206680298f, 2.722466230392456f, 2.7251524925231934f, 2.7278387546539307f,
    2.730524778366089f, 2.733211040496826f, 2.7358973026275635f, 2.7385833263397217f,
    2.741269588470459f, 2.7439558506011963f, 2.7466418743133545f, 2.749328136444092f,
    2.752014398574829f, 2.7547004222869873f, 2.7573866844177246f, 2.760072946548462f,
    2.7627599239349365f, 2.7654459476470947f, 2.768132209777832f, 2.7708184719085693f,
    2.7735044956207275f, 2.776190757751465f, 2.778877019882202f, 2.7815630435943604f,
    2.7842493057250977f, 2.786935567855835f, 2.789621591567993f, 2.7923078536987305f,
    2.7949941158294678f, 2.797680139541626f, 2.8003664016723633f, 2.8030526638031006f,
    2.805738687515259f, 2.808424949645996f, 2.8111112117767334f, 2.8137972354888916f,
    2.816483497619629f, 2.819169759750366f, 2.8218557834625244f, 2.8245420455932617f,
    2.827228307723999f, 2.8299143314361572f, 2.8326005935668945f, 2.835286855697632f,
    2.83797287940979f, 2.8406591415405273f, 2.8433454036712646f, 2.846031427383423f,
    2.84871768951416f, 2.8514039516448975f, 2.8540899753570557f, 2.856776237487793f,
    2.8594624996185303f, 2.8621485233306885f, 2.864834785461426f, 2.8675217628479004f,
    2.8702080249786377f, 2.872894048690796f, 2.875580310821533f, 2.8782665729522705f,
    2.8809525966644287f, 2.883638858795166f, 2.8863251209259033f, 2.8890111446380615f,
    2.891697406768799f, 2.894383668899536f, 2.8970696926116943f, 2.8997559547424316f,
    2.902442216873169f, 2.905128240585327f, 2.9078145027160645f, 2.9105007648468018f,
    2.91318678855896f, 2.9158730506896973f, 2.9185593128204346f, 2.921245336532592 8f,
    2.92393159866333f, 2.9266178607940674f, 2.9293038845062256f, 2.931990146636963f,
    2.9346764087677f, 2.9373624324798584f, 2.9400486946105957f, 2.942734956741333f,
    2.945420980453491f, 2.9481072425842285f, 2.950793504714966f, 2.953479528427124f,
    2.9561657905578613f, 2.9588520526885986f, 2.961538076400757f, 2.964224338531494f,
    2.9669106006622314f, 2.9695966243743896f, 2.972282886505127f, 2.9749698638916016f,
    2.977656126022339f, 2.980342149734497f, 2.9830284118652344f, 2.9857146739959717f,
    2.98840069770813f, 2.991086959838867f, 2.9937732219696045f, 2.9964592456817627f,
    2.9991455078125f, 3.001831531524658f, 3.0045177936553955f, 3.007204055786133f,
    3.009890079498291f, 3.0125763416290283f, 3.0152626037597656f, 3.017948627471924f,
    3.020634889602661f, 3.0233211517333984f, 3.0260071754455566f, 3.028693437576294f,
    3.0313796997070312f, 3.0340657234191895f, 3.0367519855499268f, 3.039438247680664f,
    3.0421242713928223f, 3.0448105335235596f, 3.047496795654297f, 3.050182819366455f,
    3.0528690814971924f, 3.0555553436279297f, 3.058241367340088f, 3.060927629470825f,
    3.0636138916015625f, 3.0662999153137207f, 3.068986177444458f, 3.0716724395751953f,
    3.0743584632873535f, 3.077044725418091f, 3.079730987548828f, 3.0824179649353027f,
    3.085103988647461f, 3.0877902507781982f, 3.0904765129089355f, 3.0931625366210938f,
    3.095848798751831f, 3.0985350608825684f, 3.1012210845947266f, 3.103907346725464f,
    3.106593608856201f, 3.1092796325683594f, 3.1119658946990967f, 3.114652156829834f,
    3.117338180541992f, 3.1200244426727295f, 3.122710704803467f, 3.125396728515625f,
    3.1280829906463623f, 3.1307692527770996f, 3.133455276489258f, 3.136141538619995f,
    3.1388278007507324f, 3.1415138244628906f, 3.144200086593628f, 3.1468863487243652f,
    3.1495723724365234f, 3.1522586345672607f, 3.154944896697998f, 3.1576309204101562f,
    3.1603171825408936f, 3.163003444671631f, 3.165689468383789f, 3.1683757305145264f,
    3.1710619926452637f, 3.173748016357422f, 3.176434278488159f, 3.1791205406188965f,
    3.1818065643310547f, 3.184492826461792f, 3.1871798038482666f, 3.189866065979004f,
    3.192552089691162f, 3.1952383518218994f, 3.1979246139526367f, 3.200610637664795f,
    3.2032968997955322f, 3.2059831619262695f, 3.2086691856384277f, 3.211355447769165f,
    3.2140417098999023f, 3.2167277336120605f, 3.219413995742798f, 3.222100257873535f,
    3.2247862815856934f, 3.2274725437164307f, 3.230158805847168f, 3.232844829559326f,
    3.2355310916900635f, 3.238217353820801f, 3.240903377532959f, 3.2435896396636963f,
    3.2462759017944336f, 3.248961925506592f, 3.251648187637329f, 3.2543344497680664f,
    3.2570204734802246f, 3.259706735610962f, 3.262392997741699f, 3.2650790214538574f,
    3.2677652835845947f, 3.270451545715332f, 3.2731375694274902f, 3.2758238315582275f,
    3.278510093688965f, 3.281196117401123f, 3.2838823795318604f, 3.2865686416625977f,
    3.289254665374756f, 3.291940927505493f, 3.2946279048919678f, 3.297314167022705f,
    3.3000001907348633f, 3.3026864528656006f, 3.305372714996338f, 3.308058738708496f,
    3.3107450008392334f, 3.3134312629699707f, 3.316117286682129f, 3.318803548812866f,
    3.3214898109436035f, 3.3241758346557617f, 3.326862096786499f, 3.3295483589172363f,
    3.3322343826293945f, 3.334920644760132f, 3.337606906890869f, 3.3402929306030273f,
    3.3429791927337646f, 3.345665454864502f, 3.34835147857666f, 3.3510377407073975f,
    3.3537240028381348f, 3.356410026550293f, 3.3590962886810303f, 3.3617825508117676f,
    3.364468574523926f, 3.367154836654663f, 3.3698410987854004f, 3.3725271224975586f,
    3.375213384628296f, 3.377899646759033f, 3.3805856704711914f, 3.3832719326019287f,
    3.385958194732666f, 3.388644218444824f, 3.3913304805755615f, 3.394016742706299f,
    3.396702766418457f, 3.3993890285491943f, 3.402076005935669f, 3.4047622680664062f,
    3.4074482917785645f, 3.4101345539093018f, 3.412820816040039f, 3.4155068397521973f,
    3.4181931018829346f, 3.420879364013672f, 3.42356538772583f, 3.4262516498565674f,
    3.4289379119873047f, 3.431623935699463f, 3.4343101978302f, 3.4369964599609375f,
    3.4396824836730957f, 3.442368745803833f, 3.4450550079345703f, 3.4477410316467285f,
    3.450427293777466f, 3.453113555908203f, 3.4557995796203613f, 3.4584858417510986f,
    3.461172103881836f, 3.463858127593994f, 3.4665443897247314f, 3.4692306518554688f,
    3.471916675567627f, 3.4746029376983643f, 3.4772891998291016f, 3.4799752235412598f,
    3.482661485671997f, 3.4853477478027344f, 3.4880337715148926f, 3.49072003364563f,
    3.493406295776367f, 3.4960923194885254f, 3.4987785816192627f, 3.50146484375f,
    3.504150867462158f, 3.506837844848633f, 3.50952410697937f, 3.5122103691101074f,
    3.5148963928222656f, 3.517582654953003f, 3.5202689170837402f, 3.5229549407958984f,
    3.5256412029266357f, 3.528327465057373f, 3.5310134887695312f, 3.5336997509002686f,
    3.536386013031006f, 3.539072036743164f, 3.5417582988739014f, 3.5444445610046387f,
    3.547130584716797f, 3.549816846847534f, 3.5525031089782715f, 3.5551891326904297f,
    3.557875394821167f, 3.5605616569519043f, 3.5632476806640625f, 3.5659339427948f,
    3.5686202049255371f, 3.5713062286376953f, 3.5739924907684326f, 3.57667875289917f,
    3.579364776611328f, 3.5820510387420654f, 3.5847373008728027f, 3.587423324584961f,
    3.5901095867156982f, 3.5927958488464355f, 3.5954818725585938f, 3.598168134689331f,
    3.6008543968200684f, 3.6035404205322266f, 3.606226682662964f, 3.608912944793701f,
    3.6115989685058594f, 3.6142859458923340f, 3.6169722080230713f, 3.6196584701538086f,
    3.622344493865967f, 3.625030755996704f, 3.6277170181274414f, 3.6304030418395996f,
    3.633089303970337f, 3.635775566101074f, 3.6384615898132324f, 3.6411478519439697f,
    3.643834114074707f, 3.6465201377868652f, 3.6492063999176025f, 3.65189266204834f,
    3.654578685760498f, 3.6572649478912354f, 3.6599512100219727f, 3.662637233734131f,
    3.665323495864868f, 3.6680097579956055f, 3.6706957817077637f, 3.673382043838501f,
    3.6760683059692383f, 3.6787543296813965f, 3.681440591812134f, 3.684126853942871f,
    3.6868128776550293f, 3.6894991397857666f, 3.692185401916504f, 3.694871425628662f,
    3.6975576877593994f, 3.7002439498901367f, 3.702929973602295f, 3.7056162357330322f,
    3.7083024978637695f, 3.7109885215759277f, 3.713674783706665f, 3.7163610458374023f,
    3.7190470695495605f, 3.721734046936035f, 3.7244203090667725f, 3.7271065711975098f,
    3.729792594909668f, 3.7324788570404053f, 3.7351651191711426f, 3.737851142883301f,
    3.740537405014038f, 3.7432236671447754f, 3.7459096908569336f, 3.748595952987671f,
    3.751282215118408f, 3.7539682388305664f, 3.7566545009613037f, 3.759340763092041f,
    3.762026786804199f, 3.7647130489349365f, 3.767399311065674f, 3.770085334777832f,
    3.7727715969085693f, 3.7754578590393066f, 3.778143882751465f, 3.780830144882202f,
    3.7835164070129395f, 3.7862024307250977f, 3.788888692855835f, 3.7915749549865723f,
    3.7942609786987305f, 3.7969472408294678f, 3.799633502960205f, 3.8023195266723633f,
    3.8050057888031006f, 3.807692050933838f, 3.810378074645996f, 3.8130643367767334f,
    3.8157505989074707f, 3.818436622619629f, 3.821122884750366f, 3.8238091468811035f,
    3.8264951705932617f, 3.829182147979736f, 3.8318684101104736f, 3.834554672241211f,
    3.837240695953369f, 3.8399269580841064f, 3.8426132202148438f, 3.845299243927002f,
    3.8479855060577393f, 3.8506717681884766f, 3.853357791900635f, 3.856044054031372f,
    3.8587303161621094f, 3.8614163398742676f, 3.864102602005005f, 3.866788864135742f,
    3.8694748878479004f, 3.8721611499786377f, 3.874847412109375f, 3.877533435821533f,
    3.8802196979522705f, 3.882905960083008f, 3.885591983795166f, 3.888278245925903f,
    3.8909645080566406f, 3.893650531768799f, 3.896336793899536f, 3.8990230560302734f,
    3.9017090797424316f, 3.904395341873169f, 3.9070816040039062f, 3.9097676277160645f,
    3.9124538898468018f, 3.915140151977539f, 3.9178261756896973f, 3.9205124378204346f,
    3.923198699951172f, 3.92588472366333f, 3.9285709857940674f, 3.9312572479248047f,
    3.9339442253112793f, 3.9366302490234375f, 3.939316511154175f, 3.942002773284912f,
    3.9446887969970703f, 3.9473750591278076f, 3.950061321258545f, 3.952747344970703f,
    3.9554336071014404f, 3.9581198692321777f, 3.960805892944336f, 3.9634921550750732f,
    3.9661784172058105f, 3.9688644409179688f, 3.971550703048706f, 3.9742369651794434f,
    3.9769229888916016f, 3.979609251022339f, 3.982295513153076f, 3.9849815368652344f,
    3.9876677989959717f, 3.990354061126709f, 3.993040084838867f, 3.9957263469696045f,
    3.998412609100342f, 4.0010986328125f, 4.003784656524658f, 4.006471157073975f,
    4.009157180786133f, 4.011843204498291f, 4.014529705047607f, 4.017215728759766f,
    4.019901752471924f, 4.02258825302124f, 4.025274276733398f, 4.027960300445557f,
    4.030646800994873f, 4.033332824707031f, 4.0360188484191895f, 4.038705348968506f,
    4.0413923263549805f, 4.044078350067139f, 4.046764373779297f, 4.049450874328613f,
    4.0521368980407715f, 4.05482292175293f, 4.057509422302246f, 4.060195446014404f,
    4.0628814697265625f, 4.065567970275879f, 4.068253993988037f, 4.070940017700195f,
    4.073626518249512f, 4.07631254196167f, 4.078998565673828f, 4.0816850662231445f,
    4.084371089935303f, 4.087057113647461f, 4.089743614196777f, 4.0924296379089355f,
    4.095115661621094f, 4.09780216217041f, 4.100488185882568f, 4.103174209594727f,
    4.105860710144043f, 4.108546733856201f, 4.111232757568359f, 4.113919258117676f,
    4.116605281829834f, 4.119291305541992f, 4.121977806091309f, 4.124663829803467f,
    4.127349853515625f, 4.130036354064941f, 4.1327223777771f, 4.135408401489258f,
    4.138094902038574f, 4.140780925750732f, 4.143466949462891f, 4.146153450012207f,
    4.148840427398682f, 4.15152645111084f, 4.154212474822998f, 4.1568989753723145f,
    4.159584999084473f, 4.162271022796631f, 4.164957523345947f, 4.1676435470581055f,
    4.170329570770264f, 4.17301607131958f, 4.175702095031738f, 4.1783881187438965f,
    4.181074619293213f, 4.183760643005371f, 4.186446666717529f, 4.189133167266846f,
    4.191819190979004f, 4.194505214691162f, 4.1971917152404785f, 4.199877738952637f,
    4.202563762664795f, 4.205250263214111f, 4.2079362869262695f, 4.210622310638428f,
    4.213308811187744f, 4.215994834899902f, 4.2186808586120605f, 4.221367359161377f,
    4.224053382873535f, 4.226739406585693f, 4.22942590713501f, 4.232111930847168f,
    4.234797954559326f, 4.237484455108643f, 4.240170478820801f, 4.242856502532959f,
    4.245543003082275f, 4.248229026794434f, 4.250915050506592f, 4.253602027893066f,
    4.256288528442383f, 4.258974552154541f, 4.261660575866699f, 4.264347076416016f,
    4.267033100128174f, 4.269719123840332f, 4.272405624389648f, 4.275091648101807f,
    4.277777671813965f, 4.280464172363281f, 4.2831501960754395f, 4.285836219787598f,
    4.288522720336914f, 4.291208744049072f, 4.2938947677612305f, 4.296581268310547f,
    4.299267292022705f, 4.301953315734863f, 4.30463981628418f, 4.307325839996338f,
    4.310011863708496f, 4.3126983642578125f, 4.315384387969971f, 4.318070411682129f,
    4.320756912231445f, 4.3234429359436035f, 4.326128959655762f, 4.328815460205078f,
    4.331501483917236f, 4.3341875076293945f, 4.336874008178711f, 4.339560031890869f,
    4.342246055603027f, 4.344932556152344f, 4.347618579864502f, 4.35030460357666f,
    4.352991104125977f, 4.355677127838135f, 4.358363151550293f, 4.361050128936768f,
    4.363736629486084f, 4.366422653198242f, 4.3691086769104f, 4.371795177459717f,
    4.374481201171875f, 4.377167224884033f, 4.37985372543335f, 4.382539749145508f,
    4.385225772857666f, 4.387912273406982f, 4.390598297119141f, 4.393284320831299f,
    4.395970821380615f, 4.398656845092773f, 4.401342868804932f, 4.404029369354248f,
    4.406715393066406f, 4.4094014167785645f, 4.412087917327881f, 4.414773941040039f,
    4.417459964752197f, 4.420146465301514f, 4.422832489013672f, 4.42551851272583f,
    4.4282050132751465f, 4.430891036987305f, 4.433577060699463f, 4.436263561248779f,
    4.4389495849609375f, 4.441635608673096f, 4.444322109222412f, 4.44700813293457f,
    4.4496941566467285f, 4.452380657196045f, 4.455066680908203f, 4.457752704620361f,
    4.460439205169678f, 4.463125228881836f, 4.465811252593994f, 4.468498229980469f,
    4.471184730529785f, 4.473870754241943f, 4.476556777954102f, 4.479243278503418f,
    4.481929302215576f, 4.484615325927734f, 4.487301826477051f, 4.489987850189209f,
    4.492673873901367f, 4.495360374450684f, 4.498046398162842f, 4.500732421875f,
    4.503418922424316f, 4.506104946136475f, 4.508790969848633f, 4.511477470397949f,
    4.514163494110107f, 4.516849517822266f, 4.519536018371582f, 4.52222204208374f,
    4.524908065795898f, 4.527594566345215f, 4.530280590057373f, 4.532966613769531f,
    4.535653114318848f, 4.538339138031006f, 4.541025161743164f, 4.5437116622924805f,
    4.546397686004639f, 4.549083709716797f, 4.551770210266113f, 4.5544562339782715f,
    4.55714225769043f, 4.559828758239746f, 4.562514781951904f, 4.5652008056640625f,
    4.567887306213379f, 4.570573329925537f, 4.573260307312012f, 4.57594633102417f,
    4.578632831573486f, 4.5813188552856445f, 4.584004878997803f, 4.586691379547119f,
    4.589377403259277f, 4.5920634269714355f, 4.594749927520752f, 4.59743595123291f,
    4.600121974945068f, 4.602808475494385f, 4.605494499206543f, 4.608180522918701f,
    4.610867023468018f, 4.613553047180176f, 4.616239070892334f, 4.61892557144165f,
    4.621611595153809f, 4.624297618865967f, 4.626984119415283f, 4.629670143127441f,
    4.6323561668396f, 4.635042667388916f, 4.637728691101074f, 4.640414714813232f,
    4.643101215362549f, 4.645787239074707f, 4.648473262786865f, 4.651159763336182f,
    4.65384578704834f, 4.656531810760498f, 4.6592183113098145f, 4.661904335021973f,
    4.664590358734131f, 4.667276859283447f, 4.6699628829956055f, 4.672648906707764f,
    4.67533540725708f, 4.678021430969238f, 4.680708408355713f, 4.683394432067871f,
    4.6860809326171875f, 4.688766956329346f, 4.691452980041504f, 4.69413948059082f,
    4.6968255043029785f, 4.699511528015137f, 4.702198028564453f, 4.704884052276611f,
    4.7075700759887695f, 4.710256576538086f, 4.712942600250244f, 4.715628623962402f,
    4.718315124511719f, 4.721001148223877f, 4.723687171936035f, 4.726373672485352f,
    4.72905969619751f, 4.731745719909668f, 4.734432220458984f, 4.737118244171143f,
    4.739804267883301f, 4.742490768432617f, 4.745176792144775f, 4.747862815856934f,
    4.75054931640625f, 4.753235340118408f, 4.755921363830566f, 4.758607864379883f,
    4.761293888092041f, 4.763979911804199f, 4.766666412353516f, 4.769352436065674f,
    4.772038459777832f, 4.774724960327148f, 4.777410984039307f, 4.780097007751465f,
    4.782783508300781f, 4.7854695320129395f, 4.788156509399414f, 4.790842533111572f,
    4.793529033660889f, 4.796215057373047f, 4.798901081085205f, 4.8015875816345215f,
    4.80427360534668f, 4.806959629058838f, 4.809646129608154f, 4.8123321533203125f,
    4.815018177032471f, 4.817704677581787f, 4.820390701293945f, 4.8230767250061035f,
    4.82576322555542f, 4.828449249267578f, 4.831135272979736f, 4.833821773529053f,
        4.836507797241211f, 4.839193820953369f, 4.8418803215026855f, 4.844566345214844f,
        4.847252368927002f, 4.849938869476318f, 4.852624893188477f, 4.855310916900635f,
        4.857997417449951f, 4.860683441162109f, 4.863369464874268f, 4.866055965423584f,
        4.868741989135742f, 4.8714280128479f, 4.874114513397217f, 4.876800537109375f,
        4.879486560821533f, 4.88217306137085f, 4.884859085083008f, 4.887545108795166f,
        4.890231609344482f, 4.892918586730957f, 4.895604610443115f, 4.898290634155273f,
        4.90097713470459f, 4.903663158416748f, 4.906349182128906f, 4.909035682678223f,
        4.911721706390381f, 4.914407730102539f, 4.9170942306518555f, 4.919780254364014f,
        4.922466278076172f, 4.925152778625488f, 4.9278388023376465f, 4.930524826049805f,
        4.933211326599121f, 4.935897350311279f, 4.9385833740234375f, 4.941269874572754f,
        4.943955898284912f, 4.94664192199707f, 4.949328422546387f, 4.952014446258545f,
        4.954700469970703f, 4.9573869705200195f, 4.960072994232178f, 4.962759017944336f,
        4.965445518493652f, 4.9681315422058105f, 4.970817565917969f, 4.973504066467285f,
        4.976190090179443f, 4.978876113891602f, 4.981562614440918f, 4.984248638153076f,
        4.986934661865234f, 4.989621162414551f, 4.992307186126709f, 4.994993209838867f,
        4.997679710388184f, 5.000366687774658f, 5.003052711486816f, 5.005738735198975f,
        5.008425235748291f, 5.011111259460449f, 5.013797283172607f, 5.016483783721924f,
        5.019169807434082f, 5.02185583114624f, 5.024542331695557f, 5.027228355407715f,
        5.029914379119873f, 5.0326008796691895f, 5.035286903381348f, 5.037972927093506f,
        5.040659427642822f, 5.0433454513549805f, 5.046031475067139f, 5.048717975616455f,
        5.051403999328613f, 5.0540900230407715f, 5.056776523590088f, 5.059462547302246f,
        5.062148571014404f, 5.064835071563721f, 5.067521095275879f, 5.070207118988037f,
        5.0728936195373535f, 5.075579643249512f, 5.07826566696167f, 5.080952167510986f,
        5.0836381912231445f, 5.086324214935303f, 5.089010715484619f, 5.091696739196777f,
        5.0943827629089355f, 5.097069263458252f, 5.09975528717041f, 5.102441310882568f,
        5.105127811431885f, 5.107814788818359f, 5.110500812530518f, 5.113186836242676f,
        5.115873336791992f, 5.11855936050415f, 5.121245384216309f, 5.123931884765625f,
        5.126617908477783f, 5.129303932189941f, 5.131990432739258f, 5.134676456451416f,
        5.137362480163574f, 5.140048980712891f, 5.142735004425049f, 5.145421028137207f,
        5.148107528686523f, 5.150793552398682f, 5.15347957611084f, 5.156166076660156f,
        5.1588521003723145f, 5.161538124084473f, 5.164224624633789f, 5.166910648345947f,
        5.1695966720581055f, 5.172283172607422f, 5.17496919631958f, 5.177655220031738f,
        5.180341720581055f, 5.183027744293213f, 5.185713768005371f, 5.1884002685546875f,
        5.191086292266846f, 5.193772315979004f, 5.19645881652832f, 5.1991448402404785f,
        5.201830863952637f, 5.204517364501953f, 5.207203388214111f, 5.2098894119262695f,
        5.212576389312744f, 5.2152628898620605f, 5.217948913574219f, 5.220634937286377f,
        5.223321437835693f, 5.226007461547852f, 5.22869348526001f, 5.231379985809326f,
        5.234066009521484f, 5.236752033233643f, 5.239438533782959f, 5.242124557495117f,
        5.244810581207275f, 5.247497081756592f, 5.25018310546875f, 5.252869129180908f,
        5.255555629730225f, 5.258241653442383f, 5.260927677154541f, 5.263614177703857f,
        5.266300201416016f, 5.268986225128174f, 5.27167272567749f, 5.274358749389648f,
        5.277044773101807f, 5.279731273651123f, 5.282417297363281f, 5.2851033210754395f,
        5.287789821624756f, 5.290475845336914f, 5.293161869049072f, 5.295848369598389f,
        5.298534393310547f, 5.301220417022705f, 5.3039069175720215f, 5.30659294128418f,
        5.309278964996338f, 5.311965465545654f, 5.3146514892578125f, 5.317337512969971f,
        5.320024490356445f, 5.322710990905762f, 5.32539701461792f, 5.328083038330078f,
        5.3307695388793945f, 5.333455562591553f, 5.336141586303711f, 5.338828086853027f,
        5.3415141105651855f, 5.344200134277344f, 5.34688663482666f, 5.349572658538818f,
        5.352258682250977f, 5.354945182800293f, 5.357631206512451f, 5.360317230224609f,
        5.363003730773926f, 5.365689754486084f, 5.368375778198242f, 5.371062278747559f,
        5.373748302459717f, 5.376434326171875f, 5.379120826721191f, 5.38180685043335f,
        5.384492874145508f, 5.387179374694824f, 5.389865398406982f, 5.392551422119141f,
        5.395237922668457f, 5.397923946380615f, 5.400609970092773f, 5.40329647064209f,
        5.405982494354248f, 5.408668518066406f, 5.411355018615723f, 5.414041042327881f,
        5.416727066040039f, 5.4194135665893555f, 5.422099590301514f, 5.424785614013672f,
        5.4274725914001465f, 5.430159091949463f, 5.432845115661621f, 5.435531139373779f,
        5.438217639923096f, 5.440903663635254f, 5.443589687347412f, 5.4462761878967285f,
        5.448962211608887f, 5.451648235321045f, 5.454334735870361f, 5.4570207595825195f,
        5.459706783294678f, 5.462393283843994f, 5.465079307556152f, 5.4677653312683105f,
        5.470451831817627f, 5.473137855529785f, 5.475823879241943f, 5.47851037979126f,
        5.481196403503418f, 5.483882427215576f, 5.486568927764893f, 5.489254951477051f,
        5.491940975189209f, 5.494627475738525f, 5.497313499450684f, 5.499999523162842f
    };
```

11.1.3 Liquid Crystal Display (LCD)

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

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))

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 Functions

- 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 Functions

- void LCD_Draw (void)

  Draw on the LCD.
- void LCD_Fill (void)

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

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

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

  Draw a rectangle of size `dx` x `dy` onto the display. The bottom-left corner will be located at `(x1, y1)`.
- void LCD_plotSample (uint16_t x, uint16_t y, uint8_t color)

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

11.1.3.1 Detailed Description

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

11.1.3.2 Enumeration Type Documentation

LCD_PLOT_INFO

```
enum LCD_PLOT_INFO
                   {
    LCD_X_MAX = ILI9341_NUM_ROWS - 1,
    LCD_Y_MAX = ILI9341_NUM_COLS - 1
};
```

LCD_COLORS

```
enum LCD_COLORS
                {
    // Bits 2, 1, 0 correspond to R, G, and B values, respectively.
    // NOTE: since the display colors are inverted, these bit patterns are too.
    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
};
```

LCD_WRITING_INFO

```
enum LCD_WRITING_INFO
                      {
    HEIGHT_CHAR = 8,
    LEN_CHAR = 5,

    NUM_LINES = 30,
    NUM_COLS = 64
};
```

11.1.3.3 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

```
                                                                          {
    assert(lineWidth > 0);
    assert((lineWidth % 2) != 0);

    // ensure line does not go out-of-bounds
    uint16_t padding = ((lineWidth - 1) / 2);
    uint16_t MAX_NUM = (is_horizontal) ? LCD_Y_MAX : LCD_X_MAX;
    if(center < padding) {
        center = padding + 1;
    }
    else if(center >= (MAX_NUM - padding)) {
        center = (MAX_NUM - padding) - 1;
    }

    // set start and end row/column, and draw
    uint16_t start = center - padding;
    uint16_t end = center + padding;
    if(is_horizontal) {
        LCD_setX(0, (LCD_X_MAX));
        LCD_setY(start, end);
    }
    else {
        LCD_setX(start, end);
        LCD_setY(0, (LCD_Y_MAX));
    }

    LCD_Draw();

    return;
}
```

LCD_updateCursor()

```
static void LCD_updateCursor (
              void  )  [static]
```

Update the cursor for after writing text on the display.

```
                                   {
    uint16_t newLineNum = lcd.lineNum / HEIGHT_CHAR;
    uint16_t newColNum = lcd.colNum / LEN_CHAR;

    newColNum = (newColNum + 2) % NUM_COLS;
    newLineNum = (newColNum == 0) ? ((newLineNum + 2) % NUM_LINES) : newLineNum;

    lcd.lineNum = newLineNum * HEIGHT_CHAR;
    lcd.colNum = newColNum * LEN_CHAR;

    return;
}
```

LCD_Init()

```
void LCD_Init (
              void  )
```

Initialize the LCD.

Postcondition

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

```
                    {
    assert(lcd.isInit == false);                              // should only be initialized once

    GpioPort_t portA = GPIO_InitPort(GPIO_PORT_A);
    Spi_t spi = SPI_Init(portA, GPIO_PIN6, SSI0);
    Timer_t timer2 = Timer_Init(TIMER2);

    ILI9341_Init(portA, GPIO_PIN7, spi, timer2);
    ILI9341_setSleepMode(SLEEP_OFF, timer2);
    Timer_Deinit(timer2);

    ILI9341_setMemAccessCtrl(1, 0, 0, 0, 1, 0);               // TODO: explain this

    ILI9341_setColorDepth(COLORDEPTH_16BIT);
    ILI9341_setColorExpression(PARTIAL_COLORS);
    ILI9341_setDisplayArea(NORMAL_AREA);
    ILI9341_setDispInversion(INVERT_ON);
    ILI9341_setDispOutput(OUTPUT_OFF);

    lcd.isInit = true;

    LCD_setColor(LCD_BLACK);
    LCD_Fill();                                               // black background

    return;
}
```

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.

```
                                  {
    outputMode_t outputMode = (isOn) ? OUTPUT_ON : OUTPUT_OFF;
    ILI9341_setDispOutput(outputMode);

    return;
}
```

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.x1 = x1;
    lcd.x2 = x2;
    ILI9341_setRowAddress(lcd.x1, lcd.x2);
    return;
}
```

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.y1 = y1;
    lcd.y2 = y2;
    ILI9341_setColAddress(lcd.y1, lcd.y2);
    return;
}
```

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.

```
                              {
    assert(color < 0x08);
    lcd.color = color;
    return;
}
```

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()

```
                 {
    // determine RGB values
    uint8_t R, G, B;
    if(lcd.color == 0) {
        R = 1;
        G = 1;
        B = 1;
    }
    else {
        R = 0x1F * ((lcd.color & 0x04) » 2);
        G = 0x3F * ((lcd.color & 0x02) » 1);
        B = 0x1F * (lcd.color & 0x01);
    }

    uint32_t numPixels = (uint32_t) ((lcd.x2 - lcd.x1) + 1) * ((lcd.y2 - lcd.y1) + 1);
    ILI9341_writeMemCmd();
    for(uint32_t count = 0; count < numPixels; count++) {
        ILI9341_writePixel(R, G, B);
    }

    return;
}
```

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_setX(0, LCD_X_MAX);
    LCD_setY(0, LCD_Y_MAX);

    LCD_Draw();

    return;
}
```

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_drawLine(yCenter, lineWidth, true);
    return;
}
```

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_drawLine(xCenter, lineWidth, false);
    return;
}
```

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()

```
{
    assert(x1 <= LCD_X_MAX);
    assert(x1 + dx <= LCD_X_MAX);
    assert(y1 <= LCD_Y_MAX);
    assert((y1 + dy) <= LCD_Y_MAX);

    uint16_t x2 = (x1 + dx) - 1;
    uint16_t y2 = (y1 + dy) - 1;
    LCD_setX(x1, x2);
    LCD_setY(y1, y2);
    LCD_Draw();

    return;
}
```

LCD_plotSample()

```
void LCD_plotSample (
            uint16_t x,
            uint16_t y,
            uint8_t color )
```

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()

```
{
    uint8_t currColor = lcd.color;
    LCD_setColor(color);

    LCD_setX(x, x);
    LCD_setY(y, y);
    LCD_Draw();

    LCD_setColor(currColor);
    return;
}
```

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)`.

```
                                                    {
    assert(lineNum < NUM_LINES);
    assert(colNum < NUM_COLS);

    lcd.lineNum = lineNum * HEIGHT_CHAR;
    lcd.colNum = colNum * LEN_CHAR;

    return;
}
```

LCD_writeChar()

```
void LCD_writeChar (
              unsigned char inputChar )
```

```
                                          {
    // determine letter
    const uint8_t * letter = FONT_ARRAY[inputChar];
    assert(((uint32_t) &letter[0]) != 0);

    uint16_t lineNum = lcd.lineNum;
    uint16_t colNum = lcd.colNum;

    for(uint8_t lineIdx = 0; lineIdx < HEIGHT_CHAR; lineIdx++) {
        uint8_t line = letter[HEIGHT_CHAR - 1 - lineIdx];
        for(uint8_t colIdx = 0; colIdx < LEN_CHAR; colIdx++) {
            uint8_t shiftVal = LEN_CHAR - 1 - colIdx;
            uint8_t pixel = line & (1 « shiftVal);

            uint8_t color = (pixel > 0) ? lcd.color : LCD_BLACK;
            LCD_plotSample(colNum + colIdx, lineNum + lineIdx, color);
        }
    }
    LCD_updateCursor();
    return;
}
```

LCD_writeStr()

```
void LCD_writeStr (
              void * asciiString )
```

```
                                    {
    unsigned char * str = (unsigned char *) asciiString;
    uint8_t idx = 0;
    while(str[idx] != ’\0’) {
        LCD_writeChar(str[idx]);
        idx += 1;
    }

    return;
}
```

LCD_writeInt()

```
void LCD_writeInt (
              int32_t num )
```

```
                             {
    //...
    if(num < 10) {
        LCD_writeChar(CONVERT_INT_TO_ASCII(num));
    }
    else {
        int32_t nearestPowOf10 = 10;
        while(num > (nearestPowOf10 * 10)) {
            nearestPowOf10 *= 10;
        }

        while(nearestPowOf10 > 0) {
            LCD_writeChar(CONVERT_INT_TO_ASCII(num / nearestPowOf10));
            num %= nearestPowOf10;
            nearestPowOf10 /= 10;
        }
    }
}
```

LCD_writeFloat()

```
void LCD_writeFloat (
               float num )
```

```
{
    //...
    int32_t intPart = num / (int32_t) 1;
    if(intPart < 100) {
        LCD_writeChar(' ');
    }
    LCD_writeInt(intPart);

    LCD_writeChar('.');

    int32_t decPart = (int32_t) ((num - intPart) * 10);
    LCD_writeChar(CONVERT_INT_TO_ASCII(decPart));

    return;
}
```

11.1.3.4 Variable Documentation

FONT_SPACE

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

Initial value:

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

```
{
    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
}
                                             {
          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
}
                                            {
          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
}
                                       {
          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
}
                                  {
      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
}
                                  {
      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
}
                                  {
      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
}
```

```
                                  {
      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
}
```

```
                                  {
      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
}
```

```
                                  {
      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
}
```

```
                                  {
      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
}
```

```
                                  {
      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
}
```

```
                                  {
      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
}
```

```
                                        {
    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
}
```

```
                                        {
    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
}
```

```
                                        {
    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
}
```

```
                                         {
      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
}
```

```
                                         {
      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
}
```

```
                                         {
      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
}
```

```
                                      {
    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
}
```

```
                                      {
  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
}
```

```
                                      {
    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
}
```

```
                                        {
          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
}
```

```
                                        {
          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
}
```

```
                                        {
          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
}
```

```
                                        {
     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
}
```

```
                                        {
     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
}
```

```
                                        {
     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
}
```

```
                                      {
    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
}
```

```
                                      {
    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
}
```

```
                                      {
    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
}
```

```
                                    {
      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
}
```

```
                                    {
      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
}
```

```
                                    {
      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
}
```

```
                                        {
    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
}
```

```
                                        {
    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
}
```

```
                                        {
    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
}
```

```
                                      {
      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
}
```

```
                                      {
      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
}
```

```
                                      {
      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
}
```

```
                                           {
      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
}
```

```
                                           {
      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
}
```

```
                                           {
      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
}
```

```
                                        {
    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
}
```

```
                                        {
    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
}
```

```
                                        {
    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
}
```

```
                                            {
     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
}
```

```
                                            {
     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
}
```

```
                                            {
     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
}
00591                                    {
00592     0x10,
00593     0x10,
00594     0x12,
00595     0x14,
00596     0x18,
00597     0x14,
00598     0x12,
00599     0x00
00600 };
```

FONT_LOWER_L

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

Initial value:

```
= {
    0x0C,
    0x04,
    0x04,
    0x04,
    0x04,
    0x04,
    0x0E,
    0x00
}
00602                                    {
00603     0x0C,
00604     0x04,
00605     0x04,
00606     0x04,
00607     0x04,
00608     0x04,
00609     0x0E,
00610     0x00
00611 };
```

FONT_LOWER_M

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

Initial value:

```
= {
    0x00,
    0x00,
    0x1A,
    0x15,
    0x15,
    0x11,
    0x11,
    0x00
}
00613                                    {
00614     0x00,
00615     0x00,
00616     0x1A,
00617     0x15,
00618     0x15,
00619     0x11,
00620     0x11,
00621     0x00
00622 };
```

FONT_LOWER_N

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

Initial value:

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

```
                                       {
    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
}
```

```
                                       {
    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
}
```

```
                                       {
    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
}
```

```
                                       {
     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
}
```

```
                                       {
     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
}
```

```
                                       {
     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
}
```

```
                                        {
     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
}
```

```
                                        {
     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
}
```

```
                                        {
     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
}
```

```
                                        {
      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
}
```

```
                                        {
      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
}
```

```
                                        {
      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
}
```

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

FONT_ARRAY [1/2]

```
const uint8_t* const FONT_ARRAY[128]
```

```
                                         {
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    FONT_SPACE,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    FONT_PERIOD,
    0,
    FONT_0,
    FONT_1,
    FONT_2,
    FONT_3,
    FONT_4,
    FONT_5,
    FONT_6,
    FONT_7,
    FONT_8,
    FONT_9,
    FONT_COLON,
    0,
    0,
    0,
    0,
    0,
    0,
    FONT_UPPER_A,
    FONT_UPPER_B,
    FONT_UPPER_C,
    FONT_UPPER_D,
    FONT_UPPER_E,
    FONT_UPPER_F,
    FONT_UPPER_G,
    FONT_UPPER_H,
    FONT_UPPER_I,
    FONT_UPPER_J,
    FONT_UPPER_K,
    FONT_UPPER_L,
    FONT_UPPER_M,
    FONT_UPPER_N,
    FONT_UPPER_O,
    FONT_UPPER_P,
    FONT_UPPER_Q,
    FONT_UPPER_R,
    FONT_UPPER_S,
    FONT_UPPER_T,
    FONT_UPPER_U,
    FONT_UPPER_V,
    FONT_UPPER_W,
    FONT_UPPER_X,
    FONT_UPPER_Y,
    FONT_UPPER_Z,
    0,
    0,
    0,
    0,
    0,
    0,
    FONT_LOWER_A,
    FONT_LOWER_B,
    FONT_LOWER_C,
    FONT_LOWER_D,
    FONT_LOWER_E,
    FONT_LOWER_F,
    FONT_LOWER_G,
    FONT_LOWER_H,
    FONT_LOWER_I,
    FONT_LOWER_J,
    FONT_LOWER_K,
    FONT_LOWER_L,
    FONT_LOWER_M,
    FONT_LOWER_N,
    FONT_LOWER_O,
    FONT_LOWER_P,
    FONT_LOWER_Q,
    FONT_LOWER_R,
    FONT_LOWER_S,
    FONT_LOWER_T,
    FONT_LOWER_U,
    FONT_LOWER_V,
    FONT_LOWER_W,
    FONT_LOWER_X,
    FONT_LOWER_Y,
    FONT_LOWER_Z,
    0,
    0,
    0,
    0,
    0
};
```

FONT_ARRAY [2/2]

```
const uint8_t* const FONT_ARRAY[128]  [extern]
```

```
                                    {
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
FONT_SPACE,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
FONT_PERIOD,
0,
FONT_0,
FONT_1,
FONT_2,
FONT_3,
FONT_4,
FONT_5,
FONT_6,
FONT_7,
FONT_8,
FONT_9,
FONT_COLON,
0,
0,
0,
0,
0,
0,
FONT_UPPER_A,
FONT_UPPER_B,
FONT_UPPER_C,
FONT_UPPER_D,
FONT_UPPER_E,
FONT_UPPER_F,
FONT_UPPER_G,
FONT_UPPER_H,
FONT_UPPER_I,
FONT_UPPER_J,
FONT_UPPER_K,
FONT_UPPER_L,
FONT_UPPER_M,
FONT_UPPER_N,
FONT_UPPER_O,
FONT_UPPER_P,
FONT_UPPER_Q,
FONT_UPPER_R,
FONT_UPPER_S,
FONT_UPPER_T,
FONT_UPPER_U,
    FONT_UPPER_V,
    FONT_UPPER_W,
    FONT_UPPER_X,
    FONT_UPPER_Y,
    FONT_UPPER_Z,
    0,
    0,
    0,
    0,
    0,
    0,
    FONT_LOWER_A,
    FONT_LOWER_B,
    FONT_LOWER_C,
    FONT_LOWER_D,
    FONT_LOWER_E,
    FONT_LOWER_F,
    FONT_LOWER_G,
    FONT_LOWER_H,
    FONT_LOWER_I,
    FONT_LOWER_J,
    FONT_LOWER_K,
    FONT_LOWER_L,
    FONT_LOWER_M,
    FONT_LOWER_N,
    FONT_LOWER_O,
    FONT_LOWER_P,
    FONT_LOWER_Q,
    FONT_LOWER_R,
    FONT_LOWER_S,
    FONT_LOWER_T,
    FONT_LOWER_U,
    FONT_LOWER_V,
    FONT_LOWER_W,
    FONT_LOWER_X,
    FONT_LOWER_Y,
    FONT_LOWER_Z,
    0,
    0,
    0,
    0,
    0
};
```

11.1.4 QRS Detector

Module for analyzing ECG data to determine heart rate.

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 **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 Filter Variables

- enum **DIGITAL_FILTER_PARAMS** {
  NUM_STAGES_BANDPASS = 4 , **NUM_COEFF_BANDPASS** = 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_BANDPASS]

  Coefficients of the bandpass filter in biquad (AKA second-order section, or "sos") form.
- static const float32_t COEFF_DERFILT [NUM_COEFF_DERFILT]

  Coefficients of the derivative filter, written in time-reversed order.
- static const float32_t COEFF_MOVAVG [NUM_COEFF_MOVAVG]

  Coefficients of the moving average (AKA moving-window integration) filter.
- 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

Pan-Tompkins Algorithm-specific Functions

- static uint8_t 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 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 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 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.

11.1.4.1 Detailed Description

Module for analyzing ECG data to determine heart rate.

Todo Add heart rate variability (HRV) calculation.

11.1.4.2 Enumeration Type Documentation

DIGITAL_FILTER_PARAMS

```
enum DIGITAL_FILTER_PARAMS
                             {
    // IIR Bandpass Filter
    NUM_STAGES_BANDPASS = 4,
    NUM_COEFF_BANDPASS = NUM_STAGES_BANDPASS * 5,
    STATE_BUFF_SIZE_BANDPASS = NUM_STAGES_BANDPASS * 4,

    // FIR Derivative Filter
    NUM_COEFF_DERFILT = 5,
    BLOCK_SIZE_DERFILT = (1 « 8),
    STATE_BUFF_SIZE_DERFILT = NUM_COEFF_DERFILT + BLOCK_SIZE_DERFILT - 1,

    // FIR Moving Average Filter
    NUM_COEFF_MOVAVG = 10,
    BLOCK_SIZE_MOVAVG = BLOCK_SIZE_DERFILT,
    STATE_BUFF_SIZE_MOVAVG = NUM_COEFF_MOVAVG + BLOCK_SIZE_MOVAVG - 1,
};
```

11.1.4.3 Function Documentation

findFiducialMarks()

```
static uint8_t 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.

```
                                                                                {
    uint8_t numMarks = 0;                      // running counter of peak candidates
    uint16_t countSincePrev = 1;               // samples checked since previous peak candidate
    uint16_t n_prevMark = 0;                   // sample number of previous peak candidate

    for(uint16_t n = 1; n < (QRS_NUM_SAMP - 1); n++) {
        if(IS_GREATER(yn[n], yn[n - 1]) &&
           IS_GREATER(yn[n], yn[n + 1])) {               // Verify `y[n]` is a peak
            if(countSincePrev >= 40) {
                fidMarkArray[numMarks] = n;
                numMarks += 1;

                n_prevMark = n;
                countSincePrev = 0;
            }
            else if(countSincePrev < 40) {
                if(IS_GREATER(yn[n], yn[n_prevMark])) {
                    fidMarkArray[numMarks - 1] = n;
                    n_prevMark = n;
                    countSincePrev = 0;
                }
                else {
                    countSincePrev += 1;
                }
            }
        }
        else {
            countSincePrev += 1;
        }
    }

    return numMarks;
}
```

initLevels()

```
static void 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.

```
                                                                                             {
    float32_t max;
    uint32_t maxIdx;
    arm_max_f32(yn, QRS_SAMP_FREQ * 2, &max, &maxIdx);
    *sigLvlPtr = 0.25f * max;

    float32_t mean;
    arm_mean_f32(yn, QRS_SAMP_FREQ * 2, &mean);
    *noiseLvlPtr = 0.5f * mean;

    return;
}
```

updateLevel()

```
static float32_t 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

This function updates the signal level or noise level using the amplitude of a peak that was marked as a QRS candidate via the following equations:

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

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

```
                                                                         {
    return ((0.125f * peakAmplitude) + (0.875f * level));
}
```

updateThreshold()

```
static float32_t 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

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

```
                                                                                  {
    return (noiseLevel + (0.25f * (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.

```
                         {
    return;
}
```

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.

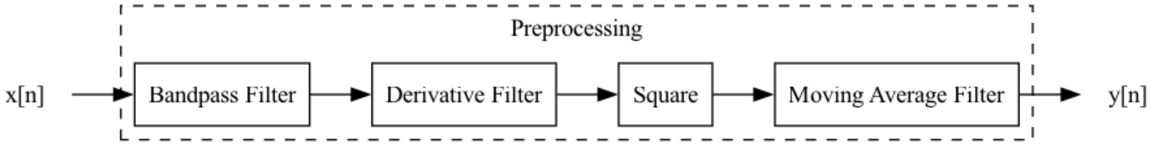
Postcondition

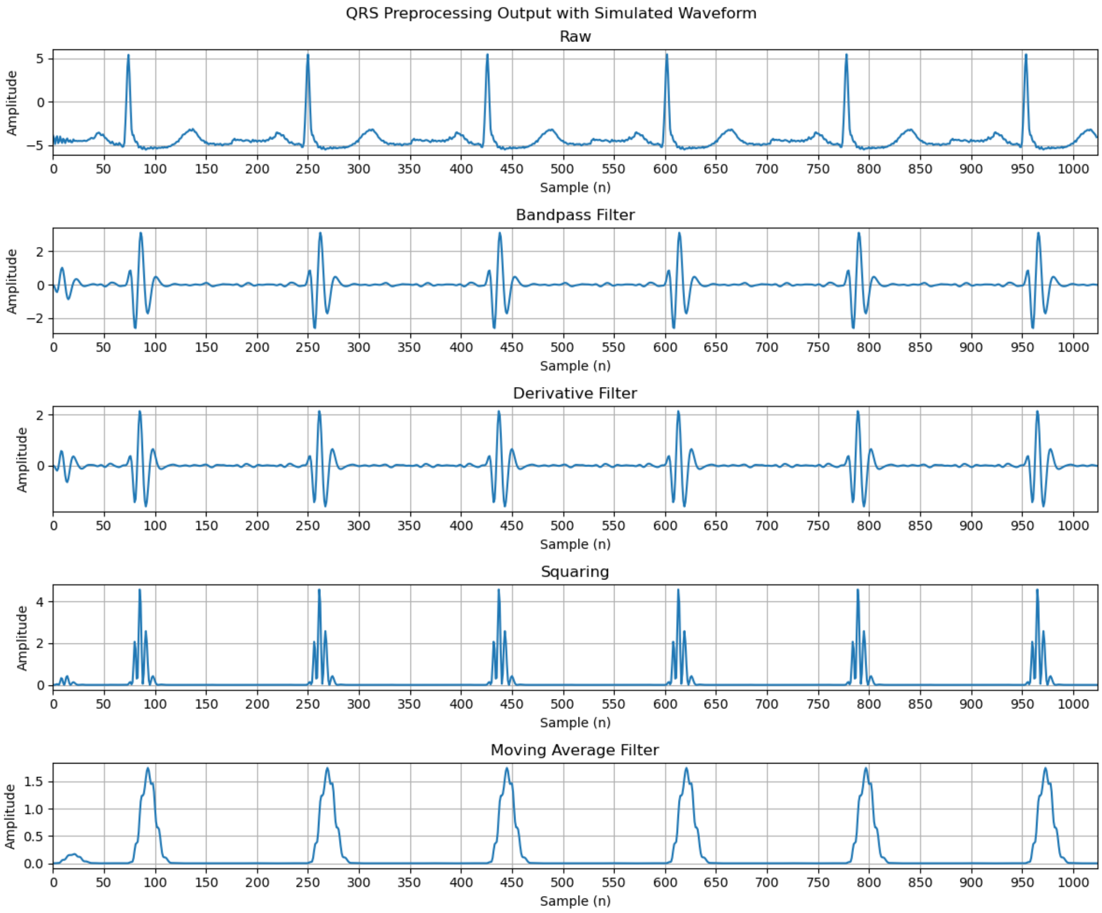
The preprocessed signal data $y[n]$ is stored in `yn` and is ready to be analyzed to calculate the heart rate in [bpm].

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.





Note

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

```
                                                            {
    // copy samples from input buffer `xn` to output buffer `yn`
    if(((uint32_t) &xn[0]) != ((uint32_t) &yn[0])) {                // skip if they're the same
        arm_copy_f32(xn, yn, QRS_NUM_SAMP);
    }

    // apply filters
    arm_biquad_cascade_df1_f32(bandpassFilter, yn, yn, QRS_NUM_SAMP);

    for(uint16_t n = 0; n < QRS_NUM_SAMP; n += BLOCK_SIZE_DERFILT) {
        arm_fir_f32(derivativeFilter, &yn[n], &yn[n], BLOCK_SIZE_DERFILT);
        arm_mult_f32(&yn[n], &yn[n], &yn[n], BLOCK_SIZE_DERFILT);                // square
        arm_fir_f32(movingAverageFilter, &yn[n], &yn[n], BLOCK_SIZE_MOVAVG);
    }

    return;
}
```

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.

Bug The current implementation processes one block of data at a time and discards the entire block immediately after. As a result, QRS complexes that are cutoff between one block and another are not being counted.

See also

QRS_Preprocess()

Todo Write implementation explanation

```
                                                       {

    // copy variables from `Detector` for readability
    float32_t signalLevel = Detector.signalLevel;
    float32_t noiseLevel = Detector.noiseLevel;
    float32_t threshold = Detector.threshold;

    uint16_t * fidMarkArray = Detector.fidMarkArray;

    float32_t * timeBuffer = Detector.utilityBuffer1;                    // time in [s] of each peak
    float32_t * heartRateBuffer = Detector.utilityBuffer2;               // HR in [BPM]

    // calibrate detector on first pass
    if(Detector.isCalibrated == false) {
        initLevels(yn, &signalLevel, &noiseLevel);
        threshold = updateThreshold(signalLevel, noiseLevel);
        Detector.isCalibrated = true;
    }

    // classify fiducial marks as signal (confirmed R peaks) or noise
    uint8_t numMarks = findFiducialMarks(yn, fidMarkArray);
    uint8_t numPeaks = 0;

    for(uint8_t idx = 0; idx < numMarks; idx++) {
        uint16_t n = fidMarkArray[idx];

        if(IS_GREATER(yn[n], threshold)) {
            timeBuffer[numPeaks] = n * QRS_SAMP_PERIOD_SEC;
            numPeaks += 1;

            signalLevel = updateLevel(yn[n], signalLevel);
        }
        else {
            noiseLevel = updateLevel(yn[n], noiseLevel);
        }

        threshold = updateThreshold(signalLevel, noiseLevel);
    }

    // store updated values in `Detector`
    Detector.signalLevel = signalLevel;
    Detector.noiseLevel = noiseLevel;
    Detector.threshold = threshold;

    // calculate RR interval and convert to HR
    for(uint8_t idx = 0; idx < (numPeaks - 1); idx++) {
        heartRateBuffer[idx] = 60.0f / (timeBuffer[idx + 1] - timeBuffer[idx]);
    }

    float32_t avgHeartRate_bpm;
    arm_mean_f32(heartRateBuffer, numPeaks, &avgHeartRate_bpm);

    return avgHeartRate_bpm;
}
```

11.1.4.4 Variable Documentation

COEFF_BANDPASS

```
const float32_t COEFF_BANDPASS[NUM_COEFF_BANDPASS]  [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,
}
```

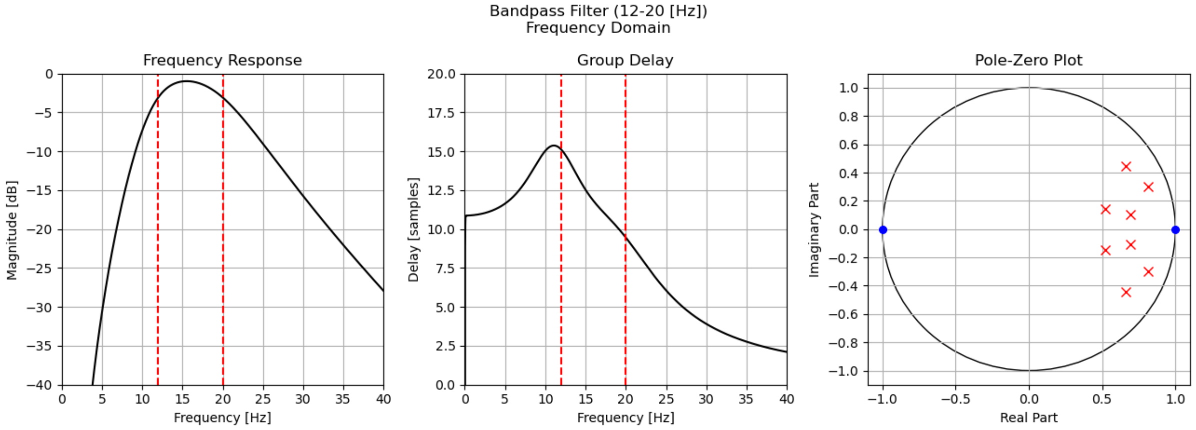
Coefficients of the bandpass filter in biquad (AKA second-order section, or "sos") form.

These coefficients were generated with the following Python code:

```
import numpy as np
from scipy import signal

fs = 200

sos_high = signal.iirfilter(N=4, Wn=12, btype='highpass', output='sos', fs=fs)
z_high, p_high, k_high = signal.sos2zpk(sos_high)

sos_low = signal.iirfilter(N=4, Wn=20, btype='lowpass', output='sos', fs=fs)
z_low, p_low, k_low = signal.sos2zpk(sos_low)

z_bpf = np.concatenate([z_high, z_low])
p_bpf = np.concatenate([p_high, p_low])
k_bpf = k_high * k_low

sos_bpf = signal.zpk2sos(z_bpf, p_bpf, k_bpf)
```

Note

> CMSIS-DSP and Scipy use different formats for biquad filters. To convert output variable `sos_bpf` to CMSIS-DSP format, the a_0 coefficients were removed from each section, and the other denominator coefficients were negated.



```
                                                            {
    // Section 1
    0.002937758108600974f, 0.005875516217201948f, 0.002937758108600974f,
    1.0485996007919312f, -0.2961403429508209f,
    // Section 2
    1.0f, 2.0f, 1.0f,
    1.3876197338104248f, -0.492422878742218f,
    // Section 3
    1.0f, -2.0f, 1.0f,
    1.3209134340286255f, -0.6327387690544128f,
    // Section 4
    1.0f, -2.0f, 1.0f,
    1.6299355030059814f, -0.7530401945114136f,
};
```

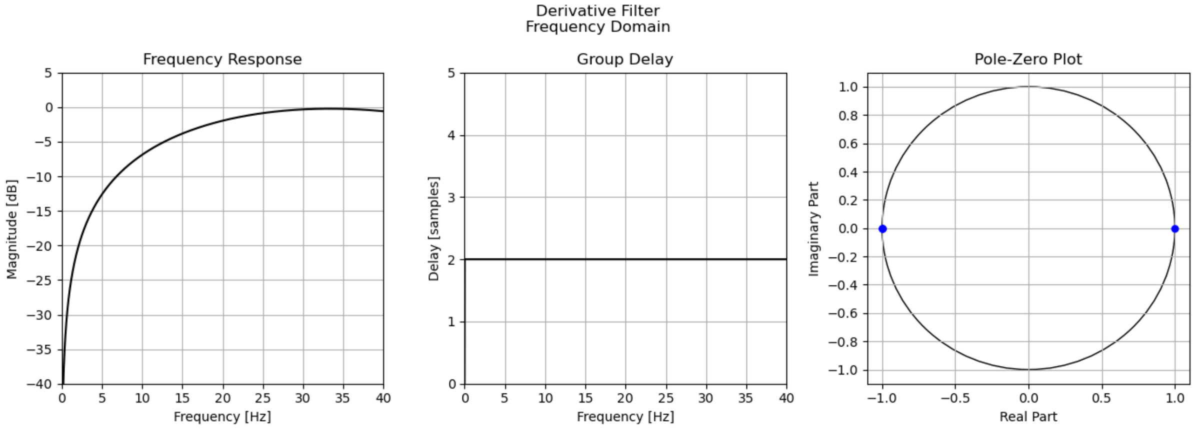
COEFF_DERFILT

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

Initial value:

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

Coefficients of the derivative filter, written in time-reversed order.



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

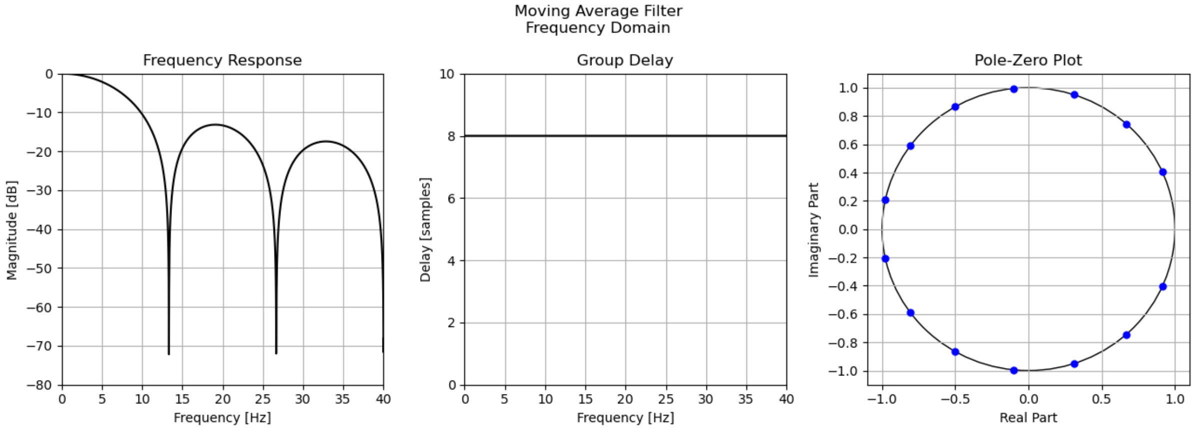
COEFF_MOVAVG

```
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
}
```

Coefficients of the moving average (AKA moving-window integration) filter.



```
                                                      {
    0.10000000149011612f, 0.10000000149011612f, 0.10000000149011612f,
    0.10000000149011612f, 0.10000000149011612f, 0.10000000149011612f,
    0.10000000149011612f, 0.10000000149011612f, 0.10000000149011612f, 0.10000000149011612f
};
```

stateBuffer_bandPass

```
float32_t stateBuffer_bandPass[STATE_BUFF_SIZE_BANDPASS] = { 0 }  [static]
{ 0 };
```

bandpassFiltStruct

```
const IIR_Filt_t bandpassFiltStruct = { NUM_STAGES_BANDPASS, stateBuffer_bandPass, COEFF_BANDPASS
}  [static]
{ NUM_STAGES_BANDPASS, stateBuffer_bandPass, COEFF_BANDPASS };
```

stateBuffer_DerFilt

```
float32_t stateBuffer_DerFilt[STATE_BUFF_SIZE_DERFILT] = { 0 }  [static]
{ 0 };
```

derivativeFiltStruct

```
const FIR_Filt_t derivativeFiltStruct = { NUM_COEFF_DERFILT, stateBuffer_DerFilt, COEFF_DERFILT
}  [static]
{ NUM_COEFF_DERFILT, stateBuffer_DerFilt, COEFF_DERFILT };
```

stateBuffer_MovingAvg

```
float32_t stateBuffer_MovingAvg[STATE_BUFF_SIZE_MOVAVG] = { 0 }  [static]
{ 0 };
```

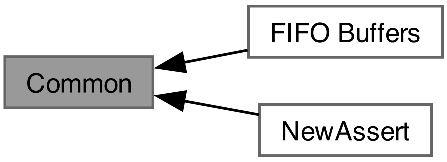
movingAvgFiltStruct

```
const FIR_Filt_t movingAvgFiltStruct = { NUM_COEFF_MOVAVG, stateBuffer_MovingAvg, COEFF_MOVAVG
}  [static]
{ NUM_COEFF_MOVAVG, stateBuffer_MovingAvg, COEFF_MOVAVG };
```

11.2 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`.

11.2.1 Detailed Description

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

11.2.2 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.

```
                            {
    if(condition) {
        return;
    }
    else {
#ifdef __arm__
        __asm__("BKPT #0");
#endif
        while(1) {}
    }
}
```

11.2.3 FIFO Buffers

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

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.

11.2.3.1 Detailed Description

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

11.2.3.2 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.

```
                                                              {
    assert(numFreeFifos > 0);

    numFreeFifos -= 1;
    volatile Fifo_t fifo = &(fifoPool[numFreeFifos]);

    fifo->buffer = buffer;
    fifo->N = N;
    fifo->frontIdx = 0;
    fifo->backIdx = 0;

    return fifo;
}
```

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->backIdx = fifo->frontIdx;
    return;
}
```

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()

```
                                                       {
    // NOTE: not using FIFO_isFull() here to reduce call stack usage
    if(((fifo->backIdx + 1) % fifo->N) != fifo->frontIdx) {
        memcpy(&fifo->buffer[fifo->backIdx], &val, sizeof(fifo->buffer[0]));
        fifo->backIdx = (fifo->backIdx + 1) % fifo->N;
    }

    return;
}
```

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()

```
                                       {
    uint32_t val;

    // NOTE: not using FIFO_isEmpty() here to reduce call stack usage
    if(fifo->frontIdx == fifo->backIdx) {
        val = 0;
    }
    else {
        memcpy(&val, &fifo->buffer[fifo->frontIdx], sizeof(fifo->buffer[0]));
        fifo->frontIdx = (fifo->frontIdx + 1) % fifo->N;
    }

    return val;
}
```

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()

```
                                                                {
    uint32_t idx = 0;

    // NOTE: not using FIFO_isEmpty() here to reduce call stack usage
    while(fifo->frontIdx != fifo->backIdx) {
        memcpy(&outputBuffer[idx], &fifo->buffer[fifo->frontIdx], sizeof(fifo->buffer[0]));
        idx += 1;
        fifo->frontIdx = (fifo->frontIdx + 1) % fifo->N;
    }

    return;
}
```

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_Put(fifo, *((uint32_t *) &val));
    return;
}
```

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.

```
                                                            {
       float val;
       *((uint32_t *) &val) = Fifo_Get(fifo);
       return val;
 }
```

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_Flush(fifo, (uint32_t *) outputBuffer);
    return;
}
```

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.

```
                                          {
    uint32_t ret_val;

    if(fifo->frontIdx == fifo->backIdx) {
        ret_val = 0;
    }
    else {
        memcpy(&ret_val, &fifo->buffer[fifo->frontIdx], sizeof(fifo->buffer[0]));
    }

    return ret_val;
}
```

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.

```
                                                              {
    uint32_t frontIdx = fifo->frontIdx;
    uint32_t idx = 0;

    while(frontIdx != fifo->backIdx) {
        memcpy(&outputBuffer[idx], &fifo->buffer[frontIdx], sizeof(fifo->buffer[0]));
        idx += 1;
        frontIdx = (frontIdx + 1) % fifo->N;               // wrap around to end
    }

    return;
}
```

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.

```
                                       {
    return (bool) (((fifo->backIdx + 1) % fifo->N) == fifo->frontIdx);
}
```

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.

```
                                        {
    return (bool) (fifo->frontIdx == fifo->backIdx);
}
```

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.

```
                                                {
    uint32_t size;

    if(fifo->frontIdx == fifo->backIdx) {                                      // empty
        size = 0;
    }
    else if(((fifo->backIdx + 1) % fifo->N) == fifo->frontIdx) {               // full
        size = fifo->N - 1;
    }
    else if(fifo->frontIdx < fifo->backIdx) {
        size = fifo->backIdx - fifo->frontIdx;
    }
    else {
        size = fifo->N - (fifo->frontIdx - fifo->backIdx);
    }

    return size;
}
```

11.2.3.3 Variable Documentation

fifoPool

`FifoStruct_t fifoPool[FIFO_POOL_SIZE] = { 0 } [static]`

pre-allocated pool

```
{ 0 };
```

11.2.4 NewAssert

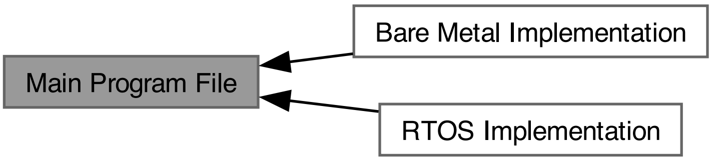
Module for using a custom `assert` implementation.

Module for using a custom `assert` implementation.

11.3 Main Program File

Files containing different implementations of the `main()` function.

Collaboration diagram for Main Program File:



Modules

- RTOS Implementation

  The project implemented with FreeRTOS.
- Bare Metal Implementation

  The project implemented on bare metal (i.e. without an operating system).

11.3.1 Detailed Description

Files containing different implementations of the `main()` function.

11.3.2 RTOS Implementation

The project implemented with FreeRTOS.

Files

- file main_rtos.c

  Main program file (RTOS implementation).

Macros

- #define Daq_Handler(void) ADC0_SS3_Handler

  ISR for the data acquisition system.
- #define **STACK_SIZE** ((UBaseType_t) 200)
- #define **DAQ_VECTOR_NUM** (INT_ADC0SS3)

Enumerations

- enum **TASK_PRIORITIES** {
  DAQ_HANDLER_PRI = 1 , **PROC_TASK_PRI** = 3 , **QRS_TASK_PRI** = 2 , **LCD_WAVEFORM_TASK_PRI** = PROC_TASK_PRI ,
  LCD_HR_TASK_PRI = QRS_TASK_PRI }
- enum QUEUE_INFO {
  QUEUE_ITEM_SIZE = sizeof(uint32_t) , DAQ_2_PROC_LEN = 3 , PROC_2_QRS_LEN = QRS_NUM_SAMP , PROC_2_LCD_LEN = DAQ_2_PROC_LEN ,
  QRS_2_LCD_LEN = 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 ProcessingTask (void ∗params)

  Task for intermediate processing of the input data.
- static void QrsDetectionTask (void ∗params)

  Task for heart rate calculation via QRS detection.
- static void LcdWaveformTask (void ∗params)

  Task for plotting the waveform on the LCD.
- static void LcdHeartRateTask (void ∗params)

  Task for outputting the heart rate to the LCD.
- int main (void)
- void vApplicationTickHook (void)

Variables

- static TaskHandle_t **ProcessingTaskHandle** = 0
- static StackType_t ProcessingStack [STACK_SIZE] = { 0 }
- static StaticTask_t ProcessingTaskBuffer = { 0 }
- static TaskHandle_t **QrsDetectionTaskHandle** = 0
- static StackType_t QrsDetectionStack [STACK_SIZE] = { 0 }
- static StaticTask_t QrsDetectionTaskBuffer = { 0 }
- static TaskHandle_t **LcdWaveformTaskHandle** = 0
- static StackType_t LcdWaveformStack [STACK_SIZE] = { 0 }
- static StaticTask_t LcdWaveformTaskBuffer = { 0 }
- static TaskHandle_t **LcdHeartRateTaskHandle** = 0
- static StackType_t LcdHeartRateStack [STACK_SIZE] = { 0 }
- static StaticTask_t LcdHeartRateTaskBuffer = { 0 }
- static volatile QueueHandle_t **Daq2ProcQueue** = 0
- static volatile StaticQueue_t Daq2ProcQueueBuffer = { 0 }
- static volatile uint8_t Daq2ProcQueueStorageArea [DAQ_2_PROC_LEN ∗QUEUE_ITEM_SIZE] = { 0 }
- static volatile QueueHandle_t **Proc2QrsQueue** = 0
- static volatile StaticQueue_t Proc2QrsQueueBuffer = { 0 }
- static volatile uint8_t Proc2QrsQueueStorageArea [PROC_2_QRS_LEN ∗QUEUE_ITEM_SIZE] = { 0 }
- static volatile QueueHandle_t **Proc2LcdQueue** = 0
- static volatile StaticQueue_t Proc2LcdQueueBuffer = { 0 }
- static volatile uint8_t Proc2LcdQueueStorageArea [PROC_2_LCD_LEN ∗QUEUE_ITEM_SIZE] = { 0 }
- static volatile QueueHandle_t **Qrs2LcdQueue** = 0
- static volatile StaticQueue_t **Qrs2LcdQueueBuffer**
- static volatile uint8_t Qrs2LcdQueueStorageArea [QRS_2_LCD_LEN ∗QUEUE_ITEM_SIZE] = { 0 }
- static float32_t qrsDetectionBuffer [QRS_NUM_SAMP] = { 0 }

    input buffer for QRS detection
- static uint16_t LCD_prevSampleBuffer [LCD_X_MAX] = { 0 }
- static volatile UBaseType_t **numTicks** = 0

11.3.2.1 Detailed Description

The project implemented with FreeRTOS.

11.3.2.2 Macro Definition Documentation

Daq_Handler

```
void Daq_Handler(
              void ) ADC0_SS3_Handler
```

ISR for the data acquisition system.

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

Precondition

Initialize the DAQ module.

Postcondition

The converted sample is placed in the Daq2ProcQueue.

The processing task is resumed.

See also

DAQ_Init(), ProcessingTask()

```
                             {
    // read sample and convert to `float32_t`
    uint16_t rawSample = DAQ_readSample();
    volatile float32_t sample = DAQ_convertToMilliVolts(rawSample);

    // send to intermediate processing task
    BaseType_t status = xQueueSendToBackFromISR(Daq2ProcQueue, &sample, NULL);
    Debug_Assert(status == pdTRUE);

    // acknowledge interrupt and unsuspend processing task
    DAQ_acknowledgeInterrupt();
    BaseType_t xYieldRequired = xTaskResumeFromISR(ProcessingTaskHandle);
    portYIELD_FROM_ISR(xYieldRequired);
}
```

11.3.2.3 Enumeration Type Documentation

TASK_PRIORITIES

```
enum TASK_PRIORITIES
                           {
    DAQ_HANDLER_PRI = 1,
    PROC_TASK_PRI = 3,
    QRS_TASK_PRI = 2,
    LCD_WAVEFORM_TASK_PRI = PROC_TASK_PRI,
    LCD_HR_TASK_PRI = QRS_TASK_PRI,
};
```

QUEUE_INFO

```
enum QUEUE_INFO
```

Enumerator

QUEUE_ITEM_SIZE	size in bytes for each queue
DAQ_2_PROC_LEN	length of DAQ-to-Processing task queue
PROC_2_QRS_LEN	length of Processing-to-QRS task queue
PROC_2_LCD_LEN	length of Processing-to-LCD task queue
QRS_2_LCD_LEN	length of QRS-to-LCD task queue

```
                      {
    QUEUE_ITEM_SIZE = sizeof(uint32_t),

    DAQ_2_PROC_LEN = 3,
    PROC_2_QRS_LEN = QRS_NUM_SAMP,
    PROC_2_LCD_LEN = DAQ_2_PROC_LEN,
    QRS_2_LCD_LEN = 1,
};
```

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

```
               {
    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
};
```

11.3.2.4 Function Documentation

ProcessingTask()

```
static void ProcessingTask (
               void * params )   [static]
```

Task for intermediate processing of the input data.

This task is triggered by the DAQ handler. It removes baseline drift and power line interference (PLI) from a sample, and then sends it to the QrsDetectionTask and LcdWaveformTask.

Postcondition

The converted sample is sent to the QrsDetectionTask.

The converted sample is sent to the LcdWaveformTask.

See also

Daq_Handler(), QrsDetectionTask(), LcdWaveformTask()

```
                                         {
    while(1) {
        static float32_t sum = 0;
        static uint32_t N = 0;

        // process sample(s) and place in queues
        while(uxQueueMessagesWaiting(Daq2ProcQueue) > 0) {
            volatile float32_t sample;
            xQueueReceive(Daq2ProcQueue, &sample, 0);

            // apply running mean subtraction to remove baseline drift
            sum += sample;
            N += 1;
            sample -= sum / ((float32_t) N);

            // apply 60 [Hz] notch filter to remove power line noise
            sample = DAQ_NotchFilter(sample);

            // place in queues
            BaseType_t status;

            status = xQueueSendToBack(Proc2QrsQueue, &sample, 0);
            Debug_Assert(status == pdTRUE);

            status = xQueueSendToBack(Proc2LcdQueue, &sample, 0);
            Debug_Assert(status == pdTRUE);
        }

        // activate next task(s) and suspend itself
        if(uxQueueSpacesAvailable(Proc2QrsQueue) == pdFALSE) {
            vTaskResume(QrsDetectionTaskHandle);
        }
        vTaskResume(LcdWaveformTaskHandle);
        vTaskSuspend(NULL);
    }
}
```

QrsDetectionTask()

```
static void QrsDetectionTask (
            void * params )  [static]
```

Task for heart rate calculation via QRS detection.

This task is triggered by the ProcessingTask. It unloads the Proc2QrsQueue within a critical section, performs QRS detection, and then sends the heart rate value to the LcdHeartRateTask.

Postcondition

The heart rate value is sent to the LcdHeartRateTask to be plotted on the display.

See also

ProcessingTask(), LcdHeartRateTask()

```
                                            {
    while(1) {
        // flush queue into QRS detection buffer
        vPortEnterCritical();
        for(uint16_t idx = 0; idx < QRS_NUM_SAMP; idx++) {
            xQueueReceive(Proc2QrsQueue, &qrsDetectionBuffer[idx], 0);
        }
        vPortExitCritical();

        // Run QRS detection
        Debug_SendMsg("Starting QRS detection...\r\n");

        QRS_Preprocess(qrsDetectionBuffer, qrsDetectionBuffer);
        float32_t heartRate_bpm = QRS_applyDecisionRules(qrsDetectionBuffer);
        Debug_Assert(isfinite(heartRate_bpm));

        // Output heart rate to serial port
        Debug_WriteFloat(heartRate_bpm);

        // Output heart rate to LCD
        xQueueSendToBack(Qrs2LcdQueue, &heartRate_bpm, 0);
        vTaskResume(LcdHeartRateTaskHandle);

        vTaskSuspend(NULL);
    }
}
```

LcdWaveformTask()

```
static void LcdWaveformTask (
            void * params )  [static]
```

Task for plotting the waveform on the LCD.

This task is triggered by the ProcessingTask. It applies a 0.5-40 [Hz] bandpass filter to the sample and plots it.

Precondition

Initialize the LCD module.

Postcondition

The bandpass-filtered sample is plotted to the LCD.

See also

LCD_Init(), ProcessingTask()

```
                                            {
    while(1) {
        static uint16_t x = 0;
        static const float32_t maxVal = DAQ_LOOKUP_MAX * 2;

        while(uxQueueMessagesWaiting(Proc2LcdQueue) > 0) {
            float32_t sample;
            xQueueReceive(Proc2LcdQueue, &sample, 0);
            sample = DAQ_BandpassFilter(sample);

            // remove previous y-value from LCD
            uint16_t y = LCD_prevSampleBuffer[x];
            LCD_plotSample(x, y, LCD_BLACK);

            // shift/scale `sample` from (est.) range [-11, 11) to [LCD_WAVE_Y_MIN, LCD_WAVE_Y_MAX)
            y = LCD_WAVE_Y_MIN + ((uint16_t) (((sample + maxVal) / (maxVal * 2)) * LCD_WAVE_Y_MAX));
            LCD_plotSample(x, y, LCD_RED);

            // store y-value and update x
            LCD_prevSampleBuffer[x] = y;
            x = (x + 1) % LCD_X_MAX;
        }

        vTaskSuspend(NULL);
    }
}
```

LcdHeartRateTask()

```
static void LcdHeartRateTask (
               void * params )  [static]
```

Task for outputting the heart rate to the LCD.

This task is triggered by the QrsDetectionTask. It outputs the heart rate.

Precondition

Initialize the LCD module.

Postcondition

The heart rate is updated after each block is analyzed.

See also

LCD_Init(), QrsDetectionTask()

```
                                             {
    while(1) {
        volatile float32_t heartRate_bpm;
        xQueueReceive(Qrs2LcdQueue, &heartRate_bpm, 0);

        LCD_setCursor(LCD_TEXT_LINE_NUM, LCD_TEXT_COL_NUM);
        LCD_writeFloat(heartRate_bpm);

        vTaskSuspend(NULL);
    }
}
```

main()

```
int main (
              void  )
```

```
          {
    static GpioPort_t portA = 0;
    static Uart_t uart0 = 0;

    PLL_Init();

    // Init. debug module
    portA = GPIO_InitPort(GPIO_PORT_A);
    uart0 = UART_Init(portA, UART0);
    Debug_Init(uart0);

    // Init./config. LCD
    LCD_Init();
    LCD_setOutputMode(false);

    LCD_setColor(LCD_WHITE);
    LCD_drawHoriLine(LCD_TOP_LINE, 1);

    LCD_setColor(LCD_RED);
    LCD_setCursor(LCD_TEXT_LINE_NUM, 0);
    LCD_writeStr("Heart Rate:      bpm");

    LCD_setOutputMode(true);

    Debug_SendFromList(DEBUG_LCD_INIT);

    // Init. other app. modules
    QRS_Init();
    Debug_SendFromList(DEBUG_QRS_INIT);

    DAQ_Init();
    Debug_SendFromList(DEBUG_DAQ_INIT);

    // Init. DAQ ISR
    ISR_GlobalDisable();
    ISR_setPriority(DAQ_VECTOR_NUM, DAQ_HANDLER_PRI);
    ISR_Enable(DAQ_VECTOR_NUM);
    ISR_GlobalEnable();

    // Init. queues and add them to registry for debugging
    Daq2ProcQueue = xQueueCreateStatic(DAQ_2_PROC_LEN, QUEUE_ITEM_SIZE, Daq2ProcQueueStorageArea,
                                       &Daq2ProcQueueBuffer);
    Proc2QrsQueue = xQueueCreateStatic(PROC_2_QRS_LEN, QUEUE_ITEM_SIZE, Proc2QrsQueueStorageArea,
                                       &Proc2QrsQueueBuffer);
    Proc2LcdQueue = xQueueCreateStatic(PROC_2_LCD_LEN, QUEUE_ITEM_SIZE, Proc2LcdQueueStorageArea,
                                       &Proc2LcdQueueBuffer);
    Qrs2LcdQueue = xQueueCreateStatic(QRS_2_LCD_LEN, QUEUE_ITEM_SIZE, Qrs2LcdQueueStorageArea,
                                      &Qrs2LcdQueueBuffer);

    // Init. tasks and start scheduler
    ProcessingTaskHandle =
        xTaskCreateStatic(ProcessingTask, "Intermediate Processing", STACK_SIZE, NULL,
                          PROC_TASK_PRI, ProcessingStack, &ProcessingTaskBuffer);
    vTaskSuspend(ProcessingTaskHandle);

    QrsDetectionTaskHandle =
        xTaskCreateStatic(QrsDetectionTask, "QRS Detection", STACK_SIZE, NULL, QRS_TASK_PRI,
                          QrsDetectionStack, &QrsDetectionTaskBuffer);
    vTaskSuspend(QrsDetectionTaskHandle);

    LcdWaveformTaskHandle =
        xTaskCreateStatic(LcdWaveformTask, "LCD (Waveform)", STACK_SIZE, NULL,
                          LCD_WAVEFORM_TASK_PRI, LcdWaveformStack, &LcdWaveformTaskBuffer);
    vTaskSuspend(LcdWaveformTaskHandle);

    LcdHeartRateTaskHandle =
        xTaskCreateStatic(LcdHeartRateTask, "LCD (Heart Rate)", STACK_SIZE, NULL, LCD_HR_TASK_PRI,
                          LcdHeartRateStack, &LcdHeartRateTaskBuffer);
    vTaskSuspend(LcdHeartRateTaskHandle);

    vTaskStartScheduler();
    while(1) {}
}
```

vApplicationTickHook()

```
void vApplicationTickHook (
              void  )
```

```
                        {
    numTicks += 1;
}
```

11.3.2.5 Variable Documentation

ProcessingStack

```
StackType_t ProcessingStack[STACK_SIZE] = { 0 }  [static]
{ 0 };
```

ProcessingTaskBuffer

```
StaticTask_t ProcessingTaskBuffer = { 0 }  [static]
{ 0 };
```

QrsDetectionStack

```
StackType_t QrsDetectionStack[STACK_SIZE] = { 0 }  [static]
{ 0 };
```

QrsDetectionTaskBuffer

```
StaticTask_t QrsDetectionTaskBuffer = { 0 }  [static]
{ 0 };
```

LcdWaveformStack

```
StackType_t LcdWaveformStack[STACK_SIZE] = { 0 }  [static]
{ 0 };
```

LcdWaveformTaskBuffer

```
StaticTask_t LcdWaveformTaskBuffer = { 0 }  [static]
{ 0 };
```

LcdHeartRateStack

```
StackType_t LcdHeartRateStack[STACK_SIZE] = { 0 }  [static]
{ 0 };
```

LcdHeartRateTaskBuffer

```
StaticTask_t LcdHeartRateTaskBuffer = { 0 }  [static]
{ 0 };
```

Daq2ProcQueueBuffer

```
volatile StaticQueue_t Daq2ProcQueueBuffer = { 0 }  [static]
00164 { 0 };
```

Daq2ProcQueueStorageArea

```
volatile uint8_t Daq2ProcQueueStorageArea[DAQ_2_PROC_LEN *QUEUE_ITEM_SIZE] = { 0 }  [static]
00165 { 0 };
```

Proc2QrsQueueBuffer

```
volatile StaticQueue_t Proc2QrsQueueBuffer = { 0 }  [static]
00168 { 0 };
```

Proc2QrsQueueStorageArea

```
volatile uint8_t Proc2QrsQueueStorageArea[PROC_2_QRS_LEN *QUEUE_ITEM_SIZE] = { 0 }  [static]
00169 { 0 };
```

Proc2LcdQueueBuffer

```
volatile StaticQueue_t Proc2LcdQueueBuffer = { 0 }  [static]
00172 { 0 };
```

Proc2LcdQueueStorageArea

```
volatile uint8_t Proc2LcdQueueStorageArea[PROC_2_LCD_LEN *QUEUE_ITEM_SIZE] = { 0 }  [static]
00173 { 0 };
```

Qrs2LcdQueueStorageArea

```
volatile uint8_t Qrs2LcdQueueStorageArea[QRS_2_LCD_LEN *QUEUE_ITEM_SIZE] = { 0 }  [static]
00177 { 0 };
```

qrsDetectionBuffer

```
float32_t qrsDetectionBuffer[QRS_NUM_SAMP] = { 0 }  [static]
```

input buffer for QRS detection

```
00184 { 0 };
```

LCD_prevSampleBuffer

```
uint16_t LCD_prevSampleBuffer[LCD_X_MAX] = { 0 }  [static]
00198 { 0 };
```

11.3.3 Bare Metal Implementation

The project implemented on bare metal (i.e. without an operating system).

Files

- file main.c

  Main program file (bare-metal implementation).

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 }

11.3.3.1 Detailed Description

The project implemented on bare metal (i.e. without an operating system).

11.3.3.2 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()

```
                     {
    DAQ_VECTOR_NUM = INT_ADC0SS3,
    PROC_VECTOR_NUM = INT_CAN0,
    LCD_VECTOR_NUM = INT_TIMER1A
};
```

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

```
               {
    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
};
```

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

```
              {
    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
};
```

11.3.3.3 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()

```
                              {
    // read sample and convert to `float32_t`
    uint16_t rawSample = DAQ_readSample();
    volatile float32_t sample = DAQ_convertToMilliVolts(rawSample);

    // send to intermediate processing handler
    Debug_Assert(Fifo_isFull(DAQ_Fifo) == false);
    Fifo_PutFloat(DAQ_Fifo, sample);
    ISR_triggerInterrupt(PROC_VECTOR_NUM);

    DAQ_acknowledgeInterrupt();
}
```

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()

```
                                     {
    static float32_t sum = 0;
    static uint32_t N = 0;

    // NOTE: this `while` is only here in case a sample arrives while the QRS FIFO is being emptied
    while(Fifo_isEmpty(DAQ_Fifo) == false) {
        volatile float32_t sample = Fifo_GetFloat(DAQ_Fifo);

        // apply running mean subtraction to remove baseline drift
        sum += sample;
        N += 1;
        sample -= sum / ((float32_t) N);

        // apply 60 [Hz] notch filter to remove power line noise
        sample = DAQ_NotchFilter(sample);

        // place in FIFO buffers
        Debug_Assert(Fifo_isFull(QRS_Fifo) == false);
        Fifo_PutFloat(QRS_Fifo, sample);

        Debug_Assert(Fifo_isFull(LCD_Fifo1) == false);
        Fifo_PutFloat(LCD_Fifo1, sample);
    }

    if(Fifo_isFull(QRS_Fifo)) {
        qrsBufferIsFull = true;
    }
    else {
        // doesn't trigger if QRS detection is ready to start
        ISR_triggerInterrupt(LCD_VECTOR_NUM);
    }
}
```

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()

```
                              {
    static uint16_t x = 0;
    static const float32_t maxVal = DAQ_LOOKUP_MAX * 2;

    Debug_Assert(Fifo_isEmpty(LCD_Fifo1) == false);

    // NOTE: this `while` is only here in case a sample arrives while the QRS FIFO is being emptied
    while(Fifo_isEmpty(LCD_Fifo1) == false) {
        // get sample and apply 0.5-40 [Hz] bandpass filter
        float32_t sample = Fifo_GetFloat(LCD_Fifo1);
        sample = DAQ_BandpassFilter(sample);

        // remove previous y-value from LCD
        uint16_t y = LCD_prevSampleBuffer[x];
        LCD_plotSample(x, y, LCD_BLACK);

        // shift/scale `sample` from (est.) range [-11, 11) to [LCD_WAVE_Y_MIN, LCD_WAVE_Y_MAX)
        y = LCD_WAVE_Y_MIN + ((uint16_t) (((sample + maxVal) / (maxVal * 2)) * LCD_WAVE_Y_MAX));
        LCD_plotSample(x, y, LCD_RED);

        // store y-value and update x
        LCD_prevSampleBuffer[x] = y;
        x = (x + 1) % LCD_X_MAX;
    }

    if(heartRateIsReady) {
        volatile float32_t heartRate_bpm = Fifo_GetFloat(LCD_Fifo2);

        LCD_setCursor(LCD_TEXT_LINE_NUM, LCD_TEXT_COL_NUM);
        LCD_writeFloat(heartRate_bpm);

        heartRateIsReady = false;
    }
}
```

main()

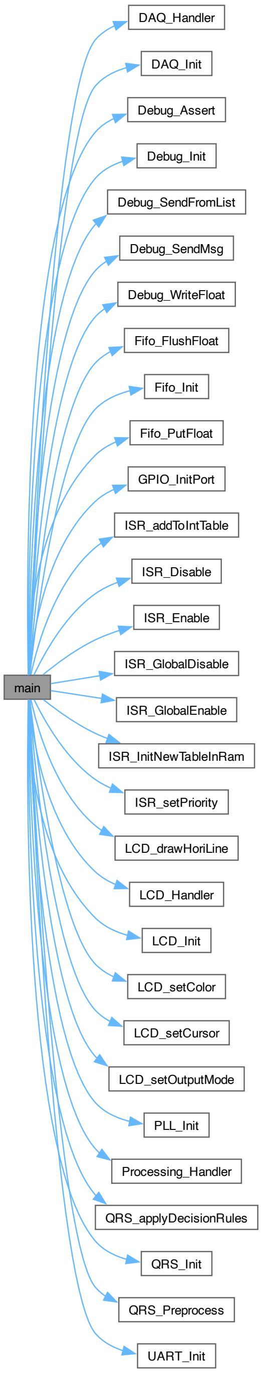
```
int main (
              void   )
```

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.

```
               {
    static GpioPort_t portA = 0;
    static Uart_t uart0 = 0;

    PLL_Init();

    // Init. debug module
    portA = GPIO_InitPort(GPIO_PORT_A);
    uart0 = UART_Init(portA, UART0);
    Debug_Init(uart0);

    // Init. vector table and ISRs
    ISR_GlobalDisable();
    ISR_InitNewTableInRam();

    ISR_addToIntTable(DAQ_Handler, DAQ_VECTOR_NUM);
    ISR_setPriority(DAQ_VECTOR_NUM, 1);
    ISR_Enable(DAQ_VECTOR_NUM);

    ISR_addToIntTable(Processing_Handler, PROC_VECTOR_NUM);
    ISR_setPriority(PROC_VECTOR_NUM, 1);
    ISR_Enable(PROC_VECTOR_NUM);
    ISR_addToIntTable(LCD_Handler, LCD_VECTOR_NUM);
    ISR_setPriority(LCD_VECTOR_NUM, 1);
    ISR_Enable(LCD_VECTOR_NUM);

    // Init. FIFOs
    DAQ_Fifo = Fifo_Init(DAQ_fifoBuffer, DAQ_ARRAY_LEN);
    QRS_Fifo = Fifo_Init(QRS_fifoBuffer, QRS_ARRAY_LEN);
    LCD_Fifo1 = Fifo_Init(LCD_fifoBuffer1, LCD_ARRAY_1_LEN);
    LCD_Fifo2 = Fifo_Init(LCD_fifoBuffer2, LCD_ARRAY_2_LEN);

    // Init./config. LCD
    LCD_Init();
    LCD_setOutputMode(false);

    LCD_setColor(LCD_WHITE);
    LCD_drawHoriLine(LCD_TOP_LINE, 1);

    LCD_setColor(LCD_RED);
    LCD_setCursor(LCD_TEXT_LINE_NUM, 0);
    LCD_writeStr("Heart Rate:      bpm");

    LCD_setOutputMode(true);

    Debug_SendFromList(DEBUG_LCD_INIT);

    // Init. other app. modules
    QRS_Init();
    Debug_SendFromList(DEBUG_QRS_INIT);

    DAQ_Init();
    Debug_SendFromList(DEBUG_DAQ_INIT);

    // Enable interrupts and start
    ISR_GlobalEnable();
    while(1) {
        if(qrsBufferIsFull) {               // flag set by Processing_Handler()
            // Transfer samples from FIFO
            ISR_Disable(PROC_VECTOR_NUM);

            Fifo_FlushFloat(QRS_Fifo, QRS_processingBuffer);
            qrsBufferIsFull = false;

            ISR_Enable(PROC_VECTOR_NUM);

            // Run QRS detection
            Debug_SendMsg("Starting QRS detection...\r\n");

            QRS_Preprocess(QRS_processingBuffer, QRS_processingBuffer);
            float32_t heartRate_bpm = QRS_applyDecisionRules(QRS_processingBuffer);
            Debug_Assert(isfinite(heartRate_bpm));

            // Output heart rate to serial port
            Debug_WriteFloat(heartRate_bpm);

            // Output heart rate to LCD
            Fifo_PutFloat(LCD_Fifo2, heartRate_bpm);
            heartRateIsReady = true;
        }
    }
}
```

Here is the call graph for this function:



11.3.3.4 Variable Documentation

DAQ_fifoBuffer

```
volatile uint32_t DAQ_fifoBuffer[DAQ_ARRAY_LEN] = { 0 }  [static]
00127 { 0 };
```

QRS_fifoBuffer

```
volatile uint32_t QRS_fifoBuffer[QRS_ARRAY_LEN] = { 0 }  [static]
00130 { 0 };
```

LCD_fifoBuffer1

```
volatile uint32_t LCD_fifoBuffer1[LCD_ARRAY_1_LEN] = { 0 }  [static]
00133 { 0 };
```

LCD_fifoBuffer2

```
volatile uint32_t LCD_fifoBuffer2[LCD_ARRAY_2_LEN] = { 0 }  [static]
00136 { 0 };
```

QRS_processingBuffer

```
float32_t QRS_processingBuffer[QRS_ARRAY_LEN] = { 0 }  [static]
00142 { 0 };
```

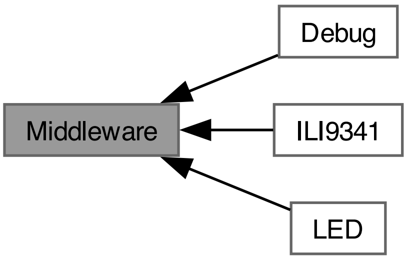
LCD_prevSampleBuffer

```
uint16_t LCD_prevSampleBuffer[LCD_X_MAX] = { 0 }  [static]
00156 { 0 };
```

11.4 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).

11.4.1 Detailed Description

High-level device driver modules.

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

11.4.2 Debug

Module for debugging functions, including serial output and assertions.

Files

- file Debug.c

  Source code for Debug module.
- file Debug.h

  Header file for Debug module.

Variables

- static Uart_t **debugUart** = 0

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.*

11.4.2.1 Detailed Description

Module for debugging functions, including serial output and assertions.

11.4.2.2 Enumeration Type Documentation

Msg_t

```
enum Msg_t
                {
    DEBUG_DAQ_INIT,
    DEBUG_QRS_INIT,
    DEBUG_LCD_INIT,
    DEBUG_QRS_START
} Msg_t;
```

11.4.2.3 Function Documentation

Debug_Init()

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

Initialize the Debug module.

Precondition

Initialize the UART.

Parameters

in	*uart*	UART to use for serial output.

Postcondition

An initialization message is sent to the serial port.

See also

UART_Init()

```
                              {
    assert(UART_isInit(uart));

    debugUart = uart;

    Debug_SendMsg((void *) "Starting transmission...\r\n");
    Debug_SendMsg((void *) "Debug module initialized.\r\n");
    return;
}
```

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()

```
                              {
    UART_WriteStr(debugUart, message);
    return;
}
```

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()

```
                                  {
    switch(msg) {
        case DEBUG_DAQ_INIT:
            Debug_SendMsg("Data acquisition module initialized.\r\n");
            break;
        case DEBUG_QRS_INIT:
            Debug_SendMsg("QRS detection module initialized.\r\n");
            break;
        case DEBUG_LCD_INIT:
            Debug_SendMsg("LCD module initialized.\r\n");
            break;
        case DEBUG_QRS_START:
            Debug_SendMsg("Starting QRS detection...\r\n");
            break;
        default:
            assert(false);
    }
    return;
}
```

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()

```
                                   {
    UART_WriteFloat(debugUart, value, 1);
    UART_WriteStr(debugUart, "\r\n");
    return;
}
```

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.

```
                                   {
    if(condition == false) {
        Debug_SendMsg((void *) "Assertion failed. Entering infinite loop.\r\n.");
        assert(false);
    }
    return;
}
```

11.4.3 ILI9341

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

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 (GpioPort_t resetPinPort, GpioPin_t resetPin, Spi_t spi, Timer_t timer)

  Initialize the LCD driver.
- 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**
  volatile uint32_t ∗ **resetPinDataRegister**
  GpioPin_t **resetPin**
  Spi_t **spi**
  bool **isInit**
  } ili9341

11.4.3.1 Detailed Description

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

11.4.3.2 Enumeration Type Documentation

anonymous enum

```
anonymous enum
```

Enumerator

ILI9341_NUM_COLS	**11.4.3.3 of columns available on the display**
ILI9341_NUM_ROWS	**11.4.3.4 of rows available on the display**

```
00039     {
00040     ILI9341_NUM_COLS = 240,
00041     ILI9341_NUM_ROWS = 320
00042 };
```

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.

Enumerator

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.

```
              {
    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,
} Cmd_t;
```

sleepMode_t

```
enum sleepMode_t
              {
    SLEEP_ON = SPLIN,
    SLEEP_OFF = SPLOUT
} sleepMode_t;
```

displayArea_t

```
enum displayArea_t
              {
    NORMAL_AREA = NORON,
    PARTIAL_AREA = PTLON
} displayArea_t;
```

colorExpr_t

```
enum colorExpr_t
                {
    FULL_COLORS = IDMOFF,
    PARTIAL_COLORS = IDMON
} colorExpr_t;
```

invertMode_t

```
enum invertMode_t
                {
    INVERT_ON = DINVON,
    INVERT_OFF = DINVOFF
} invertMode_t;
```

outputMode_t

```
enum outputMode_t
                {
    OUTPUT_ON = DISPON,
    OUTPUT_OFF = DISPOFF
} outputMode_t;
```

colorDepth_t

```
enum colorDepth_t
                {
    COLORDEPTH_16BIT = 0x55,
    COLORDEPTH_18BIT = 0x66,
} colorDepth_t;
```

11.4.3.5 Function Documentation

ILI9341_setMode()

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

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

```
                                        {
    switch(param) {
        case(SLEEP_ON):
        case(SLEEP_OFF):
            SPI_WriteCmd(ili9341.spi, param);
            ili9341.sleepMode = param;
            break;
        case(NORMAL_AREA):
        case(PARTIAL_AREA):
            SPI_WriteCmd(ili9341.spi, param);
            ili9341.displayArea = param;
            break;
        case(FULL_COLORS):
        case(PARTIAL_COLORS):
            SPI_WriteCmd(ili9341.spi, param);
            ili9341.colorExpression = param;
            break;
        case(INVERT_OFF):
        case(INVERT_ON):
            SPI_WriteCmd(ili9341.spi, param);
            ili9341.invertMode = param;
            break;
        case(OUTPUT_OFF):
        case(OUTPUT_ON):
            SPI_WriteCmd(ili9341.spi, param);
            ili9341.outputMode = param;
            break;
        case(COLORDEPTH_16BIT):
        case(COLORDEPTH_18BIT):
            SPI_WriteCmd(ili9341.spi, PIXSET);
            SPI_WriteData(ili9341.spi, param);
            break;
        default:
            assert(false);
            break;
    }

    return;
}
```

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.

```
                                                                                        {
    uint8_t cmd = (is_row) ? PASET : CASET;
    uint16_t max_num = (is_row) ? ILI9341_NUM_ROWS : ILI9341_NUM_COLS;

    // ensure `startAddress` and `endAddress` meet restrictions
    assert(endAddress < max_num);
    assert(startAddress <= endAddress);

    // configure and send command sequence
    Fifo_Put(ILI9341_Fifo, ((startAddress & 0xFF00) » 8));
    Fifo_Put(ILI9341_Fifo, (startAddress & 0x00FF));
    Fifo_Put(ILI9341_Fifo, ((endAddress & 0xFF00) » 8));
    Fifo_Put(ILI9341_Fifo, (endAddress & 0x00FF));

    ILI9341_sendParams(cmd);

    return;
}
```

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.

```
                                          {
    if(cmd != NOP) {
        SPI_WriteCmd(ili9341.spi, cmd);
    }

    uint8_t numParams = Fifo_getCurrSize(ILI9341_Fifo);
    while(numParams > 0) {
        uint8_t data = Fifo_Get(ILI9341_Fifo);
        SPI_WriteData(ili9341.spi, data);

        numParams -= 1;
    }

    return;
}
```

ILI9341_Init()

```
void ILI9341_Init (
            GpioPort_t resetPinPort,
            GpioPin_t resetPin,
            Spi_t spi,
            Timer_t timer )
```

Initialize the LCD driver.

Precondition

Initialize the GPIO port.

Initialize the SPI module.

Initialize the Timer.

Parameters

in	*resetPinPort*	The GPIO port that the `RESET` pin belongs to.
in	*resetPin*	The GPIO pin used as the `RESET` pin.
in	*spi*	The SPI module to use for communication.
in	*timer*	The hardware timer to use during initialization.

Postcondition

The `RESET` is configured as a digital `OUTPUT` pin.

The SPI is configured and enabled.

The LCD driver is initialized and ready to accept commands.

See also

GPIO_InitPort(), SPI_Init(), Timer_Init()

```
                                                                          {
    assert(ili9341.isInit == false);               // should only be initialized once
    assert(GPIO_isPortInit(resetPinPort));
    assert(SPI_isInit(spi));
    assert(Timer_isInit(timer));

    ILI9341_Fifo = Fifo_Init(ILI9341_Buffer, 8);

    GPIO_DisableDigital(resetPinPort, resetPin);
    GPIO_configDirection(resetPinPort, resetPin, GPIO_OUTPUT);
    GPIO_EnableDigital(resetPinPort, resetPin);
    ili9341.resetPinDataRegister = GPIO_getDataRegister(resetPinPort);
    ili9341.resetPin = resetPin;

    SPI_Disable(spi);
    SPI_configClock(spi, SPI_RISING_EDGE, SPI_STEADY_STATE_LOW);
    SPI_setDataSize(spi, 8);
    SPI_Enable(spi);
    ili9341.spi = spi;

    ILI9341_resetHard(timer);
    ILI9341_setInterface();
    ili9341.isInit = true;
    return;
}
```

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.

```
                                  {
    SPI_WriteCmd(ili9341.spi, IFCTL);
    SPI_WriteData(ili9341.spi, 0);
    SPI_WriteData(ili9341.spi, 0);
    SPI_WriteData(ili9341.spi, 0);
    return;
}
```

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.

```
                                              {
     assert(ili9341.resetPinDataRegister != 0);
     assert(Timer_isInit(timer));
     Timer_setMode(timer, ONESHOT, UP);

     *ili9341.resetPinDataRegister &= ~(ili9341.resetPin);
     Timer_Wait1ms(timer, 1);
     *ili9341.resetPinDataRegister |= ili9341.resetPin;
     Timer_Wait1ms(timer, 5);
     return;
 }
```

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

```
                                              {
     assert(Timer_isInit(timer));
     Timer_setMode(timer, ONESHOT, UP);

     SPI_WriteCmd(ili9341.spi, SWRESET);
     Timer_Wait1ms(timer, 5);
     return;
 }
```

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.

```
                                                        {
    assert(ili9341.isInit);
    ILI9341_setMode(sleepMode);

    Timer_setMode(timer, ONESHOT, UP);
    Timer_Wait1ms(timer, 120);

    return;
}
```

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()

```
                                                 {
    assert(ili9341.isInit);
    ILI9341_setMode(displayArea);

    return;
}
```

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.

```
                                                 {
    assert(ili9341.isInit);
    ILI9341_setMode(colorExpr);

    return;
}
```

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()

```
                                                                {
    // ensure `rowStart` and `rowEnd` meet restrictions.
    rowEnd = (rowEnd > 0) ? rowEnd : 1;
    rowEnd = (rowEnd < ILI9341_NUM_ROWS) ? rowEnd : (ILI9341_NUM_ROWS - 1);
    rowStart = (rowStart > 0) ? rowStart : 1;
    rowStart = (rowStart < rowEnd) ? rowStart : rowEnd;

    // configure and send command sequence
    Fifo_Put(ILI9341_Fifo, ((rowStart & 0xFF00) » 8));
    Fifo_Put(ILI9341_Fifo, (rowStart & 0x00FF));
    Fifo_Put(ILI9341_Fifo, ((rowEnd & 0xFF00) » 8));
    Fifo_Put(ILI9341_Fifo, (rowEnd & 0x00FF));
    ILI9341_sendParams(PLTAR);

    return;
}
```

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)

```
                                                        {
    assert(ili9341.isInit);
    ILI9341_setMode(invertMode);

    return;
}
```

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

```
                                              {
    assert(ili9341.isInit);
    ILI9341_setMode(outputMode);

    return;
}
```

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

Name	Bit #	Effect when set = 1
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`.

```
                                                              {
    uint8_t param = 0x00;
    param = (areRowsFlipped) ? (param | 0x80) : param;
    param = (areColsFlipped) ? (param | 0x40) : param;
    param = (areRowsColsSwitched) ? (param | 0x20) : param;
    param = (isVertRefreshFlipped) ? (param | 0x10) : param;
    param = (isColorOrderFlipped) ? (param | 0x08) : param;
    param = (isHorRefreshFlipped) ? (param | 0x04) : param;

    SPI_WriteCmd(ili9341.spi, MADCTL);
    SPI_WriteData(ili9341.spi, param);
    return;
}
```

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.

```
                                                          {
    assert(ili9341.isInit);
    ILI9341_setMode(colorDepth);

    return;
}
```

ILI9341_setFrameRate()

```
void ILI9341_setFrameRate (
              uint8_t divisionRatio,
              uint8_t clocksPerLine )
```

TODO: Write brief.

TODO: Write description

```
                                                                                  {

    Cmd_t cmd;
    if(ili9341.colorExpression == PARTIAL_COLORS) {
        cmd = FRMCTR2;
    }
    else {
        cmd = (ili9341.displayArea == NORMAL_AREA) ? FRMCTR1 : FRMCTR3;
    }

    SPI_WriteCmd(ili9341.spi, (uint8_t) cmd);
    SPI_WriteData(ili9341.spi, divisionRatio & 0x03);
    SPI_WriteData(ili9341.spi, clocksPerLine & 0x1F);
    return;
}
```

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_setAddress(startRow, endRow, true);
    return;
}
```

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_setAddress(startCol, endCol, false);
    return;
}
```

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()

```
                                {
    SPI_WriteCmd(ili9341.spi, RAMWR);
    return;
}
```

ILI9341_writePixel()

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

Write a single pixel to frame memory.

Precondition

Set 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	...

```
                                                             {
    // clang-format off
    // clang-format on

    if(ili9341.colorDepth == COLORDEPTH_16BIT) {
        Fifo_Put(ILI9341_Fifo, ((red & 0x1F) « 3) | ((green & 0x38) » 3));
        Fifo_Put(ILI9341_Fifo, ((green & 0x07) « 5) | (blue & 0x1F));
    }
    else {
        // bits 1 and 0 are set to prevent the TM4C from
        // attempting to right-justify the RGB data
        Fifo_Put(ILI9341_Fifo, ((red & 0x3F) « 2) + 0x03);
        Fifo_Put(ILI9341_Fifo, ((green & 0x3F) « 2) + 0x03);
        Fifo_Put(ILI9341_Fifo, ((blue & 0x3F) « 2) + 0x03);
    }

    ILI9341_sendParams(NOP);

    return;
}
```

11.4.3.6 Variable Documentation

[struct]

```
struct { ...  } ili9341  [static]
```

Initial value:

```
= { SLEEP_ON,         NORMAL_AREA, FULL_COLORS, INVERT_OFF, OUTPUT_ON,
              COLORDEPTH_16BIT, 0,           0,           0,          false }
```

11.4.4 LED

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

Files

- file Led.c

  Source code for LED module.
- file Led.h

  Interface for LED module.

Data Structures

- struct Led_t

Macros

- #define **LED_POOL_SIZE** 3

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.

11.4.4.1 Detailed Description

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

11.4.4.2 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.

```
                                                   {
    assert(GPIO_isPortInit(gpioPort));
    assert(num_free_leds > 0);

    // Initialize GPIO port pin
    GPIO_configDirection(gpioPort, pin, GPIO_OUTPUT);
    GPIO_configResistor(gpioPort, pin, PULLDOWN);

    GPIO_EnableDigital(gpioPort, pin);
    GPIO_WriteLow(gpioPort, pin);

    // Initialize LED struct
    num_free_leds -= 1;
    Led_t led = &Led_ObjPool[num_free_leds];

    led->GPIO_PORT_PTR = gpioPort;
    led->GPIO_PIN = pin;
    led->gpioDataRegister = GPIO_getDataRegister(gpioPort);
    led->isOn = false;
    led->isInit = true;

    return led;
}
```

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()

```
                                  {
    assert(led->isInit);
    return led->GPIO_PORT_PTR;
}
```

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()

```
                                {
    assert(led->isInit);
    return led->GPIO_PIN;
}
```

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()

```
                           {
    return led->isInit;
}
```

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()

```
                          {
    assert(led->isInit);
    return led->isOn;
}
```

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()

```
                            {
    assert(led->isInit);
    *led->gpioDataRegister |= led->GPIO_PIN;
    led->isOn = true;
    return;
}
```

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()

```
                           {
    assert(led->isInit);
    *led->gpioDataRegister &= ~(led->GPIO_PIN);
    led->isOn = false;
    return;
}
```

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()

```
                          {
    assert(led->isInit);
    *led->gpioDataRegister ^= led->GPIO_PIN;
    led->isOn = !led->isOn;
    return;
}
```

11.4.4.3 Variable Documentation

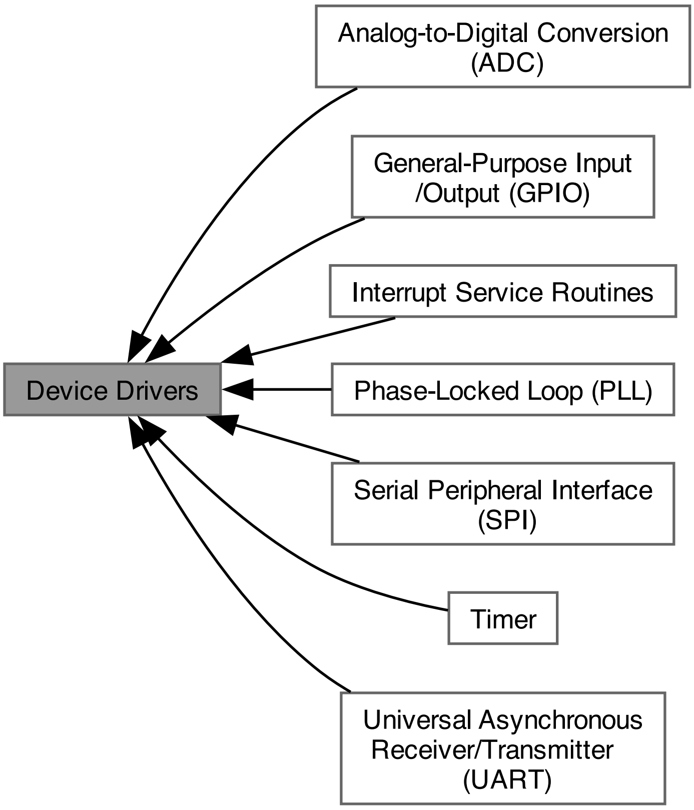
Led_ObjPool

```
LedStruct_t Led_ObjPool[LED_POOL_SIZE] = { 0 }  [static]
00038 { 0 };
```

11.5 Device Drivers

Low level device driver modules.

Collaboration diagram for Device Drivers:


Analog-to-Digital Conversion (ADC)
General-Purpose Input /Output (GPIO)
Interrupt Service Routines
Device Drivers
Phase-Locked Loop (PLL)
Serial Peripheral Interface (SPI)
Timer
Universal Asynchronous Receiver/Transmitter (UART)

Modules

- Analog-to-Digital Conversion (ADC)

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

    Functions for using GPIO ports.
- Interrupt Service Routines

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

- 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.

11.5.1 Detailed Description

Low level device driver modules.

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

11.5.2 Analog-to-Digital Conversion (ADC)

Functions for analog-to-digital conversion.

Files

- file ADC.c

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

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

Functions

- void ADC_Init (void)

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

11.5.2.1 Detailed Description

Functions for analog-to-digital conversion.

Todo Refactor to be more general.

11.5.2.2 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.

```
                    {
    // enable clock to ADC0 and wait for it to be ready
    SYSCTL_RCGCADC_R |= 0x01;
    while((SYSCTL_PRADC_R & 0x01) == 0) {
        __NOP();
    }

    // configure GPIO port
    GpioPort_t portE = GPIO_InitPort(E);
    GPIO_configDirection(portE, GPIO_PIN5, GPIO_INPUT);
    GPIO_ConfigAltMode(portE, GPIO_PIN5);
    GPIO_DisableDigital(portE, GPIO_PIN5);
    GPIO_ConfigAnalog(portE, GPIO_PIN5);

    ADC0_ACTSS_R &= ~(0x0F);                                // disable all sequencers
    ADC0_PC_R = (ADC0_PC_R & ~(0x0F)) | 0x01;               // max f_s = 125 [Hz]
    ADC0_SSPRI_R = (ADC0_SSPRI_R                            // give SS3 highest priority
                    & ~(0x3000)) |
                   0x0123;
    ADC0_EMUX_R |= 0x5000;                                  // set trigger source to Timer3A
    ADC0_SSMUX3_R = 8;                                      // analog input 8 (Ain8 = PE5)
    ADC0_SSCTL3_R = 0x06;               // disable temp. sensor, enable interrupts
    ADC0_ISC_R |= 0x08;                 // clear interrupt flag
    ADC0_IM_R |= 0x08;                  // enable interrupt

    ADC0_ACTSS_R |= 0x08;               // enable SS3
    return;
}
```

11.5.3 General-Purpose Input/Output (GPIO)

Functions for using GPIO ports.

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_PORT_BASE_ADDRESSES** {
  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_REGISTER_OFFSETS {
  DATA_REG_OFFSET = (uint32_t) 0x03FC , DIRECTION_REG_OFFSET = (uint32_t) 0x0400 , INT_SENSE_REG_OFFSET = (uint32_t) 0x0404 , INT_BOTH_EDGE_REG_OFFSET = (uint32_t) 0x0408 , INT_EVENT_REG_OFFSET = (uint32_t) 0x040C , INT_MASK_REG_OFFSET = (uint32_t) 0x0410 , INT_CLEAR_REG_OFFSET = (uint32_t) 0x041C , ALT_FUNC_REG_OFFSET = (uint32_t) 0x0420 , DRIVE_STR_2MA_REG_OFFSET = (uint32_t) 0x0500 , DRIVE_STR_4MA_REG_OFFSET = (uint32_t) 0x0504 , DRIVE_STR_8MA_REG_OFFSET = (uint32_t) 0x0508 , PULLUP_REG_OFFSET = (uint32_t) 0x0510 ,
  PULLDOWN_REG_OFFSET = (uint32_t) 0x0518 , DIGITAL_ENABLE_REG_OFFSET = (uint32_t) 0x051C , LOCK_REG_OFFSET = (uint32_t) 0x0520 , COMMIT_REG_OFFSET = (uint32_t) 0x0524 , ALT_MODE_REG_OFFSET = (uint32_t) 0x0528 , PORT_CTRL_REG_OFFSET = (uint32_t) 0x052C }

Variables

- static bool initStatusArray [6] = { false, false, false, false, false, false }
- static const GpioPortStruct_t GPIO_STRUCT_ARRAY [6]

Initialization

- 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 }
- 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.

Configuration (Digital I/O)

- 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) }
- enum **gpioDir_t** { **GPIO_INPUT** , **GPIO_OUTPUT** }
- enum **gpioResistor_t** { **PULLUP** , **PULLDOWN** }

- void GPIO_configDirection (GpioPort_t gpioPort, GpioPin_t pinMask, gpioDir_t direction)

  Configure the direction of the specified GPIO pins.
- void GPIO_configResistor (GpioPort_t gpioPort, GpioPin_t pinMask, gpioResistor_t resistor)

  Activate the specified pins' internal pull-up or 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.

Configuration (Interrupts)

- 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.

Basic Functions (Digital I/O)

- 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.

Configuration (Alternate/Analog Modes)

- 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.

11.5.3.1 Detailed Description

Functions for using GPIO ports.

11.5.3.2 Enumeration Type Documentation

GPIO_PORT_BASE_ADDRESSES

```
enum GPIO_PORT_BASE_ADDRESSES
                                {
    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,
};
```

GPIO_REGISTER_OFFSETS

```
enum GPIO_REGISTER_OFFSETS
```

Enumerator

DATA_REG_OFFSET	data
DIRECTION_REG_OFFSET	direction
INT_SENSE_REG_OFFSET	interrupt sense
INT_BOTH_EDGE_REG_OFFSET	interrupt both edges
INT_EVENT_REG_OFFSET	interrupt event
INT_MASK_REG_OFFSET	interrupt mask
INT_CLEAR_REG_OFFSET	interrupt clear
ALT_FUNC_REG_OFFSET	alternate function select
DRIVE_STR_2MA_REG_OFFSET	drive strength (2 [ma])
DRIVE_STR_4MA_REG_OFFSET	drive strength (4 [ma])
DRIVE_STR_8MA_REG_OFFSET	drive strength (8 [ma])
PULLUP_REG_OFFSET	pull-up resistor
PULLDOWN_REG_OFFSET	pull-down resistor
DIGITAL_ENABLE_REG_OFFSET	digital enable
LOCK_REG_OFFSET	lock
COMMIT_REG_OFFSET	commit
ALT_MODE_REG_OFFSET	alternate mode select
PORT_CTRL_REG_OFFSET	port control

```
                             {
    DATA_REG_OFFSET = (uint32_t) 0x03FC,
    DIRECTION_REG_OFFSET = (uint32_t) 0x0400,
    INT_SENSE_REG_OFFSET = (uint32_t) 0x0404,
    INT_BOTH_EDGE_REG_OFFSET = (uint32_t) 0x0408,
    INT_EVENT_REG_OFFSET = (uint32_t) 0x040C,
    INT_MASK_REG_OFFSET = (uint32_t) 0x0410,
    INT_CLEAR_REG_OFFSET = (uint32_t) 0x041C,
    ALT_FUNC_REG_OFFSET = (uint32_t) 0x0420,
    DRIVE_STR_2MA_REG_OFFSET = (uint32_t) 0x0500,
    DRIVE_STR_4MA_REG_OFFSET = (uint32_t) 0x0504,
    DRIVE_STR_8MA_REG_OFFSET = (uint32_t) 0x0508,
    PULLUP_REG_OFFSET = (uint32_t) 0x0510,
    PULLDOWN_REG_OFFSET = (uint32_t) 0x0518,
    DIGITAL_ENABLE_REG_OFFSET = (uint32_t) 0x051C,
    LOCK_REG_OFFSET = (uint32_t) 0x0520,
    COMMIT_REG_OFFSET = (uint32_t) 0x0524,
    ALT_MODE_REG_OFFSET = (uint32_t) 0x0528,
    PORT_CTRL_REG_OFFSET = (uint32_t) 0x052C
};
```

GPIO_PortName_t

```
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
} GPIO_PortName_t;
```

GpioPin_t

```
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))
} GpioPin_t;
```

GPIO_LAUNCHPAD_LEDS

```
enum GPIO_LAUNCHPAD_LEDS
```

Enumerator

LED_RED	PF1.
LED_GREEN	PF3.
LED_BLUE	PF2.

```
                        {
    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)
};
```

gpioDir_t

```
enum gpioDir_t
              {
    GPIO_INPUT,
    GPIO_OUTPUT
} gpioDir_t;
```

gpioResistor_t

```
enum gpioResistor_t
```

```
                  {
    PULLUP,
    PULLDOWN
} gpioResistor_t;
```

11.5.3.3 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.

```
                                                 {
    assert(portName < GPIO_NUM_PORTS);

    GpioPort_t gpioPort = &GPIO_STRUCT_ARRAY[portName];
    if(*gpioPort->isInit == false) {
        // Start clock for port and wait for it to be ready
        SYSCTL_RCGCGPIO_R |= (1 « portName);
        while((SYSCTL_PRGPIO_R & (1 « portName)) == 0) {
            __NOP();
        }

        // Disable alternate and analog modes
        REGISTER_VAL(gpioPort->BASE_ADDRESS + ALT_MODE_REG_OFFSET) &= ~(0xFF);
        REGISTER_VAL(gpioPort->BASE_ADDRESS + ALT_FUNC_REG_OFFSET) &= ~(0xFF);
        REGISTER_VAL(gpioPort->BASE_ADDRESS + PORT_CTRL_REG_OFFSET) = 0;

        if(portName == F) {
            GPIO_PORTF_LOCK_R = 0x4C4F434B;                // Unlock GPIO Port F
            GPIO_PORTF_CR_R |= 0x01;                       // Allow changes to PF0
        }

        *gpioPort->isInit = true;
    }

    return gpioPort;
}
```

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.

```
                                        {
    return *gpioPort->isInit;
}
```

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.

```
                                              {
        assert(*gpioPort->isInit);
        return gpioPort->BASE_ADDRESS;
}
```

GPIO_configDirection()

```
void GPIO_configDirection (
            GpioPort_t gpioPort,
            GpioPin_t pinMask,
            gpioDir_t direction )
```

Configure the direction of the specified GPIO pins.

Precondition

Initialize the GPIO port.

Parameters

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

Postcondition

The specified GPIO pins are now configured as inputs or outputs.

See also

GPIO_InitPort()

```
                                                                                {
        assert(*gpioPort->isInit);

        switch(direction) {
            case GPIO_INPUT:
                REGISTER_VAL(gpioPort->BASE_ADDRESS + DIRECTION_REG_OFFSET) &= ~(pinMask);
                break;
            case GPIO_OUTPUT:
                REGISTER_VAL(gpioPort->BASE_ADDRESS + DIRECTION_REG_OFFSET) |= pinMask;
                break;
            default:
                assert(false);
        }

        return;
}
```

GPIO_configResistor()

```
void GPIO_configResistor (
            GpioPort_t gpioPort,
            GpioPin_t pinMask,
            gpioResistor_t resistor )
```

Activate the specified pins' internal pull-up or pull-down resistors.

Precondition

Initialize the GPIO port.

Parameters

in	*gpioPort*	Pointer to the specified GPIO port.
in	*pinMask*	Bit mask corresponding to the intended pin(s).
in	*resistor*	The type of resistor to use.

Postcondition

The pull-up/pull-down resistor(s) are now activated.

See also

GPIO_InitPort()

```
                                                                                     {
    assert(*gpioPort->isInit);

    uint32_t registerOffset;
    switch(resistor) {
        case PULLUP:
            registerOffset = PULLUP_REG_OFFSET;
            break;
        case PULLDOWN:
            registerOffset = PULLDOWN_REG_OFFSET;
            break;
        default:
            assert(false);
    }

    REGISTER_VAL(gpioPort->BASE_ADDRESS + registerOffset) |= pinMask;
    return;
}
```

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].

```
                                                                                         {
    assert(*gpioPort->isInit);

    uint32_t driveSelectRegister_Offset;
    switch(drive_mA) {
        case 2:
            driveSelectRegister_Offset = DRIVE_STR_2MA_REG_OFFSET;
            break;
        case 4:
            driveSelectRegister_Offset = DRIVE_STR_4MA_REG_OFFSET;
            break;
        case 8:
            driveSelectRegister_Offset = DRIVE_STR_8MA_REG_OFFSET;
            break;
        default:
            driveSelectRegister_Offset = 0;
            assert(false);
    }
    REGISTER_VAL(gpioPort->BASE_ADDRESS + driveSelectRegister_Offset) |= pinMask;
    return;
}
```

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).

```
                                                                 {
    assert(*gpioPort->isInit);

    REGISTER_VAL(gpioPort->BASE_ADDRESS + DIGITAL_ENABLE_REG_OFFSET) |= pinMask;
    return;
}
```

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).

```
                                                                  {
    assert(*gpioPort->isInit);

    REGISTER_VAL(gpioPort->BASE_ADDRESS + DIGITAL_ENABLE_REG_OFFSET) &= ~pinMask;
    return;
}
```

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

```
{
    assert(*gpioPort->isInit);

    // Disable interrupts
    REGISTER_VAL(gpioPort->BASE_ADDRESS + INT_MASK_REG_OFFSET) &= ~(pinMask);

    // configure for edge-triggered interrupts
    REGISTER_VAL(gpioPort->BASE_ADDRESS + INT_SENSE_REG_OFFSET) &= ~(pinMask);
    REGISTER_VAL(gpioPort->BASE_ADDRESS + INT_BOTH_EDGE_REG_OFFSET) &= ~(pinMask);

    // select high or low edge
    if(risingEdge) {
        REGISTER_VAL(gpioPort->BASE_ADDRESS + INT_EVENT_REG_OFFSET) |= pinMask;
    }
    else {
        REGISTER_VAL(gpioPort->BASE_ADDRESS + INT_EVENT_REG_OFFSET) &= ~(pinMask);
    }

    // Clear interrupt flags and re-enable
    REGISTER_VAL(gpioPort->BASE_ADDRESS + INT_CLEAR_REG_OFFSET) |= pinMask;
    REGISTER_VAL(gpioPort->BASE_ADDRESS + INT_MASK_REG_OFFSET) |= pinMask;

    return;
}
```

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).

```
{
    assert(*gpioPort->isInit);

    // Disable interrupts
    REGISTER_VAL(gpioPort->BASE_ADDRESS + INT_MASK_REG_OFFSET) &= ~(pinMask);

    // configure for interrupts to trigger on both edges (high and low)
    REGISTER_VAL(gpioPort->BASE_ADDRESS + INT_SENSE_REG_OFFSET) &= ~(pinMask);
    REGISTER_VAL(gpioPort->BASE_ADDRESS + INT_BOTH_EDGE_REG_OFFSET) |= pinMask;

    // Clear interrupt flags and re-enable
    REGISTER_VAL(gpioPort->BASE_ADDRESS + INT_CLEAR_REG_OFFSET) |= pinMask;
    REGISTER_VAL(gpioPort->BASE_ADDRESS + INT_MASK_REG_OFFSET) |= pinMask;

    return;
}
```

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

```
                                                                                            {
    assert(*gpioPort->isInit);

    // Disable interrupts
    REGISTER_VAL(gpioPort->BASE_ADDRESS + INT_MASK_REG_OFFSET) &= ~(pinMask);

    // configure for edge-triggered interrupts
    REGISTER_VAL(gpioPort->BASE_ADDRESS + INT_SENSE_REG_OFFSET) |= pinMask;
    REGISTER_VAL(gpioPort->BASE_ADDRESS + INT_BOTH_EDGE_REG_OFFSET) &= ~(pinMask);

    // select high or low level
    if(highLevel) {
        REGISTER_VAL(gpioPort->BASE_ADDRESS + INT_EVENT_REG_OFFSET) |= pinMask;
    }
    else {
        REGISTER_VAL(gpioPort->BASE_ADDRESS + INT_EVENT_REG_OFFSET) &= ~(pinMask);
    }

    // Clear interrupt flags and re-enable
    REGISTER_VAL(gpioPort->BASE_ADDRESS + INT_CLEAR_REG_OFFSET) |= pinMask;
    REGISTER_VAL(gpioPort->BASE_ADDRESS + INT_MASK_REG_OFFSET) |= pinMask;

    return;
}
```

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).

```
                                                                 {
    assert(*gpioPort->isInit);
    assert(priority < 8);

    switch(gpioPort->BASE_ADDRESS) {
        case GPIO_PORTA_BASE_ADDRESS:
            NVIC_PRI0_R |= (priority « 5);
            NVIC_EN0_R |= (1 « 0);
            break;
        case GPIO_PORTB_BASE_ADDRESS:
            NVIC_PRI0_R |= (priority « 13);
            NVIC_EN0_R |= (1 « 1);
            break;
        case GPIO_PORTC_BASE_ADDRESS:
            NVIC_PRI0_R |= (priority « 21);
            NVIC_EN0_R |= (1 « 2);
            break;
        case GPIO_PORTD_BASE_ADDRESS:
            NVIC_PRI0_R |= (priority « 29);
            NVIC_EN0_R |= (1 « 3);
            break;
        case GPIO_PORTE_BASE_ADDRESS:
            NVIC_PRI1_R |= (priority « 5);
            NVIC_EN0_R |= (1 « 4);
            break;
        case GPIO_PORTF_BASE_ADDRESS:
            NVIC_PRI7_R |= (priority « 21);
            NVIC_EN0_R |= (1 « 30);
            break;
    }

    return;
}
```

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.

```
                                                               {
    assert(*gpioPort->isInit);
    return ((volatile uint32_t *) gpioPort->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).

```
                                                               {
    assert(*gpioPort->isInit);
    return REGISTER_VAL(gpioPort->DATA_REGISTER) & pinMask;
}
```

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).

```
                                                                  {
        assert(*gpioPort->isInit);
        REGISTER_VAL(gpioPort->DATA_REGISTER) |= pinMask;
        return;
}
```

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).

```
                                                                 {
        assert(*gpioPort->isInit);
        REGISTER_VAL(gpioPort->DATA_REGISTER) &= ~(pinMask);
        return;
}
```

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).

```
                                                               {
        assert(*gpioPort->isInit);
        REGISTER_VAL(gpioPort->DATA_REGISTER) ^= pinMask;
        return;
}
```

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).

```
                                                  {
    assert(*gpioPort->isInit);
    REGISTER_VAL(gpioPort->BASE_ADDRESS + ALT_FUNC_REG_OFFSET) |= pinMask;
    return;
}
```

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.

```
                                                                        {
    assert(*gpioPort->isInit);

    // TODO: Write explanation
    register_t portCtrlRegister = REGISTER_CAST(gpioPort->BASE_ADDRESS + PORT_CTRL_REG_OFFSET);
    for(uint8_t i = 0; i < 8; i++) {
        if(pinMask & (1 « i)) {
            *portCtrlRegister |= (fieldEncoding « (4 * i));
        }
    }
    return;
}
```

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).

```
                                                               {
    assert(*gpioPort->isInit);
    REGISTER_VAL(gpioPort->BASE_ADDRESS + ALT_MODE_REG_OFFSET) |= pinMask;
    return;
}
```

11.5.3.4 Variable Documentation

initStatusArray

```
bool initStatusArray[6] = { false, false, false, false, false, false }  [static]
{ false, false, false, false, false, false };
```

GPIO_STRUCT_ARRAY

```
const GpioPortStruct_t GPIO_STRUCT_ARRAY[6]  [static]
```

Initial value:

```
= {
    { GPIO_PORTA_BASE_ADDRESS, (GPIO_PORTA_BASE_ADDRESS + DATA_REG_OFFSET), &initStatusArray[0] },
    { GPIO_PORTB_BASE_ADDRESS, (GPIO_PORTB_BASE_ADDRESS + DATA_REG_OFFSET), &initStatusArray[1] },
    { GPIO_PORTC_BASE_ADDRESS, (GPIO_PORTC_BASE_ADDRESS + DATA_REG_OFFSET), &initStatusArray[2] },
    { GPIO_PORTD_BASE_ADDRESS, (GPIO_PORTD_BASE_ADDRESS + DATA_REG_OFFSET), &initStatusArray[3] },
    { GPIO_PORTE_BASE_ADDRESS, (GPIO_PORTE_BASE_ADDRESS + DATA_REG_OFFSET), &initStatusArray[4] },
    { GPIO_PORTF_BASE_ADDRESS, (GPIO_PORTF_BASE_ADDRESS + DATA_REG_OFFSET), &initStatusArray[5] }
}
                                                     {
    { GPIO_PORTA_BASE_ADDRESS, (GPIO_PORTA_BASE_ADDRESS + DATA_REG_OFFSET), &initStatusArray[0] },
    { GPIO_PORTB_BASE_ADDRESS, (GPIO_PORTB_BASE_ADDRESS + DATA_REG_OFFSET), &initStatusArray[1] },
    { GPIO_PORTC_BASE_ADDRESS, (GPIO_PORTC_BASE_ADDRESS + DATA_REG_OFFSET), &initStatusArray[2] },
    { GPIO_PORTD_BASE_ADDRESS, (GPIO_PORTD_BASE_ADDRESS + DATA_REG_OFFSET), &initStatusArray[3] },
    { GPIO_PORTE_BASE_ADDRESS, (GPIO_PORTE_BASE_ADDRESS + DATA_REG_OFFSET), &initStatusArray[4] },
    { GPIO_PORTF_BASE_ADDRESS, (GPIO_PORTF_BASE_ADDRESS + DATA_REG_OFFSET), &initStatusArray[5] }
};               // clang-format on
```

11.5.4 Interrupt Service Routines

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

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)

Functions

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

Variables

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

Interrupt Vector Table Configuration

- typedef void(∗ **ISR_t**) (void)

  Interrupt service routine (ISR) function pointers.
- 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.

Global Interrupt Configuration

- void ISR_GlobalDisable (void)

  Disable all interrupts globally.
- void ISR_GlobalEnable (void)

  Enable all interrupts globally.

Individual Interrupt Configuration

- 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).

11.5.4.1 Detailed Description

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

11.5.4.2 Function Documentation

ISR_setStatus()

```
static void ISR_setStatus (
              const uint8_t vectorNum,
              const bool isEnabled )  [static]
```

```
                                                                          {
    assert(vectorNum >= 16);
    assert(vectorNum < VECTOR_TABLE_SIZE);
    uint32_t interruptBitNum = (uint32_t) (vectorNum - 16);

    // Determine correct register to use
    uint32_t registerNum = 0;
    while(interruptBitNum >= ((registerNum + 1) * 32)) {
        registerNum += 1;
    }
    uint32_t REG_BASE_ADDR = (isEnabled) ? NVIC_EN_BASE_ADDR : NVIC_DIS_BASE_ADDR;
    register_t registerPtr = (register_t) (REG_BASE_ADDR + (4 * registerNum));

    // Enable/disable the ISR
    if(interruptBitNum > 31) {
        interruptBitNum -= (registerNum * 32);
    }
    *registerPtr |= (1 « interruptBitNum);

    return;
}
```

ISR_GlobalDisable()

```
void ISR_GlobalDisable (
              void  )
```

Disable all interrupts globally.

Note

Does not affect Reset, NMI, or hard faults.

See also

ISR_GlobalEnable()

```
                               {
    interruptsAreEnabled = false;
    __set_PRIMASK(1);
    return;
}
```

ISR_GlobalEnable()

```
void ISR_GlobalEnable (
              void  )
```

Enable all interrupts globally.

Note

Does not affect Reset, NMI, or hard faults.

See also

ISR_GlobalDisable()

```
                              {
    interruptsAreEnabled = true;
    __set_PRIMASK(0);
    return;
}
```

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()

```
                              {
    assert(isTableCopiedToRam == false);
    assert(interruptsAreEnabled == false);

    for(uint32_t idx = 0; idx < VECTOR_TABLE_SIZE; idx++) {
        newVectorTable[idx] = interruptVectorTable[idx];
    }

    NVIC_VTABLE_R = (uint32_t) &newVectorTable;
    isTableCopiedToRam = true;

    return;
}
```

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()

```
                                                            {
    assert(isTableCopiedToRam == true);
    assert(interruptsAreEnabled == false);
    assert(vectorNum >= 16);
    assert(vectorNum < VECTOR_TABLE_SIZE);

    newVectorTable[vectorNum] = isr;
    return;
}
```

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()

```
                                                                       {
    assert(vectorNum >= 16);
    assert(vectorNum < VECTOR_TABLE_SIZE);
    assert(priority <= 7);

    uint8_t interruptBitNum = vectorNum - 16;

    // Determine correct register and assign priority
    uint8_t priorityRegisterNum = (interruptBitNum - (interruptBitNum % 4)) / 4;
    register_t priorityRegisterPtr = (register_t) (NVIC_PRI_BASE_ADDR + (4 * priorityRegisterNum));
    switch((interruptBitNum % 4)) {
        case 0:
            *priorityRegisterPtr |= (priority « 5);
            assert(*priorityRegisterPtr & (priority « 5));
            break;

        case 1:
            *priorityRegisterPtr |= (priority « 13);
            assert(*priorityRegisterPtr & (priority « 13));
            break;

        case 2:
            *priorityRegisterPtr |= (priority « 21);
            assert(*priorityRegisterPtr & (priority « 21));
            break;

        case 3:
            *priorityRegisterPtr |= (priority « 29);
            assert(*priorityRegisterPtr & (priority « 29));
            break;
    }

    return;
}
```

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_setStatus(vectorNum, true);
    return;
}
```

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_setStatus(vectorNum, false);
    return;
}
```

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()

```
                                                        {
    assert(vectorNum >= 16);
    assert(vectorNum < VECTOR_TABLE_SIZE);

    NVIC_SW_TRIG_R = (NVIC_SW_TRIG_R & ~(0xFF)) | (vectorNum - 16);
    return;
}
```

11.5.5 Phase-Locked Loop (PLL)

Function for initializing the phase-locked loop.

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.

11.5.5.1 Detailed Description

Function for initializing the phase-locked loop.

11.5.5.2 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].

```
                    {
    // Disable PLL and system clock divider
    SYSCTL_RCC_R &= ~(1 « 22);                 // disable system clock divider
    SYSCTL_RCC2_R |= 0x80000000;                // use RCC2 register for more freq. options
    SYSCTL_RCC2_R |= (1 « 11);                 // set BYPASS2 to disable PLL

    // Select crystal value and oscillator source
    SYSCTL_RCC_R &= ~(0x1F « 6);                // zero out XTAL field
    SYSCTL_RCC_R |= (0x15 « 6);                 // 16[MHz] crystal
    SYSCTL_RCC2_R &= ~(0x70);                    // use main oscillator
    SYSCTL_RCC2_R &= ~(0x2000);                  // power on PLL

    // Set system clock divider
    SYSCTL_RCC2_R |= (1 « 30);                    // enable 7-bit divisor
    SYSCTL_RCC2_R &= ~(0x7F « 22);                // clear divisor bits
    SYSCTL_RCC2_R |= (0x04 « 22);                 // = (f_PLL / f_bus) - 1
    SYSCTL_RCC_R |= (1 « 22);                     // enable system clock divider

    // Re-activate PLL
    while((SYSCTL_RIS_R & 0x40) == 0) {                // wait for PLL to lock
        __NOP();
    }
    SYSCTL_RCC2_R &= ~(1 « 11);                       // clear BYPASS2 to enable PLL
}
```

11.5.6 Serial Peripheral Interface (SPI)

Functions for SPI-based communication via the SSI peripheral.

Files

- file SPI.c

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

  Header file for serial peripheral interface (SPI) module.

Data Structures

- struct Spi_t

Enumerations

- enum **GPIO_PORT_BASE_ADDRESSES** {
  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 **SSI_BASE_ADDRESSES** { **SSI0_BASE_ADDR** = (uint32_t) 0x40008000 , **SSI1_BASE_ADDR** = (uint32_t) 0x40009000 , **SSI2_BASE_ADDR** = (uint32_t) 0x4000A000 , **SSI3_BASE_ADDR** = (uint32_t) 0x4000B000 }
- enum **SSI_REGISTER_OFFSETS** {
  CTRL0_OFFSET = (uint32_t) 0 , **CTRL1_OFFSET** = (uint32_t) 0x004 , **DATA_OFFSET** = (uint32_t) 0x008 , **STATUS_OFFSET** = (uint32_t) 0x00C ,
  CLK_PRESCALE_OFFSET = (uint32_t) 0x010 , **INT_MASK_OFFSET** = (uint32_t) 0x014 , **RAW_INT_STATUS_OFFSET** = (uint32_t) 0x018 , **MASKED_INT_STATUS_OFFSET** = (uint32_t) 0x01C ,
  INT_CLEAR_OFFSET = (uint32_t) 0x020 }
- enum **SsiNum_t** { **SSI0** , **SSI1** , **SSI2** , **SSI3** }
- enum **SpiClockPhase_t** { **SPI_RISING_EDGE** , **SPI_FALLING_EDGE** }
- enum **SpiClockPolarity_t** { **SPI_STEADY_STATE_LOW** , **SPI_STEADY_STATE_HIGH** }

Functions

- Spi_t SPI_Init (GpioPort_t gpioPort, GpioPin_t dcPin, SsiNum_t ssiNum)

  Initialize an SSI as an SPI controller.
- bool SPI_isInit (Spi_t spi)

  Check if a given SPI is initialized.
- void SPI_configClock (Spi_t spi, SpiClockPhase_t clockPhase, SpiClockPolarity_t clockPolarity)

  Configure an SPI's clock settings.
- void SPI_setDataSize (Spi_t spi, uint8_t dataSize)
- void SPI_Enable (Spi_t spi)

  Enable an SPI.
- void SPI_Disable (Spi_t spi)

  Disable an SPI.
- uint16_t SPI_Read (Spi_t spi)

  Read data from the serial port.
- void SPI_WriteCmd (Spi_t spi, uint16_t cmd)

  Write a command to the serial port.
- void SPI_WriteData (Spi_t spi, uint16_t data)

  Write data to the serial port.

Variables

- static SpiStruct_t SPI_ARR [4]

11.5.6.1 Detailed Description

Functions for SPI-based communication via the SSI peripheral.

Todo Remove statically-allocated data structures for unused SSIs.

11.5.6.2 Enumeration Type Documentation

GPIO_PORT_BASE_ADDRESSES

```
enum GPIO_PORT_BASE_ADDRESSES
                              {
    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,
};
```

SSI_BASE_ADDRESSES

```
enum SSI_BASE_ADDRESSES
                        {
    SSI0_BASE_ADDR = (uint32_t) 0x40008000,
    SSI1_BASE_ADDR = (uint32_t) 0x40009000,
    SSI2_BASE_ADDR = (uint32_t) 0x4000A000,
    SSI3_BASE_ADDR = (uint32_t) 0x4000B000,
};
```

SSI_REGISTER_OFFSETS

```
enum SSI_REGISTER_OFFSETS
                          {
    CTRL0_OFFSET = (uint32_t) 0,
    CTRL1_OFFSET = (uint32_t) 0x004,
    DATA_OFFSET = (uint32_t) 0x008,
    STATUS_OFFSET = (uint32_t) 0x00C,
    CLK_PRESCALE_OFFSET = (uint32_t) 0x010,
    INT_MASK_OFFSET = (uint32_t) 0x014,
    RAW_INT_STATUS_OFFSET = (uint32_t) 0x018,
    MASKED_INT_STATUS_OFFSET = (uint32_t) 0x01C,
    INT_CLEAR_OFFSET = (uint32_t) 0x020,
};
```

SsiNum_t

```
enum SsiNum_t
             {
    SSI0,
    SSI1,
    SSI2,
    SSI3
} SsiNum_t;
```

SpiClockPhase_t

```
enum SpiClockPhase_t
             {
    SPI_RISING_EDGE,
    SPI_FALLING_EDGE
} SpiClockPhase_t;
```

SpiClockPolarity_t

```
enum SpiClockPolarity_t
                    {
    SPI_STEADY_STATE_LOW,
    SPI_STEADY_STATE_HIGH
} SpiClockPolarity_t;
```

11.5.6.3 Function Documentation

SPI_Init()

```
Spi_t SPI_Init (
            GpioPort_t gpioPort,
            GpioPin_t dcPin,
            SsiNum_t ssiNum )
```

Initialize an SSI as an SPI controller.

Parameters

in	*gpioPort*	GPIO port to use.
in	*dcPin*	GPIO pin to use.
in	*ssiNum*	SSI to use.
out	*Spi_t*	(Pointer to) initialized SPI peripheral.

```
                                                                      {
    assert(GPIO_isPortInit(gpioPort));
    assert(dcPin <= GPIO_PIN7);

    // check GPIO pins
    uint32_t gpio_baseAddress = GPIO_getBaseAddr(gpioPort);
    GpioPin_t gpioPins;

    switch(ssiNum) {
        case SSI0:
            assert(gpio_baseAddress == GPIO_PORTA_BASE_ADDRESS);
            gpioPins = GPIO_PIN2 | GPIO_PIN3 | GPIO_PIN4 | GPIO_PIN5;
            break;
        case SSI1:
            assert(gpio_baseAddress == GPIO_PORTF_BASE_ADDRESS);
            gpioPins = GPIO_PIN0 | GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3;
            break;
        case SSI2:
            assert(gpio_baseAddress == GPIO_PORTB_BASE_ADDRESS);
            gpioPins = GPIO_PIN4 | GPIO_PIN5 | GPIO_PIN6 | GPIO_PIN7;
            break;
        case SSI3:
            assert(gpio_baseAddress == GPIO_PORTD_BASE_ADDRESS);
            gpioPins = GPIO_PIN0 | GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3;
            break;
        default:
            assert(false);
    }

    assert((dcPin & gpioPins) == 0);

    // initialize SSI peripheral in SPI mode
    Spi_t spi = &SPI_ARR[ssiNum];
    if(spi->isInit == false) {
        // config. GPIO pins
        GPIO_ConfigAltMode(gpioPort, gpioPins);
        GPIO_ConfigPortCtrl(gpioPort, gpioPins, 2);

        GPIO_configDirection(gpioPort, dcPin, GPIO_OUTPUT);

        GPIO_EnableDigital(gpioPort, gpioPins | dcPin);

        // enable clock to SSI, and wait for it to be ready
        SYSCTL_RCGCSSI_R |= (1 « ssiNum);
        while((SYSCTL_PRSSI_R & (1 « ssiNum)) == 0) {
            __NOP();
        }

        // config control registers
        register_t ctrlRegister0 = (register_t) (spi->BASE_ADDRESS + CTRL0_OFFSET);
        register_t ctrlRegister1 = (register_t) (spi->BASE_ADDRESS + CTRL1_OFFSET);
        register_t clkPrescaleReg = (register_t) (spi->BASE_ADDRESS + CLK_PRESCALE_OFFSET);

        *ctrlRegister1 &= ~(0x02);               // disable
        *ctrlRegister1 &= ~(0x15);               // controller (master) mode, no EOT, no loopback

        *ctrlRegister0 &= ~(0x30);               // SPI frame format

        // set bit rate to 10 [MHz]
        *clkPrescaleReg = (*clkPrescaleReg & ~(0xFF)) | 4;
        *ctrlRegister0 = (*ctrlRegister0 & ~(0xFF00)) | (0x0100);

        spi->gpioDataRegister = GPIO_getDataRegister(gpioPort);
        spi->gpioDataCommPin = dcPin;
        spi->isEnabled = false;
        spi->isInit = true;
    }

    return spi;
}
```

SPI_isInit()

```
bool SPI_isInit (
               Spi_t spi )
```

Check if a given SPI is initialized.

Parameters

in	*spi*	SPI to check.
out	*true*	The SPI is initialized.
out	*false*	The SPI is not initialized.

```
                           {
    return spi->isInit;
}
```

SPI_configClock()

```
void SPI_configClock (
               Spi_t spi,
               SpiClockPhase_t clockPhase,
               SpiClockPolarity_t clockPolarity )
```

Configure an SPI's clock settings.

Precondition

Initialize the SPI.

Disable the SPI.

Parameters

in	*spi*	SPI to configure.
in	*clockPhase*	
in	*clockPolarity*	

```
                                                                                                      {
    assert(spi->isInit);
    assert(spi->isEnabled == false);

    register_t ctrlRegister0 = (register_t) (spi->BASE_ADDRESS + CTRL0_OFFSET);

    switch(clockPhase) {
        case SPI_RISING_EDGE:
            *ctrlRegister0 &= ~(1 « 7);
            break;
        case SPI_FALLING_EDGE:
            *ctrlRegister0 |= (1 « 7);
            break;
        default:
            assert(false);
    }

    switch(clockPolarity) {
        case SPI_STEADY_STATE_LOW:
            *ctrlRegister0 &= ~(1 « 6);
            break;
        case SPI_STEADY_STATE_HIGH:
            *ctrlRegister0 |= (1 « 6);
            break;
        default:
            assert(false);
    }

    return;
}
```

SPI_setDataSize()

```
void SPI_setDataSize (
              Spi_t spi,
              uint8_t dataSize )
```

Precondition

Initialize the SPI.

Disable the SPI.

Parameters

in	*spi*	
in	*dataSize*	

```
                                                   {
    assert(spi->isInit);
    assert(spi->isEnabled == false);
    assert(dataSize >= 4);
    assert(dataSize <= 16);

    register_t ctrlRegister0 = (register_t) (spi->BASE_ADDRESS + CTRL0_OFFSET);
    *ctrlRegister0 = (*ctrlRegister0 & ~(0x0F)) | (dataSize - 1);

    spi->dataSize = dataSize;
    return;
}
```

SPI_Enable()

```
void SPI_Enable (
              Spi_t spi )
```

Enable an SPI.

Precondition

Initialize the SPI.

Parameters

in	*spi*	SPI to enable.

Postcondition

The SPI is enable.

See also

SPI_Disable()

```
                             {
    assert(spi->isInit);
    assert(spi->dataSize > 0);
    if(spi->isEnabled == false) {
        register_t ctrlRegister1 = (register_t) (spi->BASE_ADDRESS + CTRL1_OFFSET);
        *ctrlRegister1 |= 0x02;
        spi->isEnabled = true;
    }
    return;
}
```

SPI_Disable()

```
void SPI_Disable (
              Spi_t spi )
```

Disable an SPI.

Precondition

Initialize the SPI.

Parameters

in	*spi*	SPI to disable.

Postcondition

The SPI is disabled.

See also

SPI_Enable()

```
                              {
    assert(spi->isInit);
    if(spi->isEnabled) {
        register_t ctrlRegister1 = (register_t) (spi->BASE_ADDRESS + CTRL1_OFFSET);
        *ctrlRegister1 &= ~(0x02);
        spi->isEnabled = false;
    }
}
```

SPI_Read()

```
uint16_t SPI_Read (
              Spi_t spi )
```

Read data from the serial port.

Precondition

Initialize the SPI.

Enable the SPI.

Parameters

in	*spi*	SPI to read from.
out	*data*	8-bit data received from the hardware's receive FIFO.

```
                             {
    assert(spi->isInit);
    assert(spi->isEnabled);

    return *spi->DATA_REGISTER;
}
```

SPI_WriteCmd()

```
void SPI_WriteCmd (
              Spi_t spi,
              uint16_t cmd )
```

Write a command to the serial port.

Precondition

Initialize the SPI.

Enable the SPI.

Parameters

in	*spi*	SPI to write to.
in	*cmd*	Command to write.

Postcondition

The D/C pin is cleared.

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

```
                                           {
    assert(spi->isInit);
    assert(spi->isEnabled);

    while(*spi->STATUS_REGISTER & SSI_SR_BSY) {                     // wait while SPI is busy
        __NOP();
    }

    *spi->gpioDataRegister &= ~(spi->gpioDataCommPin);              // signal incoming command
    *spi->DATA_REGISTER = cmd & ((1 « spi->dataSize) - 1);

    while(*spi->STATUS_REGISTER & SSI_SR_BSY) {                      // allow transmission to finish
        __NOP();
    }
    return;
}
```

SPI_WriteData()

```
void SPI_WriteData (
               Spi_t spi,
               uint16_t data )
```

Write data to the serial port.

Precondition

Initialize the SPI.

Enable the SPI.

Parameters

in	*spi*	SPI to write to.
in	*data*	Data to write.

Postcondition

The D/C pin is set.

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

```
                                          {
    assert(spi->isInit);
    assert(spi->isEnabled);

    while((*spi->STATUS_REGISTER & SSI_SR_TNF) == 0) {               // wait while TX FIFO is full
        __NOP();
    }

    *spi->gpioDataRegister |= spi->gpioDataCommPin;                  // signal incoming data
    *spi->DATA_REGISTER = data & ((1 « spi->dataSize) - 1);

    return;
}
```

11.5.6.4 Variable Documentation

SPI_ARR

```
SpiStruct_t SPI_ARR[4]  [static]
```

Initial value:

```
= {
    { SSI0_BASE_ADDR, REGISTER_CAST(SSI0_BASE_ADDR + DATA_OFFSET), REGISTER_CAST(SSI0_BASE_ADDR +
      STATUS_OFFSET),
        0, 0, 0, false, false },
    { SSI1_BASE_ADDR, REGISTER_CAST(SSI1_BASE_ADDR + DATA_OFFSET), REGISTER_CAST(SSI1_BASE_ADDR +
      STATUS_OFFSET),
        0, 0, 0, false, false },
    { SSI2_BASE_ADDR, REGISTER_CAST(SSI2_BASE_ADDR + DATA_OFFSET), REGISTER_CAST(SSI2_BASE_ADDR +
      STATUS_OFFSET),
              0, 0, 0, false, false },
    { SSI3_BASE_ADDR, REGISTER_CAST(SSI3_BASE_ADDR + DATA_OFFSET), REGISTER_CAST(SSI3_BASE_ADDR +
      STATUS_OFFSET),
              0, 0, 0, false, false },
}
                                      {
          { SSI0_BASE_ADDR, REGISTER_CAST(SSI0_BASE_ADDR + DATA_OFFSET), REGISTER_CAST(SSI0_BASE_ADDR +
      STATUS_OFFSET),
              0, 0, 0, false, false },
          { SSI1_BASE_ADDR, REGISTER_CAST(SSI1_BASE_ADDR + DATA_OFFSET), REGISTER_CAST(SSI1_BASE_ADDR +
      STATUS_OFFSET),
              0, 0, 0, false, false },
          { SSI2_BASE_ADDR, REGISTER_CAST(SSI2_BASE_ADDR + DATA_OFFSET), REGISTER_CAST(SSI2_BASE_ADDR +
      STATUS_OFFSET),
              0, 0, 0, false, false },
          { SSI3_BASE_ADDR, REGISTER_CAST(SSI3_BASE_ADDR + DATA_OFFSET), REGISTER_CAST(SSI3_BASE_ADDR +
      STATUS_OFFSET),
              0, 0, 0, false, false },
      };               // clang-format on
```

11.5.7 Timer

Functions for using hardware timers.

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 bool initStatusArray [6] = { false, false, false, false, false, false }
- static const TimerStruct_t TIMER_STRUCT_ARRAY [6]

11.5.7.1 Detailed Description

Functions for using hardware timers.

11.5.7.2 Enumeration Type Documentation

anonymous enum

```
anonymous enum
   {
  TIMER0_BASE = 0x40030000,
  TIMER1_BASE = 0x40031000,
  TIMER2_BASE = 0x40032000,
  TIMER3_BASE = 0x40033000,
  TIMER4_BASE = 0x40034000,
  TIMER5_BASE = 0x40035000
};
```

REGISTER_OFFSETS

```
enum REGISTER_OFFSETS
                        {
    CONFIG = 0x00,
    MODE = 0x04,
    CTRL = 0x0C,
    INT_MASK = 0x18,
    INT_CLEAR = 0x24,
    INTERVAL = 0x28,
    VALUE = 0x054
};
```

timerName_t

```
enum timerName_t
               {
    TIMER0,
    TIMER1,
    TIMER2,
    TIMER3,
    TIMER4,
    TIMER5,
} timerName_t;
```

timerMode_t

```
enum timerMode_t
```

Enumerator

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

```
               {
    ONESHOT,
    PERIODIC
} timerMode_t;
```

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

```
               {
    UP,
    DOWN
} timerDirection_t;
```

11.5.7.3 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_t timer = &TIMER_STRUCT_ARRAY[timerName];
    if(*timer->isInit == false) {
        // Start clock to timer
        SYSCTL_RCGCTIMER_R |= (1 « timerName);
        while((SYSCTL_PRTIMER_R & (1 « timerName)) == 0) {
            __NOP();
        }
        *timer->isInit = true;
    }

    // Disable timers and turn on concatenated mode
    *timer->controlRegister &= ~(0x0101);
    REGISTER_VAL(timer->baseAddress + CONFIG) &= ~(0x0007);

    return timer;
}
```

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()

```
                                   {
    if(*timer->isInit) {
        *timer->controlRegister &= ~(0x101);               // stop timer
        uint8_t timerNum = timer->name;

        // disable clock to timer
        SYSCTL_RCGCTIMER_R &= ~(1 « timerNum);
        while(SYSCTL_PRTIMER_R & (1 « timerNum)) {
            __NOP();
        }
        *timer->isInit = false;
    }
    return;
}
```

Timer_getName()

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

Get the name of a timer object.

Parameters

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

```
                                       {
    assert(*timer->isInit);
    return timer->name;
}
```

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()

```
                                {
    return *timer->isInit;
}
```

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.

```
                                                                                      {
    assert(*timer->isInit);
    *timer->controlRegister &= ~(0x101);                                    // disable timer

    REGISTER_VAL(timer->baseAddress + MODE) &= ~(0x13);
    switch(timerMode) {
        case ONESHOT:
            REGISTER_VAL(timer->baseAddress + MODE) |= 0x01;
            break;
        case PERIODIC:
            REGISTER_VAL(timer->baseAddress + MODE) |= 0x02;
            break;
    }

    switch(timerDirection) {
        case(UP):
            REGISTER_VAL(timer->baseAddress + MODE) |= 0x10;
            break;
        case(DOWN):
            REGISTER_VAL(timer->baseAddress + MODE) &= ~(0x10);
            break;
    }

    return;
}
```

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()

```
                                              {
    assert(*timer->isInit);

    *timer->controlRegister |= 0x20;
    return;
}
```

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()

```
                                              {
    assert(*timer->isInit);

    *timer->controlRegister &= ~(0x20);
    return;
}
```

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->controlRegister &= ~(0x101);                                     // disable timer
    *timer->interruptClearRegister |= 0x01;                                  // clear int. flag
    REGISTER_VAL(timer->baseAddress + INT_MASK) |= 0x01;
    return;
}
```

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->controlRegister &= ~(0x101);                                  // disable timer
    REGISTER_VAL(timer->baseAddress + INT_MASK) &= ~(0x01);               // disable int.
    return;
}
```

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->interruptClearRegister) |= 0x01;
    return;
}
```

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()

```
                                                           {
    assert(*timer->isInit);
    assert((time_ms > 0) && (time_ms <= 53000));

    *timer->controlRegister &= ~(0x101);                // disable timer
    uint32_t reload_val = (80000 * time_ms) - 1;
    *timer->intervalLoadRegister = reload_val;

    return;
}
```

Timer_getCurrentValue()

```
uint32_t Timer_getCurrentValue (
            Timer_t timer )
```

```
                                            {
    assert(*timer->isInit);

    return REGISTER_VAL(timer->baseAddress + VALUE);
}
```

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()

```
                             {
    assert(*timer->isInit);

    *timer->controlRegister |= 0x101;                   // enable timer
    return;
}
```

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()

```
                                {
    assert(*timer->isInit);

    *timer->controlRegister &= ~(0x101);            // stop/disable timer
    return;
}
```

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()

```
                                     {
    return (bool) (*timer->controlRegister & 0x101);
}
```

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.

```
                                                    {
    assert(*timer->isInit);

    Timer_setInterval_ms(timer, time_ms);
    Timer_Start(timer);
    while(Timer_isCounting(timer)) {
        __NOP();
    }

    return;
}
```

11.5.7.4 Variable Documentation

initStatusArray

```
bool initStatusArray[6] = { false, false, false, false, false, false }  [static]
```

```
{ false, false, false, false, false, false };
```

TIMER_STRUCT_ARRAY

```
const TimerStruct_t TIMER_STRUCT_ARRAY[6]  [static]
```

Initial value:

```
= {
    { TIMER0, TIMER0_BASE, REGISTER_CAST(TIMER0_BASE + CTRL), REGISTER_CAST(TIMER0_BASE + INTERVAL),
      REGISTER_CAST(TIMER0_BASE + INT_CLEAR), &initStatusArray[0] },
    { TIMER1, TIMER1_BASE, REGISTER_CAST(TIMER1_BASE + CTRL), REGISTER_CAST(TIMER1_BASE + INTERVAL),
      REGISTER_CAST(TIMER1_BASE + INT_CLEAR), &initStatusArray[1] },
    { TIMER2, TIMER2_BASE, REGISTER_CAST(TIMER2_BASE + CTRL), REGISTER_CAST(TIMER2_BASE + INTERVAL),
      REGISTER_CAST(TIMER2_BASE + INT_CLEAR), &initStatusArray[2] },
    { TIMER3, TIMER3_BASE, REGISTER_CAST(TIMER3_BASE + CTRL), REGISTER_CAST(TIMER3_BASE + INTERVAL),
      REGISTER_CAST(TIMER3_BASE + INT_CLEAR), &initStatusArray[3] },
    { TIMER4, TIMER4_BASE, REGISTER_CAST(TIMER4_BASE + CTRL), REGISTER_CAST(TIMER4_BASE + INTERVAL),
      REGISTER_CAST(TIMER4_BASE + INT_CLEAR), &initStatusArray[4] },
    { TIMER5, TIMER5_BASE, REGISTER_CAST(TIMER5_BASE + CTRL), REGISTER_CAST(TIMER5_BASE + INTERVAL),
      REGISTER_CAST(TIMER5_BASE + INT_CLEAR), &initStatusArray[5] }
}
```

```
                                                        {
    { TIMER0, TIMER0_BASE, REGISTER_CAST(TIMER0_BASE + CTRL), REGISTER_CAST(TIMER0_BASE + INTERVAL),
  REGISTER_CAST(TIMER0_BASE + INT_CLEAR), &initStatusArray[0] },
    { TIMER1, TIMER1_BASE, REGISTER_CAST(TIMER1_BASE + CTRL), REGISTER_CAST(TIMER1_BASE + INTERVAL),
  REGISTER_CAST(TIMER1_BASE + INT_CLEAR), &initStatusArray[1] },
    { TIMER2, TIMER2_BASE, REGISTER_CAST(TIMER2_BASE + CTRL), REGISTER_CAST(TIMER2_BASE + INTERVAL),
  REGISTER_CAST(TIMER2_BASE + INT_CLEAR), &initStatusArray[2] },
    { TIMER3, TIMER3_BASE, REGISTER_CAST(TIMER3_BASE + CTRL), REGISTER_CAST(TIMER3_BASE + INTERVAL),
  REGISTER_CAST(TIMER3_BASE + INT_CLEAR), &initStatusArray[3] },
    { TIMER4, TIMER4_BASE, REGISTER_CAST(TIMER4_BASE + CTRL), REGISTER_CAST(TIMER4_BASE + INTERVAL),
  REGISTER_CAST(TIMER4_BASE + INT_CLEAR), &initStatusArray[4] },
    { TIMER5, TIMER5_BASE, REGISTER_CAST(TIMER5_BASE + CTRL), REGISTER_CAST(TIMER5_BASE + INTERVAL),
  REGISTER_CAST(TIMER5_BASE + INT_CLEAR), &initStatusArray[5] }
};
```

11.5.8 Universal Asynchronous Receiver/Transmitter (UART)

Functions for serial communication via the UART peripheral.

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 **CONVERT_INT_TO_ASCII**(X) ((unsigned char) (X + 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 bool initStatusArray [8] = { false, false, false, false, false, false, false, false }
- static const UartStruct_t UART_STRUCT_ARRAY [8]

11.5.8.1 Detailed Description

Functions for serial communication via the UART peripheral.

11.5.8.2 Enumeration Type Documentation

GPIO_BASE_ADDRESSES

```
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
};
```

UART_BASE_ADDRESSES

```
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
};
```

UART_REG_OFFSETS

```
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
};
```

uartNum_t

```
enum uartNum_t
             {
    UART0,
    UART1,
    UART2,
    UART3,
    UART4,
    UART5,
    UART6,
    UART7
} uartNum_t;
```

11.5.8.3 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: $FBRD = int((mod(BRD,1) * 64) + 0.5)$

```
                                                      {
    // Check inputs
    assert(GPIO_isPortInit(port));
    assert(uartNum < 8);

    // Check that inputted GPIO port and UART match each other
    uint32_t gpio_baseAddress = GPIO_getBaseAddr(port);
    GpioPin_t RX_PIN_NUM;
    GpioPin_t TX_PIN_NUM;

    switch(uartNum) {
        case UART0:
            assert(gpio_baseAddress == GPIO_PORTA_BASE);
            RX_PIN_NUM = GPIO_PIN0;
            TX_PIN_NUM = GPIO_PIN1;
            break;
        case UART1:
            assert(gpio_baseAddress == GPIO_PORTB_BASE);
            RX_PIN_NUM = GPIO_PIN0;
            TX_PIN_NUM = GPIO_PIN1;
            break;
        case UART2:
            assert(gpio_baseAddress == GPIO_PORTD_BASE);
            RX_PIN_NUM = GPIO_PIN6;
            TX_PIN_NUM = GPIO_PIN7;
            break;
        case UART3:
            assert(gpio_baseAddress == GPIO_PORTC_BASE);
            RX_PIN_NUM = GPIO_PIN6;
            TX_PIN_NUM = GPIO_PIN7;
            break;
        case UART4:
            assert(gpio_baseAddress == GPIO_PORTC_BASE);
            RX_PIN_NUM = GPIO_PIN4;
            TX_PIN_NUM = GPIO_PIN5;
            break;
        case UART5:
            assert(gpio_baseAddress == GPIO_PORTE_BASE);
            RX_PIN_NUM = GPIO_PIN4;
            TX_PIN_NUM = GPIO_PIN5;
            break;
        case UART6:
            assert(gpio_baseAddress == GPIO_PORTD_BASE);
            RX_PIN_NUM = GPIO_PIN4;
            TX_PIN_NUM = GPIO_PIN5;
            break;
        case UART7:
            assert(gpio_baseAddress == GPIO_PORTE_BASE);
            RX_PIN_NUM = GPIO_PIN0;
            TX_PIN_NUM = GPIO_PIN1;
            break;
    }

    // clang-format off
    // clang-format on

    // Initialize UART
    Uart_t uart = &UART_STRUCT_ARRAY[uartNum];
    if(*uart->isInitPtr == false) {
        SYSCTL_RCGCUART_R |= (1 « uartNum);
        while((SYSCTL_PRUART_R & (1 « uartNum)) == 0) {
            __NOP();
        }

        // initialize GPIO pins
        GPIO_ConfigAltMode(port, RX_PIN_NUM | TX_PIN_NUM);
        if(gpio_baseAddress == GPIO_PORTC_BASE) {
            GPIO_ConfigPortCtrl(port, RX_PIN_NUM | TX_PIN_NUM, 2);
        }
        else {
            GPIO_ConfigPortCtrl(port, RX_PIN_NUM | TX_PIN_NUM, 1);
        }
        GPIO_ConfigDriveStrength(port, RX_PIN_NUM | TX_PIN_NUM, 8);
        GPIO_EnableDigital(port, RX_PIN_NUM | TX_PIN_NUM);

        // disable UART
        REGISTER_VAL(uart->BASE_ADDRESS + CTL_R_OFFSET) &= ~(1 « uartNum);

        // 8-bit length, FIFO
        REGISTER_VAL(uart->BASE_ADDRESS + IBRD_R_OFFSET) |= 43;
        REGISTER_VAL(uart->BASE_ADDRESS + FBRD_R_OFFSET) |= 26;

        // (NOTE: access *AFTER* `BRD‘)
        REGISTER_VAL(uart->BASE_ADDRESS + LCRH_R_OFFSET) |= 0x70;
        REGISTER_VAL(uart->BASE_ADDRESS + CC_R_OFFSET) &= ~(0x0F);               // system clock

        // re-enable
        REGISTER_VAL(uart->BASE_ADDRESS + CTL_R_OFFSET) |= (1 « uartNum);

        *uart->isInitPtr = true;
    }

    return uart;
}
```

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.

```
                              {
    return *uart->isInitPtr;
}
```

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.

```
                                                {
    while((*uart->FLAG_REGISTER & 0x10) != 0) {
        __NOP();
    }
    return (unsigned char) REGISTER_VAL(uart->BASE_ADDRESS);
}
```

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.

```
                                                                 {
    while((*uart->FLAG_REGISTER & 0x20) != 0) {
        __NOP();
    }
    REGISTER_VAL(uart->BASE_ADDRESS) = inputChar;
    return;
}
```

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.

```
                                                       {
    unsigned char * str_ptr = inputStr;
    while(*str_ptr != '\0') {
        UART_WriteChar(uart, *str_ptr);
        str_ptr += 1;
    }
    return;
}
```

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.

```
                                           {
    // Send negative sign ('-') if needed
    if(n < 0) {
        UART_WriteChar(uart, '-');
        n *= -1;
    }

    if(n < 10) {
        UART_WriteChar(uart, CONVERT_INT_TO_ASCII(n));
    }
    else {
        int32_t nearestPowOf10 = 1;
        while((n / (nearestPowOf10 * 10)) > 0) {
            nearestPowOf10 *= 10;
        }

        while(nearestPowOf10 > 0) {
            UART_WriteChar(uart, CONVERT_INT_TO_ASCII(n / nearestPowOf10));
            n %= nearestPowOf10;
            nearestPowOf10 /= 10;
        }
    }
    return;
}
```

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.

```
                                                                 {
    // Send negative sign ('-') if needed
    if(n < 0) {
        UART_WriteChar(uart, '-');
        n *= -1;
    }

    // Send the integer part
    int32_t b = n / (int32_t) 1;
    UART_WriteInt(uart, b);

    // Send the decimal part
    if(numDecimals > 0) {
        UART_WriteChar(uart, '.');
        for(uint8_t count = 0; count < numDecimals; count++) {
            n = (n - b) * (double) 10;
            b = n / (int32_t) 1;
            UART_WriteChar(uart, CONVERT_INT_TO_ASCII(b));
        }
    }
    return;
}
```

11.5.8.4 Variable Documentation

initStatusArray

```
bool initStatusArray[8] = { false, false, false, false, false, false, false, false }  [static]
```

```
{ false, false, false, false, false, false, false, false };
```

UART_STRUCT_ARRAY

```
const UartStruct_t UART_STRUCT_ARRAY[8]  [static]
```

Initial value:

```
= {
    { UART0_BASE, REGISTER_CAST(UART0_BASE + UART_FR_R_OFFSET), &initStatusArray[0] },
    { UART1_BASE, REGISTER_CAST(UART1_BASE + UART_FR_R_OFFSET), &initStatusArray[1] },
    { UART2_BASE, REGISTER_CAST(UART2_BASE + UART_FR_R_OFFSET), &initStatusArray[2] },
    { UART3_BASE, REGISTER_CAST(UART3_BASE + UART_FR_R_OFFSET), &initStatusArray[3] },
    { UART4_BASE, REGISTER_CAST(UART4_BASE + UART_FR_R_OFFSET), &initStatusArray[4] },
    { UART5_BASE, REGISTER_CAST(UART5_BASE + UART_FR_R_OFFSET), &initStatusArray[5] },
    { UART6_BASE, REGISTER_CAST(UART6_BASE + UART_FR_R_OFFSET), &initStatusArray[6] },
    { UART7_BASE, REGISTER_CAST(UART7_BASE + UART_FR_R_OFFSET), &initStatusArray[7] }
}
```

```
                                   {
    { UART0_BASE, REGISTER_CAST(UART0_BASE + UART_FR_R_OFFSET), &initStatusArray[0] },
    { UART1_BASE, REGISTER_CAST(UART1_BASE + UART_FR_R_OFFSET), &initStatusArray[1] },
    { UART2_BASE, REGISTER_CAST(UART2_BASE + UART_FR_R_OFFSET), &initStatusArray[2] },
    { UART3_BASE, REGISTER_CAST(UART3_BASE + UART_FR_R_OFFSET), &initStatusArray[3] },
    { UART4_BASE, REGISTER_CAST(UART4_BASE + UART_FR_R_OFFSET), &initStatusArray[4] },
    { UART5_BASE, REGISTER_CAST(UART5_BASE + UART_FR_R_OFFSET), &initStatusArray[5] },
    { UART6_BASE, REGISTER_CAST(UART6_BASE + UART_FR_R_OFFSET), &initStatusArray[6] },
    { UART7_BASE, REGISTER_CAST(UART7_BASE + UART_FR_R_OFFSET), &initStatusArray[7] }
};              // clang-format on
```

12 Data Structure Documentation

12.1 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

12.2 GpioPort_t Struct Reference

Data Fields

- uint32_t **BASE_ADDRESS**
- uint32_t **DATA_REGISTER**
- bool ∗ **isInit**

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

- GPIO.c

12.3 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

12.4 Spi_t Struct Reference

Data Fields

- const uint32_t **BASE_ADDRESS**
- volatile uint32_t ∗const **DATA_REGISTER**
- volatile uint32_t ∗const **STATUS_REGISTER**
- volatile uint32_t ∗ **gpioDataRegister**
- GpioPin_t **gpioDataCommPin**
- uint8_t **dataSize**
- bool **isEnabled**
- bool **isInit**

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

- SPI.c

12.5 Timer_t Struct Reference

Data Fields

- timerName_t **name**
- uint32_t **baseAddress**
- register_t **controlRegister**
- register_t **intervalLoadRegister**
- register_t **interruptClearRegister**
- bool ∗ **isInit**

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

- Timer.c

12.6 Uart_t Struct Reference

Data Fields

- uint32_t **BASE_ADDRESS**
- register_t **FLAG_REGISTER**
- bool ∗ **isInitPtr**

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

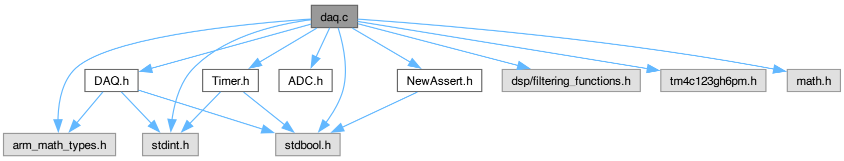
- UART.c

13 File Documentation

13.1 daq.c File Reference

Source code for DAQ module.

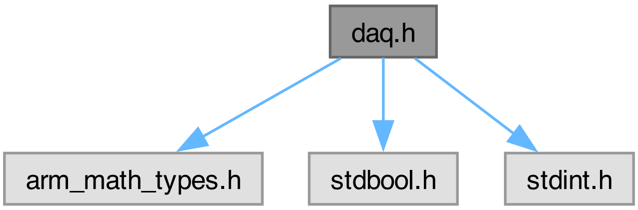
Include dependency graph for daq.c:



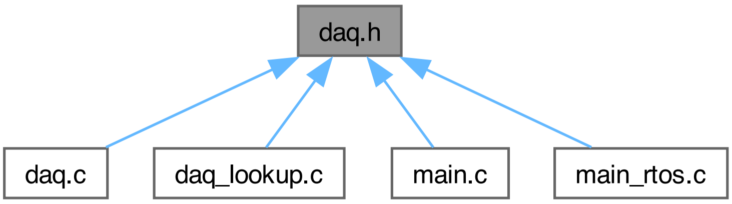
13.2 daq.h File Reference

Application software for handling data acquision (DAQ) functions.

Include dependency graph for daq.h:



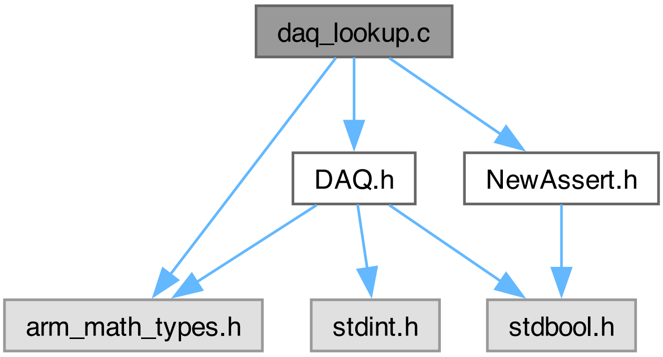
This graph shows which files directly or indirectly include this file:



13.3 daq_lookup.c File Reference

Source code for DAQ module's lookup table.

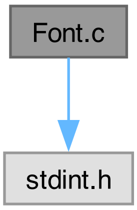
Include dependency graph for daq_lookup.c:



13.4 Font.c File Reference

Contains bitmaps for a selection of ASCII characters.

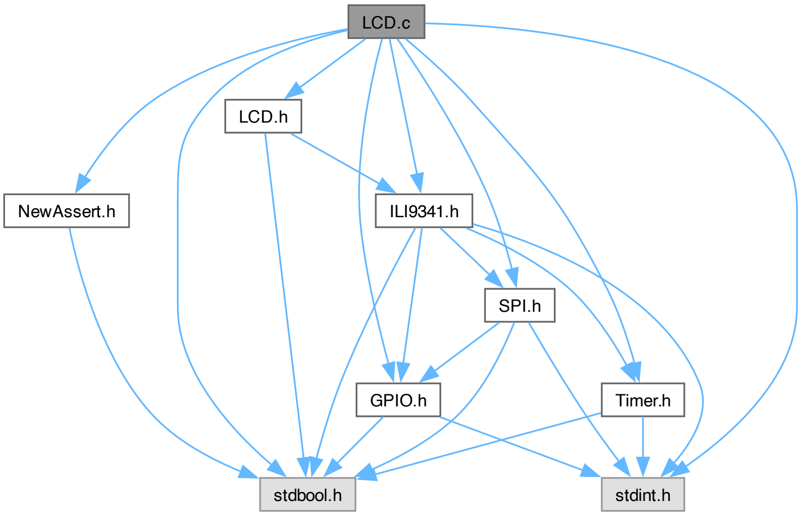
Include dependency graph for Font.c:



13.5 LCD.c File Reference

Source code for LCD module.

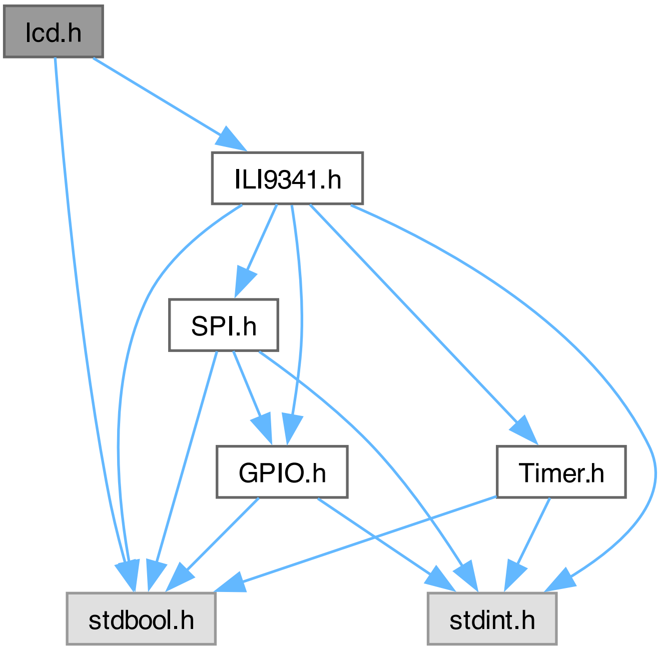
Include dependency graph for LCD.c:



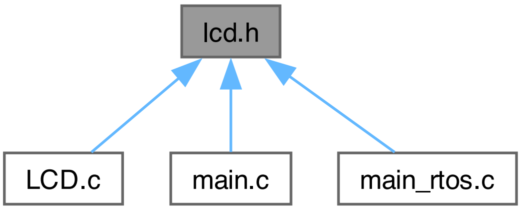
13.6 lcd.h File Reference

Header file for LCD module.

Include dependency graph for lcd.h:



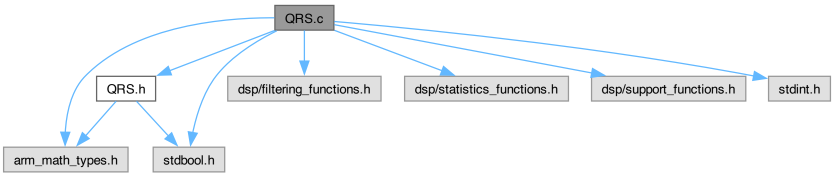
This graph shows which files directly or indirectly include this file:



13.7 QRS.c File Reference

Source code for QRS detection module.

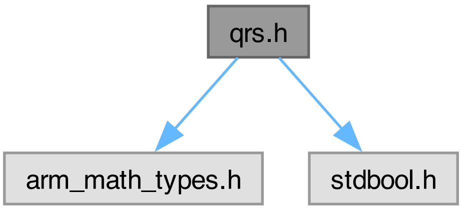
Include dependency graph for QRS.c:



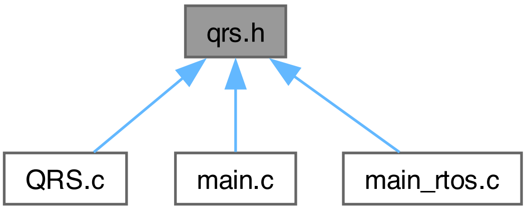
13.8 qrs.h File Reference

Header file for QRS detection module.

Include dependency graph for qrs.h:



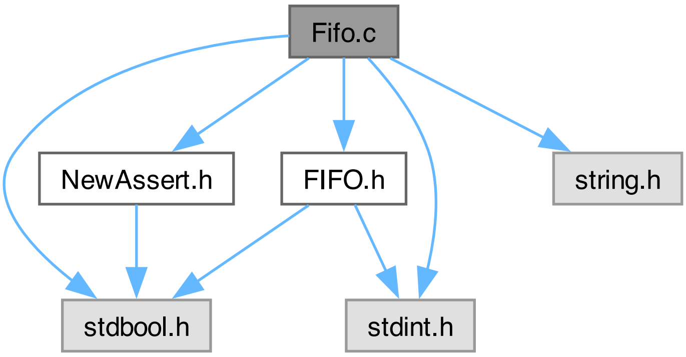
This graph shows which files directly or indirectly include this file:



13.9 Fifo.c File Reference

Source code for FIFO buffer module.

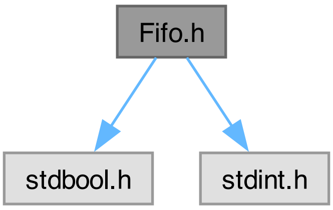
Include dependency graph for Fifo.c:



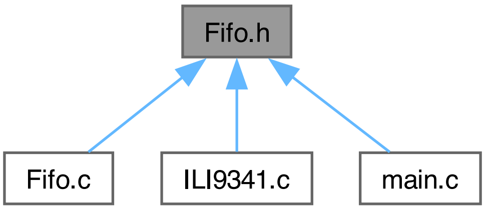
13.10 Fifo.h File Reference

Header file for FIFO buffer implementation.

Include dependency graph for Fifo.h:



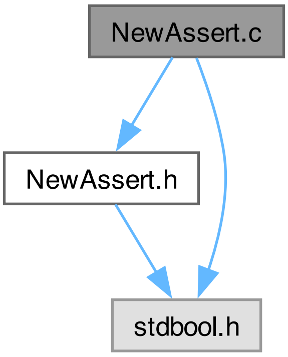
This graph shows which files directly or indirectly include this file:



13.11 NewAssert.c File Reference

Source code for custom `assert` implementation.

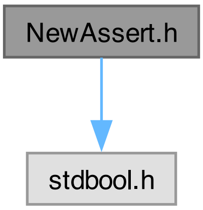
Include dependency graph for NewAssert.c:



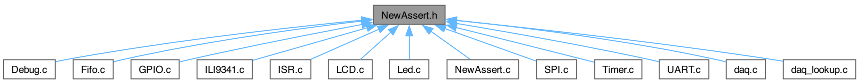
13.12 NewAssert.h File Reference

Header file for custom `assert` implementation.

Include dependency graph for NewAssert.h:



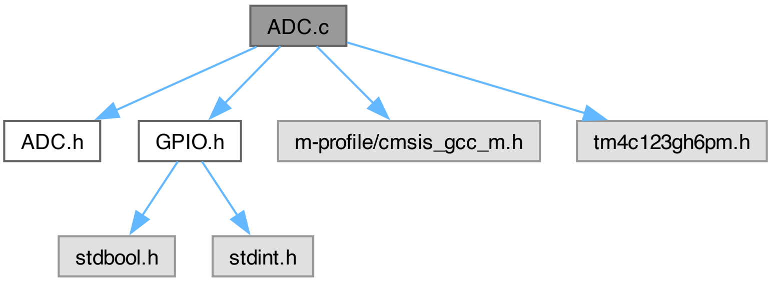
This graph shows which files directly or indirectly include this file:



13.13 ADC.c File Reference

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

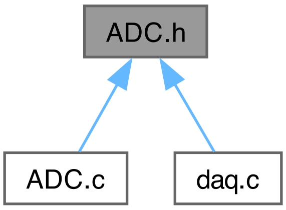
Include dependency graph for ADC.c:



13.14 ADC.h File Reference

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

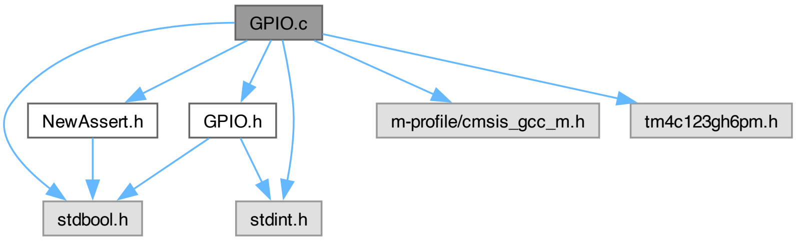
This graph shows which files directly or indirectly include this file:



13.15 GPIO.c File Reference

Source code for GPIO module.

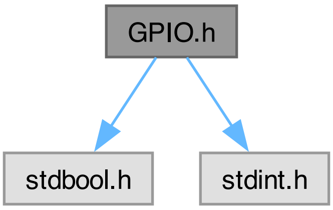
Include dependency graph for GPIO.c:



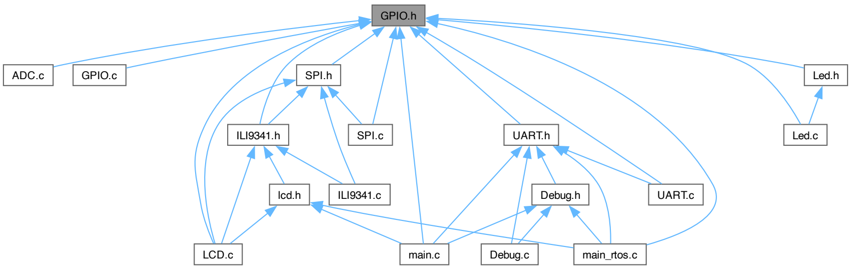
13.16 GPIO.h File Reference

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

Include dependency graph for GPIO.h:



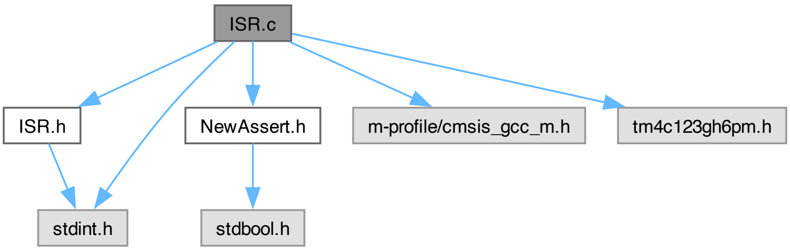
This graph shows which files directly or indirectly include this file:



13.17 ISR.c File Reference

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

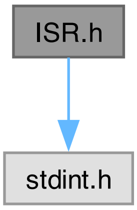
Include dependency graph for ISR.c:



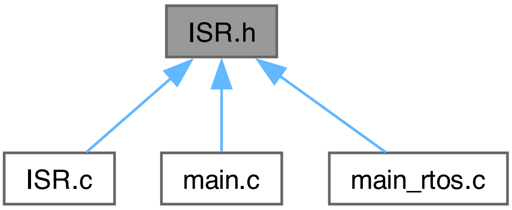
13.18 ISR.h File Reference

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

Include dependency graph for ISR.h:



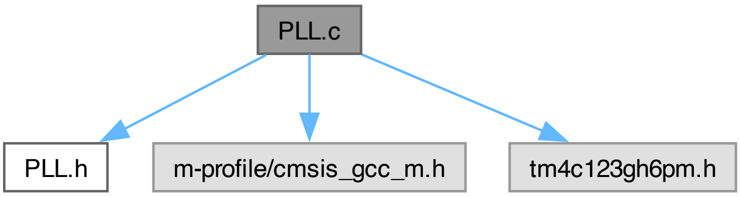
This graph shows which files directly or indirectly include this file:



13.19 PLL.c File Reference

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

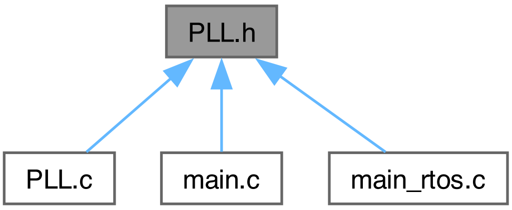
Include dependency graph for PLL.c:



13.20 PLL.h File Reference

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

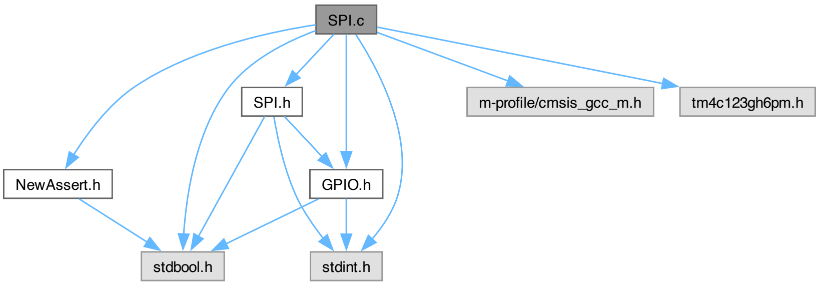
This graph shows which files directly or indirectly include this file:



13.21 SPI.c File Reference

Source code for serial peripheral interface (SPI) module.

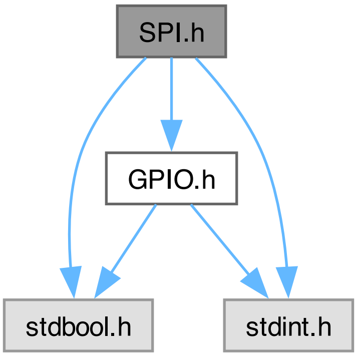
Include dependency graph for SPI.c:



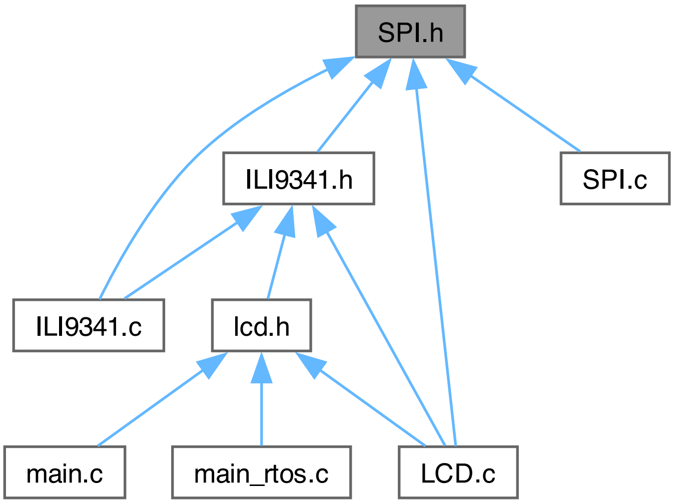
13.22 SPI.h File Reference

Header file for serial peripheral interface (SPI) module.

Include dependency graph for SPI.h:



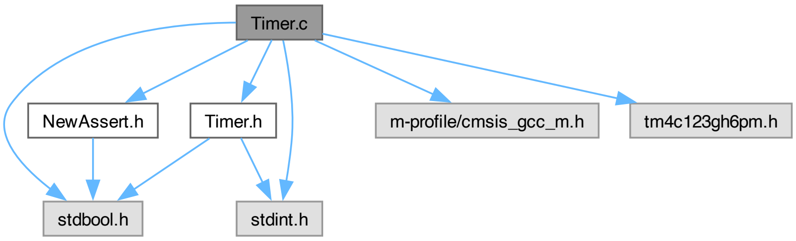
This graph shows which files directly or indirectly include this file:



13.23 Timer.c File Reference

Source code for Timer module.

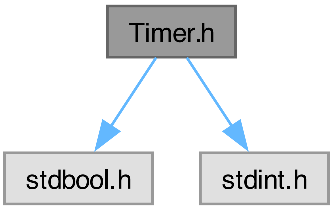
Include dependency graph for Timer.c:



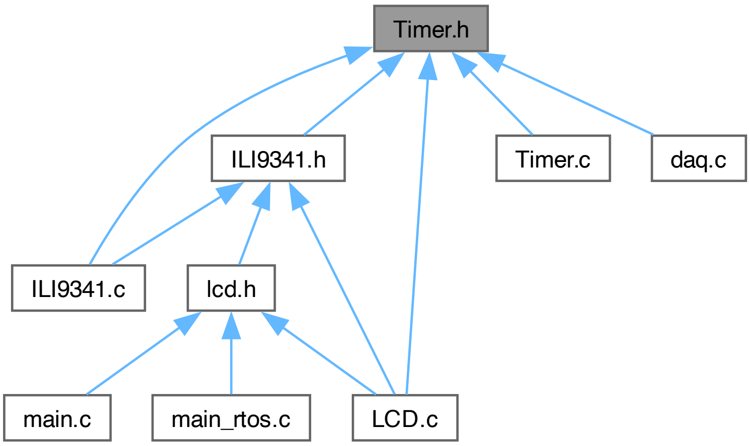
13.24 Timer.h File Reference

Device driver for general-purpose timer modules.

Include dependency graph for Timer.h:



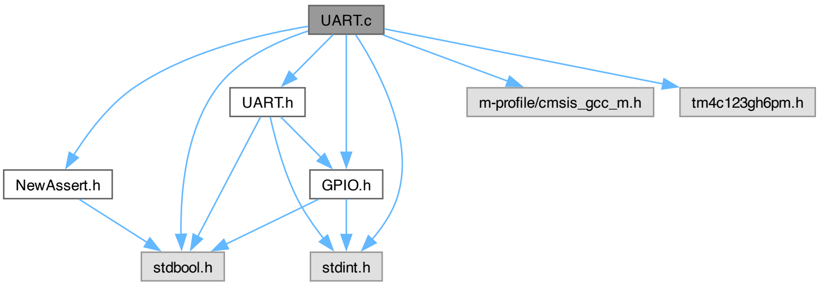
This graph shows which files directly or indirectly include this file:



13.25 UART.c File Reference

Source code for UART module.

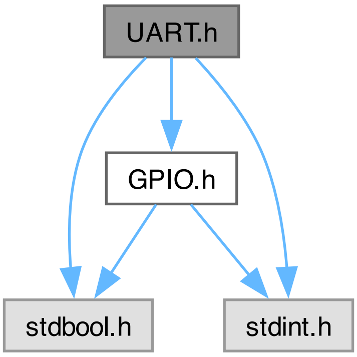
Include dependency graph for UART.c:



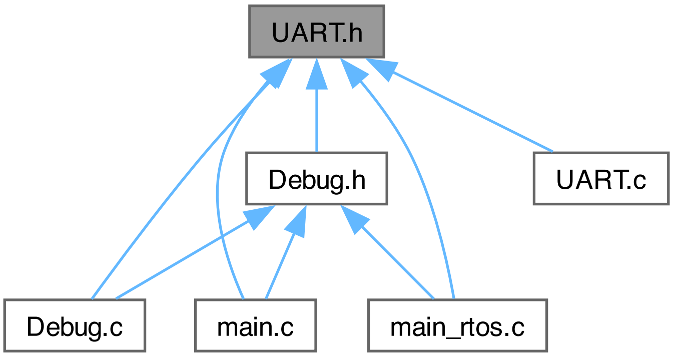
13.26 UART.h File Reference

Driver module for serial communication via UART0 and UART 1.

Include dependency graph for UART.h:



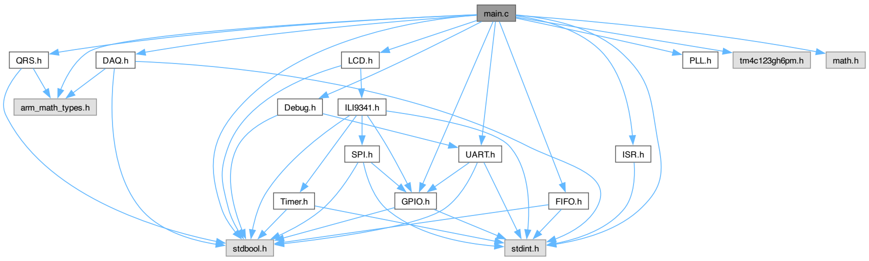
This graph shows which files directly or indirectly include this file:



13.27 main.c File Reference

Main program file (bare-metal implementation).

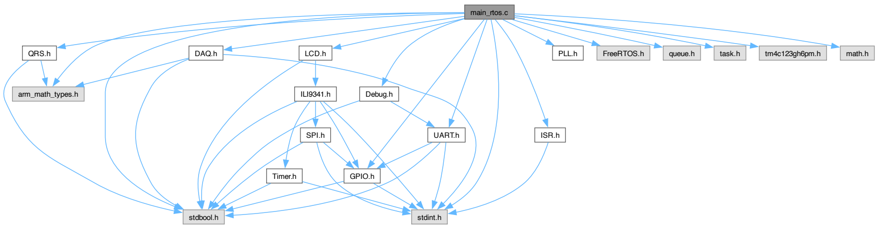
Include dependency graph for main.c:



13.28 main_rtos.c File Reference

Main program file (RTOS implementation).

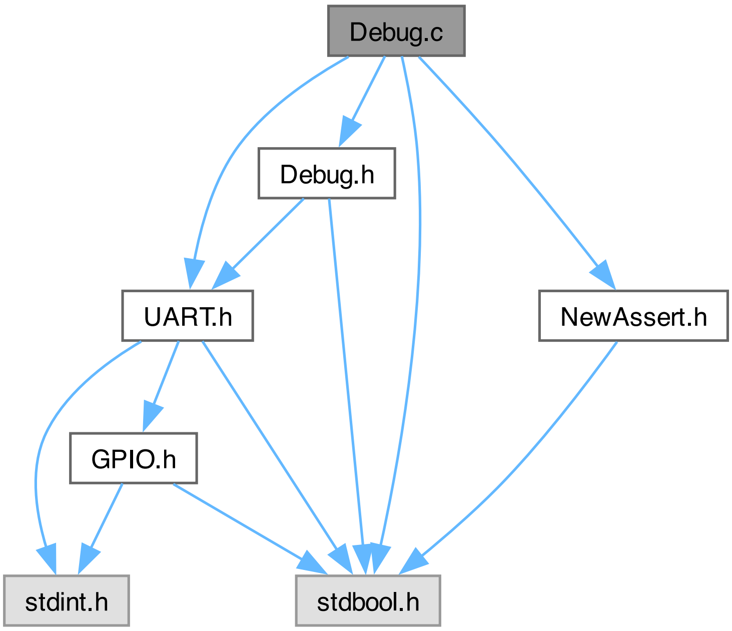
Include dependency graph for main_rtos.c:



13.29 Debug.c File Reference

Source code for Debug module.

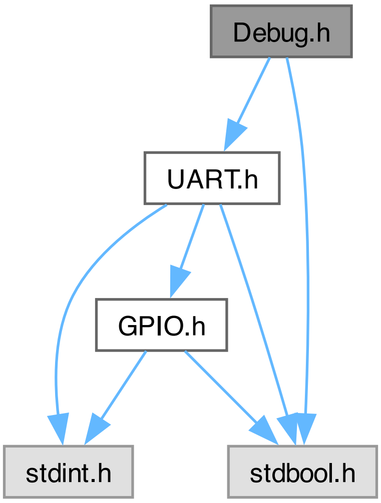
Include dependency graph for Debug.c:



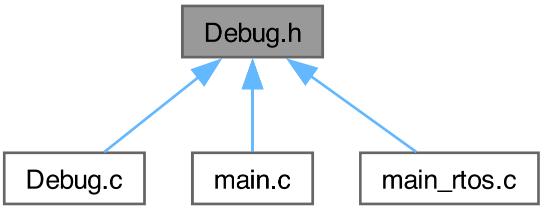
13.30 Debug.h File Reference

Header file for Debug module.

Include dependency graph for Debug.h:



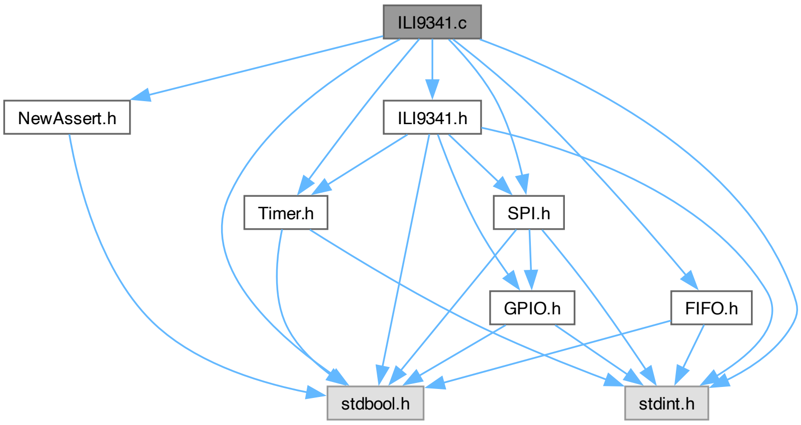
This graph shows which files directly or indirectly include this file:



13.31 ILI9341.c File Reference

Source code for ILI9341 module.

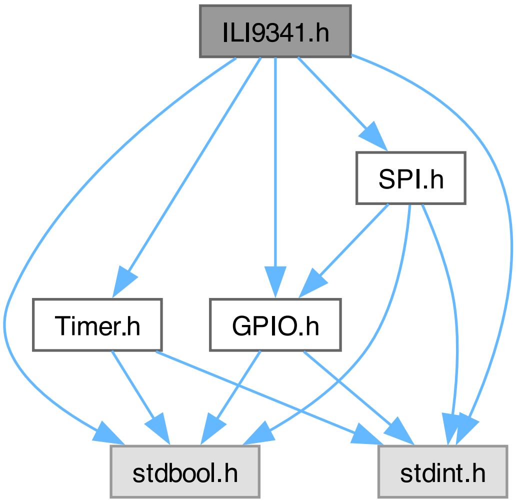
Include dependency graph for ILI9341.c:



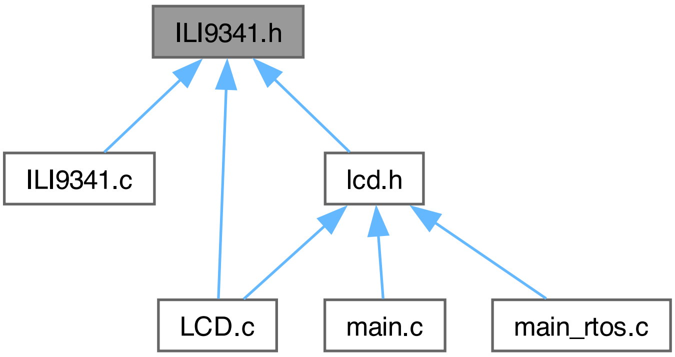
13.32 ILI9341.h File Reference

Driver module for interfacing with an ILI9341 LCD driver.

Include dependency graph for ILI9341.h:



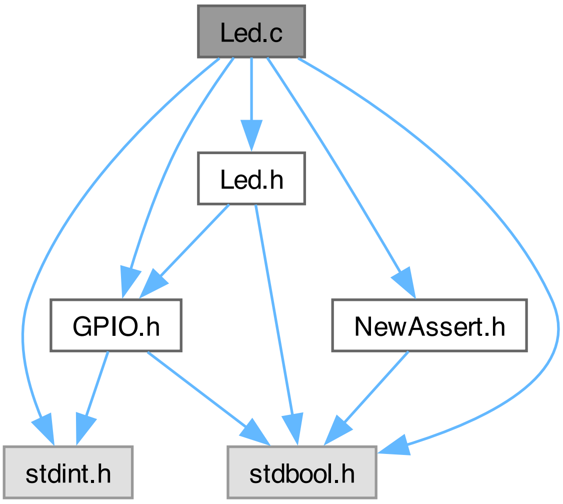
This graph shows which files directly or indirectly include this file:



13.33 Led.c File Reference

Source code for LED module.

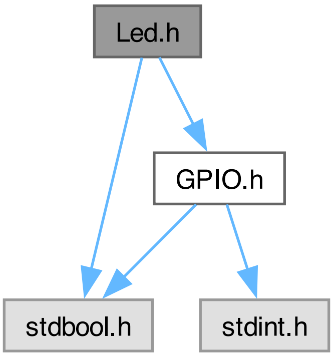
Include dependency graph for Led.c:



13.34 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: