

ECG-HRM

Generated by Doxygen 1.9.7

1 Module Index	1
1.1 Modules	1
2 Data Structure Index	2
2.1 Data Structures	2
3 File Index	2
3.1 File List	2
4 Module Documentation	4
4.1 Device Drivers	4
4.1.1 Detailed Description	4
4.1.2 Analog-to-Digital Conversion (ADC)	5
4.1.3 GPIO	6
4.1.4 ILI9341	8
4.1.5 Phase-Locked Loop (PLL)	17
4.1.6 Serial Peripheral Interface (SPI)	17
4.1.7 System Tick (SysTick)	21
4.1.8 Timer	22
4.1.9 Universal Asynchronous Receiver/Transmitter (UART)	25
4.2 Application Software	30
4.2.1 Detailed Description	31
4.2.2 Data Acquisition (DAQ)	31
4.2.3 Debug	32
4.2.4 LCD	32
4.2.5 QRS	37
4.2.6 UserCtrl	38
4.3 Program Threads	38
4.3.1 Detailed Description	38
4.3.2 Function Documentation	38
4.4 Fifo	39
4.4.1 Detailed Description	40
4.4.2 Function Documentation	40
5 Data Structure Documentation	43
5.1 FIFO_t Struct Reference	43
5.2 LCD_t Struct Reference	44
6 File Documentation	44
6.1 DAQ.c File Reference	44
6.1.1 Detailed Description	45
6.2 DAQ.h File Reference	45
6.2.1 Detailed Description	45
6.3 Debug.h File Reference	46

6.3.1 Detailed Description	46
6.3.2 Function Documentation	46
6.4 LCD.c File Reference	47
6.4.1 Detailed Description	48
6.5 LCD.h File Reference	48
6.5.1 Detailed Description	49
6.6 QRS.h File Reference	50
6.6.1 Detailed Description	50
6.7 UserCtrl.h File Reference	50
6.7.1 Detailed Description	50
6.7.2 Function Documentation	50
6.8 FIFO.c File Reference	51
6.8.1 Detailed Description	51
6.9 FIFO.h File Reference	52
6.9.1 Detailed Description	52
6.10 lookup.c File Reference	53
6.10.1 Detailed Description	53
6.10.2 Function Documentation	53
6.11 lookup.h File Reference	53
6.11.1 Detailed Description	54
6.11.2 Function Documentation	54
6.12 ADC.c File Reference	54
6.12.1 Detailed Description	54
6.13 ADC.h File Reference	55
6.13.1 Detailed Description	55
6.14 GPIO.c File Reference	55
6.14.1 Detailed Description	56
6.15 GPIO.h File Reference	56
6.15.1 Detailed Description	56
6.16 ILI9341.c File Reference	57
6.16.1 Detailed Description	58
6.17 ILI9341.h File Reference	59
6.17.1 Detailed Description	60
6.18 PLL.c File Reference	60
6.18.1 Detailed Description	60
6.19 PLL.h File Reference	61
6.19.1 Detailed Description	61
6.20 SPI.c File Reference	61
6.20.1 Detailed Description	62
6.21 SPI.h File Reference	62
6.21.1 Detailed Description	63
6.22 SysTick.c File Reference	63

6.22.1 Detailed Description	63
6.23 SysTick.h File Reference	63
6.23.1 Detailed Description	64
6.24 Timer.c File Reference	64
6.24.1 Detailed Description	65
6.25 Timer.h File Reference	65
6.25.1 Detailed Description	66
6.26 UART.c File Reference	66
6.26.1 Detailed Description	67
6.27 UART.h File Reference	67
6.27.1 Detailed Description	68
6.28 main.c File Reference	68
6.28.1 Detailed Description	68
Index	69

1 Module Index

1.1 Modules

Here is a list of all modules:

Device Drivers	4
Analog-to-Digital Conversion (ADC)	5
GPIO	6
ILI9341	8
Phase-Locked Loop (PLL)	17
Serial Peripheral Interface (SPI)	17
System Tick (SysTick)	21
Timer	22
Universal Asynchronous Receiver/Transmitter (UART)	25
Application Software	30
Data Acquisition (DAQ)	31
Debug	32
LCD	32
QRS	37
UserCtrl	38
Program Threads	38

Fifo	39
------	----

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

FIFO_t	43
LCD_t	44

3 File Index

3.1 File List

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

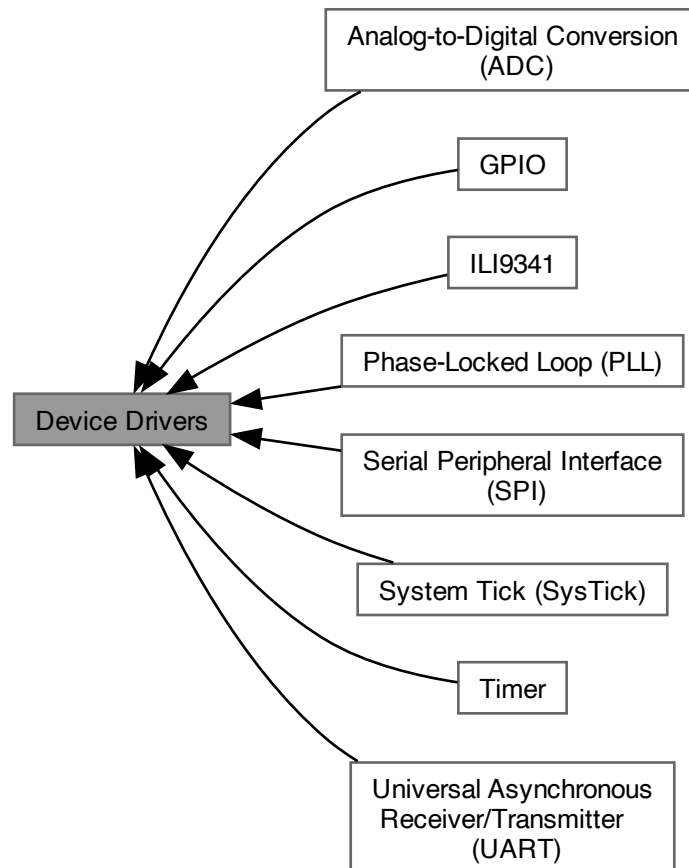
DAQ.c		44
Source code for DAQ module		
DAQ.h		45
Application software for handling data acquisition (DAQ) functions		
Debug.h		46
Functions to output debugging information to a serial port via UART		
LCD.c		47
Source code for LCD module		
LCD.h		48
Module for outputting the ECG waveform and HR to a liquid crystal display (LCD)		
QRS.h		50
QRS detection algorithm functions		
UserCtrl.h		50
Interface for user control module		
FIFO.c		51
Source code for FIFO buffer module		
FIFO.h		52
FIFO buffer data structure		
lookup.c		53
Lookup table source code		
lookup.h		53
Lookup table API		
ADC.c		54
Source code for ADC module		

ADC.h	
Driver module for analog-to-digital conversion (ADC)	55
GPIO.c	
Source code for GPIO module	55
GPIO.h	
Driver module for using the LaunchPad's onboard switches and RGB LEDs for GPIO and interrupts	56
ILI9341.c	
Source code for ILI9341 module	57
ILI9341.h	
Driver module for interfacing with an ILI9341 LCD driver	59
PLL.c	
Implementation details for phase-lock-loop (PLL) functions	60
PLL.h	
Driver module for activating the phase-locked-loop (PLL)	61
SPI.c	
Source code for SPI module	61
SPI.h	
Driver module for using the serial peripheral interface (SPI) protocol	62
SysTick.c	
Implementation details for SysTick functions	63
SysTick.h	
Driver module for using SysTick-based timing and/or interrupts	63
Timer.c	
Implementation for timer module	64
Timer.h	
Driver module for general-purpose timer modules	65
UART.c	
Source code for UART module	66
UART.h	
Driver module for serial communication via UART0 and UART 1	67
main.c	
Main program file for ECG-HRM	68

4 Module Documentation

4.1 Device Drivers

Collaboration diagram for Device Drivers:



Modules

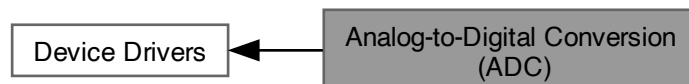
- [Analog-to-Digital Conversion \(ADC\)](#)
- [GPIO](#)
- [ILI9341](#)
- [Phase-Locked Loop \(PLL\)](#)
- [Serial Peripheral Interface \(SPI\)](#)
- [System Tick \(SysTick\)](#)
- [Timer](#)
- [Universal Asynchronous Receiver/Transmitter \(UART\)](#)

4.1.1 Detailed Description

Device driver modules.

4.1.2 Analog-to-Digital Conversion (ADC)

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



Files

- file [ADC.c](#)
Source code for ADC module.
- file [ADC.h](#)
Driver module for analog-to-digital conversion (ADC).

Functions

- void **ADC_Init** (void)
Initialize ADC0 as a differential-input analog-to-digital converter.
- void **ADC_InterruptEnable** (void)
Enable the ADC interrupt.
- void **ADC_InterruptDisable** (void)
Disable the ADC interrupt.
- volatile float32_t **ADC_ConvertToVolts** (uint16_t raw_sample)
Convert a raw ADC sample to voltage in [mV].

4.1.2.1 Detailed Description

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

4.1.2.2 Function Documentation

ADC_ConvertToVolts()

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

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

Parameters

<i>raw_sample</i>	12-bit unsigned ADC value. sample = [0, 0xFFF]
-------------------	--

Returns

double Voltage value in range $[-3.3, 3.3)$ [mV].

4.1.3 GPIO

Collaboration diagram for GPIO:



Files

- file [GPIO.c](#)
Source code for GPIO module.
- file [GPIO.h](#)
Driver module for using the LaunchPad's onboard switches and RGB LEDs for GPIO and interrupts.

Macros

- `#define LED_RED (uint8_t) 0x02`
- `#define LED_GREEN (uint8_t) 0x08`
- `#define LED_BLUE (uint8_t) 0x04`
- `#define LED_YELLOW (LED_RED + LED_GREEN)`
- `#define LED_CYAN (LED_BLUE + LED_GREEN)`
- `#define LED_PURPLE (LED_RED + LED_BLUE)`
- `#define LED_WHITE (LED_RED + LED_BLUE + LED_GREEN)`

Functions

- void [GPIO_PF_Init](#) (void)
Initialize GPIO Port F.
- void [GPIO_PF_LED_Init](#) (void)
Initialize PF1-3 to interface the LaunchPad's onboard RGB LED.
- void [GPIO_PF_LED_Write](#) (uint8_t color_mask, uint8_t on_or_off)
Write a 1 or 0 to the selected LED(s).
- void [GPIO_PF_LED_Toggle](#) (uint8_t color_mask)
Toggle the selected LED(s).
- void [GPIO_PF_Sw_Init](#) (void)
Initialize PF0/4 to interface the LaunchPad's onboard switches. PF4 is Sw1, and PF0 is Sw2.
- void [GPIO_PF_Interrupt_Init](#) (void)
Initialize GPIO Port F interrupts via Sw1 and Sw2.

4.1.3.1 Detailed Description

Functions for interfacing the LaunchPad's RGB LEDs (PF1-3) and switches (PF0/4).

4.1.3.2 Function Documentation

GPIO_PF_Init()

```
void GPIO_PF_Init (
    void )
```

Initialize GPIO Port F.

GPIO_PF_Interrupt_Init()

```
void GPIO_PF_Interrupt_Init (
    void )
```

Initialize GPIO Port F interrupts via Sw1 and Sw2.

GPIO_PF_LED_Init()

```
void GPIO_PF_LED_Init (
    void )
```

Initialize PF1-3 to interface the LaunchPad's onboard RGB LED.

GPIO_PF_LED_Toggle()

```
void GPIO_PF_LED_Toggle (
    uint8_t color_mask )
```

Toggle the selected LED(s).

Parameters

<i>color_mask</i>	Hex. number of LED pin(s) to write to. 0x02 (PF1) – RED; 0x04 (PF2) – BLUE; 0x08 (PF3) – GREEN
-------------------	--

GPIO_PF_LED_Write()

```
void GPIO_PF_LED_Write (
    uint8_t color_mask,
    uint8_t on_or_off )
```

Write a 1 or 0 to the selected LED(s).

Parameters

<i>color_mask</i>	Hex. number of LED pin(s) to write to. 0x02 (PF1) – RED; 0x04 (PF2) – BLUE; 0x08 (PF3) – GREEN
<i>on_or_off</i>	=0 for OFF, >=1 for ON

GPIO_PF_Sw_Init()

```
void GPIO_PF_Sw_Init (
    void )
```

Initialize PF0/4 to interface the LaunchPad's onboard switches. PF4 is Sw1, and PF0 is Sw2.

4.1.4 ILI9341

Collaboration diagram for ILI9341:

**Files**

- file [ILI9341.c](#)
Source code for ILI9341 module.
- file [ILI9341.h](#)
Driver module for interfacing with an ILI9341 LCD driver.

Macros

- **#define CMD_NOP** (uint8_t) 0x00
- **#define CMD_SWRESET** (uint8_t) 0x01
- **#define CMD_SPLIN** (uint8_t) 0x10
- **#define CMD_SPLOUT** (uint8_t) 0x11
- **#define CMD_PTLON** (uint8_t) 0x12
- **#define CMD_NORON** (uint8_t) 0x13
- **#define CMD_DINVOFF** (uint8_t) 0x20
- **#define CMD_DINVON** (uint8_t) 0x21
- **#define CMD_CASET** (uint8_t) 0x2A
- **#define CMD_PASET** (uint8_t) 0x2B
- **#define CMD_RAMWR** (uint8_t) 0x2C
- **#define CMD_DISPOFF** (uint8_t) 0x28
- **#define CMD_DISPON** (uint8_t) 0x29

- `#define CMD_PLTAR (uint8_t) 0x30`
- `#define CMD_VSCRDEF (uint8_t) 0x33`
- `#define CMD_MADCTL (uint8_t) 0x36`
- `#define CMD_VSCRSADD (uint8_t) 0x37`
- `#define CMD_IDMOFF (uint8_t) 0x38`
- `#define CMD_IDMON (uint8_t) 0x39`
- `#define CMD_PIXSET (uint8_t) 0x3A`
- `#define CMD_FRMCTR1 (uint8_t) 0xB1`
- `#define CMD_FRMCTR2 (uint8_t) 0xB2`
- `#define CMD_FRMCTR3 (uint8_t) 0xB3`
- `#define CMD_PRCTR (uint8_t) 0xB5`
- `#define CMD_IFCTL (uint8_t) 0xF6`
- `#define NUM_COLS (uint16_t) 240`
- `#define NUM_ROWS (uint16_t) 320`

Functions

- `void ILI9341_Init (void)`
Initialize the LCD driver, the SPI module, and Timer2A.
- `void ILI9341_resetHard (void)`
Perform a hardware reset of the LCD driver.
- `void ILI9341_resetSoft (void)`
Perform a software reset of the LCD driver.
- `void ILI9341_setSleepMode (bool is_sleeping)`
Enter or exit sleep mode. The LCD driver is in sleep mode by default upon powering on or either kind of reset.
- `void ILI9341_setDispMode (bool is_normal, bool is_full_colors)`
Set the display area and color expression.
- `void ILI9341_setPartialArea (uint16_t rowStart, uint16_t rowEnd)`
Set the partial display area for partial mode. Call before activating partial mode via ILI9341_setDisplayMode().
- `void ILI9341_setDispInversion (bool is_ON)`
Toggle display inversion. Turning ON causes colors to be inverted on the display.
- `void ILI9341_setDispOutput (bool is_ON)`
Turn display output ON or OFF. This function clears the display and stops outputting to the display area, but does not affect frame memory or power.
- `void ILI9341_setScrollArea (uint16_t topFixedArea, uint16_t vertScrollArea, uint16_t bottFixedArea)`
Set the vertical scrolling area of the display. The sum of the three parameters should be equal to the max number of rows NUM_ROWS = 320.
- `void ILI9341_setScrollStart (uint16_t startRow)`
Set the start row for vertical scrolling.
- `void ILI9341_setMemAccessCtrl (bool areRowsFlipped, bool areColsFlipped, bool areRowsColsSwitched, bool isVertRefreshFlipped, bool isColorOrderFlipped, bool isHorRefreshFlipped)`
Set how data is converted from memory to display.
- `void ILI9341_setColorDepth (bool is_16bit)`
Set the pixel format to be 16-bit (65K colors) or 18-bit (262K colors).
- `void ILI9341_NoOpCmd (void)`
Send the "No Operation" command (NOP = 0x00) to the LCD driver. Can be used to terminate the "Memory Write" (RAMWR) and "Memory Read" (RAMRD) commands, but does nothing otherwise.
- `void ILI9341 setFrameRateNorm (uint8_t div_ratio, uint8_t clocks_per_line)`
TODO: Write brief.
- `void ILI9341 setFrameRateIdle (uint8_t div_ratio, uint8_t clocks_per_line)`
TODO: Write brief.
- `void ILI9341_setBlankingPorch (uint8_t vpf, uint8_t vbp, uint8_t hfp, uint8_t hbp)`

TODO: Write.

- void `ILI9341_setInterface` (void)
Sets the interface for the ILI9341. The parameters for this command are hard-coded, so it only needs to be called once upon initialization.
- void `ILI9341_setRowAddress` (uint16_t start_row, uint16_t end_row)
not using backlight, so these aren't necessary
- void `ILI9341_setColAddress` (uint16_t start_col, uint16_t end_col)
Sets the start/end rows to be written to.
- void `ILI9341_writeMemCmd` (void)
Sends the "Write Memory" (RAMWR) command to the LCD driver, signalling that incoming data should be written to memory.
- void `ILI9341_write1px` (uint8_t red, uint8_t green, uint8_t blue, bool is_16bit)
Write a single pixel to frame memory.

4.1.4.1 Detailed Description

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

4.1.4.2 Function Documentation

ILI9341_resetHard()

```
void ILI9341_resetHard (  
    void )
```

Perform a hardware reset of the LCD driver.

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

ILI9341_resetSoft()

```
void ILI9341_resetSoft (  
    void )
```

Perform a software reset of the LCD driver.

the driver needs 5 [ms] before another command

ILI9341_setBlankingPorch()

```
void ILI9341_setBlankingPorch (  
    uint8_t vpf,  
    uint8_t vbp,  
    uint8_t hfp,  
    uint8_t hbp )
```

TODO: Write.

TODO: Write

ILI9341_setColAddress()

```
void ILI9341_setColAddress (
    uint16_t start_col,
    uint16_t end_col )
```

Sets the start/end rows to be written to.

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

Parameters

<i>start_col</i>	0 <= start_col <= end_col
<i>end_col</i>	start_col <= end_col < 240

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

ILI9341_setColorDepth()

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

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

Parameters

<i>is_16bit</i>	
-----------------	--

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

ILI9341_setDispInversion()

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

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

Parameters

<i>is_ON</i>	true to turn ON, false to turn OFF
--------------	------------------------------------

TODO: Write description

ILI9341_setDispMode()

```
void ILI9341_setDispMode (
```

```
bool is_normal,  
bool is_full_colors )
```

Set the display area and color expression.

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

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

Parameters

<i>is_normal</i>	true for normal mode, false for partial mode
<i>is_full_colors</i>	true for full colors, false for 8 colors

ILI9341_setDispOutput()

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

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

Parameters

<i>is_ON</i>	true to turn ON, false to turn OFF
--------------	------------------------------------

TODO: Write description

ILI9341_setFrameRateIdle()

```
void ILI9341_setFrameRateIdle (  
    uint8_t div_ratio,  
    uint8_t clocks_per_line )
```

TODO: Write brief.

TODO: Write description

ILI9341_setFrameRateNorm()

```
void ILI9341_setFrameRateNorm (  
    uint8_t div_ratio,  
    uint8_t clocks_per_line )
```

TODO: Write brief.

TODO: Write description

ILI9341_setInterface()

```
void ILI9341_setInterface (
    void )
```

Sets the interface for the ILI9341. The parameters for this command are hard-coded, so it only needs to be called once upon initialization.

This function implements the "Interface Control" `CMD_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 CMD_MADCTL bit
MX_EOR	6		flips value of corresponding CMD_MADCTL bit
MV_EOR	5		flips value of corresponding CMD_MADCTL bit
BGR_EOR	3		flips value of corresponding CMD_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 CMD_MADCTL bits (`ILI9341_setMemoryAccessCtrl()`) are unaffected and overflowing pixel data is ignored. The EPF bits are cleared so that the LSB of the R and B values is copied from the MSB when using 16-bit color depth. The TM4C123 sends the MSB first, so the ENDIAN bit is cleared. The other bits are cleared and/or irrelevant since the RGB and VSYNC interfaces aren't used.

ILI9341_setMemAccessCtrl()

```
void ILI9341_setMemAccessCtrl (
    bool areRowsFlipped,
    bool areColsFlipped,
    bool areRowsColsSwitched,
    bool isVertRefreshFlipped,
    bool isColorOrderFlipped,
    bool isHorRefreshFlipped )
```

Set how data is converted from memory to display.

Parameters

<i>areRowsFlipped</i>	
<i>areColsFlipped</i>	
<i>areRowsColsSwitched</i>	
<i>isVertRefreshFlipped</i>	
<i>isColorOrderFlipped</i>	
<i>isHorRefreshFlipped</i>	

This function implements the "Memory Access Control" (`CMD_MADCTL`) command from p. 127-128 of the ILI9341

datasheet, which controls how the LCD driver displays data upon writing to memory.

Name	Bit #	Effect when set = 1
MY	7	flip row (AKA "page") addresses
MX	6	flip column addresses
MV	5	exchange rows and column addresses
ML	4	reverse horizontal refresh order
BGR	3	reverse color input order (RGB -> BGR)
MH	2	reverse vertical refresh order

All bits are clear after powering on or HWRESET.

ILI9341_setPartialArea()

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

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

Parameters

<i>rowStart</i>	
<i>rowEnd</i>	

ILI9341_setRowAddress()

```
void ILI9341_setRowAddress (
    uint16_t start_row,
    uint16_t end_row )
```

not using backlight, so these aren't necessary

Sets the start/end rows to be written to.

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

Parameters

<i>start_row</i>	$0 \leq \text{start_row} \leq \text{end_row}$
<i>end_row</i>	$\text{start_row} \leq \text{end_row} < 320$

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

ILI9341_setScrollArea()

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

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

Parameters

<i>topFixedArea</i>	Number of rows fixed at the top of the screen.
<i>vertScrollArea</i>	Number of rows that scroll.
<i>bottFixedArea</i>	Number of rows fixed at the bottom of the screen.

ILI9341_setScrollStart()

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

Set the start row for vertical scrolling.

Parameters

<i>startRow</i>	Start row for scrolling. Should be $\geq \text{topFixedArea} - 1$
-----------------	---

ILI9341_setSleepMode()

```
void ILI9341_setSleepMode (
    bool is_sleeping )
```

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

Parameters

<i>is_sleeping</i>	true to enter sleep mode, false to exit
--------------------	---

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

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

ILI9341_write1px()

```
void ILI9341_write1px (
    uint8_t red,
```

```
uint8_t green,
uint8_t blue,
bool is_16bit )
```

Write a single pixel to frame memory.

Call `ILI9341_writeMemCmd()` before this one.

Parameters

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

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

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

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

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

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

ILI9341_writeMemCmd()

```
void ILI9341_writeMemCmd (
    void )
```

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

Should be called after setting the row ([ILI9341_setRowAddress\(\)](#)) and/or and/or column ([ILI9341_setRowAddress\(\)](#)) addresses, but before writing image data ([ILI9341_writelpx\(\)](#)).

4.1.5 Phase-Locked Loop (PLL)

Collaboration diagram for Phase-Locked Loop (PLL):



Files

- file [PLL.c](#)
Implementation details for phase-lock-loop (PLL) functions.
- file [PLL.h](#)
Driver module for activating the phase-locked-loop (PLL).

Functions

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

4.1.5.1 Detailed Description

Function for initializing the phase-locked loop.

4.1.5.2 Function Documentation

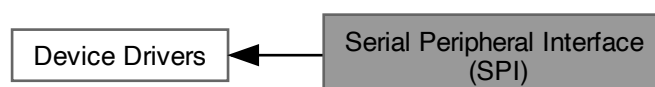
PLL_Init()

```
void PLL_Init (  
    void )
```

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

4.1.6 Serial Peripheral Interface (SPI)

Collaboration diagram for Serial Peripheral Interface (SPI):



Files

- file [SPI.c](#)
Source code for SPI module.
- file [SPI.h](#)
Driver module for using the serial peripheral interface (SPI) protocol.

Macros

- `#define NVIC_SSI0_NUM 7`
- `#define SPI_INT_START() (NVIC_SW_TRIG_R = (NVIC_SW_TRIG_R & ~(0xFF)) | NVIC_SSI0_NUM)`
- `#define SPI_SET_DC() (GPIO_PORTA_DATA_R |= 0x40)`
- `#define SPI_CLEAR_DC() (GPIO_PORTA_DATA_R &= ~(0x40))`
- `#define SPI_IS_BUSY (SSI0_SR_R & 0x10)`
- `#define SPI_TX_ISNOTFULL ((bool) (SSI0_SR_R & 0x02))`
- `#define SPI_BUFFER_SIZE 9`

Functions

- void [SPI_Init](#) (void)
Initialize SSI0 to act as an SPI Controller (AKA Master) in mode 0.
- uint8_t [SPI_Read](#) (void)
Read data from the peripheral.
- void [SPI_WriteCmd](#) (uint8_t cmd)
Write an 8-bit command to the peripheral.
- void [SPI_WriteData](#) (uint8_t data)
Write 8-bit data to the peripheral.
- void [SPI_IRQ_WriteCmd](#) (uint8_t cmd)
Add an 8-bit command to the SPI queue. If no data or other command is written, should directly precede a call to [SPI_IRQ_StartWriting\(\)](#).
- void [SPI_IRQ_WriteData](#) (uint8_t data)
Add 8-bit data to the SPI queue. Should directly precede either another call to the same function or a call to [SPI_IRQ_StartWriting\(\)](#).
- void [SPI_IRQ_StartWriting](#) (void)
Start writing data to the Tx FIFO. Should be used after 1+ calls to [SPI_IRQ_WriteCmd\(\)](#) and/or [SPI_IRQ_WriteData\(\)](#). If unused, writing will start when the SPI queue is full.
- void [SSI0_Handler](#) (void)
Sends parameters (data or commands) over SPI via SSI0.

4.1.6.1 Detailed Description

Functions for SPI-based communication via SSI0 peripheral.

4.1.6.2 Macro Definition Documentation

NVIC_SSI0_NUM

```
#define NVIC_SSI0_NUM 7
```

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

Clk. Polarity = steady state low (0)

Clk. Phase = rising clock edge (0)

4.1.6.3 Function Documentation

SPI_Init()

```
void SPI_Init (
    void )
```

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

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

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

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

SPI_IRQ_WriteCmd()

```
void SPI_IRQ_WriteCmd (
    uint8_t cmd )
```

Add an 8-bit command to the SPI queue. If no data or other command is written, should directly precede a call to [SPI_IRQ_StartWriting\(\)](#).

Parameters

<i>cmd</i>	command for peripheral
------------	------------------------

SPI_IRQ_WriteData()

```
void SPI_IRQ_WriteData (
    uint8_t data )
```

Add 8-bit data to the SPI queue. Should directly precede either another call to the same function or a call to [SPI_IRQ_StartWriting\(\)](#).

Parameters

<i>data</i>	input data for peripheral
-------------	---------------------------

SPI_Read()

```
uint8_t SPI_Read (
    void )
```

Read data from the peripheral.

Returns

uint8_t

SPI_WriteCmd()

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

Write an 8-bit command to the peripheral.

Parameters

<i>cmd</i>	command for peripheral
------------	------------------------

SPI_WriteData()

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

Write 8-bit data to the peripheral.

Parameters

<i>data</i>	input data for peripheral
-------------	---------------------------

SSI0_Handler()

```
void SSI0_Handler (
    void )
```

Sends parameters (data or commands) over SPI via SSI0.

The interrupt is enabled by the 'SPI_Init()' function and triggered by a call to 'SPI_IRQ_StartWriting()'. The handler determines whether to signal for data or a command via the D/C pin, and then writes to the data register.

The interrupt is unpended at the start of the function.

4.1.7 System Tick (SysTick)

Collaboration diagram for System Tick (SysTick):



Files

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

Functions

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

4.1.7.1 Detailed Description

Functions for timing and periodic interrupts via SysTick.

4.1.7.2 Function Documentation

SysTick_Interrupt_Init()

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

Initialize SysTick for interrupts.

Parameters

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

SysTick_Timer_Init()

```
void SysTick_Timer_Init (
    void )
```

Initialize SysTick for timing purposes.

4.1.8 Timer

Collaboration diagram for Timer:



Files

- file [Timer.c](#)
Implementation for timer module.
- file [Timer.h](#)
Driver module for general-purpose timer modules.

Timer0A

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

Timer1A

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

Timer2A

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

4.1.8.1 Detailed Description

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

4.1.8.2 Function Documentation**Timer0A_Init()**

```
void Timer0A_Init (
    void )
```

Initialize timer 0 as 32-bit, one-shot, countdown timer.

Timer0A_isCounting()

```
uint8_t Timer0A_isCounting (
    void )
```

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

Returns

uint8_t status

Timer0A_Start()

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

Count down starting from the inputted value.

Parameters

<i>time_ms</i>	Time in [ms] to load into Timer 0. Must be <= 53 seconds.
----------------	---

Timer0A_Wait1ms()

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

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

Parameters

<i>time_ms</i>	Time in [ms] to load into Timer 0. Must be \leq 53 seconds.
----------------	---

Timer1A_Init()

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

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

Parameters

<i>time_ms</i>	Time in [ms] between interrupts. Must be \leq 53 seconds.
----------------	---

Timer2A_Init()

```
void Timer2A_Init (
    void )
```

Initialize timer 2 as 32-bit, one-shot, countdown timer.

Timer2A_isCounting()

```
uint8_t Timer2A_isCounting (
    void )
```

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

Returns

uint8_t status

Timer2A_Start()

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

Count down starting from the inputted value.

Parameters

<i>time_ms</i>	Time in [ms] to load into Timer 2. Must be ≤ 53 seconds.
----------------	---

Timer2A_Wait1ms()

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

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

Parameters

<i>time_ms</i>	Time in [ms] to load into Timer 2. Must be ≤ 53 seconds.
----------------	---

Timer3A_Init()

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

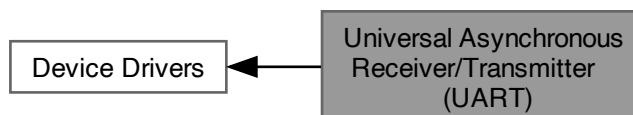
Initialize Timer3A as a 32-bit, periodic, countdown timer that triggers ADC sample capture.

Parameters

<i>time_ms</i>	Time in [ms] to load into Timer3A. Must be ≤ 53 seconds.
----------------	---

4.1.9 Universal Asynchronous Receiver/Transmitter (UART)

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

**Files**

- file [UART.c](#)
Source code for UART module.
- file [UART.h](#)
Driver module for serial communication via UART0 and UART 1.

Macros

- #define **ASCII_CONVERSION** 0x30
- #define **UART0_TX_FULL** (UART0_FR_R & 0x20)
- #define **UART0_BUFFER_SIZE** 16
- #define **UART0_INTERRUPT_NUM** 5

Functions

- void **UART0_Init** (void)
Initialize UART0 to a baud rate of 115200, 8-bit data length, 1 start bit, and 1 stop bit.
- unsigned char **UART0_ReadChar** (void)
Read a single character from UART0.
- void **UART0_WriteChar** (unsigned char input_char)
Write a single character to UART0.
- void **UART0_WriteStr** (void *input_str)
Write a C string to UART0.
- void **UART0_WriteInt** (uint32_t n)
Write a 32-bit unsigned integer to UART0.
- void **UART0_WriteFloat** (double n, uint8_t num_decimals)
Write a floating-point number to UART0.
- void **UART0_IRQ_AddChar** (unsigned char input_char)
Add a single character to UART0's FIFO.
- void **UART0_IRQ_AddStr** (void *input_str)
Add a string to UART0's FIFO.
- void **UART0_IRQ_AddInt** (uint32_t n)
Add an integer to UART0's FIFO.
- void **UART0_IRQ_Start** (void)
Transmit the UART0's FIFO's contents via interrupt.
- void **UART0_Handler** (void)
- void **UART1_Init** (void)
Initialize UART1 to a baud rate of 115200, 8-bit data length, 1 start bit, and 1 stop bit.
- unsigned char **UART1_ReadChar** (void)
Read a single character from UART1.
- void **UART1_WriteChar** (unsigned char input_char)
Write a single character to UART1.
- void **UART1_WriteStr** (void *input_str)
Write a C string to UART1.

4.1.9.1 Detailed Description

Functions for UART-based communication.

4.1.9.2 Function Documentation

UART0_Init()

```
void UART0_Init (
    void )
```

Initialize UART0 to a baud rate of 115200, 8-bit data length, 1 start bit, and 1 stop bit.

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

UART0_IRQ_AddChar()

```
void UART0_IRQ_AddChar (
    unsigned char input_char )
```

Add a single character to UART0's FIFO.

Parameters

<i>input_char</i>	ASCII character.
-------------------	------------------

UART0_IRQ_AddInt()

```
void UART0_IRQ_AddInt (
    uint32_t n )
```

Add an integer to UART0's FIFO.

Parameters

<i>n</i>	32-bit integer to be converted and transmitted.
----------	---

UART0_IRQ_AddStr()

```
void UART0_IRQ_AddStr (
    void * input_str )
```

Add a string to UART0's FIFO.

Parameters

<i>input_str</i>	(Pointer to) array of ASCII characters.
------------------	---

UART0_IRQ_Start()

```
void UART0_IRQ_Start (
    void )
```

Transmit the UART0's FIFO's contents via interrupt.

This function writes to the Software Trigger Interrupt (SWTRIG) register to activate the `UART0_Handler()` function rather than relying on the TM4C123's built-in UART0 interrupt sources.

UART0_ReadChar()

```
unsigned char UART0_ReadChar (
    void )
```

Read a single character from UART0.

Returns

`input_char`

This function uses busy-wait synchronization to read a character from UART0.

UART0_WriteChar()

```
void UART0_WriteChar (
    unsigned char input_char )
```

Write a single character to UART0.

Parameters

<code><i>input_char</i></code>	
--------------------------------	--

This function uses busy-wait synchronization to write a character to UART0.

UART0_WriteFloat()

```
void UART0_WriteFloat (
    double n,
    uint8_t num_decimals )
```

Write a floating-point number to UART0.

Parameters

<code><i>n</i></code>	Floating-point number to be converted and transmitted.
<code><i>num_decimals</i></code>	Number of digits after the decimal point to include.

UART0_WriteInt()

```
void UART0_WriteInt (
    uint32_t n )
```

Write a 32-bit unsigned integer to UART0.

Parameters

<i>n</i>	32-bit unsigned integer to be converted and transmitted
----------	---

UART0_WriteStr()

```
void UART0_WriteStr (
    void * input_str )
```

Write a C string to UART0.

Parameters

<i>input_str</i>	(Pointer to) array of ASCII characters.
------------------	---

This function uses [UART0_WriteChar\(\)](#) function to write a C string to UART0. The function writes until either the entire string has been written or a null-terminated character has been reached.

UART1_Init()

```
void UART1_Init (
    void )
```

Initialize UART1 to a baud rate of 115200, 8-bit data length, 1 start bit, and 1 stop bit.

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

NOTE: LCRH must be accessed *AFTER* setting the BRD register

UART1_ReadChar()

```
unsigned char UART1_ReadChar (
    void )
```

Read a single character from UART1.

Returns

input_char

This function uses busy-wait synchronization to read a character from UART1.

UART1_WriteChar()

```
void UART1_WriteChar (
    unsigned char input_char )
```

Write a single character to UART1.

Parameters

<i>input_char</i>	
-------------------	--

This function uses busy-wait synchronization to write a character to UART1.

UART1_WriteStr()

```
void UART1_WriteStr (
    void * input_str )
```

Write a C string to UART1.

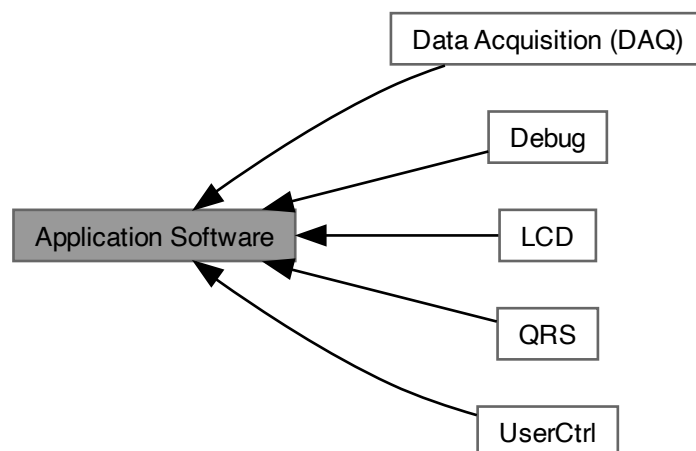
Parameters

<i>input_str</i>	C string
------------------	----------

This function uses [UART1_WriteChar\(\)](#) function to write a C string to UART1. The function writes until either the entire string has been written or a null-terminated character has been reached.

4.2 Application Software

Collaboration diagram for Application Software:



Modules

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

4.2.1 Detailed Description

Application-specific software modules.

4.2.2 Data Acquisition (DAQ)

Collaboration diagram for Data Acquisition (DAQ):



Files

- file [DAQ.c](#)
Source code for DAQ module.
- file [DAQ.h](#)
Application software for handling data acquisition (DAQ) functions.

Macros

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

Typedefs

- `typedef arm_biquad_casd_df1_inst_f32 filt_t`

Functions

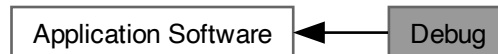
- `void DAQ_Init (void)`
- `volatile float32_t DAQ_Filter (volatile float32_t inputSample)`

4.2.2.1 Detailed Description

Module for managing data acquisition (DAQ) functions.

4.2.3 Debug

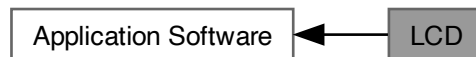
Collaboration diagram for Debug:



Module for debugging functions, including serial output and assertion.

4.2.4 LCD

Collaboration diagram for LCD:



Files

- file [LCD.c](#)
Source code for LCD module.
- file [LCD.h](#)
Module for outputting the ECG waveform and HR to a liquid crystal display (LCD).

Macros

- `#define X_MAX NUM_ROWS`
- `#define Y_MAX NUM_COLS`
- `#define LCD_BLACK (uint8_t) 0x00`
- `#define LCD_RED (uint8_t) 0x04`
- `#define LCD_GREEN (uint8_t) 0x02`
- `#define LCD_BLUE (uint8_t) 0x01`
- `#define LCD_YELLOW (uint8_t) 0x06`
- `#define LCD_CYAN (uint8_t) 0x03`
- `#define LCD_PURPLE (uint8_t) 0x05`
- `#define LCD_WHITE (uint8_t) 0x07`

Init./Config. Functions

- void **LCD_Init** (void)
Initialize the LCD driver and its internal independencies.
- void **LCD_toggleOutput** (void)
Toggle display output ON or OFF (OFF by default). Turning output OFF prevents the LCD driver from refreshing the display, which can prevent abnormalities like screen tearing while attempting to update the image.
- void **LCD_toggleInversion** (void)
Toggle color inversion ON or OFF (OFF by default).
- void **LCD_toggleColorDepth** (void)
Toggle 16-bit or 18-bit color depth (16-bit by default).

Drawing Area Definition Functions

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

Color Setting Functions

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

Drawing Functions

- void **LCD_draw** (void)
*Draw on the LCD display. Call this function after setting the drawable area via **LCD_setArea()**, or after individually calling **LCD_setX()** and/or **LCD_setY()**.*
- void **LCD_fill** (void)
Fill the display with a single color.
- void **LCD_drawHLine** (uint16_t yCenter, uint16_t lineWidth)
Draw a horizontal line across the entire display.
- void **LCD_drawVLine** (uint16_t xCenter, uint16_t lineWidth)
Draw a vertical line across the entire display.
- void **LCD_drawRectangle** (uint16_t x1, uint16_t dx, uint16_t y1, uint16_t dy, bool is_filled)
Draw a rectangle of size $dx \times dy$ onto the display. The bottom-left corner will be located at $(x1, y1)$.
- void **LCD_graphSample** (uint16_t x1, uint16_t dx, uint16_t y1, uint16_t dy, uint16_t y_min, uint16_t y_max, uint16_t color_code)
Draw a rectangle of size $dx \times dy$ and blank out all other pixels between y_min and y_max .

4.2.4.1 Detailed Description

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

4.2.4.2 Function Documentation

LCD_draw()

```
void LCD_draw (
    void )
```

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

References `LCD_t::B_val`, `LCD_t::G_val`, `ILI9341_write1px()`, `ILI9341_writeMemCmd()`, `LCD_t::is_16bit`, `LCD_t::numPixels`, and `LCD_t::R_val`.

LCD_drawHLine()

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

Draw a horizontal line across the entire display.

Parameters

<i>yCenter</i>	y-coordinate to center the line on
<i>lineWidth</i>	width of the line; should be a positive, odd number

LCD_drawRectangle()

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

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

Parameters

<i>x1</i>	lowest (left-most) x-coordinate
<i>dx</i>	length (horizontal distance) of the rectangle
<i>y1</i>	lowest (bottom-most) y-coordinate
<i>dy</i>	height (vertical distance) of the rectangle
<i>is_filled</i>	true to fill the rectangle, false to leave it unfilled

LCD_drawVLine()

```
void LCD_drawVLine (
```

```
uint16_t xCenter,
uint16_t lineWidth )
```

Draw a vertical line across the entire display.

Parameters

<i>xCenter</i>	x-coordinate to center the line on
<i>lineWidth</i>	width of the line; should be a positive, odd number

LCD_graphSample()

```
void LCD_graphSample (
    uint16_t x1,
    uint16_t dx,
    uint16_t y1,
    uint16_t dy,
    uint16_t y_min,
    uint16_t y_max,
    uint16_t color_code )
```

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

Parameters

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

TODO: Write description

LCD_Init()

```
void LCD_Init (
    void )
```

Initialize the LCD driver and its internal independencies.

LCD_setArea()

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

Set the area of the display to be written to. $0 \leq x1 \leq x2 < X_MAX$ $0 \leq y1 \leq y2 < Y_MAX$

Parameters

<i>x1_new</i>	left-most x-coordinate
<i>x2_new</i>	right-most x-coordinate
<i>y1_new</i>	lowest y-coordinate
<i>y2_new</i>	highest y-coordinate

LCD_setColor()

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

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

Parameters

<i>R_val</i>	5-bit ([0-31]) R value; 6-bit ([0-63]) if color depth is 18-bit
<i>G_val</i>	6-bit ([0-63]) G value
<i>B_val</i>	5-bit ([0-31]) B value; 6-bit ([0-63]) if color depth is 18-bit

LCD_setColor_3bit()

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

Set the color value via a 3-bit code.

Parameters

<i>color_code</i>	3-bit color value to use. Bits 2, 1, 0 correspond to R, G, and B values, respectively.
-------------------	--

This is simply a convenience function for setting the color using the macros defined in the header file.

When the display is inverted, a chosen color can be selected by subtracting it's macro from `LCD_WHITE` (e.g. to select red, the `color_code` argument should be `LCD_WHITE - LCD_RED`).

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

LCD_setX()

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

Set only new x-coordinates to be written to. $0 \leq x1 \leq x2 < X_MAX$

Parameters

<i>x1_new</i>	left-most x-coordinate
<i>x2_new</i>	right-most x-coordinate

LCD_setY()

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

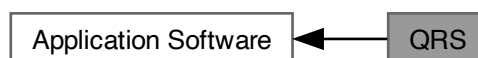
Set only new y-coordinates to be written to. $0 \leq y1 \leq y2 < Y_MAX$

Parameters

<i>y1_new</i>	lowest y-coordinate
<i>y2_new</i>	highest y-coordinate

4.2.5 QRS

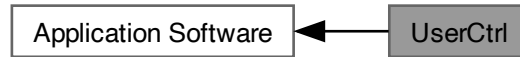
Collaboration diagram for QRS:



Module for analyzing ECG data to determine heart rate.

4.2.6 UserCtrl

Collaboration diagram for UserCtrl:



User control module.

4.3 Program Threads

Functions

- int **main** (void)
- void [GPIO_PortF_Handler](#) (void)
Interrupt service routine (ISR) for the UserCtrl module via GPIO Port F.
- void [ADC0_SS3_Handler](#) (void)
Interrupt service routine (ISR) for collecting ADC samples.
- void [Timer1A_Handler](#) (void)
Interrupt service routine (ISR) for outputting data to the LCD.

4.3.1 Detailed Description

Primary threads of execution.

4.3.2 Function Documentation

ADC0_SS3_Handler()

```
void ADC0_SS3_Handler (  
    void )
```

Interrupt service routine (ISR) for collecting ADC samples.

GPIO_PortF_Handler()

```
void GPIO_PortF_Handler (  
    void )
```

Interrupt service routine (ISR) for the UserCtrl module via GPIO Port F.

Timer1A_Handler()

```
void Timer1A_Handler (
    void )
```

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

4.4 Fifo

Files

- file [FIFO.c](#)
Source code for FIFO buffer module.
- file [FIFO.h](#)
FIFO buffer data structure.

Data Structures

- struct [FIFO_t](#)

Macros

- `#define FIFO_POOL_SIZE 5`

Functions

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

Basic Operations

- void [FIFO_Put](#) (volatile [FIFO_t](#) *fifo_ptr, const uint32_t val)
Add a value to the end of the buffer.
- volatile uint32_t [FIFO_Get](#) (volatile [FIFO_t](#) *fifo_ptr)
Remove the first value of the buffer.
- void [FIFO_TransferOne](#) (volatile [FIFO_t](#) *src_fifo_ptr, volatile [FIFO_t](#) *dest_fifo_ptr)
Transfer a value from one FIFO buffer to another.

Bulk Removal

- void [FIFO_Flush](#) (volatile [FIFO_t](#) *fifo_ptr, uint32_t output_buffer[])
Empty the FIFO buffer's contents into an array.
- void [FIFO_TransferAll](#) (volatile [FIFO_t](#) *src_fifo_ptr, volatile [FIFO_t](#) *dest_fifo_ptr)
Transfer the contents of one FIFO buffer to another.

Status Checks

- `uint32_t FIFO_PeekOne` (volatile `FIFO_t` *`fifo_ptr`)
See the first element in the FIFO without removing it.
- `void FIFO_PeekAll` (volatile `FIFO_t` *`fifo_ptr`, `uint32_t` `output_buffer[]`)
See the FIFO buffer's contents without removing them.
- `bool FIFO_isFull` (volatile `FIFO_t` *`fifo_ptr`)
Check if the FIFO buffer is full.
- `bool FIFO_isEmpty` (volatile `FIFO_t` *`fifo_ptr`)
Check if the FIFO buffer is empty.
- `uint32_t FIFO_getCurrSize` (volatile `FIFO_t` *`fifo_ptr`)
Get the current size of the FIFO buffer.

4.4.1 Detailed Description

4.4.2 Function Documentation

FIFO_Flush()

```
void FIFO_Flush (  
    volatile FIFO_t * fifo_ptr,  
    uint32_t output_buffer[] )
```

Empty the FIFO buffer's contents into an array.

Parameters

<i>fifo_ptr</i>	Pointer to source FIFO buffer.
<i>output_buffer</i>	Array to output values to. Should be the same length as the FIFO buffer.

FIFO_Get()

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

Remove the first value of the buffer.

Parameters

<i>fifo_ptr</i>	Pointer to FIFO object
-----------------	------------------------

Returns

First sample in the FIFO.

FIFO_getCurrSize()

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

Get the current size of the FIFO buffer.

Parameters

<i>fifo_ptr</i>	Pointer to the FIFO buffer.
-----------------	-----------------------------

FIFO_Init()

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

Initialize a FIFO buffer of length N.

Parameters

<i>buffer</i>	Array of size N to be used as FIFO buffer
<i>N</i>	Length of <i>buffer</i> . Usable length is $N - 1$.

Returns

pointer to the FIFO buffer

TODO: Add details

FIFO_isEmpty()

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

Check if the FIFO buffer is empty.

Parameters

<i>fifo_ptr</i>	Pointer to the FIFO buffer.
-----------------	-----------------------------

Return values

<i>true</i>	The buffer is empty.
<i>false</i>	The buffer is not empty.

FIFO_isFull()

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

Check if the FIFO buffer is full.

Parameters

<i>fifo_ptr</i>	Pointer to the FIFO buffer.
-----------------	-----------------------------

Return values

<i>true</i>	The buffer is full.
<i>false</i>	The buffer is not full.

FIFO_PeekAll()

```
void FIFO_PeekAll (
    volatile FIFO_t * fifo_ptr,
    uint32_t output_buffer[] )
```

See the FIFO buffer's contents without removing them.

Parameters

<i>fifo_ptr</i>	Pointer to FIFO object
<i>output_buffer</i>	Array to output values to. Should be the same length as the FIFO buffer.

FIFO_PeekOne()

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

See the first element in the FIFO without removing it.

Parameters

<i>fifo_ptr</i>	Pointer to FIFO object
-----------------	------------------------

Returns

First sample in the FIFO.

FIFO_Put()

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

Add a value to the end of the buffer.

Parameters

<i>fifo_ptr</i>	Pointer to FIFO object
<i>val</i>	last value in the buffer

FIFO_TransferAll()

```
void FIFO_TransferAll (
    volatile FIFO_t * src_fifo_ptr,
    volatile FIFO_t * dest_fifo_ptr )
```

Transfer the contents of one FIFO buffer to another.

Parameters

<i>src_fifo_ptr</i>	Pointer to source FIFO buffer.
<i>dest_fifo_ptr</i>	Pointer to destination FIFO buffer.

FIFO_TransferOne()

```
void FIFO_TransferOne (
    volatile FIFO_t * src_fifo_ptr,
    volatile FIFO_t * dest_fifo_ptr )
```

Transfer a value from one FIFO buffer to another.

Parameters

<i>src_fifo_ptr</i>	Pointer to source FIFO buffer.
<i>dest_fifo_ptr</i>	Pointer to destination FIFO buffer.

5 Data Structure Documentation

5.1 FIFO_t Struct Reference

Data Fields

- volatile uint32_t * **buffer**
(pointer to) array to use as FIFO buffer
- volatile uint32_t **N**
length of buffer
- volatile uint32_t **front_idx**
idx of front of FIFO
- volatile uint32_t **back_idx**
idx of back of FIFO

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

- [FIFO.c](#)

5.2 LCD_t Struct Reference

Data Fields

- **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)
- **uint32_t numPixels**
*number of pixels to write to; = (x2 - x1 + 1) * (y2 - y1 + 1)*
- **uint8_t R_val**
5 or 6-bit R value
- **uint8_t G_val**
6-bit G value
- **uint8_t B_val**
5 or 6-bit B value
- **bool is_outputON**
if true, the LCD driver is writing from its memory to display
- **bool is_inverted**
if true, the display's colors are inverted
- **bool is_16bit**
true for 16-bit color depth (65K colors, 2 transfers), false for 18-bit
- **bool is_init**
if true, LCD has been initialized

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

- [LCD.c](#)

6 File Documentation

6.1 DAQ.c File Reference

Source code for DAQ module.

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

Macros

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

Typedefs

- `typedef arm_biquad_casd_df1_inst_f32 filt_t`

Functions

- void **DAQ_Init** (void)
- volatile float32_t **DAQ_Filter** (volatile float32_t inputSample)

6.1.1 Detailed Description

Source code for DAQ module.

Author

Bryan McElvy

6.2 DAQ.h File Reference

Application software for handling data acquisition (DAQ) functions.

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

Functions

- void **DAQ_Init** (void)
- volatile float32_t **DAQ_Filter** (volatile float32_t inputSample)

6.2.1 Detailed Description

Application software for handling data acquisition (DAQ) functions.

Author

Bryan McElvy

6.3 Debug.h File Reference

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

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

Enumerations

- enum **messages** { **START_MSG** , **DAQ_INIT** , **QRS_INIT** , **LCD_INIT** }

Functions

- void **Debug_Init** (void)
Initialize the Debug module and send a start message to the serial port.
- void **Debug_SendMsg** (void *message)
Send a message to the serial port.
- void **Debug_SendFromList** (uint8_t msg_idx)
- void **Debug_WriteFloat** (float64_t value)

6.3.1 Detailed Description

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

Author

Bryan McElvy

6.3.2 Function Documentation

Debug_Init()

```
void Debug_Init (
    void )
```

Initialize the Debug module and send a start message to the serial port.

Debug_SendMsg()

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

Send a message to the serial port.

Parameters

<code>message</code>	(Pointer to) array of ASCII characters.
----------------------	---

6.4 LCD.c File Reference

Source code for LCD module.

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

Data Structures

- struct [LCD_t](#)

Functions

- void [LCD_Init](#) (void)
Initialize the LCD driver and its internal independencies.
- void [LCD_toggleOutput](#) (void)
Toggle display output ON or OFF (OFF by default). Turning output OFF prevents the LCD driver from refreshing the display, which can prevent abnormalities like screen tearing while attempting to update the image.
- void [LCD_toggleInversion](#) (void)
Toggle color inversion ON or OFF (OFF by default).
- void [LCD_toggleColorDepth](#) (void)
Toggle 16-bit or 18-bit color depth (16-bit by default).
- void [LCD_setArea](#) (uint16_t x1, uint16_t x2, uint16_t y1, uint16_t y2)
Set the area of the display to be written to. $0 \leq x1 \leq x2 < X_MAX$ $0 \leq y1 \leq y2 < Y_MAX$
- void [LCD_setX](#) (uint16_t x1, uint16_t x2)
Set only new x-coordinates to be written to. $0 \leq x1 \leq x2 < X_MAX$
- void [LCD_setY](#) (uint16_t y1, uint16_t y2)
Set only new y-coordinates to be written to. $0 \leq y1 \leq y2 < Y_MAX$
- void [LCD_setColor](#) (uint8_t R_val, uint8_t G_val, uint8_t B_val)
Set the current color value for the display. Only the first 5-6 bits of each inputted value are used.
- void [LCD_setColor_3bit](#) (uint8_t color_code)
Set the color value via a 3-bit code.
- void [LCD_draw](#) (void)
Draw on the LCD display. Call this function after setting the drawable area via [LCD_setArea\(\)](#), or after individually calling [LCD_setX\(\)](#) and/or [LCD_setY\(\)](#).
- void [LCD_fill](#) (void)
Fill the display with a single color.
- void [LCD_drawHLine](#) (uint16_t yCenter, uint16_t lineWidth)
Draw a horizontal line across the entire display.

- void `LCD_drawVLine` (uint16_t xCenter, uint16_t lineWidth)
Draw a vertical line across the entire display.
- void `LCD_drawRectangle` (uint16_t x1, uint16_t dx, uint16_t y1, uint16_t dy, bool is_filled)
Draw a rectangle of size $dx \times dy$ onto the display. The bottom-left corner will be located at $(x1, y1)$.
- void `LCD_graphSample` (uint16_t x1, uint16_t dx, uint16_t y1, uint16_t dy, uint16_t y_min, uint16_t y_max, uint16_t color_code)
Draw a rectangle of size $dx \times dy$ and blank out all other pixels between y_{min} and y_{max} .

6.4.1 Detailed Description

Source code for LCD module.

Author

Bryan McElvy

6.5 LCD.h File Reference

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

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

Macros

- #define `X_MAX` NUM_ROWS
- #define `Y_MAX` NUM_COLS
- #define `LCD_BLACK` (uint8_t) 0x00
- #define `LCD_RED` (uint8_t) 0x04
- #define `LCD_GREEN` (uint8_t) 0x02
- #define `LCD_BLUE` (uint8_t) 0x01
- #define `LCD_YELLOW` (uint8_t) 0x06
- #define `LCD_CYAN` (uint8_t) 0x03
- #define `LCD_PURPLE` (uint8_t) 0x05
- #define `LCD_WHITE` (uint8_t) 0x07

Functions

Init./Config. Functions

- void `LCD_Init` (void)
Initialize the LCD driver and its internal independencies.
- void `LCD_toggleOutput` (void)
Toggle display output ON or OFF (OFF by default). Turning output OFF prevents the LCD driver from refreshing the display, which can prevent abnormalities like screen tearing while attempting to update the image.
- void `LCD_toggleInversion` (void)
Toggle color inversion ON or OFF (OFF by default).
- void `LCD_toggleColorDepth` (void)
Toggle 16-bit or 18-bit color depth (16-bit by default).

Drawing Area Definition Functions

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

Color Setting Functions

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

Drawing Functions

- void `LCD_draw` (void)
Draw on the LCD display. Call this function after setting the drawable area via `LCD_setArea()`, or after individually calling `LCD_setX()` and/or `LCD_setY()`.
- void `LCD_fill` (void)
Fill the display with a single color.
- void `LCD_drawHLine` (uint16_t yCenter, uint16_t lineWidth)
Draw a horizontal line across the entire display.
- void `LCD_drawVLine` (uint16_t xCenter, uint16_t lineWidth)
Draw a vertical line across the entire display.
- void `LCD_drawRectangle` (uint16_t x1, uint16_t dx, uint16_t y1, uint16_t dy, bool is_filled)
Draw a rectangle of size $dx \times dy$ onto the display. The bottom-left corner will be located at $(x1, y1)$.
- void `LCD_graphSample` (uint16_t x1, uint16_t dx, uint16_t y1, uint16_t dy, uint16_t y_min, uint16_t y_max, uint16_t color_code)
Draw a rectangle of size $dx \times dy$ and blank out all other pixels between y_min and y_max .

6.5.1 Detailed Description

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

Author

Bryan McElvy

6.6 QRS.h File Reference

QRS detection algorithm functions.

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

6.6.1 Detailed Description

QRS detection algorithm functions.

Author

Bryan McElvy

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

6.7 UserCtrl.h File Reference

Interface for user control module.

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

Functions

- void [UserCtrl_Init](#) ()
Initializes the UserCtrl module and its dependencies (Timer0B and GPIO_PortF)

6.7.1 Detailed Description

Interface for user control module.

Author

Bryan McElvy

6.7.2 Function Documentation

UserCtrl_Init()

```
void UserCtrl_Init ( )
```

Initializes the UserCtrl module and its dependencies (Timer0B and GPIO_PortF)

6.8 FIFO.c File Reference

Source code for FIFO buffer module.

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

Data Structures

- struct [FIFO_t](#)

Functions

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

Basic Operations

- void [FIFO_Put](#) (volatile [FIFO_t](#) *fifo_ptr, const uint32_t val)
Add a value to the end of the buffer.
- volatile uint32_t [FIFO_Get](#) (volatile [FIFO_t](#) *fifo_ptr)
Remove the first value of the buffer.
- void [FIFO_TransferOne](#) (volatile [FIFO_t](#) *src_fifo_ptr, volatile [FIFO_t](#) *dest_fifo_ptr)
Transfer a value from one FIFO buffer to another.

Bulk Removal

- void [FIFO_Flush](#) (volatile [FIFO_t](#) *fifo_ptr, uint32_t output_buffer[])
Empty the FIFO buffer's contents into an array.
- void [FIFO_TransferAll](#) (volatile [FIFO_t](#) *src_fifo_ptr, volatile [FIFO_t](#) *dest_fifo_ptr)
Transfer the contents of one FIFO buffer to another.

Status Checks

- uint32_t [FIFO_PeekOne](#) (volatile [FIFO_t](#) *fifo_ptr)
See the first element in the FIFO without removing it.
- void [FIFO_PeekAll](#) (volatile [FIFO_t](#) *fifo_ptr, uint32_t output_buffer[])
See the FIFO buffer's contents without removing them.
- bool [FIFO_isFull](#) (volatile [FIFO_t](#) *fifo_ptr)
Check if the FIFO buffer is full.
- bool [FIFO_isEmpty](#) (volatile [FIFO_t](#) *fifo_ptr)
Check if the FIFO buffer is empty.
- uint32_t [FIFO_getCurrSize](#) (volatile [FIFO_t](#) *fifo_ptr)
Get the current size of the FIFO buffer.

6.8.1 Detailed Description

Source code for FIFO buffer module.

Author

Bryan McElvy

6.9 FIFO.h File Reference

FIFO buffer data structure.

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

Macros

- `#define FIFO_POOL_SIZE 5`

Functions

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

Basic Operations

- void [FIFO_Put](#) (volatile [FIFO_t](#) *fifo_ptr, const uint32_t val)
Add a value to the end of the buffer.
- volatile uint32_t [FIFO_Get](#) (volatile [FIFO_t](#) *fifo_ptr)
Remove the first value of the buffer.
- void [FIFO_TransferOne](#) (volatile [FIFO_t](#) *src_fifo_ptr, volatile [FIFO_t](#) *dest_fifo_ptr)
Transfer a value from one FIFO buffer to another.

Bulk Removal

- void [FIFO_Flush](#) (volatile [FIFO_t](#) *fifo_ptr, uint32_t output_buffer[])
Empty the FIFO buffer's contents into an array.
- void [FIFO_TransferAll](#) (volatile [FIFO_t](#) *src_fifo_ptr, volatile [FIFO_t](#) *dest_fifo_ptr)
Transfer the contents of one FIFO buffer to another.

Status Checks

- uint32_t [FIFO_PeekOne](#) (volatile [FIFO_t](#) *fifo_ptr)
See the first element in the FIFO without removing it.
- void [FIFO_PeekAll](#) (volatile [FIFO_t](#) *fifo_ptr, uint32_t output_buffer[])
See the FIFO buffer's contents without removing them.
- bool [FIFO_isFull](#) (volatile [FIFO_t](#) *fifo_ptr)
Check if the FIFO buffer is full.
- bool [FIFO_isEmpty](#) (volatile [FIFO_t](#) *fifo_ptr)
Check if the FIFO buffer is empty.
- uint32_t [FIFO_getCurrSize](#) (volatile [FIFO_t](#) *fifo_ptr)
Get the current size of the FIFO buffer.

6.9.1 Detailed Description

FIFO buffer data structure.

Author

Bryan McElvy

6.10 lookup.c File Reference

Lookup table source code.

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

Functions

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

6.10.1 Detailed Description

Lookup table source code.

Author

Bryan McElvy

6.10.2 Function Documentation

Lookup_GetPtr_ADC()

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

Return a pointer to the ADC lookup table.

Returns

`const float32_t*`

6.11 lookup.h File Reference

Lookup table API.

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

Macros

- `#define LOOKUP_ADC_MAX (float32_t) 3.3`

Functions

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

6.11.1 Detailed Description

Lookup table API.

Author

Bryan McElvy

6.11.2 Function Documentation

Lookup_GetPtr_ADC()

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

Return a pointer to the ADC lookup table.

Returns

const float32_t*

6.12 ADC.c File Reference

Source code for ADC module.

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

Functions

- void **ADC_Init** (void)
Initialize ADC0 as a differential-input analog-to-digital converter.
- void **ADC_InterruptEnable** (void)
Enable the ADC interrupt.
- void **ADC_InterruptDisable** (void)
Disable the ADC interrupt.
- volatile float32_t **ADC_ConvertToVolts** (uint16_t raw_sample)
Convert a raw ADC sample to voltage in [mV].

6.12.1 Detailed Description

Source code for ADC module.

Author

Bryan McElvy

6.13 ADC.h File Reference

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

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

Functions

- void **ADC_Init** (void)
Initialize ADC0 as a differential-input analog-to-digital converter.
- void **ADC_InterruptEnable** (void)
Enable the ADC interrupt.
- void **ADC_InterruptDisable** (void)
Disable the ADC interrupt.
- volatile float32_t **ADC_ConvertToVolts** (uint16_t raw_sample)
Convert a raw ADC sample to voltage in [mV].

6.13.1 Detailed Description

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

Author

Bryan McElvy

6.14 GPIO.c File Reference

Source code for GPIO module.

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

Functions

- void **GPIO_PF_Init** (void)
Initialize GPIO Port F.
- void **GPIO_PF_LED_Init** (void)
Initialize PF1-3 to interface the LaunchPad's onboard RGB LED.
- void **GPIO_PF_LED_Write** (uint8_t color_mask, uint8_t on_or_off)
Write a 1 or 0 to the selected LED(s).
- void **GPIO_PF_LED_Toggle** (uint8_t color_mask)
Toggle the selected LED(s).
- void **GPIO_PF_Sw_Init** (void)
Initialize PF0/4 to interface the LaunchPad's onboard switches. PF4 is Sw1, and PF0 is Sw2.
- void **GPIO_PF_Interrupt_Init** (void)
Initialize GPIO Port F interrupts via Sw1 and Sw2.

6.14.1 Detailed Description

Source code for GPIO module.

Author

Bryan McElvy

6.15 GPIO.h File Reference

Driver module for using the LaunchPad's onboard switches and RGB LEDs for GPIO and interrupts.

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

Macros

- `#define LED_RED (uint8_t) 0x02`
- `#define LED_GREEN (uint8_t) 0x08`
- `#define LED_BLUE (uint8_t) 0x04`
- `#define LED_YELLOW (LED_RED + LED_GREEN)`
- `#define LED_CYAN (LED_BLUE + LED_GREEN)`
- `#define LED_PURPLE (LED_RED + LED_BLUE)`
- `#define LED_WHITE (LED_RED + LED_BLUE + LED_GREEN)`

Functions

- void `GPIO_PF_Init` (void)
Initialize GPIO Port F.
- void `GPIO_PF_LED_Init` (void)
Initialize PF1-3 to interface the LaunchPad's onboard RGB LED.
- void `GPIO_PF_LED_Write` (uint8_t color_mask, uint8_t on_or_off)
Write a 1 or 0 to the selected LED(s).
- void `GPIO_PF_LED_Toggle` (uint8_t color_mask)
Toggle the selected LED(s).
- void `GPIO_PF_Sw_Init` (void)
Initialize PF0/4 to interface the LaunchPad's onboard switches. PF4 is Sw1, and PF0 is Sw2.
- void `GPIO_PF_Interrupt_Init` (void)
Initialize GPIO Port F interrupts via Sw1 and Sw2.

6.15.1 Detailed Description

Driver module for using the LaunchPad's onboard switches and RGB LEDs for GPIO and interrupts.

Author

Bryan McElvy

6.16 ILI9341.c File Reference

Source code for ILI9341 module.

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

Macros

- #define **CMD_NOP** (uint8_t) 0x00
- #define **CMD_SWRESET** (uint8_t) 0x01
- #define **CMD_SPLIN** (uint8_t) 0x10
- #define **CMD_SPLOUT** (uint8_t) 0x11
- #define **CMD_PTLON** (uint8_t) 0x12
- #define **CMD_NORON** (uint8_t) 0x13
- #define **CMD_DINVOFF** (uint8_t) 0x20
- #define **CMD_DINVON** (uint8_t) 0x21
- #define **CMD_CASET** (uint8_t) 0x2A
- #define **CMD_PASET** (uint8_t) 0x2B
- #define **CMD_RAMWR** (uint8_t) 0x2C
- #define **CMD_DISPOFF** (uint8_t) 0x28
- #define **CMD_DISPON** (uint8_t) 0x29
- #define **CMD_PLTAR** (uint8_t) 0x30
- #define **CMD_VSCRDEF** (uint8_t) 0x33
- #define **CMD_MADCTL** (uint8_t) 0x36
- #define **CMD_VSCRADD** (uint8_t) 0x37
- #define **CMD_IDMOFF** (uint8_t) 0x38
- #define **CMD_IDMON** (uint8_t) 0x39
- #define **CMD_PIXSET** (uint8_t) 0x3A
- #define **CMD_FRMCTR1** (uint8_t) 0xB1
- #define **CMD_FRMCTR2** (uint8_t) 0xB2
- #define **CMD_FRMCTR3** (uint8_t) 0xB3
- #define **CMD_PRCTR** (uint8_t) 0xB5
- #define **CMD_IFCTL** (uint8_t) 0xF6

Functions

- void **ILI9341_Init** (void)
Initialize the LCD driver, the SPI module, and Timer2A.
- void **ILI9341_resetHard** (void)
Perform a hardware reset of the LCD driver.
- void **ILI9341_resetSoft** (void)
Perform a software reset of the LCD driver.
- void **ILI9341_setSleepMode** (bool is_sleeping)
Enter or exit sleep mode. The LCD driver is in sleep mode by default upon powering on or either kind of reset.
- void **ILI9341_setDispMode** (bool is_normal, bool is_full_colors)
Set the display area and color expression.

- void `ILI9341_setPartialArea` (uint16_t rowStart, uint16_t rowEnd)
Set the partial display area for partial mode. Call before activating partial mode via `ILI9341_setDisplayMode()`.
- void `ILI9341_setDisplInversion` (bool is_ON)
Toggle display inversion. Turning ON causes colors to be inverted on the display.
- void `ILI9341_setDispOutput` (bool is_ON)
Turn display output ON or OFF. This function clears the display and stops outputting to the display area, but does not affect frame memory or power.
- void `ILI9341_setScrollArea` (uint16_t topFixedArea, uint16_t vertScrollArea, uint16_t bottFixedArea)
Set the vertical scrolling area of the display. The sum of the three parameters should be equal to the max number of rows `NUM_ROWS = 320`.
- void `ILI9341_setScrollStart` (uint16_t startRow)
Set the start row for vertical scrolling.
- void `ILI9341_setMemAccessCtrl` (bool areRowsFlipped, bool areColsFlipped, bool areRowsColsSwitched, bool isVertRefreshFlipped, bool isColorOrderFlipped, bool isHorRefreshFlipped)
Set how data is converted from memory to display.
- void `ILI9341_setColorDepth` (bool is_16bit)
Set the pixel format to be 16-bit (65K colors) or 18-bit (262K colors).
- void `ILI9341_NoOpCmd` (void)
Send the "No Operation" command (`NOP = 0x00`) to the LCD driver. Can be used to terminate the "Memory Write" (`RAMWR`) and "Memory Read" (`RAMRD`) commands, but does nothing otherwise.
- void `ILI9341_setFrameRateNorm` (uint8_t div_ratio, uint8_t clocks_per_line)
TODO: Write brief.
- void `ILI9341_setFrameRateIdle` (uint8_t div_ratio, uint8_t clocks_per_line)
TODO: Write brief.
- void `ILI9341_setBlankingPorch` (uint8_t vpf, uint8_t vbp, uint8_t hfp, uint8_t hbp)
TODO: Write.
- void `ILI9341_setInterface` (void)
Sets the interface for the ILI9341. The parameters for this command are hard-coded, so it only needs to be called once upon initialization.
- void `ILI9341_setRowAddress` (uint16_t start_row, uint16_t end_row)
not using backlight, so these aren't necessary
- void `ILI9341_setColAddress` (uint16_t start_col, uint16_t end_col)
Sets the start/end rows to be written to.
- void `ILI9341_writeMemCmd` (void)
Sends the "Write Memory" (`RAMWR`) command to the LCD driver, signalling that incoming data should be written to memory.
- void `ILI9341_write1px` (uint8_t red, uint8_t green, uint8_t blue, bool is_16bit)
Write a single pixel to frame memory.

6.16.1 Detailed Description

Source code for ILI9341 module.

Author

Bryan McElvy

6.17 ILI9341.h File Reference

Driver module for interfacing with an ILI9341 LCD driver.

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

Macros

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

Functions

- `void ILI9341_Init (void)`
Initialize the LCD driver, the SPI module, and Timer2A.
- `void ILI9341_resetHard (void)`
Perform a hardware reset of the LCD driver.
- `void ILI9341_resetSoft (void)`
Perform a software reset of the LCD driver.
- `void ILI9341_setSleepMode (bool is_sleeping)`
Enter or exit sleep mode. The LCD driver is in sleep mode by default upon powering on or either kind of reset.
- `void ILI9341_setDispMode (bool is_normal, bool is_full_colors)`
Set the display area and color expression.
- `void ILI9341_setPartialArea (uint16_t rowStart, uint16_t rowEnd)`
Set the partial display area for partial mode. Call before activating partial mode via ILI9341_setDisplayMode().
- `void ILI9341_setDispInversion (bool is_ON)`
Toggle display inversion. Turning ON causes colors to be inverted on the display.
- `void ILI9341_setDispOutput (bool is_ON)`
Turn display output ON or OFF. This function clears the display and stops outputting to the display area, but does not affect frame memory or power.
- `void ILI9341_setScrollArea (uint16_t topFixedArea, uint16_t vertScrollArea, uint16_t bottFixedArea)`
Set the vertical scrolling area of the display. The sum of the three parameters should be equal to the max number of rows NUM_ROWS = 320.
- `void ILI9341_setScrollStart (uint16_t startRow)`
Set the start row for vertical scrolling.
- `void ILI9341_setMemAccessCtrl (bool areRowsFlipped, bool areColsFlipped, bool areRowsColsSwitched, bool isVertRefreshFlipped, bool isColorOrderFlipped, bool isHorRefreshFlipped)`
Set how data is converted from memory to display.
- `void ILI9341_setColorDepth (bool is_16bit)`
Set the pixel format to be 16-bit (65K colors) or 18-bit (262K colors).
- `void ILI9341_NoOpCmd (void)`
Send the "No Operation" command (NOP = 0x00) to the LCD driver. Can be used to terminate the "Memory Write" (RAMWR) and "Memory Read" (RAMRD) commands, but does nothing otherwise.
- `void ILI9341_setFrameRateNorm (uint8_t div_ratio, uint8_t clocks_per_line)`
TODO: Write brief.
- `void ILI9341_setFrameRateIdle (uint8_t div_ratio, uint8_t clocks_per_line)`

TODO: Write brief.

- void `ILI9341_setBlankingPorch` (uint8_t vpf, uint8_t vbp, uint8_t hfp, uint8_t hbp)

TODO: Write.

- void `ILI9341_setInterface` (void)

Sets the interface for the ILI9341. The parameters for this command are hard-coded, so it only needs to be called once upon initialization.

- void `ILI9341_setRowAddress` (uint16_t start_row, uint16_t end_row)

not using backlight, so these aren't necessary

- void `ILI9341_setColAddress` (uint16_t start_col, uint16_t end_col)

Sets the start/end rows to be written to.

- void `ILI9341_writeMemCmd` (void)

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

- void `ILI9341_write1px` (uint8_t red, uint8_t green, uint8_t blue, bool is_16bit)

Write a single pixel to frame memory.

6.17.1 Detailed Description

Driver module for interfacing with an ILI9341 LCD driver.

Author

Bryan McElvy

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

6.18 PLL.c File Reference

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

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

Functions

- void `PLL_Init` (void)

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

6.18.1 Detailed Description

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

Author

Bryan McElvy

6.19 PLL.h File Reference

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

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

Functions

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

6.19.1 Detailed Description

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

Author

Bryan McElvy

6.20 SPI.c File Reference

Source code for SPI module.

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

Macros

- #define [NVIC_SSI0_NUM](#) 7
- #define [SPI_INT_START](#)() (NVIC_SW_TRIG_R = (NVIC_SW_TRIG_R & ~(0xFF)) | [NVIC_SSI0_NUM](#))
- #define [SPI_SET_DC](#)() (GPIO_PORTA_DATA_R |= 0x40)
- #define [SPI_CLEAR_DC](#)() (GPIO_PORTA_DATA_R &= ~(0x40))
- #define [SPI_IS_BUSY](#) (SSI0_SR_R & 0x10)
- #define [SPI_TX_ISNOTFULL](#) ((bool) (SSI0_SR_R & 0x02))
- #define [SPI_BUFFER_SIZE](#) 9

Functions

- void [SPI_Init](#) (void)
Initialize SSI0 to act as an SPI Controller (AKA Master) in mode 0.
- uint8_t [SPI_Read](#) (void)
Read data from the peripheral.
- void [SPI_WriteCmd](#) (uint8_t cmd)
Write an 8-bit command to the peripheral.
- void [SPI_WriteData](#) (uint8_t data)
Write 8-bit data to the peripheral.
- void [SPI_IRQ_WriteCmd](#) (uint8_t cmd)
Add an 8-bit command to the SPI queue. If no data or other command is written, should directly precede a call to [SPI_IRQ_StartWriting\(\)](#).
- void [SPI_IRQ_WriteData](#) (uint8_t data)
Add 8-bit data to the SPI queue. Should directly precede either another call to the same function or a call to [SPI_IRQ_StartWriting\(\)](#).
- void [SPI_IRQ_StartWriting](#) (void)
Start writing data to the Tx FIFO. Should be used after 1+ calls to [SPI_IRQ_WriteCmd\(\)](#) and/or [SPI_IRQ_WriteData\(\)](#). If unused, writing will start when the SPI queue is full.
- void [SSI0_Handler](#) (void)
Sends parameters (data or commands) over SPI via SSI0.

6.20.1 Detailed Description

Source code for SPI module.

Author

Bryan McElvy

6.21 SPI.h File Reference

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

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

Functions

- void [SPI_Init](#) (void)
Initialize SSI0 to act as an SPI Controller (AKA Master) in mode 0.
- uint8_t [SPI_Read](#) (void)
Read data from the peripheral.
- void [SPI_WriteCmd](#) (uint8_t cmd)
Write an 8-bit command to the peripheral.
- void [SPI_WriteData](#) (uint8_t data)
Write 8-bit data to the peripheral.

- void [SPI_IRQ_WriteCmd](#) (uint8_t cmd)
Add an 8-bit command to the SPI queue. If no data or other command is written, should directly precede a call to [SPI_IRQ_StartWriting\(\)](#).
- void [SPI_IRQ_WriteData](#) (uint8_t data)
Add 8-bit data to the SPI queue. Should directly precede either another call to the same function or a call to [SPI_IRQ_StartWriting\(\)](#).
- void [SPI_IRQ_StartWriting](#) (void)
Start writing data to the Tx FIFO. Should be used after 1+ calls to [SPI_IRQ_WriteCmd\(\)](#) and/or [SPI_IRQ_WriteData\(\)](#). If unused, writing will start when the SPI queue is full.

6.21.1 Detailed Description

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

Author

Bryan McElvy

6.22 SysTick.c File Reference

Implementation details for SysTick functions.

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

Functions

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

6.22.1 Detailed Description

Implementation details for SysTick functions.

Author

Bryan McElvy

6.23 SysTick.h File Reference

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

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

Functions

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

6.23.1 Detailed Description

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

Author

Bryan McElvy

6.24 Timer.c File Reference

Implementation for timer module.

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

Functions

Timer0A

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

Timer1A

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

Timer2A

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

6.24.1 Detailed Description

Implementation for timer module.

Author

Bryan McElvy

6.25 Timer.h File Reference

Driver module for general-purpose timer modules.

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

Functions

Timer0A

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

Timer1A

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

Timer2A

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

6.25.1 Detailed Description

Driver module for general-purpose timer modules.

Author

Bryan McElvy

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

6.26 UART.c File Reference

Source code for UART module.

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

Macros

- `#define ASCII_CONVERSION 0x30`
- `#define UART0_TX_FULL (UART0_FR_R & 0x20)`
- `#define UART0_BUFFER_SIZE 16`
- `#define UART0_INTERRUPT_NUM 5`

Functions

- void [UART0_Init](#) (void)
Initialize UART0 to a baud rate of 115200, 8-bit data length, 1 start bit, and 1 stop bit.
- unsigned char [UART0_ReadChar](#) (void)
Read a single character from UART0.
- void [UART0_WriteChar](#) (unsigned char input_char)
Write a single character to UART0.
- void [UART0_WriteStr](#) (void *input_str)
Write a C string to UART0.
- void [UART0_Writelnt](#) (uint32_t n)
Write a 32-bit unsigned integer to UART0.
- void [UART0_WriteFloat](#) (double n, uint8_t num_decimals)
Write a floating-point number to UART0.
- void [UART0_IRQ_AddChar](#) (unsigned char input_char)
Add a single character to UART0's FIFO.
- void [UART0_IRQ_AddStr](#) (void *input_str)
Add a string to UART0's FIFO.
- void [UART0_IRQ_AddInt](#) (uint32_t n)

- *Add an integer to UART0's FIFO.*
- void [UART0_IRQ_Start](#) (void)
 - *Transmit the UART0's FIFO's contents via interrupt.*
- void **UART0_Handler** (void)
- void [UART1_Init](#) (void)
 - *Initialize UART1 to a baud rate of 115200, 8-bit data length, 1 start bit, and 1 stop bit.*
- unsigned char [UART1_ReadChar](#) (void)
 - *Read a single character from UART1.*
- void [UART1_WriteChar](#) (unsigned char input_char)
 - *Write a single character to UART1.*
- void [UART1_WriteStr](#) (void *input_str)
 - *Write a C string to UART1.*

6.26.1 Detailed Description

Source code for UART module.

Author

Bryan McElvy

6.27 UART.h File Reference

Driver module for serial communication via UART0 and UART 1.

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

Functions

- void [UART0_Init](#) (void)
 - *Initialize UART0 to a baud rate of 115200, 8-bit data length, 1 start bit, and 1 stop bit.*
- unsigned char [UART0_ReadChar](#) (void)
 - *Read a single character from UART0.*
- void [UART0_WriteChar](#) (unsigned char input_char)
 - *Write a single character to UART0.*
- void [UART0_WriteStr](#) (void *input_str)
 - *Write a C string to UART0.*
- void [UART0_WriteInt](#) (uint32_t n)
 - *Write a 32-bit unsigned integer to UART0.*
- void [UART0_WriteFloat](#) (double n, uint8_t num_decimals)
 - *Write a floating-point number to UART0.*
- void [UART0_IRQ_AddChar](#) (unsigned char input_char)
 - *Add a single character to UART0's FIFO.*
- void [UART0_IRQ_AddStr](#) (void *input_str)
 - *Add a string to UART0's FIFO.*
- void [UART0_IRQ_AddInt](#) (uint32_t n)
 - *Add an integer to UART0's FIFO.*

- void `UART0_IRQ_Start` (void)
Transmit the UART0's FIFO's contents via interrupt.
- void `UART1_Init` (void)
Initialize UART1 to a baud rate of 115200, 8-bit data length, 1 start bit, and 1 stop bit.
- unsigned char `UART1_ReadChar` (void)
Read a single character from UART1.
- void `UART1_WriteChar` (unsigned char input_char)
Write a single character to UART1.
- void `UART1_WriteStr` (void *input_str)
Write a C string to UART1.

6.27.1 Detailed Description

Driver module for serial communication via UART0 and UART 1.

Author

Bryan McElvy

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

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

6.28 main.c File Reference

Main program file for ECG-HRM.

```
#include "ADC.h"
#include "ILI9341.h"
#include "PLL.h"
#include "DAQ.h"
#include "Debug.h"
#include "QRS.h"
#include "UserCtrl.h"
```

Functions

- int `main` (void)
- void `GPIO_PortF_Handler` (void)
Interrupt service routine (ISR) for the UserCtrl module via GPIO Port F.
- void `ADC0_SS3_Handler` (void)
Interrupt service routine (ISR) for collecting ADC samples.
- void `Timer1A_Handler` (void)
Interrupt service routine (ISR) for outputting data to the LCD.

6.28.1 Detailed Description

Main program file for ECG-HRM.

Author

Bryan McElvy

Index

- ADC.c, [54](#)
- ADC.h, [55](#)
- ADC0_SS3_Handler
 - Program Threads, [38](#)
- ADC_ConvertToVolts
 - Analog-to-Digital Conversion (ADC), [5](#)
- Analog-to-Digital Conversion (ADC), [5](#)
 - ADC_ConvertToVolts, [5](#)
- Application Software, [30](#)

- DAQ.c, [44](#)
- DAQ.h, [45](#)
- Data Acquisition (DAQ), [31](#)
- Debug, [32](#)
- Debug.h, [46](#)
 - Debug_Init, [46](#)
 - Debug_SendMsg, [46](#)
- Debug_Init
 - Debug.h, [46](#)
- Debug_SendMsg
 - Debug.h, [46](#)
- Device Drivers, [4](#)

- Fifo, [39](#)
 - FIFO_Flush, [40](#)
 - FIFO_Get, [40](#)
 - FIFO_getCurrSize, [40](#)
 - FIFO_Init, [41](#)
 - FIFO_isEmpty, [41](#)
 - FIFO_isFull, [41](#)
 - FIFO_PeekAll, [42](#)
 - FIFO_PeekOne, [42](#)
 - FIFO_Put, [42](#)
 - FIFO_TransferAll, [43](#)
 - FIFO_TransferOne, [43](#)
- FIFO.c, [51](#)
- FIFO.h, [52](#)
- FIFO_Flush
 - Fifo, [40](#)
- FIFO_Get
 - Fifo, [40](#)
- FIFO_getCurrSize
 - Fifo, [40](#)
- FIFO_Init
 - Fifo, [41](#)
- FIFO_isEmpty
 - Fifo, [41](#)
- FIFO_isFull
 - Fifo, [41](#)
- FIFO_PeekAll
 - Fifo, [42](#)
- FIFO_PeekOne
 - Fifo, [42](#)
- FIFO_Put
 - Fifo, [42](#)
- FIFO_t, [43](#)

- FIFO_TransferAll
 - Fifo, [43](#)
- FIFO_TransferOne
 - Fifo, [43](#)

- GPIO, [6](#)
 - GPIO_PF_Init, [7](#)
 - GPIO_PF_Interrupt_Init, [7](#)
 - GPIO_PF_LED_Init, [7](#)
 - GPIO_PF_LED_Toggle, [7](#)
 - GPIO_PF_LED_Write, [7](#)
 - GPIO_PF_Sw_Init, [8](#)
- GPIO.c, [55](#)
- GPIO.h, [56](#)
- GPIO_PF_Init
 - GPIO, [7](#)
- GPIO_PF_Interrupt_Init
 - GPIO, [7](#)
- GPIO_PF_LED_Init
 - GPIO, [7](#)
- GPIO_PF_LED_Toggle
 - GPIO, [7](#)
- GPIO_PF_LED_Write
 - GPIO, [7](#)
- GPIO_PF_Sw_Init
 - GPIO, [8](#)
- GPIO_PortF_Handler
 - Program Threads, [38](#)

- ILI9341, [8](#)
 - ILI9341_resetHard, [10](#)
 - ILI9341_resetSoft, [10](#)
 - ILI9341_setBlankingPorch, [10](#)
 - ILI9341_setColAddress, [10](#)
 - ILI9341 setColorDepth, [11](#)
 - ILI9341_setDispInversion, [11](#)
 - ILI9341_setDispMode, [11](#)
 - ILI9341_setDispOutput, [12](#)
 - ILI9341 setFrameRateIdle, [12](#)
 - ILI9341 setFrameRateNorm, [12](#)
 - ILI9341_setInterface, [12](#)
 - ILI9341_setMemAccessCtrl, [13](#)
 - ILI9341_setPartialArea, [14](#)
 - ILI9341_setRowAddress, [14](#)
 - ILI9341_setScrollArea, [14](#)
 - ILI9341_setScrollStart, [15](#)
 - ILI9341_setSleepMode, [15](#)
 - ILI9341_write1px, [15](#)
 - ILI9341_writeMemCmd, [16](#)
- ILI9341.c, [57](#)
- ILI9341.h, [59](#)
- ILI9341_resetHard
 - ILI9341, [10](#)
- ILI9341_resetSoft
 - ILI9341, [10](#)
- ILI9341_setBlankingPorch

- ILI9341, [10](#)
- ILI9341_setColAddress
 - ILI9341, [10](#)
- ILI9341_setColorDepth
 - ILI9341, [11](#)
- ILI9341_setDisplnVersion
 - ILI9341, [11](#)
- ILI9341_setDispMode
 - ILI9341, [11](#)
- ILI9341_setDispOutput
 - ILI9341, [12](#)
- ILI9341_setFrameRateIdle
 - ILI9341, [12](#)
- ILI9341_setFrameRateNorm
 - ILI9341, [12](#)
- ILI9341_setInterface
 - ILI9341, [12](#)
- ILI9341_setMemAccessCtrl
 - ILI9341, [13](#)
- ILI9341_setPartialArea
 - ILI9341, [14](#)
- ILI9341_setRowAddress
 - ILI9341, [14](#)
- ILI9341_setScrollArea
 - ILI9341, [14](#)
- ILI9341_setScrollStart
 - ILI9341, [15](#)
- ILI9341_setSleepMode
 - ILI9341, [15](#)
- ILI9341_write1px
 - ILI9341, [15](#)
- ILI9341_writeMemCmd
 - ILI9341, [16](#)
- LCD, [32](#)
 - LCD_draw, [34](#)
 - LCD_drawHLine, [34](#)
 - LCD_drawRectangle, [34](#)
 - LCD_drawVLine, [34](#)
 - LCD_graphSample, [35](#)
 - LCD_Init, [35](#)
 - LCD_setArea, [35](#)
 - LCD_setColor, [36](#)
 - LCD_setColor_3bit, [36](#)
 - LCD_setX, [36](#)
 - LCD_setY, [37](#)
- LCD.c, [47](#)
- LCD.h, [48](#)
- LCD_draw
 - LCD, [34](#)
- LCD_drawHLine
 - LCD, [34](#)
- LCD_drawRectangle
 - LCD, [34](#)
- LCD_drawVLine
 - LCD, [34](#)
- LCD_graphSample
 - LCD, [35](#)
- LCD_Init
 - LCD, [35](#)
- LCD, [35](#)
- LCD_setArea
 - LCD, [35](#)
- LCD_setColor
 - LCD, [36](#)
- LCD_setColor_3bit
 - LCD, [36](#)
- LCD_setX
 - LCD, [36](#)
- LCD_setY
 - LCD, [37](#)
- LCD_t, [44](#)
- lookup.c, [53](#)
 - Lookup_GetPtr_ADC, [53](#)
- lookup.h, [53](#)
 - Lookup_GetPtr_ADC, [54](#)
- Lookup_GetPtr_ADC
 - lookup.c, [53](#)
 - lookup.h, [54](#)
- main.c, [68](#)
- NVIC_SSI0_NUM
 - Serial Peripheral Interface (SPI), [18](#)
- Phase-Locked Loop (PLL), [17](#)
 - PLL_Init, [17](#)
- PLL.c, [60](#)
- PLL.h, [61](#)
- PLL_Init
 - Phase-Locked Loop (PLL), [17](#)
- Program Threads, [38](#)
 - ADC0_SS3_Handler, [38](#)
 - GPIO_PortF_Handler, [38](#)
 - Timer1A_Handler, [38](#)
- QRS, [37](#)
- QRS.h, [50](#)
- Serial Peripheral Interface (SPI), [17](#)
 - NVIC_SSI0_NUM, [18](#)
 - SPI_Init, [19](#)
 - SPI_IRQ_WriteCmd, [19](#)
 - SPI_IRQ_WriteData, [19](#)
 - SPI_Read, [20](#)
 - SPI_WriteCmd, [20](#)
 - SPI_WriteData, [20](#)
 - SSI0_Handler, [20](#)
- SPI.c, [61](#)
- SPI.h, [62](#)
- SPI_Init
 - Serial Peripheral Interface (SPI), [19](#)
- SPI_IRQ_WriteCmd
 - Serial Peripheral Interface (SPI), [19](#)
- SPI_IRQ_WriteData
 - Serial Peripheral Interface (SPI), [19](#)
- SPI_Read
 - Serial Peripheral Interface (SPI), [20](#)
- SPI_WriteCmd

- Serial Peripheral Interface (SPI), 20
- SPI_WriteData
 - Serial Peripheral Interface (SPI), 20
- SSI0_Handler
 - Serial Peripheral Interface (SPI), 20
- System Tick (SysTick), 21
 - SysTick_Interrupt_Init, 21
 - SysTick_Timer_Init, 21
- SysTick.c, 63
- SysTick.h, 63
- SysTick_Interrupt_Init
 - System Tick (SysTick), 21
- SysTick_Timer_Init
 - System Tick (SysTick), 21
- Timer, 22
 - Timer0A_Init, 23
 - Timer0A_isCounting, 23
 - Timer0A_Start, 23
 - Timer0A_Wait1ms, 24
 - Timer1A_Init, 24
 - Timer2A_Init, 24
 - Timer2A_isCounting, 24
 - Timer2A_Start, 24
 - Timer2A_Wait1ms, 25
 - Timer3A_Init, 25
- Timer.c, 64
- Timer.h, 65
- Timer0A_Init
 - Timer, 23
- Timer0A_isCounting
 - Timer, 23
- Timer0A_Start
 - Timer, 23
- Timer0A_Wait1ms
 - Timer, 24
- Timer1A_Handler
 - Program Threads, 38
- Timer1A_Init
 - Timer, 24
- Timer2A_Init
 - Timer, 24
- Timer2A_isCounting
 - Timer, 24
- Timer2A_Start
 - Timer, 24
- Timer2A_Wait1ms
 - Timer, 25
- Timer3A_Init
 - Timer, 25
- UART.c, 66
- UART.h, 67
- UART0_Init
 - Universal Asynchronous Receiver/Transmitter (UART), 27
- UART0_IRQ_AddChar
 - Universal Asynchronous Receiver/Transmitter (UART), 27
- UART0_IRQ_AddInt
 - Universal Asynchronous Receiver/Transmitter (UART), 27
- UART0_IRQ_AddStr
 - Universal Asynchronous Receiver/Transmitter (UART), 27
- UART0_IRQ_Start
 - Universal Asynchronous Receiver/Transmitter (UART), 28
- UART0_ReadChar
 - Universal Asynchronous Receiver/Transmitter (UART), 28
- UART0_WriteChar
 - Universal Asynchronous Receiver/Transmitter (UART), 28
- UART0_WriteFloat
 - Universal Asynchronous Receiver/Transmitter (UART), 28
- UART0_WriteInt
 - Universal Asynchronous Receiver/Transmitter (UART), 29
- UART0_WriteStr
 - Universal Asynchronous Receiver/Transmitter (UART), 29
- UART1_Init
 - Universal Asynchronous Receiver/Transmitter (UART), 29
- UART1_ReadChar
 - Universal Asynchronous Receiver/Transmitter (UART), 29
- UART1_WriteChar
 - Universal Asynchronous Receiver/Transmitter (UART), 29
- UART1_WriteStr
 - Universal Asynchronous Receiver/Transmitter (UART), 30
- Universal Asynchronous Receiver/Transmitter (UART), 25
 - UART0_Init, 27
 - UART0_IRQ_AddChar, 27
 - UART0_IRQ_AddInt, 27
 - UART0_IRQ_AddStr, 27
 - UART0_IRQ_Start, 28
 - UART0_ReadChar, 28
 - UART0_WriteChar, 28
 - UART0_WriteFloat, 28
 - UART0_WriteInt, 29
 - UART0_WriteStr, 29
 - UART1_Init, 29
 - UART1_ReadChar, 29
 - UART1_WriteChar, 29
 - UART1_WriteStr, 30
- UserCtrl, 38
- UserCtrl.h, 50
 - UserCtrl_Init, 50
- UserCtrl_Init
 - UserCtrl.h, 50