

## ECG-HRM

Generated by Doxygen 1.9.7

<b>1 Module Index</b>	<b>2</b>
1.1 Modules	2
<b>2 Data Structure Index</b>	<b>2</b>
2.1 Data Structures	2
<b>3 File Index</b>	<b>2</b>
3.1 File List	2
<b>4 Module Documentation</b>	<b>4</b>
4.1 Device Drivers	4
4.1.1 Detailed Description	4
4.1.2 ADC	4
4.1.3 GPIO	5
4.1.4 ILI9341	7
4.1.5 PLL	17
4.1.6 SPI	18
4.1.7 SysTick	21
4.1.8 Timer	22
4.1.9 UART	27
4.2 Application Software	31
4.2.1 Detailed Description	31
4.2.2 Debug	32
4.2.3 Filter	32
4.2.4 LCD	32
4.2.5 QRS	38
4.2.6 UserCtrl	38
4.3 Program Threads	38
4.3.1 Detailed Description	39
4.3.2 Function Documentation	39
<b>5 Data Structure Documentation</b>	<b>39</b>
5.1 LCD_t Struct Reference	39
<b>6 File Documentation</b>	<b>40</b>
6.1 Debug.h File Reference	40
6.1.1 Detailed Description	40
6.2 Filter.h File Reference	41
6.2.1 Detailed Description	41
6.3 LCD.c File Reference	41
6.3.1 Detailed Description	43
6.4 LCD.h File Reference	43
6.4.1 Detailed Description	45
6.5 QRS.h File Reference	45

6.5.1 Detailed Description . . . . .	46
6.6 UserCtrl.h File Reference . . . . .	46
6.6.1 Detailed Description . . . . .	47
6.6.2 Function Documentation . . . . .	47
6.7 ADC.c File Reference . . . . .	47
6.7.1 Detailed Description . . . . .	48
6.8 ADC.h File Reference . . . . .	48
6.8.1 Detailed Description . . . . .	49
6.9 GPIO.c File Reference . . . . .	49
6.9.1 Detailed Description . . . . .	50
6.10 GPIO.h File Reference . . . . .	50
6.10.1 Detailed Description . . . . .	51
6.11 ILI9341.c File Reference . . . . .	52
6.11.1 Detailed Description . . . . .	54
6.12 ILI9341.h File Reference . . . . .	54
6.12.1 Detailed Description . . . . .	56
6.13 PLL.c File Reference . . . . .	56
6.13.1 Detailed Description . . . . .	56
6.14 PLL.h File Reference . . . . .	57
6.14.1 Detailed Description . . . . .	57
6.15 SPI.c File Reference . . . . .	58
6.15.1 Detailed Description . . . . .	58
6.16 SPI.h File Reference . . . . .	59
6.16.1 Detailed Description . . . . .	60
6.17 SysTick.c File Reference . . . . .	60
6.17.1 Detailed Description . . . . .	60
6.18 SysTick.h File Reference . . . . .	61
6.18.1 Detailed Description . . . . .	61
6.19 Timer.c File Reference . . . . .	62
6.19.1 Detailed Description . . . . .	62
6.20 Timer.h File Reference . . . . .	63
6.20.1 Detailed Description . . . . .	64
6.21 UART.c File Reference . . . . .	64
6.21.1 Detailed Description . . . . .	65
6.22 UART.h File Reference . . . . .	65
6.22.1 Detailed Description . . . . .	66
6.23 main.c File Reference . . . . .	67
6.23.1 Detailed Description . . . . .	67
<b>Index</b>	<b>69</b>

## 1 Module Index

### 1.1 Modules

Here is a list of all modules:

<b>Device Drivers</b>	<b>4</b>
ADC	4
GPIO	5
ILI9341	7
PLL	17
SPI	18
SysTick	21
Timer	22
UART	27
<b>Application Software</b>	<b>31</b>
Debug	32
Filter	32
LCD	32
QRS	38
UserCtrl	38
<b>Program Threads</b>	<b>38</b>

## 2 Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">LCD_t</a>	39
-----------------------	----

## 3 File Index

### 3.1 File List

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

<b>Debug.h</b>	
Functions to output debugging information to a serial port via UART	40
<b>Filter.h</b>	
Functions to implement digital filters via linear constant coefficient difference equations (LC-CDEs)	41
<b>LCD.c</b>	
Source code for LCD module	41
<b>LCD.h</b>	
Module for outputting the ECG waveform and HR to a liquid crystal display (LCD)	43
<b>QRS.h</b>	
QRS detection algorithm functions	45
<b>UserCtrl.h</b>	
Interface for user control module	46
<b>ADC.c</b>	
Source code for ADC module	47
<b>ADC.h</b>	
Driver module for analog-to-digital conversion (ADC)	48
<b>GPIO.c</b>	
Source code for GPIO module	49
<b>GPIO.h</b>	
Driver module for using the LaunchPad's onboard switches and RGB LEDs for GPIO and interrupts	50
<b>ILI9341.c</b>	
Source code for ILI9341 module	52
<b>ILI9341.h</b>	
Driver module for interfacing with an ILI9341 LCD driver	54
<b>PLL.c</b>	
Implementation details for phase-lock-loop (PLL) functions	56
<b>PLL.h</b>	
Driver module for activating the phase-locked-loop (PLL)	57
<b>SPI.c</b>	
Source code for SPI module	58
<b>SPI.h</b>	
Driver module for using the serial peripheral interface (SPI) protocol	59
<b>SysTick.c</b>	
Implementation details for SysTick functions	60
<b>SysTick.h</b>	
Driver module for using SysTick-based timing and/or interrupts	61
<b>Timer.c</b>	
Implementation for timer module	62
<b>Timer.h</b>	
Driver module for timing (Timer0) and interrupts (Timer1)	63

<a href="#">UART.c</a>	
Source code for UART module	64
<a href="#">UART.h</a>	
Driver module for serial communication via UART0 and UART 1	65
<a href="#">main.c</a>	
Main program file for ECG-HRM	67

## 4 Module Documentation

### 4.1 Device Drivers

Device driver modules.

#### Modules

- [ADC](#)  
*Functions for differential-input analog-to-digital conversion.*
- [GPIO](#)  
*Functions for interfacing the LaunchPad's RGB LEDs (PF1-3) and switches (PF0/4).*
- [ILI9341](#)  
*Functions for interfacing an ILI9341-based 240RGBX320 LCD via [SPI](#).*
- [PLL](#)  
*Function for initializing the phase-locked loop.*
- [SPI](#)  
*Functions for SPI-based communication via SSI0 peripheral.*
- [SysTick](#)  
*Functions for timing and periodic interrupts via SysTick.*
- [Timer](#)  
*Functions for timing and periodic interrupts via general-purpose timer modules (GPTM).*
- [UART](#)  
*Functions for UART-based communication.*

#### 4.1.1 Detailed Description

Device driver modules.

#### 4.1.2 ADC

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

#### Files

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

#### 4.1.2.1 Detailed Description

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

#### 4.1.3 GPIO

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

##### 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).

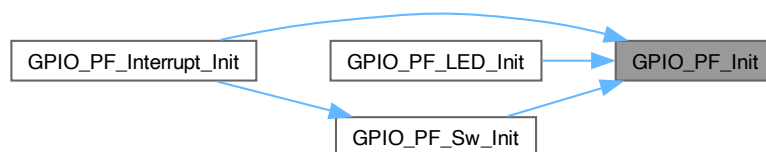
#### 4.1.3.2 Function Documentation

##### GPIO\_PF\_Init()

```
void GPIO_PF_Init (
    void )
```

Initialize GPIO Port F.

Here is the caller graph for this function:



##### 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.

Here is the caller graph for this function:

**4.1.4 ILI9341**

Functions for interfacing an ILI9341-based 240RGBX320 LCD via [SPI](#).

**Files**

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

## Macros

- `#define NOP (uint8_t) 0x00`
- `#define SWRESET (uint8_t) 0x01`
- `#define SPLIN (uint8_t) 0x10`
- `#define SPLOUT (uint8_t) 0x11`
- `#define PTLON (uint8_t) 0x12`
- `#define NORON (uint8_t) 0x13`
- `#define DINVOFF (uint8_t) 0x20`
- `#define DINVON (uint8_t) 0x21`
- `#define CASET (uint8_t) 0x2A`
- `#define PASET (uint8_t) 0x2B`
- `#define RAMWR (uint8_t) 0x2C`
- `#define DISPOFF (uint8_t) 0x28`
- `#define DISPON (uint8_t) 0x29`
- `#define PLTAR (uint8_t) 0x30`
- `#define VSCRDEF (uint8_t) 0x33`
- `#define MADCTL (uint8_t) 0x36`
- `#define VSCRSADD (uint8_t) 0x37`
- `#define IDMOFF (uint8_t) 0x38`
- `#define IDMON (uint8_t) 0x39`
- `#define PIXSET (uint8_t) 0x3A`
- `#define FRMCTR1 (uint8_t) 0xB1`
- `#define PRCTR (uint8_t) 0xB5`
- `#define 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_setFrameRate` (uint8\_t div\_ratio, uint8\_t clocks\_per\_line)  
TODO: Write.
- 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 [SPI](#).

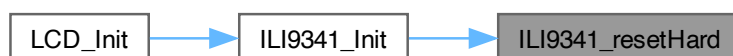
#### 4.1.4.2 Function Documentation

##### `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  $\geq 10$  [us] and an additional 5 [ms] before further commands can be sent. Here is the caller graph for this function:



**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>	$0 \leq \text{start\_col} \leq \text{end\_col}$
<i>end_col</i>	$\text{start\_col} \leq \text{end\_col} < 240$

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()**

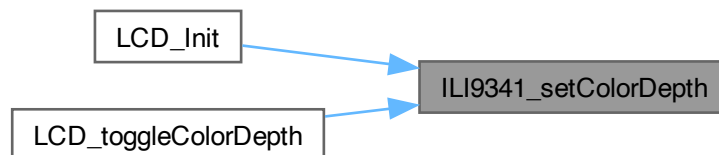
```
void ILI9341_setColorDepth (
    bool is_16bit )
```

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. Here is the caller graph for this function:

**ILI9341\_setDispInversion()**

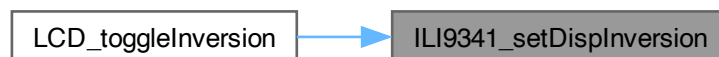
```
void ILI9341_setDispInversion (
    bool is_ON )
```

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 Here is the caller graph for this function:

**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

Here is the caller graph for this function:



#### 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 descriptionHere is the caller graph for this function:



#### ILI9341\_setFrameRate()

```
void ILI9341_setFrameRate (
```

```
uint8_t div_ratio,
uint8_t clocks_per_line )
```

TODO: Write.

TODO: Write

### ILI9341\_setInterface()

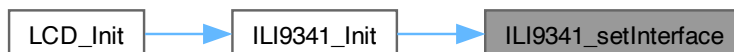
```
void ILI9341_setInterface (
    void )
```

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

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

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

The first param's bits are cleared so that the corresponding MADCTL bits (ILI9341\_setMemoryAccessCtrl()) are unaffected and overflowing pixel data is ignored. The EPF bits are cleared so that the LSB of the R and B values is copied from the MSB when using 16-bit color depth. The TM4C123 sends the MSB first, so the ENDIAN bit is cleared. The other bits are cleared and/or irrelevant since the RGB and VSYNC interfaces aren't used. Here is the caller graph for this function:



### 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.

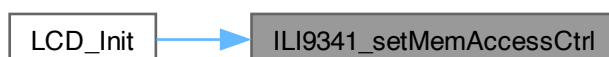
#### Parameters

<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" (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. Here is the caller graph for this function:



#### 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  $\geq 5$  [ms] before sending further commands.

It's also necessary to wait 120 [ms] before sending `SPL0UT` after sending `SPLIN` or a reset, so this function waits 120 [ms] regardless of the preceding event. Here is the caller graph for this function:

**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>	true for 16-bit (65K colors, 2 transfers) color depth, false for 18-bit (262K colors, 3 transfer) color depth NOTE: set color depth via <a href="#">ILI9341_setColorDepth()</a>

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.

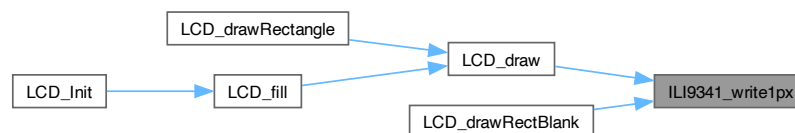
The following table (adapted from p. 63 of the datasheet) visualizes how the RGB data is sent to the display when using 16-bit color depth.

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	...

Here is the caller graph for this function:

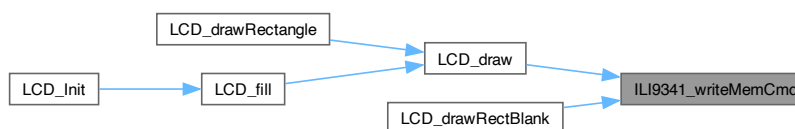


### ILI9341\_writeMemCmd()

```
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()`). Here is the caller graph for this function:



### 4.1.5 PLL

Function for initializing the phase-locked loop.

## Files

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

## Functions

- void [PLL\\_Init](#) (void)  
*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 SPI

Functions for SPI-based communication via SSI0 peripheral.

## Files

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

## 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 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\\_WriteSequence](#) (uint8\_t cmd, uint8\_t \*param\_sequence, uint8\_t num\_params)  
*Write a sequence of data to the peripheral, with or without a preceding command.*

#### 4.1.6.1 Detailed Description

Functions for SPI-based communication via SSI0 peripheral.

#### 4.1.6.2 Function Documentation

##### SPI\_Init()

```
void SPI_Init (
    void )
```

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

TM4C Pin	Function	ILI9341 Pin	Description
PA2	SSI0Clk	CLK	Serial clock signal
PA3	SSI0Fss	CS	Chip select signal
PA4	SSI0Rx	MISO	TM4C (M) input, LCD (S) output
PA5	SSI0Tx	MOSI	TM4C (M) output, LCD (S) input
PA6	GPIO	D/C	Data = 1, Command = 0
PA7	GPIO	RESET	Reset the display (negative logic/active LOW)

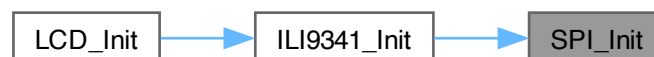
Clk. Polarity = steady state low (0)

Clk. Phase = rising clock edge (0)

The bit rate  $BR$  is set using the clock prescale divisor  $CPSDVSR$  and  $SCR$  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 12, allowing a bit rate of 6.67 [MHz], or a period of 150 [ns]. Here is the caller graph for this function:



##### SPI\_Read()

```
uint8_t SPI_Read (
    void )
```

Read data from peripheral.

##### Returns

uint8\_t

## SPI\_WriteCmd()

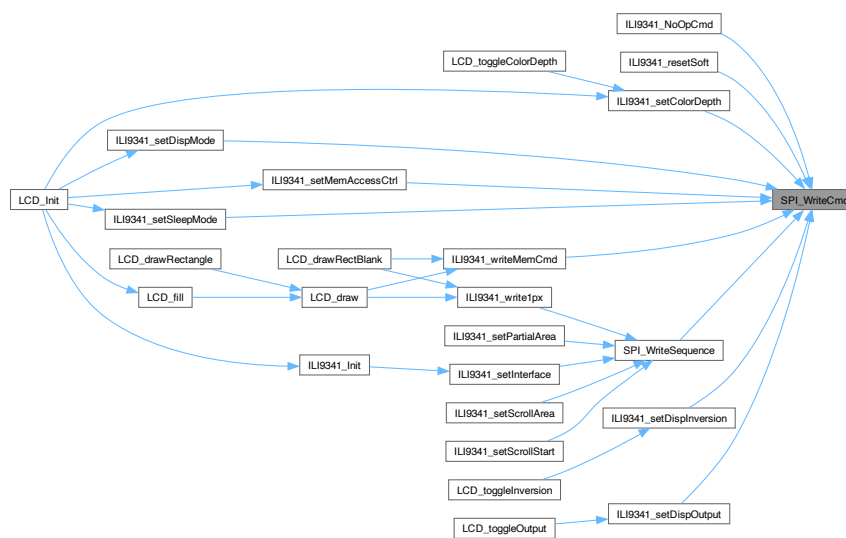
```
void SPI_WriteCmd (
    uint8_t cmd )
```

Write an 8-bit command to the peripheral.

### Parameters

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

Here is the caller graph for this function:



## SPI\_WriteData()

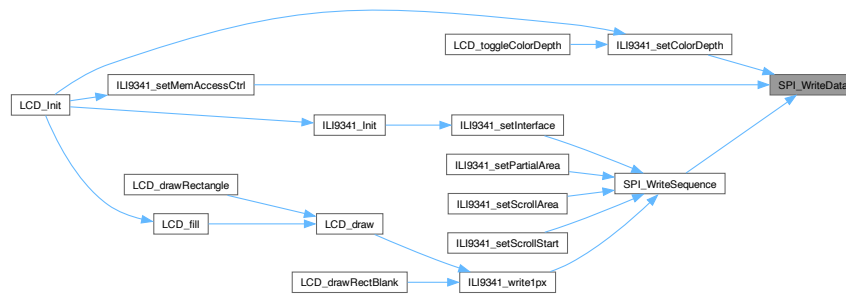
```
void SPI_WriteData (
    uint8_t data )
```

Write 8-bit data to the peripheral.

### Parameters

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

Here is the caller graph for this function:



## SPI\_WriteSequence()

```

void SPI_WriteSequence (
    uint8_t cmd,
    uint8_t * param_sequence,
    uint8_t num_params )

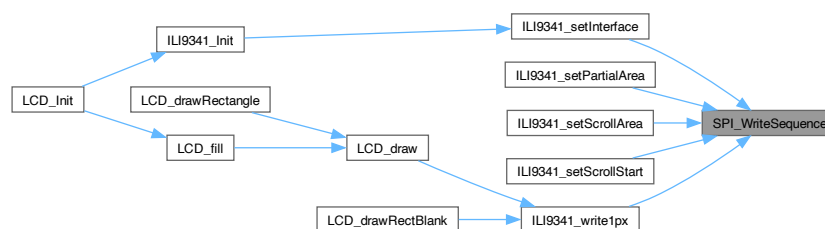
```

Write a sequence of data to the peripheral, with or without a preceding command.

### Parameters

<i>cmd</i>	8-bit command (using <code>cmd = 0</code> omits the command)
<i>param_sequence</i>	sequence of parameters to send after <code>cmd</code>
<i>num_params</i>	number of parameters to send; should be $\leq$ size of <code>param_sequence</code>

Here is the caller graph for this function:



### 4.1.7 SysTick

Functions for timing and periodic interrupts via 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

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



## Files

- file [Timer.c](#)  
*Implementation for timer module.*
- file [Timer.h](#)  
*Driver module for timing (Timer0) and interrupts (Timer1).*

## Functions

- 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].*
- void [Timer1A\\_Init](#) (uint32\_t time\_ms)  
*Initialize timer 1 as a 32-bit, periodic, countdown timer with interrupts.*
- 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].*

### 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

Here is the caller graph for this function:

**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 $\leq$ 53 seconds.
----------------	---

Here is the caller graph for this function:

**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**

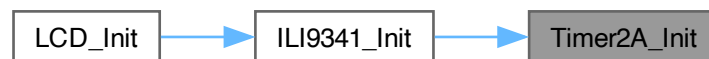
<i>time_ms</i>	Time in [ms] between interrupts. Must be $\leq 53$ seconds.
----------------	---

**Timer2A\_Init()**

```
void Timer2A_Init (
    void )
```

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

Here is the caller graph for this function:

**Timer2A\_isCounting()**

```
uint8_t Timer2A_isCounting (
    void )
```

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

**Returns**

`uint8_t` status

Here is the caller graph for this function:



**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 $\leq 53$ seconds.
----------------	---

Here is the caller graph for this function:

**Timer2A\_Wait1ms()**

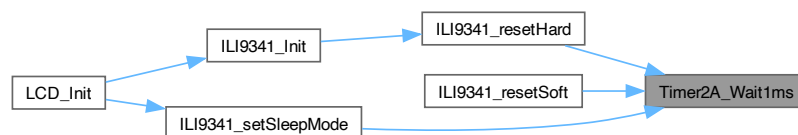
```
void Timer2A_Wait1ms (
    uint32_t time_ms )
```

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

**Parameters**

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

Here is the caller graph for this function:

**4.1.9 UART**

Functions for UART-based communication.

## Files

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

## 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](#) (unsigned char \*str\_ptr)  
*Write a C string to UART0.*
- 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](#) (unsigned char \*str\_ptr)  
*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:  $FBRD = int((mod(BRD, 1) * 64) + 0.5)$

NOTE: LCRH must be accessed *AFTER* setting the BRD register0

**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**

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

This function uses busy-wait synchronization to write a character to UART0. Here is the caller graph for this function:

**UART0\_WriteStr()**

```
void UART0_WriteStr (
    unsigned char * str_ptr )
```

Write a C string to UART0.

**Parameters**

<i>str_ptr</i>	pointer to C string
----------------	---------------------

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

input_char	
------------	--

This function uses busy-wait synchronization to write a character to UART1. Here is the caller graph for this



function:



### UART1\_WriteStr()

```
void UART1_WriteStr (
    unsigned char * str_ptr )
```

Write a C string to UART1.

#### Parameters

<code>str_ptr</code>	pointer to C string
----------------------	---------------------

This function uses [UART1\\_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.

## 4.2 Application Software

Application-specific modules.

### Modules

- [Debug](#)  
*Debug module.*
- [Filter](#)  
*Filter module.*
- [LCD](#)  
*Module for displaying graphs on an LCD via the [ILI9341](#) module.*
- [QRS](#)  
*QRS detection algorithm.*
- [UserCtrl](#)  
*User control module.*

### 4.2.1 Detailed Description

Application-specific modules.

### 4.2.2 Debug

Debug module.

Debug module.

### 4.2.3 Filter

Filter module.

Filter module.

### 4.2.4 LCD

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

#### 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`

## 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\_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.  $0 \leq y1 \leq y2 < Y\_MAX$*
- 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\_drawRectBlank** (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

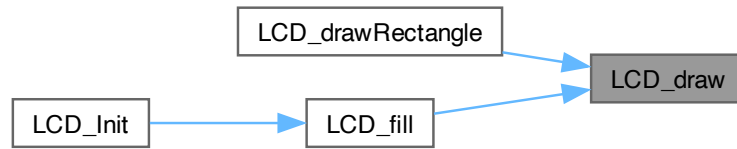
#### LCD\_draw()

```
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 [ILI9341\\_write1px\(\)](#), and [ILI9341\\_writeMemCmd\(\)](#).

Here is the caller graph for this function:



### 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  $dx \times dy$  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>	<code>true</code> to fill the rectangle, <code>false</code> to leave it unfilled

**LCD\_drawRectBlank()**

```
void LCD_drawRectBlank (
    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\_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\_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 )

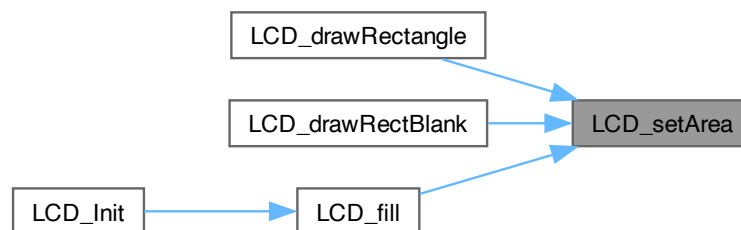
```

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

Here is the caller graph for this function:



### LCD\_setColor()

```

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

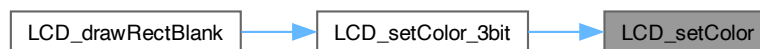
```

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

Here is the caller graph for this function:



### 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.

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

Here is the caller graph for this function:



### 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

QRS detection algorithm.

QRS detection algorithm.

### 4.2.6 UserCtrl

User control module.

User control module.

## 4.3 Program Threads

Primary threads of execution.

### 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.

## 5 Data Structure Documentation

### 5.1 LCD\_t Struct Reference

#### Data Fields

- uint16\_t **x1**
- uint16\_t **x2**
- uint16\_t **y1**
- uint16\_t **y2**
- uint32\_t **numPixels**
- uint8\_t **R\_val**
- uint8\_t **G\_val**
- uint8\_t **B\_val**
- bool **is\_outputON**
- bool **is\_inverted**
- bool **is\_16bit**
- bool **is\_init**

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

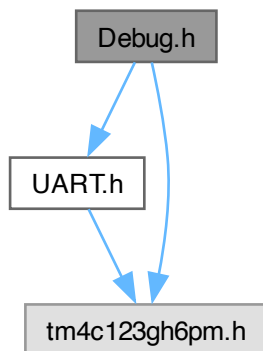
- [LCD.c](#)

## 6 File Documentation

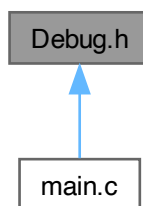
### 6.1 Debug.h File Reference

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

```
#include "UART.h"  
#include "tm4c123gh6pm.h"  
Include dependency graph for Debug.h:
```



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



#### 6.1.1 Detailed Description

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

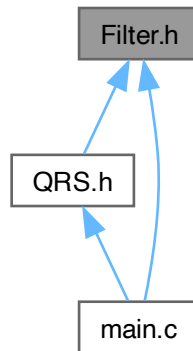
Author

Bryan McElvy

## 6.2 Filter.h File Reference

Functions to implement digital filters via linear constant coefficient difference equations (LCCDEs).

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



### 6.2.1 Detailed Description

Functions to implement digital filters via linear constant coefficient difference equations (LCCDEs).

#### Author

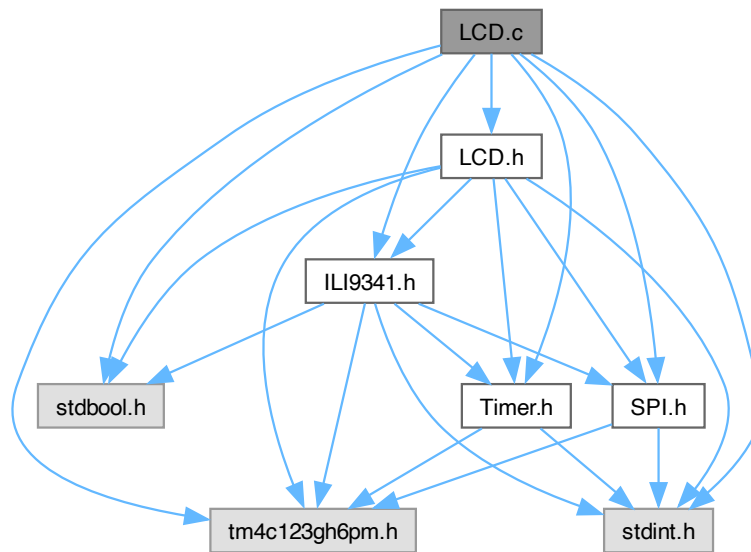
Bryan McElvy

## 6.3 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>
Include dependency graph for LCD.c:
```



## 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\_drawRectBlank** (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  $y_{max}$ .

### 6.3.1 Detailed Description

Source code for LCD module.

Author

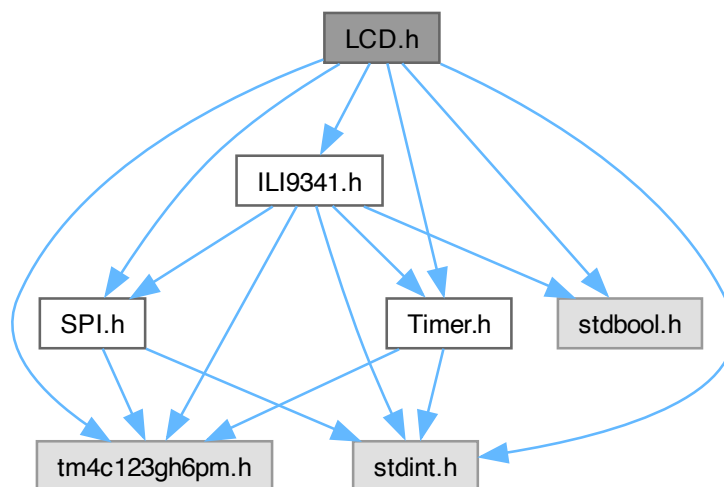
Bryan McElvy

## 6.4 LCD.h File Reference

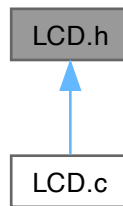
Module for outputting the ECG waveform and HR to a liquid crystal display (LCD).

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

Include dependency graph for LCD.h:



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



## 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

- 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\_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.  $0 \leq y1 \leq y2 < Y\_MAX$*
- 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\_drawRectBlank** (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

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

Author

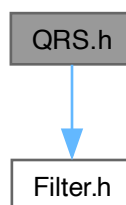
Bryan McElvy

## 6.5 QRS.h File Reference

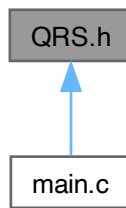
QRS detection algorithm functions.

```
#include "Filter.h"
```

Include dependency graph for QRS.h:



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



### 6.5.1 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.

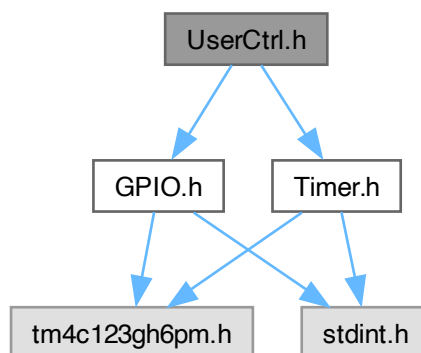
## 6.6 UserCtrl.h File Reference

Interface for user control module.

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

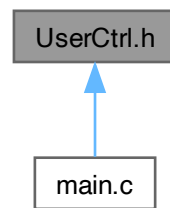
```
#include "Timer.h"
```

Include dependency graph for UserCtrl.h:





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



## Functions

- void [UserCtrl\\_Init](#) ()  
*Initializes the UserCtrl module and its dependencies (Timer0B and GPIO\_PortF)*

### 6.6.1 Detailed Description

Interface for user control module.

#### Author

Bryan McElvy

### 6.6.2 Function Documentation

#### UserCtrl\_Init()

```
void UserCtrl_Init ( )
```

Initializes the UserCtrl module and its dependencies (Timer0B and GPIO\_PortF)

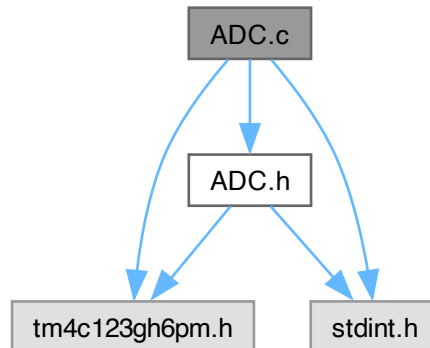
## 6.7 ADC.c File Reference

Source code for ADC module.

```
#include "ADC.h"  
#include "tm4c123gh6pm.h"
```

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

Include dependency graph for ADC.c:



### 6.7.1 Detailed Description

Source code for ADC module.

Author

Bryan McElvy

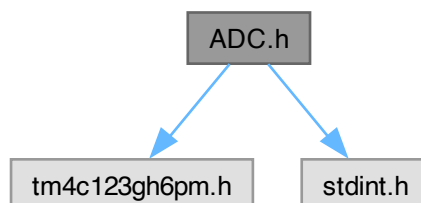
## 6.8 ADC.h File Reference

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

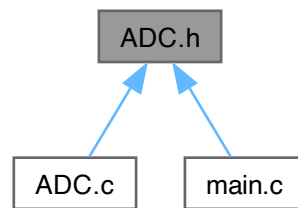
```
#include "tm4c123gh6pm.h"
```

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

Include dependency graph for ADC.h:



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



### 6.8.1 Detailed Description

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

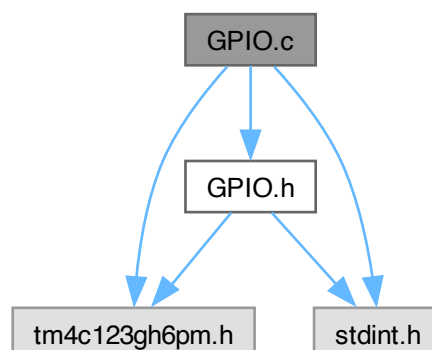
#### Author

Bryan McElvy

## 6.9 GPIO.c File Reference

Source code for GPIO module.

```
#include "GPIO.h"
#include "tm4c123gh6pm.h"
#include <stdint.h>
Include dependency graph for GPIO.c:
```



## 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.*

### 6.9.1 Detailed Description

Source code for GPIO module.

#### Author

Bryan McElvy

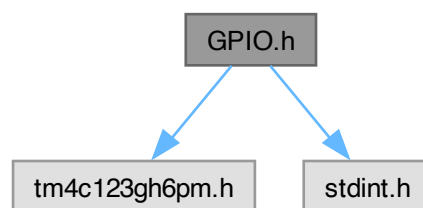
## 6.10 GPIO.h File Reference

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

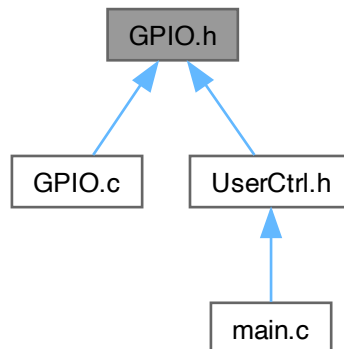
```
#include "tm4c123gh6pm.h"
```

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

Include dependency graph for GPIO.h:



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



## 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.*

### 6.10.1 Detailed Description

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

#### Author

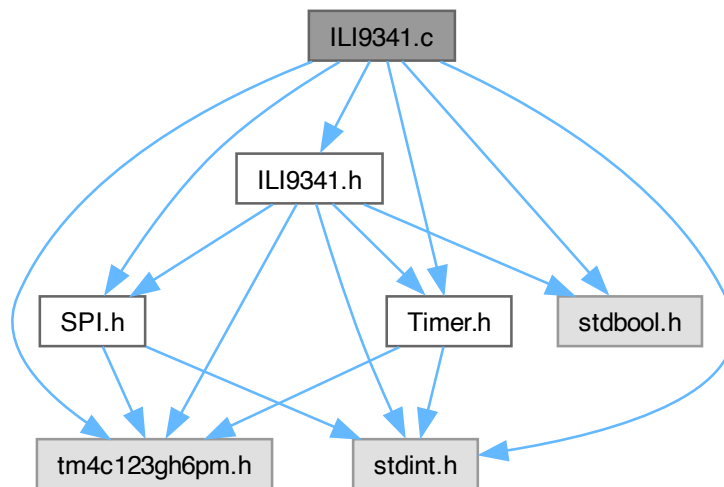
Bryan McElvy

## 6.11 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>
```

Include dependency graph for ILI9341.c:



### Macros

- #define **NOP** (uint8\_t) 0x00
- #define **SWRESET** (uint8\_t) 0x01
- #define **SPLIN** (uint8\_t) 0x10
- #define **SPLOUT** (uint8\_t) 0x11
- #define **PTLON** (uint8\_t) 0x12
- #define **NORON** (uint8\_t) 0x13
- #define **DINVOFF** (uint8\_t) 0x20
- #define **DINVON** (uint8\_t) 0x21
- #define **CASET** (uint8\_t) 0x2A
- #define **PASET** (uint8\_t) 0x2B
- #define **RAMWR** (uint8\_t) 0x2C
- #define **DISPOFF** (uint8\_t) 0x28
- #define **DISPON** (uint8\_t) 0x29
- #define **PLTAR** (uint8\_t) 0x30
- #define **VSCRDEF** (uint8\_t) 0x33
- #define **MADCTL** (uint8\_t) 0x36
- #define **VSCRSADD** (uint8\_t) 0x37
- #define **IDMOFF** (uint8\_t) 0x38

- `#define IDMON (uint8_t) 0x39`
- `#define PIXSET (uint8_t) 0x3A`
- `#define FRMCTR1 (uint8_t) 0xB1`
- `#define PRCTR (uint8_t) 0xB5`
- `#define 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\_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\_setFrameRate** (uint8\_t div\_ratio, uint8\_t clocks\_per\_line)  
*TODO: Write.*
- 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.*

### 6.11.1 Detailed Description

Source code for ILI9341 module.

Author

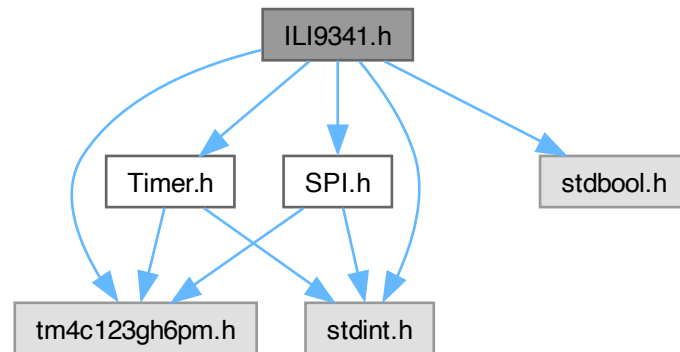
Bryan McElvy

## 6.12 ILI9341.h File Reference

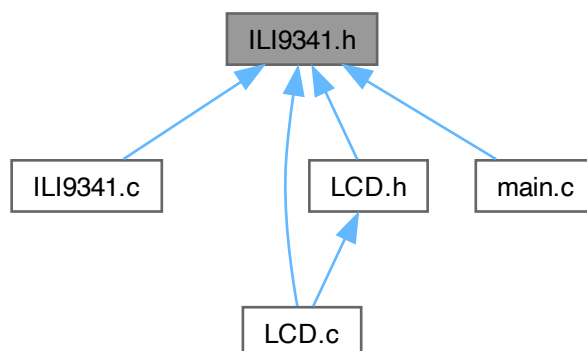
Driver module for interfacing with an ILI9341 LCD driver.

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

Include dependency graph for ILI9341.h:



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





## 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 setFrameRate** (uint8\_t div\_ratio, uint8\_t clocks\_per\_line)  
*TODO: Write.*
- 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.*

### 6.12.1 Detailed Description

Driver module for interfacing with an ILI9341 LCD driver.

#### Author

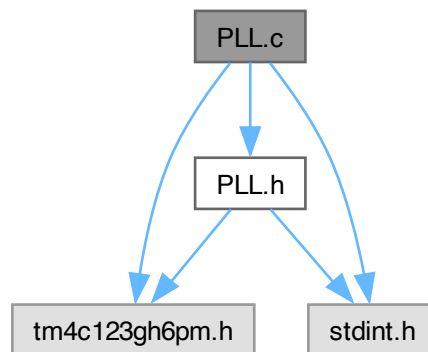
Bryan McElvy

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.

### 6.13 PLL.c File Reference

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

```
#include "PLL.h"
#include "tm4c123gh6pm.h"
#include <stdint.h>
Include dependency graph for PLL.c:
```



#### Functions

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

### 6.13.1 Detailed Description

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

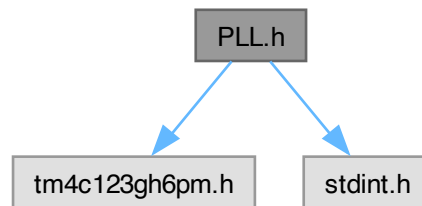
#### Author

Bryan McElvy

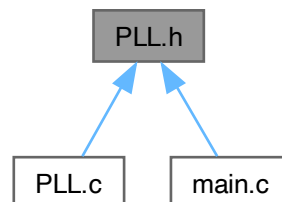
## 6.14 PLL.h File Reference

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

```
#include "tm4c123gh6pm.h"
#include <stdint.h>
Include dependency graph for PLL.h:
```



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



### Functions

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

#### 6.14.1 Detailed Description

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

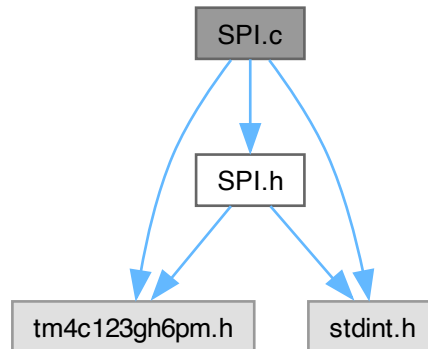
#### Author

Bryan McElvy

## 6.15 SPI.c File Reference

Source code for SPI module.

```
#include "SPI.h"
#include "tm4c123gh6pm.h"
#include <stdint.h>
Include dependency graph for SPI.c:
```



### Functions

- void [SPI\\_Init](#) (void)  
*Initialize SSIO to act as an SPI Controller (AKA Master) in mode 0.*
- uint8\_t [SPI\\_Read](#) (void)  
*Read data from 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\\_WriteSequence](#) (uint8\_t cmd, uint8\_t \*param\_sequence, uint8\_t num\_params)  
*Write a sequence of data to the peripheral, with or without a preceding command.*

### 6.15.1 Detailed Description

Source code for SPI module.

Author

Bryan McElvy

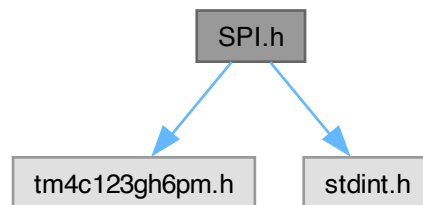
## 6.16 SPI.h File Reference

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

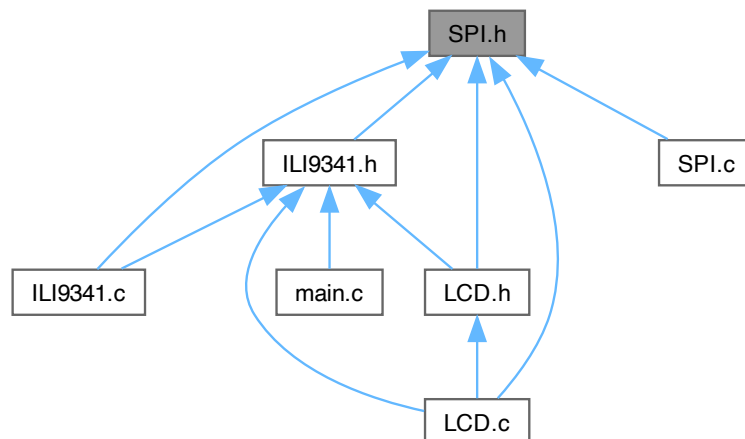
```
#include "tm4c123gh6pm.h"
```

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

Include dependency graph for SPI.h:



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



### Functions

- void [SPI\\_Init](#) (void)  
*Initialize SSIO to act as an SPI Controller (AKA Master) in mode 0.*
- uint8\_t [SPI\\_Read](#) (void)  
*Read data from 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\\_WriteSequence](#) (uint8\_t cmd, uint8\_t \*param\_sequence, uint8\_t num\_params)  
*Write a sequence of data to the peripheral, with or without a preceding command.*

### 6.16.1 Detailed Description

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

Author

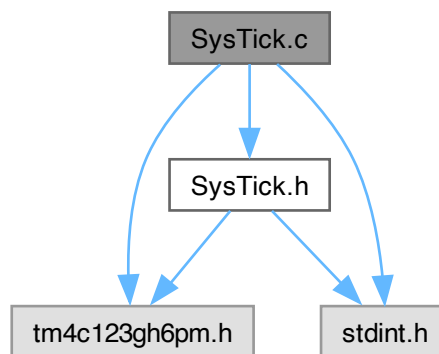
Bryan McElvy

## 6.17 SysTick.c File Reference

Implementation details for SysTick functions.

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

Include dependency graph for SysTick.c:



### Functions

- void [SysTick\\_Timer\\_Init](#) (void)  
*Initialize SysTick for timing purposes.*
- void **SysTick\_Wait1ms** (uint32\_t delay\_ms)  
*Delay for specified amount of time in [ms]. Assumes  $f_{bus} = 80[MHz]$ .*
- void [SysTick\\_Interrupt\\_Init](#) (uint32\_t time\_ms)  
*Initialize SysTick for interrupts.*

### 6.17.1 Detailed Description

Implementation details for SysTick functions.

Author

Bryan McElvy

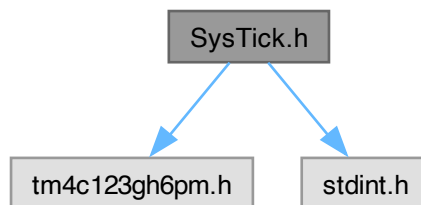
## 6.18 SysTick.h File Reference

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

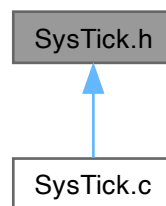
```
#include "tm4c123gh6pm.h"
```

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

Include dependency graph for SysTick.h:



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



### Functions

- void [SysTick\\_Timer\\_Init](#) (void)  
*Initialize SysTick for timing purposes.*
- void **SysTick\_Wait1ms** (uint32\_t delay\_ms)  
*Delay for specified amount of time in [ms]. Assumes  $f_{bus} = 80[MHz]$ .*
- void [SysTick\\_Interrupt\\_Init](#) (uint32\_t time\_ms)  
*Initialize SysTick for interrupts.*

#### 6.18.1 Detailed Description

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

Author

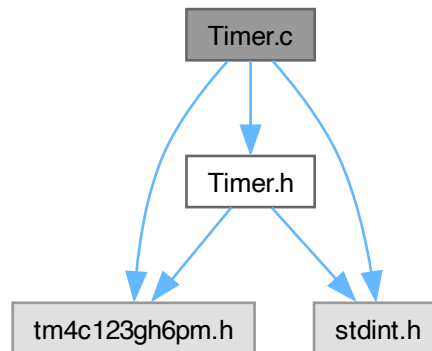
Bryan McElvy

## 6.19 Timer.c File Reference

Implementation for timer module.

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

Include dependency graph for Timer.c:



### Functions

- 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].*
- void [Timer1A\\_Init](#) (uint32\_t time\_ms)  
*Initialize timer 1 as a 32-bit, periodic, countdown timer with interrupts.*
- 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].*

### 6.19.1 Detailed Description

Implementation for timer module.

Author

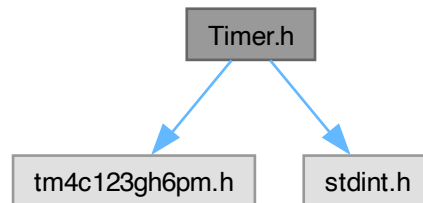
Bryan McElvy



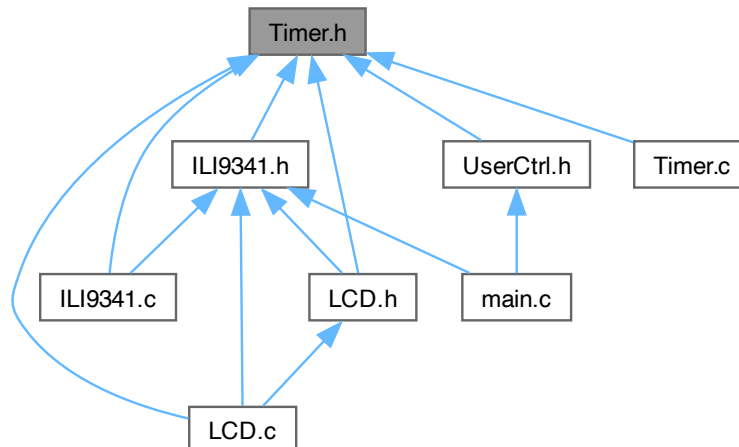
## 6.20 Timer.h File Reference

Driver module for timing (Timer0) and interrupts (Timer1).

```
#include "tm4c123gh6pm.h"
#include <stdint.h>
Include dependency graph for Timer.h:
```



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



### Functions

- 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].*
- void `Timer1A_Init` (uint32\_t time\_ms)  
*Initialize timer 1 as a 32-bit, periodic, countdown timer with interrupts.*
- 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].*

### 6.20.1 Detailed Description

Driver module for timing (Timer0) and interrupts (Timer1).

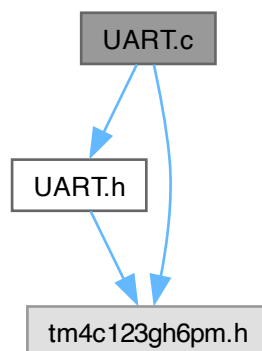
#### Author

Bryan McElvy

## 6.21 UART.c File Reference

Source code for UART module.

```
#include "UART.h"
#include "tm4c123gh6pm.h"
Include dependency graph for UART.c:
```



## 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](#) (unsigned char \*str\_ptr)  
*Write a C string to UART0.*
- 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](#) (unsigned char \*str\_ptr)  
*Write a C string to UART1.*

### 6.21.1 Detailed Description

Source code for UART module.

#### Author

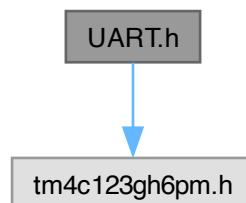
Bryan McElvy

## 6.22 UART.h File Reference

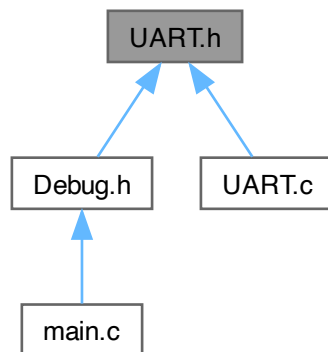
Driver module for serial communication via UART0 and UART 1.

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

Include dependency graph for UART.h:



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



## 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` (unsigned char \*str\_ptr)  
*Write a C string to UART0.*
- 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` (unsigned char \*str\_ptr)  
*Write a C string to UART1.*

### 6.22.1 Detailed Description

Driver module for serial communication via UART0 and UART 1.

#### Author

Bryan McElvy

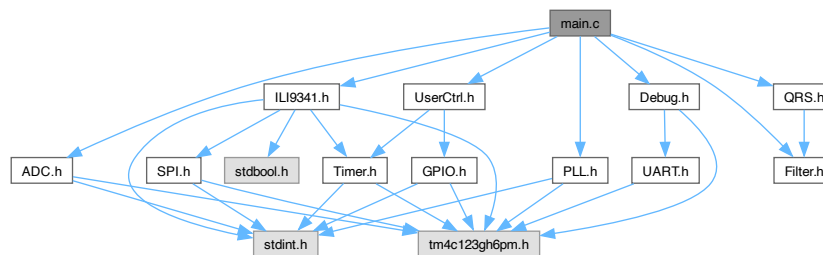
UART0 uses PA0 and PA1, which are not broken out but can connect to a PC's serial port via USB.

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

## 6.23 main.c File Reference

Main program file for ECG-HRM.

```
#include "ADC.h"
#include "ILI9341.h"
#include "PLL.h"
#include "Debug.h"
#include "Filter.h"
#include "QRS.h"
#include "UserCtrl.h"
Include dependency graph for main.c:
```



### 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.*

#### 6.23.1 Detailed Description

Main program file for ECG-HRM.

Author

Bryan McElvy



## Index

ADC, [4](#)  
ADC.c, [47](#)  
ADC.h, [48](#)  
ADC0\_SS3\_Handler  
    Program Threads, [39](#)  
Application Software, [31](#)  
  
Debug, [32](#)  
Debug.h, [40](#)  
Device Drivers, [4](#)  
  
Filter, [32](#)  
Filter.h, [41](#)  
  
GPIO, [5](#)  
    GPIO\_PF\_Init, [6](#)  
    GPIO\_PF\_Interrupt\_Init, [6](#)  
    GPIO\_PF\_LED\_Init, [6](#)  
    GPIO\_PF\_LED\_Toggle, [6](#)  
    GPIO\_PF\_LED\_Write, [6](#)  
    GPIO\_PF\_Sw\_Init, [7](#)  
GPIO.c, [49](#)  
GPIO.h, [50](#)  
GPIO\_PF\_Init  
    GPIO, [6](#)  
GPIO\_PF\_Interrupt\_Init  
    GPIO, [6](#)  
GPIO\_PF\_LED\_Init  
    GPIO, [6](#)  
GPIO\_PF\_LED\_Toggle  
    GPIO, [6](#)  
GPIO\_PF\_LED\_Write  
    GPIO, [6](#)  
GPIO\_PF\_Sw\_Init  
    GPIO, [7](#)  
GPIO\_PortF\_Handler  
    Program Threads, [39](#)  
  
ILI9341, [7](#)  
    ILI9341\_resetHard, [9](#)  
    ILI9341\_resetSoft, [9](#)  
    ILI9341\_setBlankingPorch, [10](#)  
    ILI9341\_setColAddress, [10](#)  
    ILI9341\_setColorDepth, [10](#)  
    ILI9341\_setDisplInversion, [11](#)  
    ILI9341\_setDispMode, [11](#)  
    ILI9341\_setDispOutput, [12](#)  
    ILI9341 setFrameRate, [12](#)  
    ILI9341\_setInterface, [13](#)  
    ILI9341\_setMemAccessCtrl, [13](#)  
    ILI9341\_setPartialArea, [14](#)  
    ILI9341\_setRowAddress, [15](#)  
    ILI9341\_setScrollArea, [15](#)  
    ILI9341\_setScrollStart, [15](#)  
    ILI9341\_setSleepMode, [15](#)  
    ILI9341\_write1px, [16](#)  
    ILI9341\_writeMemCmd, [17](#)  
ILI9341.c, [52](#)  
ILI9341.h, [54](#)  
ILI9341\_resetHard  
    ILI9341, [9](#)  
ILI9341\_resetSoft  
    ILI9341, [9](#)  
ILI9341\_setBlankingPorch  
    ILI9341, [10](#)  
ILI9341\_setColAddress  
    ILI9341, [10](#)  
ILI9341\_setColorDepth  
    ILI9341, [10](#)  
ILI9341\_setDisplInversion  
    ILI9341, [11](#)  
ILI9341\_setDispMode  
    ILI9341, [11](#)  
ILI9341\_setDispOutput  
    ILI9341, [12](#)  
ILI9341 setFrameRate  
    ILI9341, [12](#)  
ILI9341\_setInterface  
    ILI9341, [13](#)  
ILI9341\_setMemAccessCtrl  
    ILI9341, [13](#)  
ILI9341\_setPartialArea  
    ILI9341, [14](#)  
ILI9341\_setRowAddress  
    ILI9341, [15](#)  
ILI9341\_setScrollArea  
    ILI9341, [15](#)  
ILI9341\_setScrollStart  
    ILI9341, [15](#)  
ILI9341\_setSleepMode  
    ILI9341, [15](#)  
ILI9341\_write1px  
    ILI9341, [16](#)  
ILI9341\_writeMemCmd  
    ILI9341, [17](#)  
  
LCD, [32](#)  
    LCD\_draw, [33](#)  
    LCD\_drawHLine, [34](#)  
    LCD\_drawRectangle, [34](#)  
    LCD\_drawRectBlank, [34](#)  
    LCD\_drawVLine, [35](#)  
    LCD\_Init, [35](#)  
    LCD\_setArea, [35](#)  
    LCD\_setColor, [36](#)  
    LCD\_setColor\_3bit, [37](#)  
    LCD\_setX, [37](#)  
    LCD\_setY, [38](#)  
LCD.c, [41](#)  
LCD.h, [43](#)  
LCD\_draw

- LCD, [33](#)
- LCD\_drawHLine
  - LCD, [34](#)
- LCD\_drawRectangle
  - LCD, [34](#)
- LCD\_drawRectBlank
  - LCD, [34](#)
- LCD\_drawVLine
  - LCD, [35](#)
- LCD\_Init
  - LCD, [35](#)
- LCD\_setArea
  - LCD, [35](#)
- LCD\_setColor
  - LCD, [36](#)
- LCD\_setColor\_3bit
  - LCD, [37](#)
- LCD\_setX
  - LCD, [37](#)
- LCD\_setY
  - LCD, [38](#)
- LCD\_t, [39](#)
- main.c, [67](#)
- PLL, [17](#)
  - PLL\_Init, [18](#)
- PLL.c, [56](#)
- PLL.h, [57](#)
- PLL\_Init
  - PLL, [18](#)
- Program Threads, [38](#)
  - ADC0\_SS3\_Handler, [39](#)
  - GPIO\_PortF\_Handler, [39](#)
  - Timer1A\_Handler, [39](#)
- QRS, [38](#)
- QRS.h, [45](#)
- SPI, [18](#)
  - SPI\_Init, [19](#)
  - SPI\_Read, [19](#)
  - SPI\_WriteCmd, [19](#)
  - SPI\_WriteData, [20](#)
  - SPI\_WriteSequence, [21](#)
- SPI.c, [58](#)
- SPI.h, [59](#)
- SPI\_Init
  - SPI, [19](#)
- SPI\_Read
  - SPI, [19](#)
- SPI\_WriteCmd
  - SPI, [19](#)
- SPI\_WriteData
  - SPI, [20](#)
- SPI\_WriteSequence
  - SPI, [21](#)
- SysTick, [21](#)
  - SysTick\_Interrupt\_Init, [22](#)

- SysTick\_Timer\_Init, [22](#)
- SysTick.c, [60](#)
- SysTick.h, [61](#)
- SysTick\_Interrupt\_Init
  - SysTick, [22](#)
- SysTick\_Timer\_Init
  - SysTick, [22](#)
- Timer, [22](#)
  - Timer0A\_Init, [23](#)
  - Timer0A\_isCounting, [23](#)
  - Timer0A\_Start, [24](#)
  - Timer0A\_Wait1ms, [24](#)
  - Timer1A\_Init, [26](#)
  - Timer2A\_Init, [26](#)
  - Timer2A\_isCounting, [26](#)
  - Timer2A\_Start, [26](#)
  - Timer2A\_Wait1ms, [27](#)
- Timer.c, [62](#)
- Timer.h, [63](#)
- Timer0A\_Init
  - Timer, [23](#)
- Timer0A\_isCounting
  - Timer, [23](#)
- Timer0A\_Start
  - Timer, [24](#)
- Timer0A\_Wait1ms
  - Timer, [24](#)
- Timer1A\_Handler
  - Program Threads, [39](#)
- Timer1A\_Init
  - Timer, [26](#)
- Timer2A\_Init
  - Timer, [26](#)
- Timer2A\_isCounting
  - Timer, [26](#)
- Timer2A\_Start
  - Timer, [26](#)
- Timer2A\_Wait1ms
  - Timer, [27](#)
- UART, [27](#)
  - UART0\_Init, [28](#)
  - UART0\_ReadChar, [28](#)
  - UART0\_WriteChar, [29](#)
  - UART0\_WriteStr, [29](#)
  - UART1\_Init, [30](#)
  - UART1\_ReadChar, [30](#)
  - UART1\_WriteChar, [30](#)
  - UART1\_WriteStr, [31](#)
- UART.c, [64](#)
- UART.h, [65](#)
- UART0\_Init
  - UART, [28](#)
- UART0\_ReadChar
  - UART, [28](#)
- UART0\_WriteChar
  - UART, [29](#)
- UART0\_WriteStr



- UART, [29](#)
- UART1\_Init
  - UART, [30](#)
- UART1\_ReadChar
  - UART, [30](#)
- UART1\_WriteChar
  - UART, [30](#)
- UART1\_WriteStr
  - UART, [31](#)
- UserCtrl, [38](#)
- UserCtrl.h, [46](#)
  - UserCtrl\_Init, [47](#)
- UserCtrl\_Init
  - UserCtrl.h, [47](#)