ECG-HRM

Generated by Doxygen 1.9.8

1 Topic Index

1.1 Topics

Here is a list of all topics with brief descriptions:

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

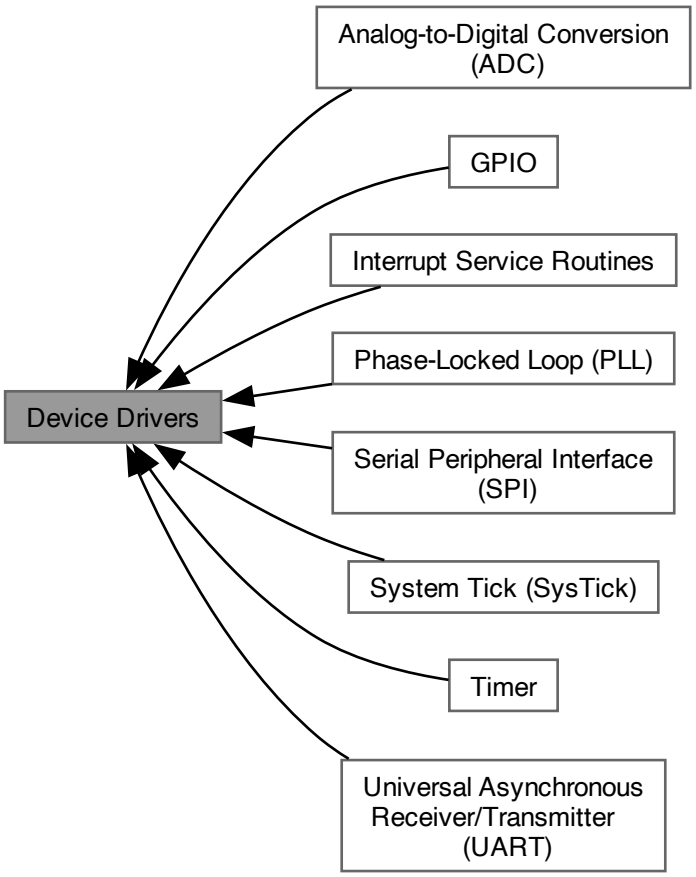
Here is a list of all documented files with brief descriptions:

4 Topic Documentation

4.1 Device Drivers

Low level device driver modules.

Collaboration diagram for Device Drivers:



Modules

- Analog-to-Digital Conversion (ADC)
- GPIO
- Phase-Locked Loop (PLL)
- Serial Peripheral Interface (SPI)
- System Tick (SysTick)
- Timer
- Universal Asynchronous Receiver/Transmitter (UART)
- Interrupt Service Routines

4.1.1 Detailed Description

Low level device driver modules.

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

4.1.2 Analog-to-Digital Conversion (ADC)

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

Files

- file ADC.c

  Source code for ADC module.
- file ADC.h

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

Functions

- void **ADC_Init** (void)

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

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

  Disable the ADC interrupt.
- void **ADC_InterruptAcknowledge** (void)

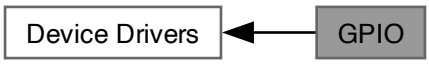
  Acknowledge the ADC interrupt, clearing the flag.

4.1.2.1 Detailed Description

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

4.1.3 GPIO

Collaboration diagram for GPIO:



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

4.1.4 Phase-Locked Loop (PLL)

Collaboration diagram for Phase-Locked Loop (PLL):

Files

- file PLL.c

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

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

Functions

- void **PLL_Init** (void)

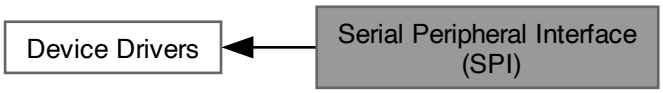
  Initializes the phase-locked-loop (PLL), allowing a bus frequency of 80[MHz].

4.1.4.1 Detailed Description

Function for initializing the phase-locked loop.

4.1.5 Serial Peripheral Interface (SPI)

Collaboration diagram for Serial Peripheral Interface (SPI):



Files

- file SPI.c

  Source code for SPI module.
- file SPI.h

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

Macros

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

Enumerations

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

Functions

- void SPI_Init (void)

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

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

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

  Write 8-bit data to the peripheral.

4.1.5.1 Detailed Description

Functions for SPI-based communication via SSI0 peripheral.

4.1.5.2 Macro Definition Documentation

SPI_SET_DC

```
#define SPI_SET_DC( ) (GPIO_PORTA_DATA_R |= 0x40)
```

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

Clk. Polarity = steady state low (0)
Clk. Phase = rising clock edge (0)

4.1.5.3 Function Documentation

SPI_Init()

```
void SPI_Init (
             void  )
```

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

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

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

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

SPI_Read()

```
uint8_t SPI_Read (
             void  )
```

Read data from the peripheral.

Returns

uint8_t

SPI_WriteCmd()

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

Write an 8-bit command to the peripheral.

Parameters

cmd	command for peripheral

SPI_WriteData()

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

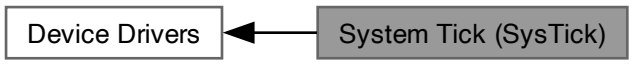
Write 8-bit data to the peripheral.

Parameters

data	input data for peripheral

4.1.6 System Tick (SysTick)

Collaboration diagram for System Tick (SysTick):



Files

- file SysTick.c

  Implementation details for SysTick functions.
- file SysTick.h

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

Functions

- void **SysTick_Timer_Init** (void)

  Initialize SysTick for timing purposes.
- void **SysTick_Wait1ms** (uint32_t delay_ms)

  Delay for specified amount of time in [ms]. Assumes f_bus = 80[MHz].
- void SysTick_Interrupt_Init (uint32_t time_ms)

  Initialize SysTick for interrupts.

4.1.6.1 Detailed Description

Functions for timing and periodic interrupts via SysTick.

4.1.6.2 Function Documentation

SysTick_Interrupt_Init()

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

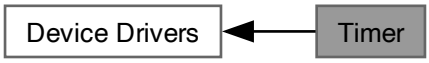
Initialize SysTick for interrupts.

Parameters

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

4.1.7 Timer

Collaboration diagram for Timer:



Files

- file Timer.c

  Source code for Timer module.
- file Timer.h

  Device driver for general-purpose timer modules.

Data Structures

- struct Timer_t

Typedefs

- typedef volatile uint32_t * **register_t**

Enumerations

- enum {
  TIMER0_BASE = 0x40030000 , **TIMER1_BASE** = 0x40031000 , **TIMER2_BASE** = 0x40032000 , **TIMER3_BASE** = 0x40033000 ,
  TIMER4_BASE = 0x40034000 , **TIMER5_BASE** = 0x40035000 }
- enum **REGISTER_OFFSETS** {
  CONFIG = 0x00 , **MODE** = 0x04 , **CTRL** = 0x0C , **INT_MASK** = 0x18 ,
  INT_CLEAR = 0x24 , **INTERVAL** = 0x28 , **VALUE** = 0x054 }
- enum **timerName_t** {
  TIMER0 , **TIMER1** , **TIMER2** , **TIMER3** ,
  TIMER4 , **TIMER5** }
- enum **timerMode_t** { **ONESHOT** , **PERIODIC** }
- enum { **UP** = true , **DOWN** = false }

Functions

- Timer_t **Timer_Init** (timerName_t timerName)
- timerName_t **Timer_getName** (Timer_t timer)
- void **Timer_setMode** (Timer_t timer, timerMode_t timerMode, bool isCountingUp)
- void **Timer_enableAdcTrigger** (Timer_t timer)
- void **Timer_disableAdcTrigger** (Timer_t timer)
- void **Timer_enableInterruptOnTimeout** (Timer_t timer, uint8_t priority)
- void **Timer_disableInterruptOnTimeout** (Timer_t timer)
- void **Timer_clearInterruptFlag** (Timer_t timer)
- void **Timer_setInterval_ms** (Timer_t timer, uint32_t time_ms)
- uint32_t **Timer_getCurrentValue** (Timer_t timer)
- void **Timer_Start** (Timer_t timer)
- void **Timer_Stop** (Timer_t timer)
- bool **Timer_isCounting** (Timer_t timer)
- void **Timer_Wait1ms** (Timer_t timer, uint32_t time_ms)

Variables

- static TimerStruct_t TIMER_POOL [6]

4.1.7.1 Detailed Description

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

4.1.7.2 Variable Documentation

TIMER_POOL

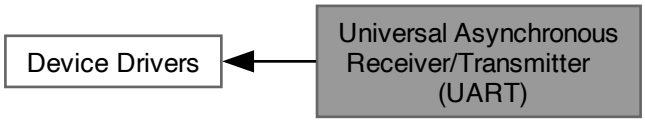
```
TimerStruct_t TIMER_POOL[6]  [static]
```

Initial value:

```
= {
    { TIMER0, TIMER0_BASE, (register_t) (TIMER0_BASE + CTRL), (register_t) (TIMER0_BASE + INTERVAL),
      (register_t) (TIMER0_BASE + INT_CLEAR), false },
    { TIMER1, TIMER1_BASE, (register_t) (TIMER1_BASE + CTRL), (register_t) (TIMER1_BASE + INTERVAL),
      (register_t) (TIMER1_BASE + INT_CLEAR), false },
    { TIMER2, TIMER2_BASE, (register_t) (TIMER2_BASE + CTRL), (register_t) (TIMER2_BASE + INTERVAL),
      (register_t) (TIMER2_BASE + INT_CLEAR), false },
    { TIMER3, TIMER3_BASE, (register_t) (TIMER3_BASE + CTRL), (register_t) (TIMER3_BASE + INTERVAL),
      (register_t) (TIMER3_BASE + INT_CLEAR), false },
    { TIMER4, TIMER4_BASE, (register_t) (TIMER4_BASE + CTRL), (register_t) (TIMER4_BASE + INTERVAL),
      (register_t) (TIMER4_BASE + INT_CLEAR), false },
    { TIMER5, TIMER5_BASE, (register_t) (TIMER5_BASE + CTRL), (register_t) (TIMER5_BASE + INTERVAL),
      (register_t) (TIMER5_BASE + INT_CLEAR), false }
}
```

4.1.8 Universal Asynchronous Receiver/Transmitter (UART)

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



Files

- file UART.c

  Source code for UART module.
- file UART.h

  Driver module for serial communication via UART0 and UART 1.

Data Structures

- struct UART_t

Macros

- #define **ASCII_CONVERSION** 0x30

Typedefs

- typedef volatile uint32_t * **register_t**

Enumerations

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

Functions

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

  Initialize the specified UART peripheral.
- unsigned char UART_ReadChar (UART_t *uart)

  Read a single ASCII character from the UART.
- void UART_WriteChar (UART_t *uart, unsigned char input_char)

  Write a single character to the UART.
- void UART_WriteStr (UART_t *uart, void *input_str)

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

  Write a 32-bit unsigned integer the UART.
- void UART_WriteFloat (UART_t *uart, double n, uint8_t num_decimals)

  Write a floating-point number the UART.

Variables

- static UART_t UART_ARR [8]

4.1.8.1 Detailed Description

Functions for UART-based communication.

4.1.8.2 Function Documentation

UART_Init()

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

Initialize the specified UART peripheral.

Parameters

in	*port*	GPIO port to use.
in	*uartNum*	UART number. Should be either one of the enumerated constants or an int in range [0, 7].
out	*UART_t**	(Pointer to) initialized UART peripheral.

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

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

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

UART_ReadChar()

```
unsigned char UART_ReadChar (
            UART_t * uart )
```

Read a single ASCII character from the UART.

Parameters

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

UART_WriteChar()

```
void UART_WriteChar (
            UART_t * uart,
            unsigned char input_char )
```

Write a single character to the UART.

Parameters

in	*uart*	UART to read from.
in	*input_char*	ASCII character to send.

UART_WriteStr()

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

Write a C string to the UART.

Parameters

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

UART_WriteInt()

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

Write a 32-bit unsigned integer the UART.

Parameters

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

UART_WriteFloat()

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

Write a floating-point number the UART.

Parameters

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

4.1.8.3 Variable Documentation

UART_ARR

```
UART_t UART_ARR[8]  [static]
```

Initial value:

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

4.1.9 Interrupt Service Routines

Collaboration diagram for Interrupt Service Routines:

Files

- file ISR.c

  Source code for interrupt vector handling module.
- file ISR.h

  Module for configuring interrupt service routines (ISRs).

Macros

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

Typedefs

- typedef volatile uint32_t * **register_t**
- typedef void(* **ISR_t**) (void)

  Type definition for function pointers representing ISRs.

Functions

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

  Disable all interrupts globally.
- void ISR_GlobalEnable (void)

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

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

  Add an ISR to the interrupt table.
- void ISR_setPriority (const uint8_t vectorNum, const uint8_t priority)

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

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

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

  Generate a software-generated interrupt (SGI).
- void ISR_clearPending (const uint8_t vectorNum)

  Clear an ISR's pending bit.

Variables

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

4.1.9.1 Detailed Description

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

4.1.9.2 Function Documentation

ISR_GlobalDisable()

```
void ISR_GlobalDisable (
             void  )
```

Disable all interrupts globally.

See also

ISR_GlobalEnable()

ISR_GlobalEnable()

```
void ISR_GlobalEnable (
             void  )
```

Enable all interrupts globally.

See also

ISR_GlobalDisable()

ISR_InitNewTableInRam()

```
void ISR_InitNewTableInRam (
             void  )
```

Relocate the vector table to RAM.

Precondition

Call this after disabling interrupts globally.

Postcondition

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

See also

ISR_GlobalDisable(), ISR_addToIntTable()

ISR_addToIntTable()

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

Add an ISR to the interrupt table.

Precondition

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

Parameters

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

Postcondition

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

See also

ISR_relocateIntTableToRam()

ISR_setPriority()

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

Set the priority for an interrupt.

Parameters

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

ISR_Enable()

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

Enable an interrupt in the NVIC.

Precondition

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

Parameters

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

See also

ISR_setPriority(), ISR_Disable()

ISR_Disable()

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

Disable an interrupt in the NVIC.

Parameters

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

See also

ISR_Enable()

ISR_triggerInterrupt()

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

Generate a software-generated interrupt (SGI).

Precondition

Enable the ISR (and set priority as needed) for calling this.

Enable all interrupts before calling this.

Parameters

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

Postcondition

The ISR should trigger once any higher priority ISRs return.

See also

ISR_clearPending()

ISR_clearPending()

```
void ISR_clearPending (
            const uint8_t vectorNum )
```

Clear an ISR's pending bit.

Precondition

This should be called during the ISR for an SGI.

Parameters

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

Postcondition

The ISR should not trigger again until re-activated.

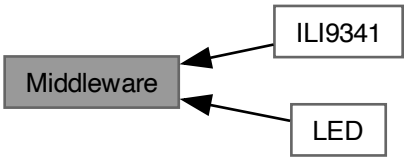
See also

ISR_triggerInterrupt()

4.2 Middleware

High-level device driver modules.

Collaboration diagram for Middleware:



Modules

- ILI9341
- LED

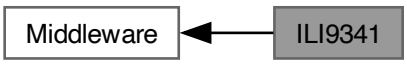
4.2.1 Detailed Description

High-level device driver modules.

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

4.2.2 ILI9341

Collaboration diagram for ILI9341:



Files

- file ILI9341.c

  Source code for ILI9341 module.
- file ILI9341.h

  Driver module for interfacing with an ILI9341 LCD driver.

Enumerations

- enum Cmd_t {
  NOP = 0x00 , SWRESET = 0x01 , SPLIN = 0x10 , SPLOUT = 0x11 ,
  PTLON = 0x12 , NORON = 0x13 , DINVOFF = 0x20 , DINVON = 0x21 ,
  CASET = 0x2A , PASET = 0x2B , RAMWR = 0x2C , DISPOFF = 0x28 ,
  DISPON = 0x29 , PLTAR = 0x30 , VSCRDEF = 0x33 , MADCTL = 0x36 ,
  VSCRSADD = 0x37 , IDMOFF = 0x38 , IDMON = 0x39 , PIXSET = 0x3A ,
  FRMCTR1 = 0xB1 , FRMCTR2 = 0xB2 , FRMCTR3 = 0xB3 , PRCTR = 0xB5 ,
  IFCTL = 0xF6 }
- enum { **ILI9341_NUM_COLS** = 240 , **ILI9341_NUM_ROWS** = 320 }

Functions

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

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

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

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

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

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

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

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

- void ILI9341_setDispInversion (bool is_ON)

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

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

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

  Set the start row for vertical scrolling.
- void ILI9341_setMemAccessCtrl (bool areRowsFlipped, bool areColsFlipped, bool areRowsAndCols↩ Switched, bool isVertRefreshFlipped, bool isColorOrderFlipped, bool isHorRefreshFlipped)

  Set how data is converted from memory to display.
- void ILI9341_setColorDepth (bool is_16bit)

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

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

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

  TODO: Write brief.
- void ILI9341_setInterface (void)

  Sets the interface for the ILI9341. The parameters for this command are hard-coded, so it only needs to be called once upon initialization.
- void ILI9341_setRowAddress (uint16_t startRow, uint16_t endRow)

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

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

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

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

  TODO: Write.

Variables

- static uint32_t ILI9341_Buffer [8]
- static FIFO_t * **ILI9341_Fifo**

4.2.2.1 Detailed Description

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

4.2.2.2 Enumeration Type Documentation

Cmd_t

```
enum Cmd_t
```

Enumerator

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

4.2.2.3 Function Documentation

ILI9341_setAddress()

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

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

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

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

ILI9341_sendParams()

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

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

Parameters

in	*cmd*	Command to send.

ILI9341_resetHard()

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

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 (
            Timer_t timer )
```

Perform a software reset of the LCD driver.

the driver needs 5 [ms] before another command

ILI9341_setSleepMode()

```
void ILI9341_setSleepMode (
            bool isSleeping,
            Timer_t timer )
```

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

Parameters

isSleeping	`true` to enter sleep mode, `false` to exit

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

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

ILI9341_setDispMode()

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

Set the display area and color expression.

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

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

Parameters

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

ILI9341_setPartialArea()

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

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

Parameters

rowStart	
rowEnd	

ILI9341_setDispInversion()

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

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

Parameters

is_ON	`true` to turn ON, `false` to turn OFF

TODO: Write description

ILI9341_setDispOutput()

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

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

Parameters

is_ON	`true` to turn ON, `false` to turn OFF

TODO: Write description

ILI9341_setScrollArea()

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

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

Parameters

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

ILI9341_setScrollStart()

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

Set the start row for vertical scrolling.

Parameters

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

ILI9341_setMemAccessCtrl()

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

Set how data is converted from memory to display.

Parameters

in	*areRowsFlipped*	
in	*areColsFlipped*	
in	*areRowsAndColsSwitched*	
in	*isVertRefreshFlipped*	
in	*isColorOrderFlipped*	
in	*isHorRefreshFlipped*	

This function implements the "Memory Access Control" (MADCTL) command from p. 127-128 of the ILI9341 datasheet, which controls how the LCD driver displays data upon writing to memory.

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

All bits are clear after powering on or HWRESET.

ILI9341_setColorDepth()

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

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

Parameters

is_16bit	

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

ILI9341_setFrameRateNorm()

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

TODO: Write brief.

TODO: Write description

ILI9341_setFrameRateIdle()

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

TODO: Write brief.

TODO: Write description

ILI9341_setInterface()

```
void ILI9341_setInterface (
             void  )
```

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

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

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

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

ILI9341_setRowAddress()

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

not using backlight, so these aren't necessary

Sets the start/end rows to be written to.

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

Parameters

startRow	0 <= `startRow` <= `endRow`
endRow	`startRow` <= `endRow` < 320

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

ILI9341_setColAddress()

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

Sets the start/end rows to be written to.

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

Parameters

startCol	0 <= `startCol` <= `endCol`
endCol	`startCol` <= `endCol` < 240

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

ILI9341_writeMemCmd()

```
void ILI9341_writeMemCmd (
             void  )
```

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

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

ILI9341_writePixel()

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

Write a single pixel to frame memory.

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

Parameters

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

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

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

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

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

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

4.2.2.4 Variable Documentation

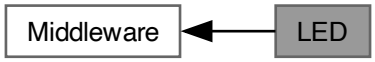
ILI9341_Buffer

```
uint32_t ILI9341_Buffer[8]  [static]
```

Currently unused commands #define RDDST (uint8_t) 0x09 /// Read Display Status #define RDDMADCTL (uint8↩_t) 0x0B /// Read Display MADCTL #define RDDCOLMOD (uint8_t) 0x0C /// Read Display Pixel Format #define RGBSET (uint8_t) 0x2D /// Color Set #define RAMRD (uint8_t) 0x2E /// Memory Read #define WRITE_MEMORY↩_CONTINUE (uint8_t) 0x3C /// Write_Memory_Continue #define READ_MEMORY_CONTINUE (uint8_t) 0x3E /// Read_Memory_Continue #define WRDISBV (uint8_t) 0x51 /// Write Display Brightness #define RDDISBV (uint8_t) 0x52 /// Read Display Brightness #define IFMODE (uint8_t) 0xB0 /// RGB Interface Signal Control (i.e. Interface Mode Control) #define INVTR (uint8_t) 0xB4 /// Display Inversion Control

4.2.3 LED

Collaboration diagram for LED:



Files

- file Led.c

  Source code for LED module.
- file Led.h

  Interface for LED module.

Data Structures

- struct Led_t

Macros

- #define **LED_POOL_SIZE** 3

Functions

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

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

  Get the GPIO port associated with the LED.
- GPIO_Pin_t Led_GetPin (Led_t *led)

  Get the GPIO pin associated with the LED.
- bool Led_isOn (Led_t *led)

  Check the LED's status.
- void Led_TurnOn (Led_t *led)

  Turn the LED `ON`.
- void Led_TurnOff (Led_t *led)

  Turn the LED `OFF`.
- void Led_Toggle (Led_t *led)

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

Variables

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

4.2.3.1 Detailed Description

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

4.2.3.2 Function Documentation

Led_Init()

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

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

Parameters

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

Led_GetPort()

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

Get the GPIO port associated with the LED.

Parameters

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

Led_GetPin()

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

Get the GPIO pin associated with the LED.

Parameters

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

Led_isOn()

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

Check the LED's status.

Parameters

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

Led_TurnOn()

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

Turn the LED `ON`.

Parameters

in	*led*	Pointer to LED data structure.

Led_TurnOff()

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

Turn the LED OFF.

Parameters

in	*led*	Pointer to LED data structure.

Led_Toggle()

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

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

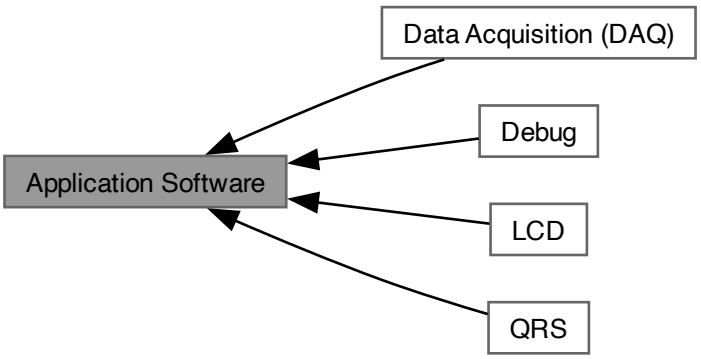
Parameters

in	*led*	Pointer to LED data structure.

4.3 Application Software

Application-specific software modules.

Collaboration diagram for Application Software:



Modules

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

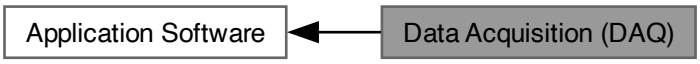
4.3.1 Detailed Description

Application-specific software modules.

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

4.3.2 Data Acquisition (DAQ)

Collaboration diagram for Data Acquisition (DAQ):



Files

- file DAQ.c
  Source code for DAQ module.
- file DAQ.h
  Application software for handling data acquision (DAQ) functions.
- file lookup.c
  Source code for DAQ module's lookup table.
- file lookup.h
  Lookup table for DAQ module.

Macros

- #define **SAMPLING_PERIOD_MS** 5
  sampling period in ms ($T_s = 1/f_s$)
- #define **LOOKUP_DAQ_MAX** (float32_t) 5.5
- #define **LOOKUP_DAQ_MIN** (float32_t)(-5.5)

Typedefs

- typedef arm_biquad_casd_df1_inst_f32 **Filter_t**

Enumerations

- enum {
  NUM_STAGES_NOTCH = 6 , **NUM_COEFFS_NOTCH** = NUM_STAGES_NOTCH $*$ 5 , **STATE_BUFF_↩**
  SIZE_NOTCH = NUM_STAGES_NOTCH $*$ 4 , **NUM_STAGES_BANDPASS** = 4 ,
  NUM_COEFFS_DAQ_BANDPASS = NUM_STAGES_BANDPASS $*$ 5 , **STATE_BUFF_SIZE_BANDPASS** = NUM_STAGES_BANDPASS $*$ 4 }

Functions

- const float32_t * Lookup_GetPtr (void)

  Return a pointer to the DAQ lookup table.

Variables

- static const float32_t * **DAQ_LOOKUP_TABLE** = 0
- static const float32_t COEFFS_NOTCH [NUM_COEFFS_NOTCH]
- static const float32_t COEFFS_BANDPASS [NUM_COEFFS_DAQ_BANDPASS]
- static float32_t **stateBuffer_Notch** [STATE_BUFF_SIZE_NOTCH]
- static const Filter_t **notchFiltStruct** = { NUM_STAGES_NOTCH, stateBuffer_Notch, COEFFS_NOTCH }
- static const Filter_t *const **notchFilter** = &notchFiltStruct
- static float32_t **stateBuffer_Bandpass** [STATE_BUFF_SIZE_BANDPASS]
- static const Filter_t bandpassFiltStruct
- static const Filter_t *const **bandpassFilter** = &bandpassFiltStruct
- static const float32_t **LOOKUP_DAQ_TABLE** [4096]

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

Initialization

- void DAQ_Init (void)

  Initialize the data acquisition (DAQ) module.

Reading Input Data

- uint16_t DAQ_readSample (void)

  Read a sample from the ADC.
- float32_t DAQ_convertToMilliVolts (uint16_t sample)

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

Digital Filtering Functions

- float32_t DAQ_NotchFilter (volatile float32_t xn)

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

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

4.3.2.1 Detailed Description

Module for managing data acquisition (DAQ) functions.

4.3.2.2 Function Documentation

DAQ_Init()

```
void DAQ_Init (
            void  )
```

Initialize the data acquisition (DAQ) module.

Postcondition

The ADC and Timer are initialized, and the DAQ module has access to its lookup table (LUT).

DAQ_readSample()

```
uint16_t DAQ_readSample (
            void  )
```

Read a sample from the ADC.

Precondition

Initialize the DAQ module.

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

Parameters

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

Postcondition

The sample can now be converted to millivolts.

See also

DAQ_convertToMilliVolts()

DAQ_convertToMilliVolts()

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

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

Precondition

Read a sample from the ADC.

Parameters

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

Postcondition

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

See also

DAQ_readSample()

DAQ_NotchFilter()

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

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

Precondition

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

Parameters

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

Postcondition

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

See also

DAQ_BandpassFilter()

DAQ_BandpassFilter()

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

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

Precondition

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

Parameters

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

Postcondition

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

See also

DAQ_NotchFilter()

Lookup_GetPtr()

```
const float32_t * Lookup_GetPtr (
            void  )
```

Return a pointer to the DAQ lookup table.

Returns

const float32_t*

4.3.2.3 Variable Documentation

COEFFS_NOTCH

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

Initial value:

```
= {

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

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

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

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

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

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

COEFFS_BANDPASS

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

Initial value:

```
= {

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

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

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

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

bandpassFiltStruct

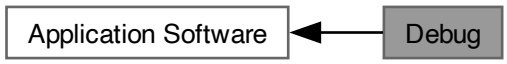
```
const Filter_t bandpassFiltStruct  [static]
```

Initial value:

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

4.3.3 Debug

Collaboration diagram for Debug:



Files

- file Debug.h

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

Serial Output

- enum **Msg_t** { **DEBUG_DAQ_INIT** , **DEBUG_QRS_INIT** , **DEBUG_LCD_INIT** , **DEBUG_QRS_START** }
- void Debug_SendMsg (void ∗message)

  Send a message to the serial port.
- void Debug_SendFromList (Msg_t msg)

  Send a message from the message list.
- void Debug_WriteFloat (double value)

  Write a floating-point value to the serial port.

Initialization

- void Debug_Init (void)

  Initialize the Debug module.

Assertions

- void Debug_Assert (bool condition)

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

4.3.3.1 Detailed Description

Module for debugging functions, including serial output and assertion.

4.3.3.2 Function Documentation

Debug_Init()

```
void Debug_Init (
              void  )
```

Initialize the Debug module.

Postcondition

An initialization message is sent to the serial port (UART0).

Debug_SendMsg()

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

Send a message to the serial port.

Precondition

Initialize the Debug module.

Parameters

message	(Pointer to) array of ASCII characters.

Postcondition

A floating point value is written to the serial port.

See also

Debug_SendMsg()

Debug_SendFromList()

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

Send a message from the message list.

Precondition

Initialize the Debug module.

Parameters

in	*msg*	An entry from the enumeration.

Postcondition

The corresponding message is sent to the serial port.

See also

Debug_SendMsg()

Debug_WriteFloat()

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

Write a floating-point value to the serial port.

Precondition

Initialize the Debug module.

Parameters

in	*value*	Floating-point value.

Postcondition

A floating point value is written to the serial port.

See also

Debug_SendMsg()

Debug_Assert()

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

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

Precondition

Initialize the Debug module.

Parameters

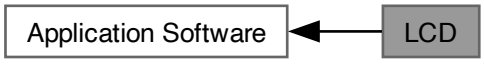
in	*condition*	Conditional statement to evaluate.

Postcondition

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

4.3.4 LCD

Collaboration diagram for LCD:



Files

- file LCD.c

  Source code for LCD module.
- file LCD.h

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

Enumerations

- enum { **LCD_X_MAX** = ILI9341_NUM_ROWS , **LCD_Y_MAX** = ILI9341_NUM_COLS }

Functions

- static void **LCD_updateNumPixels** (void)

  Updates `lcd`'s `numPixels` parameter after changing rows/columns.
- static void LCD_setDim (uint16_t d1, uint16_t d2, bool is_x, bool update_num_pixels)

  Set new `x` or `y` parameters, and optionally update `numPixels`.
- static void LCD_drawLine (uint16_t center, uint16_t lineWidth, bool is_horizontal)

  Helper function for drawing straight lines.

Variables

- struct {
  uint16_t **x1**
  starting x-value in range [0, x2]
  uint16_t **x2**
  ending x-value in range [0, NUM_ROWS)
  uint16_t **y1**
  starting y-value in range [0, y2]
  uint16_t **y2**
  ending x-value in range [0, NUM_COLS)
  uint32_t **numPixels**
  num. of pixels to write; = `(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 **isOutputOn**
  if `true`, LCD driver writes from its memory to display
  bool **isInverted**
  if `true`, the display's colors are inverted
  bool **using16bitColors**
  `true` for 16-bit color depth, `false` for 18-bit
  bool **isInit**
  if `true`, LCD has been initialized
  } **lcd**

Color Setting Functions

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

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

  Set the color value via a 3-bit code.

Init./Config. Functions

- void **LCD_Init** (void)

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

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

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

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

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

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

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

Drawing Area Definition Functions

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

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

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

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

Drawing Functions

- void LCD_Draw (void)

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

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

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

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

  Draw a rectangle of size `dx` x `dy` onto the display. The bottom-left corner will be located at `(x1, y1)`.
- void LCD_graphSample (uint16_t x1, uint16_t dx, uint16_t y1, uint16_t dy, uint16_t y_min, uint16_t y_max, uint16_t color_code)

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

4.3.4.1 Detailed Description

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

4.3.4.2 Function Documentation

LCD_setDim()

```
static void LCD_setDim (
             uint16_t d1,
             uint16_t d2,
             bool is_x,
             bool update_num_pixels )  [inline], [static]
```

Set new `x` or `y` parameters, and optionally update `numPixels`.

Parameters

d1	start index of selected dimension
d2	end index of selected dimension
is_x	`true` if dimension is `x`, `false` if `y`
update_num_pixels	`true` to update `lcd.numPixels`, `false` if not

LCD_drawLine()

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

Helper function for drawing straight lines.

Parameters

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

LCD_setOutputMode()

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

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

Parameters

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

See also

LCD_toggleOutput()

LCD_toggleOutput()

```
void LCD_toggleOutput (
            void  )
```

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

See also

LCD_setOutputMode()

LCD_setColorInversionMode()

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

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

Parameters

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

See also

LCD_toggleColorInversion(), LCD_setColor(), LCD_setColor_3bit()

LCD_toggleColorInversion()

```
void LCD_toggleColorInversion (
            void  )
```

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

See also

LCD_setColorInversionMode(), LCD_setColor(), LCD_setColor_3bit()

LCD_setColorDepth()

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

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

Parameters

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

See also

LCD_toggleColorDepth(), LCD_setColor(), LCD_setColor_3bit()

LCD_toggleColorDepth()

```
void LCD_toggleColorDepth (
            void  )
```

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

See also

LCD_setColorDepth(), LCD_setColor(), LCD_setColor_3bit()

LCD_setArea()

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

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

Parameters

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

See also

LCD_setX(), LCD_setY()

LCD_setX()

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

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

Parameters

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

See also

LCD_setY(), LCD_setArea()

LCD_setY()

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

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

Parameters

y1_new	lowest y-coordinate
y2_new	highest y-coordinate

See also

LCD_setX(), LCD_setArea()

LCD_setColor()

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

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

Parameters

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

See also

LCD_setColorDepth(), LCD_toggleColorDepth(), LCD_setColor_3bit()

LCD_setColor_3bit()

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

Set the color value via a 3-bit code.

Parameters

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

See also

LCD_setColorDepth(), LCD_toggleColorDepth(), LCD_setColor()

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

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

LCD_Draw()

```
void LCD_Draw (
            void  )
```

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

LCD_drawHoriLine()

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

Draw a horizontal line across the entire display.

Parameters

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

@seeLCD_drawVertLine, LCD_drawRectangle()

LCD_drawVertLine()

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

Draw a vertical line across the entire display.

Parameters

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

@seeLCD_drawHoriLine, LCD_drawRectangle()

LCD_drawRectangle()

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

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

Parameters

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

LCD_graphSample()

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

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

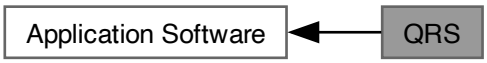
Parameters

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

TODO: Write description

4.3.5 QRS

Collaboration diagram for QRS:



Files

- file QRS.c

  Source code for QRS module.
- file QRS.h

  QRS detection algorithm functions.

Macros

- #define **QRS_NUM_FID_MARKS** 20
- #define **FLOAT_COMPARE_TOLERANCE** (float32_t)(1E-5f)
- #define **IS_GREATER**(X, Y) (bool) ((X - Y) > FLOAT_COMPARE_TOLERANCE)
- #define **IS_LESSER**(X, Y) (bool) ((Y - X) > FLOAT_COMPARE_TOLERANCE)
- #define **IS_PEAK**(X_MINUS_1, X, X_PLUS_1) (bool) (IS_GREATER(X, X_MINUS_1) && IS_GREATER(X, X_PLUS_1))
- #define **QRS_SAMP_FREQ** ((uint32_t) 200)
- #define **QRS_SAMP_PERIOD_SEC** ((float32_t) 0.005f)
- #define **QRS_NUM_SAMP** ((uint16_t) (1200))

Typedefs

- typedef arm_biquad_casd_df1_inst_f32 **IIR_Filt_t**
- typedef arm_fir_instance_f32 **FIR_Filt_t**

Enumerations

- enum {
  NUM_STAGES_BANDPASS = 4 , **NUM_COEFF_HIGHPASS** = NUM_STAGES_BANDPASS ∗ 5 , **STATE↩
  _BUFF_SIZE_BANDPASS** = NUM_STAGES_BANDPASS ∗ 4 , **NUM_COEFF_DERFILT** = 5 ,
  STATE_BUFF_SIZE_DERFILT = NUM_COEFF_DERFILT + QRS_NUM_SAMP - 1 , **NUM_COEFF_↩
  MOVAVG** = 10 , **STATE_BUFF_SIZE_MOVAVG** = NUM_COEFF_MOVAVG + QRS_NUM_SAMP - 1 }

Functions

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

  Mark local peaks in the input signal y as potential candidates for QRS complexes (AKA "fiducial marks").
- static void QRS_initLevels (const float32_t yn[ ])

  Initialize the signal and noise levels for the QRS detector using the initial block of input signal data.
- static float32_t QRS_updateLevel (float32_t peakAmplitude, float32_t level)

  Update signal or noise level based on a confirmed peak's amplitude.
- static float32_t QRS_updateThreshold (void)

  Update the amplitude threshold used to identify peaks based on the signal and noise levels.
- void **QRS_Init** (void)

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

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

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

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

Variables

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

- static const float32_t COEFF_BANDPASS [NUM_COEFF_HIGHPASS]
- static const float32_t **COEFF_DERFILT** [NUM_COEFF_DERFILT] = { -0.125f, -0.25f, 0.0f, 0.25f, 0.125f }
- static const float32_t COEFF_MOVAVG [NUM_COEFF_MOVAVG]
- static float32_t **stateBuffer_bandPass** [STATE_BUFF_SIZE_BANDPASS] = { 0 }
- static const IIR_Filt_t **bandpassFiltStruct** = { NUM_STAGES_BANDPASS, stateBuffer_bandPass, COEFF↩
  _BANDPASS }
- static const IIR_Filt_t ∗const **bandpassFilter** = &bandpassFiltStruct
- static float32_t **stateBuffer_DerFilt** [STATE_BUFF_SIZE_DERFILT] = { 0 }
- static const FIR_Filt_t **derivativeFiltStruct** = { NUM_COEFF_DERFILT, stateBuffer_DerFilt, COEFF_↩
  DERFILT }
- static const FIR_Filt_t ∗const **derivativeFilter** = &derivativeFiltStruct
- static float32_t **stateBuffer_MovingAvg** [STATE_BUFF_SIZE_MOVAVG] = { 0 }
- static const FIR_Filt_t **movingAvgFiltStruct** = { NUM_COEFF_MOVAVG, stateBuffer_MovingAvg, COEFF↩
  _MOVAVG }
- static const FIR_Filt_t ∗const **movingAverageFilter** = &movingAvgFiltStruct

4.3.5.1 Detailed Description

Module for analyzing ECG data to determine heart rate.

4.3.5.2 Function Documentation

QRS_findFiducialMarks()

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

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

Parameters

in	*yn*	Array containing the preprocessed ECG signal $y[n]$
in	*fidMarkArray*	Array to place the fiducial mark's sample indices into.
out	*uint8_t*	Number of identified fiducial marks

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

QRS_initLevels()

```
static void QRS_initLevels (
            const float32_t yn[] )  [static]
```

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

Parameters

in	*yn*	Array containing the preprocessed ECG signal $y[n]$

Postcondition

The detector's signal and noise levels are initialized.

QRS_updateLevel()

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

Update signal or noise level based on a confirmed peak's amplitude.

Parameters

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

QRS_updateThreshold()

```
static float32_t QRS_updateThreshold (
            void  )  [static]
```

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

Parameters

out	*threshold*	New threshold to use for next comparison.

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

QRS_Preprocess()

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

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

Precondition

Fill `inputBuffer` with raw or lightly preprocessed ECG data.

Parameters

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

Postcondition

`yn` will contain the preprocessed data, which is ready to be analyzed to calculate HR.

See also

QRS_applyDecisionRules()

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

QRS_applyDecisionRules()

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

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

Precondition

Preprocess the raw ECG data.

Parameters

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

Postcondition

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

See also

QRS_Preprocess()

QRS_runDetection()

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

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

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

Parameters

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

Postcondition

`yn` will contain the preprocessed data.

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

See also

QRS_Preprocess(), QRS_applyDecisionRules()

4.3.5.3 Variable Documentation

COEFF_BANDPASS

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

Initial value:

```
= {

    0.002937758108600974f, 0.005875516217201948f, 0.002937758108600974f,
    1.0485996007919312f, -0.2961403429508209f,

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

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

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

COEFF_MOVAVG

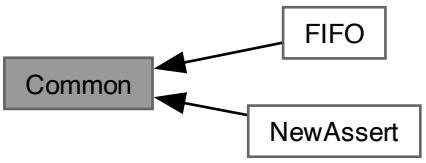
```
const float32_t COEFF_MOVAVG[NUM_COEFF_MOVAVG]  [static]
```

Initial value:

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

4.4 Common

Collaboration diagram for Common:



Modules

- FIFO
- NewAssert

Files

- file NewAssert.c

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

  Header file for custom `assert` implementation.

Functions

- void Assert (bool condition)

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

4.4.1 Detailed Description

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

4.4.2 Function Documentation

Assert()

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

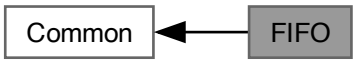
Custom `assert` implementation that is more lightweight than the one from `newlib`.

Parameters

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

4.4.3 FIFO

Collaboration diagram for FIFO:



Files

- file FIFO.c

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

  FIFO buffer data structure.

Data Structures

- struct FIFO_t

Macros

- #define **FIFO_POOL_SIZE** 5

Functions

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

  Initialize a FIFO buffer of length N.

Variables

- static FIFO_t **buffer_pool** [FIFO_POOL_SIZE] = { 0 }

  pre-allocated buffer pool
- static uint8_t **free_buffers** = FIFO_POOL_SIZE

  no. of remaining buffers

Basic Operations

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

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

  Remove the first value of the buffer.
- void FIFO_TransferOne (volatile FIFO_t *srcFifo, volatile FIFO_t *destFifo)

  Transfer a value from one FIFO buffer to another.

Bulk Removal

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

  Empty the FIFO buffer's contents into an array.
- void FIFO_Reset (volatile FIFO_t *fifo)

  Reset the FIFO buffer.
- void FIFO_TransferAll (volatile FIFO_t *srcFifo, volatile FIFO_t *destFifo)

  Transfer the contents of one FIFO buffer to another.

Peeking

- uint32_t FIFO_PeekOne (volatile FIFO_t *fifo)

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

  See the FIFO buffer's contents without removing them.

Status Checks

- bool FIFO_isFull (volatile FIFO_t *fifo)

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

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

  Get the current size of the FIFO buffer.

4.4.3.1 Detailed Description

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

4.4.3.2 Function Documentation

FIFO_Init()

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

Initialize a FIFO buffer of length `N`.

Parameters

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

Returns

pointer to the FIFO buffer

TODO: Add details

FIFO_Put()

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

Add a value to the end of the buffer.

Parameters

fifo	Pointer to FIFO object
val	last value in the buffer

FIFO_Get()

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

Remove the first value of the buffer.

Parameters

fifo	Pointer to FIFO object

Returns

First sample in the FIFO.

FIFO_TransferOne()

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

Transfer a value from one FIFO buffer to another.

Parameters

srcFifo	Pointer to source FIFO buffer.
destFifo	Pointer to destination FIFO buffer.

FIFO_Flush()

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

Empty the FIFO buffer's contents into an array.

Parameters

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

FIFO_Reset()

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

Reset the FIFO buffer.

Parameters

in	*fifo*	Pointer to FIFO buffer.

FIFO_TransferAll()

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

Transfer the contents of one FIFO buffer to another.

Parameters

srcFifo	Pointer to source FIFO buffer.
destFifo	Pointer to destination FIFO buffer.

FIFO_PeekOne()

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

See the first element in the FIFO without removing it.

Parameters

fifo	Pointer to FIFO object

Returns

First sample in the FIFO.

FIFO_PeekAll()

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

See the FIFO buffer's contents without removing them.

Parameters

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

FIFO_isFull()

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

Check if the FIFO buffer is full.

Parameters

fifo	Pointer to the FIFO buffer.

Return values

true	The buffer is full.
false	The buffer is not full.

FIFO_isEmpty()

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

Check if the FIFO buffer is empty.

Parameters

fifo	Pointer to the FIFO buffer.

Return values

true	The buffer is empty.
false	The buffer is not empty.

FIFO_getCurrSize()

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

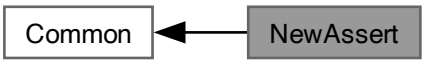
Get the current size of the FIFO buffer.

Parameters

fifo	Pointer to the FIFO buffer.

4.4.4 NewAssert

Collaboration diagram for NewAssert:



Module for using a custom `assert` implementation.

4.5 Main

Files

- file main.c

  Main program file.

Enumerations

- enum { **DAQ_VECTOR_NUM** = INT_ADC0SS3 , **PROC_VECTOR_NUM** = INT_CAN0 , **LCD_VECTOR_NUM** = INT_TIMER1A }
- enum { **DAQ_BUFFER_CAPACITY** = 3 , **DAQ_BUFFER_SIZE** = DAQ_BUFFER_CAPACITY + 1 , **QRS_BUFFER_SIZE** = QRS_NUM_SAMP + 1 , **LCD_BUFFER_CAPACITY** = DAQ_BUFFER_CAPACITY , **LCD_BUFFER_SIZE** = LCD_BUFFER_CAPACITY + 1 }
- enum { **LCD_TOP_LINE** = (LCD_Y_MAX - 48) , **LCD_WAVE_NUM_Y** = 128 , **LCD_WAVE_X_OFFSET** = 0 , **LCD_WAVE_Y_MIN** = (0 + LCD_WAVE_X_OFFSET) , **LCD_WAVE_Y_MAX** = (LCD_WAVE_NUM_Y + LCD_WAVE_X_OFFSET) }

Functions

- static void DAQ_Handler (void)

  Reads ADC output, converts to raw voltage sample, and sends to next FIFO.
- static void Processing_Handler (void)

  Removes noise from the signal and sends it to the QRS and LCD FIFO buffers.
- static void LCD_Handler (void)

  Applies a 0.5-40 [Hz] bandpass filter and plots the sample to the waveform.
- static void **LCD_plotNewSample** (uint16_t x, volatile const float32_t sample)
- int **main** (void)

Variables

- static volatile FIFO_t * **DAQ_Fifo** = 0
- static volatile uint32_t **DAQ_Buffer** [DAQ_BUFFER_SIZE] = { 0 }
- static volatile FIFO_t * **QRS_Fifo** = 0
- static volatile uint32_t **QRS_FifoBuffer** [QRS_BUFFER_SIZE] = { 0 }
- static volatile bool **QRS_bufferIsFull** = false
- static volatile FIFO_t * **LCD_Fifo** = 0
- static volatile uint32_t **LCD_FifoBuffer** [LCD_BUFFER_SIZE] = { 0 }
- static volatile float32_t **QRS_Buffer** [QRS_BUFFER_SIZE] = { 0 }

4.5.1 Detailed Description

4.5.2 Function Documentation

DAQ_Handler()

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

Reads ADC output, converts to raw voltage sample, and sends to next FIFO.

This ISR has a priority level of 1, is triggered when the ADC has finished capturing a sample, and also triggers the intermediate processing handler.

Precondition

Initialize the DAQ module.

Postcondition

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

See also

DAQ_Init(), Processing_Handler()

Processing_Handler()

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

Removes noise from the signal and sends it to the QRS and LCD FIFO buffers.

This ISR has a priority level of 1, is triggered by the DAQ ISR, and triggers the LCD Handler. It also notifies the superloop in main() that the QRS buffer is full.

Postcondition

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

See also

DAQ_Handler(), main(), LCD_Handler()

LCD_Handler()

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

Applies a 0.5-40 [Hz] bandpass filter and plots the sample to the waveform.

This ISR has a priority level of 1 and is triggered by the Processing ISR. This ISR also plots an intermediate sample to the display to make the waveform look more continuous.

Precondition

Initialize the LCD module.

Postcondition

The bandpass-filtered sample is plotted to the LCD.

See also

LCD_Init(), Processing_Handler()

5 Data Structure Documentation

5.1 FIFO_t Struct Reference

Data Fields

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

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

- FIFO.c

5.2 GPIO_Port_t Struct Reference

Data Fields

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

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

- GPIO.c

5.3 Led_t Struct Reference

Data Fields

- GPIO_Port_t * **GPIO_PORT_PTR**
  pointer to GPIO port data structure
- GPIO_Pin_t **GPIO_PIN**
  GPIO pin number.
- bool **is_ON**
  state indicator

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

- Led.c

5.4 Timer_t Struct Reference

Data Fields

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

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

- Timer.c

5.5 UART_t Struct Reference

Data Fields

- const uint32_t **BASE_ADDRESS**
- register_t const **FLAG_R_ADDRESS**
- GPIO_Port_t * **GPIO_PORT**
  pointer to GPIO port data structure
- GPIO_Pin_t **RX_PIN_NUM**
  GPIO pin number.
- GPIO_Pin_t **TX_PIN_NUM**
  GPIO pin number.
- bool **isInit**

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

- UART.c

6 File Documentation

6.1 DAQ.c File Reference

Source code for DAQ module.

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

Macros

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

Typedefs

- typedef arm_biquad_casd_df1_inst_f32 **Filter_t**

Enumerations

- enum {
  NUM_STAGES_NOTCH = 6 , **NUM_COEFFS_NOTCH** = NUM_STAGES_NOTCH ∗ 5 , **STATE_BUFF_SIZE_NOTCH** = NUM_STAGES_NOTCH ∗ 4 , **NUM_STAGES_BANDPASS** = 4 ,
  NUM_COEFFS_DAQ_BANDPASS = NUM_STAGES_BANDPASS ∗ 5 , **STATE_BUFF_SIZE_BANDPASS** = NUM_STAGES_BANDPASS ∗ 4 }

Functions

Initialization

- void DAQ_Init (void)
  Initialize the data acquisition (DAQ) module.

Reading Input Data

- uint16_t DAQ_readSample (void)
  Read a sample from the ADC.
- float32_t DAQ_convertToMilliVolts (uint16_t sample)
  Convert a 12-bit ADC sample to a floating-point voltage value via LUT.

Digital Filtering Functions

- float32_t DAQ_NotchFilter (volatile float32_t xn)
  Apply a 60 [Hz] notch filter to an input sample.
- float32_t DAQ_BandpassFilter (volatile float32_t xn)
  Apply a 0.5-40 [Hz] bandpass filter to an input sample.

Variables

- static const float32_t * **DAQ_LOOKUP_TABLE** = 0
- static const float32_t COEFFS_NOTCH [NUM_COEFFS_NOTCH]
- static const float32_t COEFFS_BANDPASS [NUM_COEFFS_DAQ_BANDPASS]
- static float32_t **stateBuffer_Notch** [STATE_BUFF_SIZE_NOTCH]
- static const Filter_t **notchFiltStruct** = { NUM_STAGES_NOTCH, stateBuffer_Notch, COEFFS_NOTCH }
- static const Filter_t *const **notchFilter** = &notchFiltStruct
- static float32_t **stateBuffer_Bandpass** [STATE_BUFF_SIZE_BANDPASS]
- static const Filter_t bandpassFiltStruct
- static const Filter_t *const **bandpassFilter** = &bandpassFiltStruct

6.1.1 Detailed Description

Source code for DAQ module.

Author

Bryan McElvy

6.2 DAQ.h File Reference

Application software for handling data acquision (DAQ) functions.

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

Functions

Initialization

- void DAQ_Init (void)
  Initialize the data acquisition (DAQ) module.

Reading Input Data

- uint16_t DAQ_readSample (void)
  Read a sample from the ADC.
- float32_t DAQ_convertToMilliVolts (uint16_t sample)
  Convert a 12-bit ADC sample to a floating-point voltage value via LUT.

Digital Filtering Functions

- float32_t DAQ_NotchFilter (volatile float32_t xn)
  Apply a 60 [Hz] notch filter to an input sample.
- float32_t DAQ_BandpassFilter (volatile float32_t xn)
  Apply a 0.5-40 [Hz] bandpass filter to an input sample.

6.2.1 Detailed Description

Application software for handling data acquision (DAQ) functions.

Author

Bryan McElvy

6.3 Debug.h File Reference

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

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

Functions

Initialization

- void Debug_Init (void)
  Initialize the Debug module.

Assertions

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

Serial Output

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

6.3.1 Detailed Description

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

Author

Bryan McElvy

6.4 LCD.c File Reference

Source code for LCD module.

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

Functions

- static void **LCD_updateNumPixels** (void)

  Updates `lcd`'s `numPixels` parameter after changing rows/columns.
- static void LCD_setDim (uint16_t d1, uint16_t d2, bool is_x, bool update_num_pixels)

  Set new `x` or `y` parameters, and optionally update `numPixels`.
- static void LCD_drawLine (uint16_t center, uint16_t lineWidth, bool is_horizontal)

  Helper function for drawing straight lines.

Init./Config. Functions

- void **LCD_Init** (void)

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

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

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

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

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

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

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

Drawing Area Definition Functions

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

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

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

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

Color Setting Functions

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

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

- void LCD_setColor_3bit (uint8_t color_code)

  Set the color value via a 3-bit code.

Drawing Functions

- void LCD_Draw (void)

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

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

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

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

  Draw a rectangle of size `dx` x `dy` onto the display. The bottom-left corner will be located at `(x1, y1)`.
- void LCD_graphSample (uint16_t x1, uint16_t dx, uint16_t y1, uint16_t dy, uint16_t y_min, uint16_t y_max, uint16_t color_code)

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

Variables

- struct {
  uint16_t **x1**
  starting x-value in range [0, x2]
  uint16_t **x2**
  ending x-value in range [0, NUM_ROWS)
  uint16_t **y1**
  starting y-value in range [0, y2]
  uint16_t **y2**
  ending x-value in range [0, NUM_COLS)
  uint32_t **numPixels**
  num. of pixels to write; = `(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 **isOutputOn**
  if `true`, LCD driver writes from its memory to display
  bool **isInverted**
  if `true`, the display's colors are inverted
  bool **using16bitColors**
  `true` for 16-bit color depth, `false` for 18-bit
  bool **isInit**
  if `true`, LCD has been initialized
  } **lcd**

6.4.1 Detailed Description

Source code for LCD module.

Author

Bryan McElvy

6.5 LCD.h File Reference

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

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

Enumerations

- enum { **LCD_X_MAX** = ILI9341_NUM_ROWS , **LCD_Y_MAX** = ILI9341_NUM_COLS }

Functions

Init./Config. Functions

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

Drawing Area Definition Functions

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

Drawing Functions

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

Color Setting Functions

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

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

  Set the color value via a 3-bit code.

6.5.1 Detailed Description

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

Author

Bryan McElvy

6.6 QRS.c File Reference

Source code for QRS module.

```
#include "QRS.h"
#include "arm_math_types.h"
#include "dsp/filtering_functions.h"
#include "dsp/statistics_functions.h"
#include <stdbool.h>
#include <stdint.h>
```

Macros

- #define **QRS_NUM_FID_MARKS** 20
- #define **FLOAT_COMPARE_TOLERANCE** (float32_t)(1E-5f)
- #define **IS_GREATER**(X, Y) (bool) ((X - Y) > FLOAT_COMPARE_TOLERANCE)
- #define **IS_LESSER**(X, Y) (bool) ((Y - X) > FLOAT_COMPARE_TOLERANCE)
- #define **IS_PEAK**(X_MINUS_1, X, X_PLUS_1) (bool) (IS_GREATER(X, X_MINUS_1) && IS_GREATER(X, X_PLUS_1))

Typedefs

- typedef arm_biquad_casd_df1_inst_f32 **IIR_Filt_t**
- typedef arm_fir_instance_f32 **FIR_Filt_t**

Enumerations

- enum {
  NUM_STAGES_BANDPASS = 4 , **NUM_COEFF_HIGHPASS** = NUM_STAGES_BANDPASS ∗ 5 , **STATE↩
  _BUFF_SIZE_BANDPASS** = NUM_STAGES_BANDPASS ∗ 4 , **NUM_COEFF_DERFILT** = 5 ,
  STATE_BUFF_SIZE_DERFILT = NUM_COEFF_DERFILT + QRS_NUM_SAMP - 1 , **NUM_COEFF_↩
  MOVAVG** = 10 , **STATE_BUFF_SIZE_MOVAVG** = NUM_COEFF_MOVAVG + QRS_NUM_SAMP - 1 }

Functions

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

  Mark local peaks in the input signal y as potential candidates for QRS complexes (AKA "fiducial marks").
- static void QRS_initLevels (const float32_t yn[ ])

  Initialize the signal and noise levels for the QRS detector using the initial block of input signal data.
- static float32_t QRS_updateLevel (float32_t peakAmplitude, float32_t level)

  Update signal or noise level based on a confirmed peak's amplitude.
- static float32_t QRS_updateThreshold (void)

  Update the amplitude threshold used to identify peaks based on the signal and noise levels.
- void **QRS_Init** (void)

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

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

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

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

Variables

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

- static const float32_t COEFF_BANDPASS [NUM_COEFF_HIGHPASS]
- static const float32_t **COEFF_DERFILT** [NUM_COEFF_DERFILT] = { -0.125f, -0.25f, 0.0f, 0.25f, 0.125f }
- static const float32_t COEFF_MOVAVG [NUM_COEFF_MOVAVG]
- static float32_t **stateBuffer_bandPass** [STATE_BUFF_SIZE_BANDPASS] = { 0 }
- static const IIR_Filt_t **bandpassFiltStruct** = { NUM_STAGES_BANDPASS, stateBuffer_bandPass, COEFF↩
  _BANDPASS }
- static const IIR_Filt_t ∗const **bandpassFilter** = &bandpassFiltStruct
- static float32_t **stateBuffer_DerFilt** [STATE_BUFF_SIZE_DERFILT] = { 0 }
- static const FIR_Filt_t **derivativeFiltStruct** = { NUM_COEFF_DERFILT, stateBuffer_DerFilt, COEFF_↩
  DERFILT }
- static const FIR_Filt_t ∗const **derivativeFilter** = &derivativeFiltStruct
- static float32_t **stateBuffer_MovingAvg** [STATE_BUFF_SIZE_MOVAVG] = { 0 }
- static const FIR_Filt_t **movingAvgFiltStruct** = { NUM_COEFF_MOVAVG, stateBuffer_MovingAvg, COEFF↩
  _MOVAVG }
- static const FIR_Filt_t ∗const **movingAverageFilter** = &movingAvgFiltStruct

6.6.1 Detailed Description

Source code for QRS module.

Author

Bryan McElvy

6.7 QRS.h File Reference

QRS detection algorithm functions.

```
#include "arm_math_types.h"
#include "dsp/filtering_functions.h"
#include "dsp/statistics_functions.h"
#include <stdbool.h>
#include <stdint.h>
```

Macros

- #define **QRS_SAMP_FREQ** ((uint32_t) 200)
- #define **QRS_SAMP_PERIOD_SEC** ((float32_t) 0.005f)
- #define **QRS_NUM_SAMP** ((uint16_t) (1200))

Functions

- void **QRS_Init** (void)

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

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

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

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

6.7.1 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.
```

6.8 FIFO.c File Reference

Source code for FIFO buffer module.

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

Data Structures

- struct FIFO_t

Functions

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

Basic Operations

- void FIFO_Put (volatile FIFO_t *fifo, const uint32_t val)
  Add a value to the end of the buffer.
- uint32_t FIFO_Get (volatile FIFO_t *fifo)
  Remove the first value of the buffer.
- void FIFO_TransferOne (volatile FIFO_t *srcFifo, volatile FIFO_t *destFifo)
  Transfer a value from one FIFO buffer to another.

Bulk Removal

- void FIFO_Flush (volatile FIFO_t *fifo, uint32_t outputBuffer[ ])
  Empty the FIFO buffer's contents into an array.
- void FIFO_Reset (volatile FIFO_t *fifo)
  Reset the FIFO buffer.
- void FIFO_TransferAll (volatile FIFO_t *srcFifo, volatile FIFO_t *destFifo)
  Transfer the contents of one FIFO buffer to another.

Peeking

- uint32_t FIFO_PeekOne (volatile FIFO_t *fifo)
  See the first element in the FIFO without removing it.
- void FIFO_PeekAll (volatile FIFO_t *fifo, uint32_t outputBuffer[ ])
  See the FIFO buffer's contents without removing them.

Status Checks

- bool FIFO_isFull (volatile FIFO_t *fifo)
  Check if the FIFO buffer is full.
- bool FIFO_isEmpty (volatile FIFO_t *fifo)
  Check if the FIFO buffer is empty.
- uint32_t FIFO_getCurrSize (volatile FIFO_t *fifo)
  Get the current size of the FIFO buffer.

Variables

- static FIFO_t **buffer_pool** [FIFO_POOL_SIZE] = { 0 }
  pre-allocated buffer pool
- static uint8_t **free_buffers** = FIFO_POOL_SIZE
  no. of remaining buffers

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

Macros

- #define **FIFO_POOL_SIZE** 5

Functions

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

Basic Operations

- void FIFO_Put (volatile FIFO_t *fifo, const uint32_t val)
  Add a value to the end of the buffer.
- uint32_t FIFO_Get (volatile FIFO_t *fifo)
  Remove the first value of the buffer.
- void FIFO_TransferOne (volatile FIFO_t *srcFifo, volatile FIFO_t *destFifo)
  Transfer a value from one FIFO buffer to another.

Bulk Removal

- void FIFO_Flush (volatile FIFO_t *fifo, uint32_t outputBuffer[ ])
  Empty the FIFO buffer's contents into an array.
- void FIFO_Reset (volatile FIFO_t *fifo)
  Reset the FIFO buffer.
- void FIFO_TransferAll (volatile FIFO_t *srcFifo, volatile FIFO_t *destFifo)
  Transfer the contents of one FIFO buffer to another.

Peeking

- uint32_t FIFO_PeekOne (volatile FIFO_t *fifo)
  See the first element in the FIFO without removing it.
- void FIFO_PeekAll (volatile FIFO_t *fifo, uint32_t outputBuffer[ ])
  See the FIFO buffer's contents without removing them.

Status Checks

- bool FIFO_isFull (volatile FIFO_t *fifo)
  Check if the FIFO buffer is full.
- bool FIFO_isEmpty (volatile FIFO_t *fifo)
  Check if the FIFO buffer is empty.
- uint32_t FIFO_getCurrSize (volatile FIFO_t *fifo)
  Get the current size of the FIFO buffer.

6.9.1 Detailed Description

FIFO buffer data structure.

Author

Bryan McElvy

6.10 ISR.c File Reference

Source code for interrupt vector handling module.

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

Macros

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

Typedefs

- typedef volatile uint32_t * **register_t**

Functions

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

  Disable all interrupts globally.
- void ISR_GlobalEnable (void)

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

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

  Add an ISR to the interrupt table.
- void ISR_setPriority (const uint8_t vectorNum, const uint8_t priority)

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

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

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

  Generate a software-generated interrupt (SGI).
- void ISR_clearPending (const uint8_t vectorNum)

  Clear an ISR's pending bit.

Variables

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

6.10.1 Detailed Description

Source code for interrupt vector handling module.

Author

Bryan McElvy

6.11 ISR.h File Reference

Module for configuring interrupt service routines (ISRs).

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

Typedefs

- typedef void(∗ **ISR_t**) (void)

  Type definition for function pointers representing ISRs.

Functions

- void ISR_GlobalDisable (void)

  Disable all interrupts globally.
- void ISR_GlobalEnable (void)

  Enable all interrupts globally.
- void ISR_InitNewTableInRam (void)

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

  Add an ISR to the interrupt table.
- void ISR_setPriority (const uint8_t vectorNum, const uint8_t priority)

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

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

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

  Generate a software-generated interrupt (SGI).
- void ISR_clearPending (const uint8_t vectorNum)

  Clear an ISR's pending bit.

6.11.1 Detailed Description

Module for configuring interrupt service routines (ISRs).

Author

Bryan McElvy

6.12 lookup.c File Reference

Source code for DAQ module's lookup table.

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

Functions

- const float32_t * Lookup_GetPtr (void)

  Return a pointer to the DAQ lookup table.

Variables

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

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

6.12.1 Detailed Description

Source code for DAQ module's lookup table.

Author

Bryan McElvy

6.13 lookup.h File Reference

Lookup table for DAQ module.

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

Macros

- #define **LOOKUP_DAQ_MAX** (float32_t) 5.5
- #define **LOOKUP_DAQ_MIN** (float32_t)(-5.5)

Functions

- const float32_t * Lookup_GetPtr (void)

  Return a pointer to the DAQ lookup table.

6.13.1 Detailed Description

Lookup table for DAQ module.

Author

Bryan McElvy

6.14 NewAssert.c File Reference

Source code for custom `assert` implementation.

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

Functions

- void Assert (bool condition)

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

6.14.1 Detailed Description

Source code for custom `assert` implementation.

Author

Bryan McElvy

6.15 NewAssert.h File Reference

Header file for custom `assert` implementation.

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

Functions

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

6.15.1 Detailed Description

Header file for custom `assert` implementation.

Author

Bryan McElvy

6.16 ADC.c File Reference

Source code for ADC module.

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

Functions

- void **ADC_Init** (void)
  Initialize ADC0 as a single-input analog-to-digital converter.
- void **ADC_InterruptEnable** (void)
  Enable the ADC interrupt.
- void **ADC_InterruptDisable** (void)
  Disable the ADC interrupt.
- void **ADC_InterruptAcknowledge** (void)
  Acknowledge the ADC interrupt, clearing the flag.

6.16.1 Detailed Description

Source code for ADC module.

Author

Bryan McElvy

6.17 ADC.h File Reference

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

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

Functions

- void **ADC_Init** (void)

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

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

  Disable the ADC interrupt.
- void **ADC_InterruptAcknowledge** (void)

  Acknowledge the ADC interrupt, clearing the flag.

6.17.1 Detailed Description

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

Author

Bryan McElvy

6.18 GPIO.c File Reference

Source code for GPIO module.

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

Data Structures

- struct GPIO_Port_t

Macros

- #define **GPIO_NUM_PORTS** 6

Typedefs

- typedef volatile uint32_t * **register_t**

Enumerations

- enum {
  GPIO_PORTA_BASE_ADDRESS = (uint32_t) 0x40004000 , **GPIO_PORTB_BASE_ADDRESS** = (uint32_t) 0x40005000 , **GPIO_PORTC_BASE_ADDRESS** = (uint32_t) 0x40006000 , **GPIO_PORTD_BASE_ADDRESS** = (uint32_t) 0x40007000 ,
  GPIO_PORTE_BASE_ADDRESS = (uint32_t) 0x40024000 , **GPIO_PORTF_BASE_ADDRESS** = (uint32_t) 0x40025000 }
- enum {
  GPIO_DATA_R_OFFSET = (uint32_t) 0x03FC , **GPIO_DIR_R_OFFSET** = (uint32_t) 0x0400 , **GPIO_IS_R_OFFSET** = (uint32_t) 0x0404 , **GPIO_IBE_R_OFFSET** = (uint32_t) 0x0408 ,
  GPIO_IEV_R_OFFSET = (uint32_t) 0x040C , **GPIO_IM_R_OFFSET** = (uint32_t) 0x0410 , **GPIO_ICR_R_OFFSET** = (uint32_t) 0x041C , **GPIO_AFSEL_R_OFFSET** = (uint32_t) 0x0420 ,
  GPIO_DR2R_R_OFFSET = (uint32_t) 0x0500 , **GPIO_DR4R_R_OFFSET** = (uint32_t) 0x0504 , **GPIO_DR8R_R_OFFSET** = (uint32_t) 0x0508 , **GPIO_PUR_R_OFFSET** = (uint32_t) 0x0510 ,
  GPIO_PDR_R_OFFSET = (uint32_t) 0x0518 , **GPIO_DEN_R_OFFSET** = (uint32_t) 0x051C , **GPIO_LOCK_R_OFFSET** = (uint32_t) 0x0520 , **GPIO_COMMIT_R_OFFSET** = (uint32_t) 0x0524 ,
  GPIO_AMSEL_R_OFFSET = (uint32_t) 0x0528 , **GPIO_PCTL_R_OFFSET** = (uint32_t) 0x052C }

Functions

- GPIO_Port_t * GPIO_InitPort (GPIO_PortName_t portName)

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

  Check if the GPIO port is initialized.
- uint32_t **GPIO_getBaseAddr** (GPIO_Port_t *gpioPort)
- void GPIO_ConfigDirOutput (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

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

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

  Activate the specified pins' internal pull-up resistors.
- void GPIO_ConfigPullDown (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

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

  Configure the specified pins' drive strength. Pins are initialized with 2[mA] drive strength, so this is only needed for a drive strength of 4[mA] or 8[mA].
- void GPIO_EnableDigital (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

Enable digital I/O for the specified pins.

- void GPIO_DisableDigital (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

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

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

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

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

  Configure interrupts for the selected port in the NVIC.
- uint8_t GPIO_ReadPins (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

  Read from the specified GPIO pin.
- void GPIO_WriteHigh (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

  Write a `1` to the specified GPIO pins.
- void GPIO_WriteLow (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

  Write a `0` to the specified GPIO pins.
- void GPIO_Toggle (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

  Toggle the specified GPIO pins.
- void GPIO_ConfigAltMode (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

  Activate the alternate mode for the specified pins.
- void GPIO_ConfigPortCtrl (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask, uint8_t fieldEncoding)

  Specify the alternate mode to use for the specified pins.
- void GPIO_ConfigAnalog (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

  Activate analog mode for the specified GPIO pins.

Variables

- static GPIO_Port_t GPIO_PTR_ARR [6]

6.18.1 Detailed Description

Source code for GPIO module.

Author

Bryan McElvy

6.18.2 Function Documentation

GPIO_InitPort()

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

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

Parameters

in	*portName*	Name of the chosen port.

Returns

GPIO_Port_t* Pointer to the GPIO port's `struct`.

GPIO_isPortInit()

```
bool GPIO_isPortInit (
            GPIO_Port_t * gpioPort )
```

Check if the GPIO port is initialized.

Parameters

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

GPIO_ConfigDirOutput()

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

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

Parameters

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

GPIO_ConfigDirInput()

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

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

Parameters

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

GPIO_ConfigPullUp()

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

Activate the specified pins' internal pull-up resistors.

Parameters

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

GPIO_ConfigPullDown()

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

Activate the specified pins' internal pull-down resistors.

Parameters

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

GPIO_ConfigDriveStrength()

```
void GPIO_ConfigDriveStrength (
            GPIO_Port_t * gpioPort,
            GPIO_Pin_t pinMask,
            uint8_t drive_mA )
```

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

Parameters

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

GPIO_EnableDigital()

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

Enable digital I/O for the specified pins.

Parameters

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

GPIO_DisableDigital()

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

Disable digital I/O for the specified pins.

Parameters

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

GPIO_ConfigInterrupts_Edge()

```
void GPIO_ConfigInterrupts_Edge (
            GPIO_Port_t * gpioPort,
            GPIO_Pin_t pinMask,
            bool risingEdge )
```

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

Parameters

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

GPIO_ConfigInterrupts_BothEdges()

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

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

Parameters

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

GPIO_ConfigInterrupts_LevelTrig()

```
void GPIO_ConfigInterrupts_LevelTrig (
            GPIO_Port_t * gpioPort,
            GPIO_Pin_t pinMask,
            bool highLevel )
```

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

Parameters

in	*gpioPort*	Pointer to the specified GPIO port.
in	*pinMask*	Bit mask corresponding to the intended pin(s).
in	*highLevel*	`true` for high level, `false` for low level

GPIO_ConfigNVIC()

```
void GPIO_ConfigNVIC (
            GPIO_Port_t * gpioPort,
            uint8_t priority )
```

Configure interrupts for the selected port in the NVIC.

Parameters

in	*gpioPort*	Pointer to the specified GPIO port.
in	*priority*	Priority number between 0 (highest) and 7 (lowest).

GPIO_ReadPins()

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

Read from the specified GPIO pin.

Parameters

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

GPIO_WriteHigh()

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

Write a `1` to the specified GPIO pins.

Parameters

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

GPIO_WriteLow()

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

Write a `0` to the specified GPIO pins.

Parameters

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

GPIO_Toggle()

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

Toggle the specified GPIO pins.

Parameters

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

GPIO_ConfigAltMode()

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

Activate the alternate mode for the specified pins.

Parameters

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

GPIO_ConfigPortCtrl()

```
void GPIO_ConfigPortCtrl (
            GPIO_Port_t * gpioPort,
            GPIO_Pin_t pinMask,
            uint8_t fieldEncoding )
```

Specify the alternate mode to use for the specified pins.

Parameters

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

GPIO_ConfigAnalog()

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

Activate analog mode for the specified GPIO pins.

Parameters

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

6.18.3 Variable Documentation

GPIO_PTR_ARR

```
GPIO_Port_t GPIO_PTR_ARR[6]  [static]
```

Initial value:

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

6.19 GPIO.h File Reference

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

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

Enumerations

- enum **GPIO_Pin_t** {
  GPIO_PIN0 = ((uint8_t) 1) , **GPIO_PIN1** = ((uint8_t) (1 << 1)) , **GPIO_PIN2** = ((uint8_t) (1 << 2)) , **GPIO_PIN3** = ((uint8_t) (1 << 3)) ,
  GPIO_PIN4 = ((uint8_t) (1 << 4)) , **GPIO_PIN5** = ((uint8_t) (1 << 5)) , **GPIO_PIN6** = ((uint8_t) (1 << 6)) ,
  GPIO_PIN7 = ((uint8_t) (1 << 7)) ,
  GPIO_ALL_PINS = ((uint8_t) (0xFF)) }
- enum {
  LED_RED = GPIO_PIN1 , **LED_GREEN** = GPIO_PIN3 , **LED_BLUE** = GPIO_PIN2 , **LED_YELLOW** = (LED_RED + LED_GREEN) ,
  LED_CYAN = (LED_BLUE + LED_GREEN) , **LED_PURPLE** = (LED_RED + LED_BLUE) , **LED_WHITE** = (LED_RED + LED_BLUE + LED_GREEN) }
- enum **GPIO_PortName_t** {
  A , **B** , **C** , **D** ,
  E , **F** }

Functions

- GPIO_Port_t * GPIO_InitPort (GPIO_PortName_t portName)

  Initialize a GPIO Port and return a pointer to its `struct`.
- uint32_t **GPIO_getBaseAddr** (GPIO_Port_t *gpioPort)
- bool GPIO_isPortInit (GPIO_Port_t *gpioPort)

  Check if the GPIO port is initialized.
- void GPIO_ConfigDirOutput (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

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

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

  Activate the specified pins' internal pull-up resistors.
- void GPIO_ConfigPullDown (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

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

  Configure the specified pins' drive strength. Pins are initialized with 2[mA] drive strength, so this is only needed for a drive strength of 4[mA] or 8[mA].
- void GPIO_EnableDigital (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

  Enable digital I/O for the specified pins.
- void GPIO_DisableDigital (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

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

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

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

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

  Configure interrupts for the selected port in the NVIC.
- uint8_t GPIO_ReadPins (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

  Read from the specified GPIO pin.
- void GPIO_WriteHigh (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

  Write a `1` to the specified GPIO pins.

- void GPIO_WriteLow (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

  Write a 0 to the specified GPIO pins.
- void GPIO_Toggle (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

  Toggle the specified GPIO pins.
- void GPIO_ConfigAltMode (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

  Activate the alternate mode for the specified pins.
- void GPIO_ConfigPortCtrl (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask, uint8_t fieldEncoding)

  Specify the alternate mode to use for the specified pins.
- void GPIO_ConfigAnalog (GPIO_Port_t *gpioPort, GPIO_Pin_t pinMask)

  Activate analog mode for the specified GPIO pins.

6.19.1 Detailed Description

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

Author

Bryan McElvy

6.19.2 Function Documentation

GPIO_InitPort()

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

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

Parameters

`in`	*portName*	Name of the chosen port.

Returns

GPIO_Port_t* Pointer to the GPIO port's `struct`.

GPIO_isPortInit()

```
bool GPIO_isPortInit (
            GPIO_Port_t * gpioPort )
```

Check if the GPIO port is initialized.

Parameters

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

GPIO_ConfigDirOutput()

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

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

Parameters

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

GPIO_ConfigDirInput()

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

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

Parameters

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

GPIO_ConfigPullUp()

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

Activate the specified pins' internal pull-up resistors.

Parameters

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

GPIO_ConfigPullDown()

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

Activate the specified pins' internal pull-down resistors.

Parameters

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

GPIO_ConfigDriveStrength()

```
void GPIO_ConfigDriveStrength (
            GPIO_Port_t * gpioPort,
            GPIO_Pin_t pinMask,
            uint8_t drive_mA )
```

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

Parameters

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

GPIO_EnableDigital()

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

Enable digital I/O for the specified pins.

Parameters

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

GPIO_DisableDigital()

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

Disable digital I/O for the specified pins.

Parameters

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

GPIO_ConfigInterrupts_Edge()

```
void GPIO_ConfigInterrupts_Edge (
            GPIO_Port_t * gpioPort,
            GPIO_Pin_t pinMask,
            bool risingEdge )
```

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

Parameters

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

GPIO_ConfigInterrupts_BothEdges()

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

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

Parameters

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

GPIO_ConfigInterrupts_LevelTrig()

```
void GPIO_ConfigInterrupts_LevelTrig (
            GPIO_Port_t * gpioPort,
            GPIO_Pin_t pinMask,
            bool highLevel )
```

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

Parameters

in	*gpioPort*	Pointer to the specified GPIO port.
in	*pinMask*	Bit mask corresponding to the intended pin(s).
in	*highLevel*	`true` for high level, `false` for low level

GPIO_ConfigNVIC()

```
void GPIO_ConfigNVIC (
            GPIO_Port_t * gpioPort,
            uint8_t priority )
```

Configure interrupts for the selected port in the NVIC.

Parameters

in	*gpioPort*	Pointer to the specified GPIO port.
in	*priority*	Priority number between 0 (highest) and 7 (lowest).

GPIO_ReadPins()

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

Read from the specified GPIO pin.

Parameters

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

GPIO_WriteHigh()

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

Write a `1` to the specified GPIO pins.

Parameters

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

GPIO_WriteLow()

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

Write a `0` to the specified GPIO pins.

Parameters

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

GPIO_Toggle()

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

Toggle the specified GPIO pins.

Parameters

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

GPIO_ConfigAltMode()

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

Activate the alternate mode for the specified pins.

Parameters

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

GPIO_ConfigPortCtrl()

```
void GPIO_ConfigPortCtrl (
            GPIO_Port_t * gpioPort,
            GPIO_Pin_t pinMask,
            uint8_t fieldEncoding )
```

Specify the alternate mode to use for the specified pins.

Parameters

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

GPIO_ConfigAnalog()

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

Activate analog mode for the specified GPIO pins.

Parameters

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

6.20 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].

6.20.1 Detailed Description

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

Author

Bryan McElvy

6.21 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].

6.21.1 Detailed Description

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

Author

Bryan McElvy

6.22 SPI.c File Reference

Source code for SPI module.

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

Macros

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

Enumerations

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

Functions

- void SPI_Init (void)

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

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

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

  Write 8-bit data to the peripheral.

6.22.1 Detailed Description

Source code for SPI module.

Author

Bryan McElvy

6.23 SPI.h File Reference

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

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

Functions

- void SPI_Init (void)

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

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

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

  Write 8-bit data to the peripheral.

6.23.1 Detailed Description

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

Author

Bryan McElvy

6.24 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.

6.24.1 Detailed Description

Implementation details for SysTick functions.

Author

Bryan McElvy

6.25 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.

6.25.1 Detailed Description

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

Author

Bryan McElvy

6.26 Timer.c File Reference

Source code for Timer module.

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

Data Structures

- struct Timer_t

Typedefs

- typedef volatile uint32_t * **register_t**

Enumerations

- enum {
  TIMER0_BASE = 0x40030000 , **TIMER1_BASE** = 0x40031000 , **TIMER2_BASE** = 0x40032000 , **TIMER3_BASE** = 0x40033000 ,
  TIMER4_BASE = 0x40034000 , **TIMER5_BASE** = 0x40035000 }
- enum **REGISTER_OFFSETS** {
  CONFIG = 0x00 , **MODE** = 0x04 , **CTRL** = 0x0C , **INT_MASK** = 0x18 ,
  INT_CLEAR = 0x24 , **INTERVAL** = 0x28 , **VALUE** = 0x054 }

Functions

- Timer_t **Timer_Init** (timerName_t timerName)
- timerName_t **Timer_getName** (Timer_t timer)
- void **Timer_setMode** (Timer_t timer, timerMode_t timerMode, bool isCountingUp)
- void **Timer_enableAdcTrigger** (Timer_t timer)
- void **Timer_disableAdcTrigger** (Timer_t timer)
- void **Timer_enableInterruptOnTimeout** (Timer_t timer, uint8_t priority)
- void **Timer_disableInterruptOnTimeout** (Timer_t timer)
- void **Timer_clearInterruptFlag** (Timer_t timer)
- void **Timer_setInterval_ms** (Timer_t timer, uint32_t time_ms)
- uint32_t **Timer_getCurrentValue** (Timer_t timer)
- void **Timer_Start** (Timer_t timer)
- void **Timer_Stop** (Timer_t timer)
- bool **Timer_isCounting** (Timer_t timer)
- void **Timer_Wait1ms** (Timer_t timer, uint32_t time_ms)

Variables

- static TimerStruct_t TIMER_POOL [6]

6.26.1 Detailed Description

Source code for Timer module.

Author

Bryan McElvy

6.27 Timer.h File Reference

Device driver for general-purpose timer modules.

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

Enumerations

- enum **timerName_t** {
  TIMER0 , **TIMER1** , **TIMER2** , **TIMER3** ,
  TIMER4 , **TIMER5** }
- enum **timerMode_t** { **ONESHOT** , **PERIODIC** }
- enum { **UP** = true , **DOWN** = false }

Functions

- Timer_t **Timer_Init** (timerName_t timerName)
- timerName_t **Timer_getName** (Timer_t timer)
- void **Timer_setMode** (Timer_t timer, timerMode_t timerMode, bool isCountingUp)
- void **Timer_enableAdcTrigger** (Timer_t timer)
- void **Timer_disableAdcTrigger** (Timer_t timer)
- void **Timer_enableInterruptOnTimeout** (Timer_t timer, uint8_t priority)
- void **Timer_disableInterruptOnTimeout** (Timer_t timer)
- void **Timer_clearInterruptFlag** (Timer_t timer)
- void **Timer_setInterval_ms** (Timer_t timer, uint32_t time_ms)
- uint32_t **Timer_getCurrentValue** (Timer_t timer)
- void **Timer_Start** (Timer_t timer)
- void **Timer_Stop** (Timer_t timer)
- bool **Timer_isCounting** (Timer_t timer)
- void **Timer_Wait1ms** (Timer_t timer, uint32_t time_ms)

6.27.1 Detailed Description

Device driver for general-purpose timer modules.

Author

Bryan McElvy

6.28 UART.c File Reference

Source code for UART module.

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

Data Structures

- struct UART_t

Macros

- #define **ASCII_CONVERSION** 0x30

Typedefs

- typedef volatile uint32_t * **register_t**

Enumerations

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

Functions

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

  Initialize the specified UART peripheral.
- unsigned char UART_ReadChar (UART_t *uart)

  Read a single ASCII character from the UART.
- void UART_WriteChar (UART_t *uart, unsigned char input_char)

  Write a single character to the UART.
- void UART_WriteStr (UART_t *uart, void *input_str)

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

  Write a 32-bit unsigned integer the UART.
- void UART_WriteFloat (UART_t *uart, double n, uint8_t num_decimals)

  Write a floating-point number the UART.

Variables

- static UART_t UART_ARR [8]

6.28.1 Detailed Description

Source code for UART module.

Author

Bryan McElvy

6.29 UART.h File Reference

Driver module for serial communication via UART0 and UART 1.

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

Enumerations

- enum **UART_Num_t** {
  UART0 , **UART1** , **UART2** , **UART3** ,
  UART4 , **UART5** , **UART6** , **UART7** }

Functions

- UART_t ∗ UART_Init (GPIO_Port_t ∗port, UART_Num_t uartNum)

  Initialize the specified UART peripheral.
- unsigned char UART_ReadChar (UART_t ∗uart)

  Read a single ASCII character from the UART.
- void UART_WriteChar (UART_t ∗uart, unsigned char input_char)

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

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

  Write a 32-bit unsigned integer the UART.
- void UART_WriteFloat (UART_t ∗uart, double n, uint8_t num_decimals)

  Write a floating-point number the UART.

6.29.1 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.
```

6.30 main.c File Reference

Main program file.

```
#include "DAQ.h"
#include "Debug.h"
#include "LCD.h"
#include "QRS.h"
#include "FIFO.h"
#include "ISR.h"
#include "PLL.h"
#include "arm_math_types.h"
#include "tm4c123gh6pm.h"
#include <math.h>
#include <stdbool.h>
#include <stdint.h>
```

Enumerations

- enum { **DAQ_VECTOR_NUM** = INT_ADC0SS3 , **PROC_VECTOR_NUM** = INT_CAN0 , **LCD_VECTOR_NUM** = INT_TIMER1A }
- enum { **DAQ_BUFFER_CAPACITY** = 3 , **DAQ_BUFFER_SIZE** = DAQ_BUFFER_CAPACITY + 1 , **QRS_BUFFER_SIZE** = QRS_NUM_SAMP + 1 , **LCD_BUFFER_CAPACITY** = DAQ_BUFFER_CAPACITY , **LCD_BUFFER_SIZE** = LCD_BUFFER_CAPACITY + 1 }
- enum { **LCD_TOP_LINE** = (LCD_Y_MAX - 48) , **LCD_WAVE_NUM_Y** = 128 , **LCD_WAVE_X_OFFSET** = 0 , **LCD_WAVE_Y_MIN** = (0 + LCD_WAVE_X_OFFSET) , **LCD_WAVE_Y_MAX** = (LCD_WAVE_NUM_Y + LCD_WAVE_X_OFFSET) }

Functions

- static void DAQ_Handler (void)

  Reads ADC output, converts to raw voltage sample, and sends to next FIFO.
- static void Processing_Handler (void)

  Removes noise from the signal and sends it to the QRS and LCD FIFO buffers.
- static void LCD_Handler (void)

  Applies a 0.5-40 [Hz] bandpass filter and plots the sample to the waveform.
- static void **LCD_plotNewSample** (uint16_t x, volatile const float32_t sample)
- int **main** (void)

Variables

- static volatile FIFO_t * **DAQ_Fifo** = 0
- static volatile uint32_t **DAQ_Buffer** [DAQ_BUFFER_SIZE] = { 0 }
- static volatile FIFO_t * **QRS_Fifo** = 0
- static volatile uint32_t **QRS_FifoBuffer** [QRS_BUFFER_SIZE] = { 0 }
- static volatile bool **QRS_bufferIsFull** = false
- static volatile FIFO_t * **LCD_Fifo** = 0
- static volatile uint32_t **LCD_FifoBuffer** [LCD_BUFFER_SIZE] = { 0 }
- static volatile float32_t **QRS_Buffer** [QRS_BUFFER_SIZE] = { 0 }

6.30.1 Detailed Description

Main program file.

Author

Bryan McElvy

6.31 ILI9341.c File Reference

Source code for ILI9341 module.

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

Enumerations

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

Functions

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

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

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

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

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

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

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

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

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

- void ILI9341_setDispOutput (bool is_ON)

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

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

  Set the start row for vertical scrolling.
- void ILI9341_setMemAccessCtrl (bool areRowsFlipped, bool areColsFlipped, bool areRowsAndColsSwitched, bool isVertRefreshFlipped, bool isColorOrderFlipped, bool isHorRefreshFlipped)

  Set how data is converted from memory to display.
- void ILI9341_setColorDepth (bool is_16bit)

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

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

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

  TODO: Write brief.
- void ILI9341_setInterface (void)

  Sets the interface for the ILI9341. The parameters for this command are hard-coded, so it only needs to be called once upon initialization.
- void ILI9341_setRowAddress (uint16_t startRow, uint16_t endRow)

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

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

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

  Write a single pixel to frame memory.

Variables

- static uint32_t ILI9341_Buffer [8]
- static FIFO_t * **ILI9341_Fifo**

6.31.1 Detailed Description

Source code for ILI9341 module.

Author

Bryan McElvy

6.32 ILI9341.h File Reference

Driver module for interfacing with an ILI9341 LCD driver.

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

Enumerations

- enum { **ILI9341_NUM_COLS** = 240 , **ILI9341_NUM_ROWS** = 320 }

Functions

- void **ILI9341_Init** (Timer_t timer)

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

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

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

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

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

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

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

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

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

  Set the start row for vertical scrolling.
- void ILI9341_setMemAccessCtrl (bool areRowsFlipped, bool areColsFlipped, bool areRowsAndCols↩ Switched, bool isVertRefreshFlipped, bool isColorOrderFlipped, bool isHorRefreshFlipped)

  Set how data is converted from memory to display.
- void ILI9341_setColorDepth (bool is_16bit)

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

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

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

TODO: Write brief.

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

  TODO: Write.

- void ILI9341_setInterface (void)

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

- void ILI9341_setRowAddress (uint16_t startRow, uint16_t endRow)

  not using backlight, so these aren't necessary

- void ILI9341_setColAddress (uint16_t startCol, uint16_t endCol)

  Sets the start/end rows to be written to.

- void ILI9341_writeMemCmd (void)

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

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

  Write a single pixel to frame memory.

6.32.1 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.

6.33 Led.c File Reference

Source code for LED module.

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

Data Structures

- struct Led_t

Functions

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

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

  Get the GPIO port associated with the LED.
- GPIO_Pin_t Led_GetPin (Led_t *led)

  Get the GPIO pin associated with the LED.
- bool Led_isOn (Led_t *led)

  Check the LED's status.
- void Led_TurnOn (Led_t *led)

  Turn the LED `ON`.
- void Led_TurnOff (Led_t *led)

  Turn the LED `OFF`.
- void Led_Toggle (Led_t *led)

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

Variables

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

6.33.1 Detailed Description

Source code for LED module.

Author

Bryan McElvy

6.34 Led.h File Reference

Interface for LED module.

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

Macros

- #define **LED_POOL_SIZE** 3

Functions

- Led_t * Led_Init (GPIO_Port_t *gpioPort, GPIO_Pin_t pin)
  Initialize a light-emitting diode (LED) as an `Led_t`.
- GPIO_Port_t * Led_GetPort (Led_t *led)
  Get the GPIO port associated with the LED.
- GPIO_Pin_t Led_GetPin (Led_t *led)
  Get the GPIO pin associated with the LED.
- bool Led_isOn (Led_t *led)
  Check the LED's status.
- void Led_TurnOn (Led_t *led)
  Turn the LED `ON`.
- void Led_TurnOff (Led_t *led)
  Turn the LED `OFF`.
- void Led_Toggle (Led_t *led)
  Toggle the LED (i.e. `OFF` -> `ON` or `ON` -> `OFF`).

6.34.1 Detailed Description

Interface for LED module.

Author

Bryan McElvy

6.35 test_adc.c File Reference

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

```
#include "ADC.h"
#include "PLL.h"
#include "GPIO.h"
#include "Timer.h"
#include "FIFO.h"
#include "tm4c123gh6pm.h"
#include <stdint.h>
```

Macros

- #define **LED_PINS** (GPIO_Pin_t)(GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3)
- #define **SAMPLING_PERIOD_MS** (uint32_t) 5
- #define **NUM_SAMPLES** (uint32_t) 1000

Functions

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

Variables

- volatile bool **buffer_is_full** = false
- volatile FIFO_t * **fifo_ptr** = 0
- volatile uint32_t **fifo_buffer** [NUM_SAMPLES]

6.35.1 Detailed Description

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

Author

Bryan McElvy

6.36 test_daq.c File Reference

Test script for the data acquisition (DAQ) module.

```
#include "DAQ.h"
#include "Debug.h"
#include "LCD.h"
#include "ADC.h"
#include "PLL.h"
#include "FIFO.h"
#include "ISR.h"
#include "lookup.h"
#include "arm_math_types.h"
#include <stdbool.h>
#include <stdint.h>
```

Macros

- #define **DAQ_BUFFER_SIZE** 128
- #define **LCD_TOP_LINE** (LCD_Y_MAX - 48)
- #define **LCD_NUM_Y_VALS** 128
- #define **LCD_X_AXIS_OFFSET** 32
- #define **LCD_Y_MIN** (0 + LCD_X_AXIS_OFFSET)
- #define **LCD_Y_MAX** (LCD_NUM_Y_VALS + LCD_X_AXIS_OFFSET)

Functions

- void **LCD_plotNewSample** (uint16_t x, volatile const float32_t sample)
- int **main** (void)
- void **ADC0_SS3_Handler** (void)

Variables

- volatile FIFO_t * **inputFifo** = 0
- volatile uint32_t **inputBuffer** [DAQ_BUFFER_SIZE] = { 0 }
- volatile bool **sampleReady** = false

6.36.1 Detailed Description

Test script for the data acquisition (DAQ) module.

Author

Bryan McElvy

6.37 test_debug.c File Reference

Test script for Debug module.

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

Functions

- int **main** (void)

6.37.1 Detailed Description

Test script for Debug module.

Author

Bryan McElvy

6.38 test_fifo.c File Reference

Test script for FIFO buffer.

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

Macros

- #define **FIFO_LEN** 10
- #define **LED_PINS** (GPIO_Pin_t)(GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3)

Functions

- void **FIFO_reportStatus** (FIFO_t *fifo_ptr)
- int **main** (void)

Variables

- UART_t * **uart**

6.38.1 Detailed Description

Test script for FIFO buffer.

Author

Bryan McElvy

6.39 test_lcd_image.c File Reference

Test script for writing images onto the display.

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

Macros

- #define **X_OFFSET** (uint16_t) 0
- #define **SIZE** (uint16_t) 4
- #define **LED_PINS** (GPIO_Pin_t)(GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3)

Functions

- int **main** (void)

Variables

- const uint8_t **COLOR_ARR** [6] = { LCD_RED, LCD_YELLOW, LCD_GREEN, LCD_CYAN, LCD_BLUE, LCD_PURPLE }
- uint8_t **color_idx**

6.39.1 Detailed Description

Test script for writing images onto the display.

Author

Bryan McElvy

6.40 test_lcd_scroll.c File Reference

Test script for writing different colors on the LCD.

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

Macros

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

Functions

- int **main** (void)

6.40.1 Detailed Description

Test script for writing different colors on the LCD.

Author

Bryan McElvy

6.41 test_pll.c File Reference

Test script for the PLL module.

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

Macros

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

Functions

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

6.41.1 Detailed Description

Test script for the PLL module.

Author

Bryan McElvy

6.42 test_qrs.c File Reference

QRS detector test script.

```
#include "DAQ.h"
#include "Debug.h"
#include "QRS.h"
#include "PLL.h"
#include "FIFO.h"
#include "ISR.h"
#include "arm_math_types.h"
#include <math.h>
#include <stdbool.h>
#include <stdint.h>
```

Enumerations

- enum { **ADC_VECTOR_NUM** = INT_ADC0SS3 , **DAQ_VECTOR_NUM** = INT_CAN0 }
- enum { **DAQ_BUFFER_CAPACITY** = 8 , **DAQ_BUFFER_SIZE** = DAQ_BUFFER_CAPACITY + 1 , **QRS_↩ BUFFER_SIZE** = QRS_NUM_SAMP + 1 }

Functions

- static void **ADC_Handler** (void)
- static void **DAQ_Handler** (void)
- int **main** (void)

Variables

- static volatile FIFO_t * **DAQ_Fifo** = 0
- static volatile uint32_t **DAQ_Buffer** [DAQ_BUFFER_SIZE] = { 0 }
- static volatile FIFO_t * **QRS_Fifo** = 0
- static volatile uint32_t **QRS_FifoBuffer** [QRS_BUFFER_SIZE] = { 0 }
- static volatile bool **QRS_bufferIsFull** = false
- volatile float32_t **QRS_InputBuffer** [QRS_BUFFER_SIZE] = { 0 }
- volatile float32_t **QRS_OutputBuffer** [QRS_BUFFER_SIZE] = { 0 }

6.42.1 Detailed Description

QRS detector test script.

Author

Bryan McElvy

6.43 test_spi.c File Reference

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

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

Functions

- int **main** ()

6.43.1 Detailed Description

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

Author

Bryan McElvy

6.44 test_systick_int.c File Reference

Test script for SysTick interrupts.

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

Functions

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

Variables

- const uint8_t **color_table** [6] = { 0x02, 0x06, 0x04, 0x0C, 0x08, 0x0A }
- volatile uint8_t **color_idx** = 0
- volatile uint8_t **led_is_on** = 0

6.44.1 Detailed Description

Test script for SysTick interrupts.

Author

Bryan McElvy

6.45 test_timer1_int.c File Reference

Test script for relocating the vector table to RAM.

```
#include "GPIO.h"
#include "PLL.h"
#include "Timer.h"
#include "ISR.h"
#include "tm4c123gh6pm.h"
#include <stdbool.h>
#include <stdint.h>
```

Macros

- #define **LED_PINS** (GPIO_Pin_t)(GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3)

Functions

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

Variables

- GPIO_Port_t * **portF** = 0
- Timer_t **timer1** = 0
- bool **isLedOn** = false

6.45.1 Detailed Description

Test script for relocating the vector table to RAM.

Test script for Timer1A interrupts.

Author

Bryan McElvy

6.46 test_uart_interrupt.c File Reference

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

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

Functions

- int **main** (void)

Variables

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

6.46.1 Detailed Description

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

Author

Bryan McElvy

6.46.2 Variable Documentation

COLOR_LIST

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

Initial value:

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

COLOR_NAMES

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

Initial value:

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

6.47 test_uart_la.c File Reference

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

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

Functions

- int **main** (void)

6.47.1 Detailed Description

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

Author

Bryan McElvy

6.48 test_uart_write.c File Reference

Test script for writing to serial port via UART0.

```
#include "PLL.h"
#include "GPIO.h"
#include "Led.h"
#include "UART.h"
```

Functions

- int **main** (void)

Variables

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

6.48.1 Detailed Description

Test script for writing to serial port via UART0.

Author

Bryan McElvy

6.49 test_userctrl.c File Reference

Test file for GPIO/UserCtrl modules and GPIO interrupts.

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

Functions

- int **main** ()

6.49.1 Detailed Description

Test file for GPIO/UserCtrl modules and GPIO interrupts.

Author

Bryan McElvy

Index