Module Descriptions: Export | PACT

# Module Description

1. Module Description ID: MD-0020

**Description:**

This module controls timing.

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

**Low-Level Requirements:**

|  |  |
| --- | --- |
| CD284022-LLR-0130 | The software SHALL keep track of the number of ticks. |
| CD284022-LLR-0140 | The Software SHALL ensure the proper timing of operations. |

**File Names:**

|  |  |
| --- | --- |
| src/i2c.c |  |
| src/serial.c |  |
| src/ticks.c |  |

**Revision:**

**Draft Revision:**

1.0

**Revision Date:**

**Version:**

10

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0030

**Description:**

This module handles communications.

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

**Low-Level Requirements:**

**File Names:**

**Revision:**

**Draft Revision:**

1.0

**Revision Date:**

**Version:**

10

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0040

**Description:**

This module handles interrupts.

![](data:image/png;base64;base64,)

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/interrupts.c |  |

**Revision:**

**Draft Revision:**

0.2

**Revision Date:**

**Version:**

2

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0050

**Description:**

This module handles math routines.

![](data:image/png;base64;base64,)

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/math64.c |  |

**Revision:**

**Draft Revision:**

0.2

**Revision Date:**

**Version:**

2

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0060

**Description:**

This module contains hardware-level routines.

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/global.c |  |
| src/hardware.c |  |
| src/reflash.c |  |
| src/testinput.c |  |

**Revision:**

**Draft Revision:**

1.3

**Revision Date:**

**Version:**

13

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0070

**Description:**

This module contains error handling routines.

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/error\_handler.c |  |

**Revision:**

**Draft Revision:**

0.5

**Revision Date:**

**Version:**

5

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0080

**Description:**

This module contains support routines.

![](data:image/png;base64;base64,)

**Low-Level Requirements:**

**File Names:**

**Revision:**

**Draft Revision:**

0.2

**Revision Date:**

**Version:**

2

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0090

**Description:**

This module contains the Pitch Compensation Logic.

![](data:image/png;base64;base64,)

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/pitch.c |  |

**Revision:**

**Draft Revision:**

0.2

**Revision Date:**

**Version:**

2

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0095

**Description:**

This module contains This code for processing low fuel.

![](data:image/png;base64;base64,)

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/lowfuelalarm.c |  |

**Revision:**

**Draft Revision:**

0.3

**Revision Date:**

**Version:**

3

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0100

**Description:**

This module handles frequency input/output.

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/freqin.c |  |

**Revision:**

**Draft Revision:**

0.3

**Revision Date:**

**Version:**

3

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0110

**Description:**

This module handles logging.

![](data:image/png;base64;base64,)

**Low-Level Requirements:**

**File Names:**

**Revision:**

**Draft Revision:**

0.3

**Revision Date:**

**Version:**

3

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0120

**Description:**

This module controls indexes.

![](data:image/png;base64;base64,)

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/indexes.c |  |

**Revision:**

**Draft Revision:**

0.3

**Revision Date:**

**Version:**

3

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0130

**Description:**

This module contains This code is for driving an external Capacitive Interface Module, which serially clocks in 8-bits of data that then drives 8 capacitors.  The module is basically a 8-bit Capacitor Box.

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/capacitive.c |  |

**Revision:**

**Draft Revision:**

0.3

**Revision Date:**

**Version:**

3

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0140

**Description:**

crc16.c

This module handles CRC routines.

CalcCrc routine is called as needed

![](data:image/png;base64;base64,)

**Inputs:**uint8\_t Char, uint16\_t Crc1 (current CRC16 value)

**Outputs:** Crc1

CalcCrcArray routine is called as needed

![](data:image/png;base64;base64,)

Inputs:    uint8\_t \*Buffer, int32\_t Length

Outputs:  Crc16

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/crc16.c |  |

**Revision:**

**Draft Revision:**

0.5

**Revision Date:**

**Version:**

5

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0141

**Description:**

This is the code that gets called when the processor first starts execution following a reset event. Only the absolutely necessary set is performed, after which the application supplied main() routine is called.

When power is applied to the Microcontroller the Initialization Mode takes control. This mode is in charge of initializing the system environment and performing the setup procedures to execute the main program. During this stage of the program, the microcontroller components and all the external peripherals required to accomplish the software requirements are initialized.

The software boot code performs the following:

Copy the data segment initializers from FLASH to SRAM

Set unitialized variables to zeros

Set unused memory to 0x12345678

Store interrupt vectors

**Inputs:**    None

**Outputs:**  None

**Low-Level Requirements:**

|  |  |
| --- | --- |
| CD284022-LLR-0010 | 1. LLR Description  Software shallhave boot code written in Assembly Language that performs the following:  * Move Initialized variable data from FLASH to SRAM * Sets uninitialized variables to zero * Set all unused memory to 0x12345678 * Set Stack Pointer to end of SRAM * Call main() * Interrupt Vectors will also be stored in the boot code |

**File Names:**

|  |  |
| --- | --- |
| startup/startup\_stm32.s |  |

**Revision:**

**Draft Revision:**

1.0

**Revision Date:**

**Version:**

10

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0142

**Description:**

### main.c is entry point to application and main program loop.

After performing basic processor initialization and setting up the C runtime environment, control is passed to the "main()" function contained in �main.c�.  
  
Interrupts are left disabled by the startup (assembly) code so that the initialization of the various peripheral interfaces can be performed using C code. Several of the peripherals utilize the interrupt subsystem and startup is cleaner if the peripherals are completely initialized prior to enabling interrupts.

main.c contains the following program loop:

**InitializeSystem**() � Initialization routine for the microcontrollers parameters.  Function is defined in *initsystem.c*.

LOOP{

**Runs every clock cycle**

ExecutionStart()

ProcessTickCounters()

ProcessI2C1()

ProcessI2C2()

ProcessMcp3421()

**Runs every millisecond**

ProcessRxSerial(&SerialUsart1)

ProcessRxSerial(&SerialUsart2)

ProcessReadAdc()

ProcessAccelerometer()

**Runs every 5 milliseconds**

ProcessAdcFilter()

ProcessCapacitive()

**Runs every 10 milliseconds**

ProcessMasterIndex()

ProcessSlaveIndexes()

CalculuateFrequencyInput()

ProcessAnalogReadings()

ProcessPwmOutput()

ProcessSignalStatus()

IWDG\_TriggerWatchdog()

ProcessClearEventLog()

ProcessEventMessageQueue()

**Runs every 20 milliseconds**

ProcessFrequencyOutputs()

ProcessMasterFrequencyOutput()

ProcessLowFuelAlarm()

ProcessSerialSlave()

ProcessPitch()

ProcessTestInput()

ProcessErrorCondition()

**Runs every second**

ProcessVerifyStack()

ExecutionAverage()

ExecutionEnd(EXECUTION\_MAIN\_LOOP)

}

![](data:image/png;base64;base64,)

**Inputs:**  None

**Outputs:** None

### 

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/main.c |  |

**Revision:**

**Draft Revision:**

1.1

**Revision Date:**

**Version:**

11

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0143

**Description:**

in

initsystem.c - System Initialization

This module contains code that initializes the hardware.

initsystem.c performs the following:

Initialize System

Initialize the ADC Channels

Pre-fetch FLASH

Enable clocks

Configure SystTick Module for 1ms interrupt rate

Enable the System Interrupts

Configure HSE, HSI14, PLL

Configure Flash Latency

Configure Watchdog for a 0.75 to 1.25 second window

Setup pointers to configuration structures stored at end of FLASH.

Enable TSEN, VBATEN and VREFEN

Initialize STM32 GPIO

Initialize STEM32 I2C1

Initialize STEM32 I2C2

Initialize DMA

Set ADC GPIO Configuration

Verify Configuration PDI image has correct CRC16

Initialize Instance Number of SerialUsartx data structure

Initialize UART1 and UART2

Verify Configuration Data

Set SysVars.Version to FIRMWARE\_VERSION

InitializeDigitalIO - Configure DIG\_IO1 as SERIAL\_INOUT, all other DIG I/O as Digital Inputs (disabled)

InitializeEventLog

InitializeAccel

InitializeExecutionTimers

InitializePWM

STM32\_Delay

InitializeMcp3421

LogEvent

![](data:image/png;base64;base64,)

### **Inputs:**    None

### **Outputs**: None

**Low-Level Requirements:**

|  |  |
| --- | --- |
| CD284022-LLR-0020 | The software shall check to see that the factory configuration is valid. |
| CD284022-LLR-0030 | The software shall check to see that an accelerator is present. |
| CD284022-LLR-0040 | The software shall set the accelerometor scaling.   1. Send '0' to CTRL\_REG1 (0x2A) register of Accelerometer. 2. Send Accelerometer Scaling Value to XYZ\_DATA\_CFG (0x0E) register of Accelerometer. 3. Send '1' to CTRL\_REG1 (0x2A) of Accelerometer. |
| CD284022-LLR-0050 | Before reading the accelerometer's X, Y, and Z registers, the software shall check that:   * The I2C bus is idle (not being used by EEPROM) * The Factory Configuration is valid * The Accelerometer is flagged as being present * There are no messages waiting to be written to EEPROM |
| CD284022-LLR-0060 | When reading the data from the accelerometer the software shall convert data so 0 Counts == -Full Scale, and 4095 counts == +Full Scale. |
| CD284022-LLR-0070 | Upon initialization of the ADC, the software shall set the priority and subpriority of the ADC DMA Channel 1 to 0. |
| CD284022-LLR-0080 | Upon initialization of the ADC, the software shall enable I/O Requests on DMA Channel 1. |
| CD284022-LLR-0090 | Upon Initialization of the ADC, the software shall assign Processor A Pin 1 to the ADC IN 0 alternate. |
| CD284022-LLR-0100 | Upon Initialization of the ADC, the software SHALL assign Processor A Pin 5 to the ADC IN 5 alternate. |
| CD284022-LLR-0110 | Upon Initialization of the ADC, the software SHALL assign Processor A Pin 6 to the ADC IN 6 alternate. |
| CD284022-LLR-0120 | Upon Initialization of the ADC, the software SHALL assign Processor A Pin 7 to the ADC IN 7 alternate. |

**File Names:**

|  |  |
| --- | --- |
| src/initsystem.c |  |

**Revision:**

**Draft Revision:**

0.8

**Revision Date:**

**Version:**

8

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0144

**Description:**

### ticks.c � Tick Counters

This module contains code that updates the tick counters enabled through STM32 built-in timer.  STM32\_GetTick increments the 1ms, 5ms, 10ms, 20ms and 1sec tick counters, used by main.c to schedule the control loop.

![](data:image/png;base64;base64,)

**Inputs**:     None

**Outputs**:  None

**Low-Level Requirements:**

|  |  |
| --- | --- |
| CD284022-LLR-0130 | The software SHALL keep track of the number of ticks. |

**File Names:**

|  |  |
| --- | --- |
| src/ticks.c |  |

**Revision:**

**Draft Revision:**

0.4

**Revision Date:**

**Version:**

4

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0145

**Description:**

### i2c.c � This module contains I2C routines

There are 2 routines in i2c.c and both are called from Main.c. One routine is ProcessI2C1 and the other is ProcessI2C2.

IC21 is connected to the MCP3421 chip (A/D chip). This routine is used to read and write from the MCP33421 chip. This module controls the I2C bus and is called from the main loop. At 400KHz, a character is sent/received every 44.4KHz (400/9) or every 22.5uS. Currently, Main Loop runs at 40KHz, or 25uS.

The following actions are performed:

Execution Start

 I2C\_MODE\_IDLE

     Stay here if idle

 I2C\_MODE\_READ

 Wait for busy bit to be reset, but if >50ms, reset I2C Interface and declare problem

 Once reset then do READ

 I2C\_MODE\_WRITE

      Wait for busy bit to be reset, but if >50ms, reset I2C Interface and declare problem

         Once reset then do READ

 Execution End

![](data:image/png;base64;base64,)

**Inputs**:    None

**Outputs**: None

I2C2 routine is called from Main Loop and during initialization of EventLog and Accelerometer. This routine is used to read and write from the EEPROM and Accelerometer.

The following actions are performed:

Execution Start

State Machine of I2C

I2C\_MODE\_IDLE

    Stay here if idle

I2C\_MODE\_READ\_ACCEL

I2C\_MODE\_WRITE\_EEPROM

I2C\_MODE\_READ\_EEPROM

I2C\_MODE\_WRITE\_ACCEL

Execution End

![](data:image/png;base64;base64,)

**Inputs**:    None

**Outputs**:  None

**Low-Level Requirements:**

|  |  |
| --- | --- |
| CD284022-LLR-0140 | The Software SHALL ensure the proper timing of operations. |

**File Names:**

|  |  |
| --- | --- |
| src/i2c.c |  |

**Revision:**

**Draft Revision:**

0.6

**Revision Date:**

**Version:**

6

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0146

**Description:**

### mcp3421.c � Interface with MCP3421 A/D chip

This module is called by main loop. When I2C, which is connected to MCP3421 A/D chip, is idle, place the I2C back into read mode.

Mc3421.c contains InitializeMcp3421 and ProcessMcp3421 routines.

InitializeMcp3421 routine is used to initialize Mcp3421 chip.

![](data:image/png;base64;base64,)

Inputs:   None

Outputs: None

ProcessMcp3421 routine contains a state machine which performs the following functions:

State Machine (State)

            State = 0

            STM32\_GPIO\_WritePin

                        Initialization of A/D chip, Sample rate 60 SPS, Gain = 1

                        I2C\_MODE\_WRITE\_MCP3421

State = 1

Wait for Write to finish, and set up for READ Mode

Read 3 bytes

I2C\_MODE\_READ\_MCP3421

            State  = 2

                        Check if A/D Conversion has finished

                        If not finished go to State 1 and read again

                        Store in Sysvars.RVOutputVoltageCounts

            State = 3, 4, and 5

                        Read RV Output Current

                        I2C\_MODE\_READ\_MCP3421

![](data:image/png;base64;base64,)

**Inputs:**  None

**Outputs:** None

**Low-Level Requirements:**

|  |  |
| --- | --- |
| CD284022-LLR-0150 | The CD Series Liquid Level Sender **SHALL** be able to interface with an aircraft instrument system via any of the following interfaces:  � Six (6) Configurable Digital I/O signals � Resistance/Voltage (R/V) Analog Output � RS485 Serial � CAN Bus |

**File Names:**

|  |  |
| --- | --- |
| src/mcp3421.c |  |

**Revision:**

**Draft Revision:**

0.6

**Revision Date:**

**Version:**

6

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0147

**Description:**

### serial.c � Communicate with UART

Code to handle serial communications.

ProcessSerialData is called for each byte received. There should never be a 100ms gap between characters coming from host. If there is then reset the State Machine back to looking for sync characters.

![](data:image/png;base64;base64,)

**Inputs:**   stSerial \*SerialStruct

**Outputs:** None

ProcessRxSerial is called every 1ms from Main Loop. The following actions are performed in this routine:

- Prevent Timer from continuing to count then roll-over back to zero

- If DIG\_Mode is still in Powerup Mode and USART2 hasn�t received a command for 3 seconds, then switch to Normal Mode

- RS485 Message Output, but only if no RS485 commands have been received for 2 seconds.

- Process any Serial Input that has been received.

![](data:image/png;base64;base64,)

**Inputs:**   stSerial \*SerialStruct

**Outputs:** None

SerialSend is routine is called to send a serial message to Host. The following actions are performed by this routine:

- Calculate CRC

- Send Message via selected SerialMode Mode

- If InstanceNumber = 0, Turn on RS422/485 Driver

- If InstanceNumber = 1, Switch Bi-Directional Buffer to an Output

   - Set DIG\_IO1 buffer as output

- As data is transmitted, it will be received also. A flag is used to prevent the received data from being placed into the Receiver Buffer.

![](data:image/png;base64;base64,)

**Inputs:**     stSerial \*SerialStruct

**Outputs:** None

VerifyFlashData routine is used to verify all parameter data items are within range. This routine is called when the WriteFlashConfiguration data serial command is received.

The following actions are performed in this routine:

 - Verify all FilteringGains are within range of 1 to 16384

 - LowFuelLevelActiveState must be 0, 1 or 2

 - Verify Trip Point is within the range of 400 to 102300 (4.00 to 1023.00), but only if LowFuel isn�t disabled

 - Verify Mode is within range of 0 to 2

 - Verify DIGI01 0 to 7, or 10 to 12

 - Verify DIGI02 0 to 6, 9 to 12, or 14

 - Verify DIGI03 0 to 6, or 10 to 12

 - Verify DIGI04 0 to 6, or 10 to 12

 - Verify DIGI05 is 0, 2, 4, 6, 8, 10, 11, 12, 13

 - Verify DIGI06 0 to 6, or 10 to 12

- If DIGI01 is configured as SERIAL\_IOUT, then DIGI02 cannot be SERIAL\_SLAVE\_OUT and DIGI05 can not be configured as SERIAL\_SLAVE\_IN

- If DIGI02 is SERIAL\_SLAVE\_OUT, then DIGI05 must be SERIAL\_SLAVE\_IN

- If DIGI05 is SERIAL\_SLAVE\_IN, then DIGI02 must be SERIAL\_SLAVE\_OUT

- DIGI02 and DIGI06 can not both be configured as SLAVE\_FREQ\_INPUTS

- If DIG2/DIG5 are configured as Serial Slave/Master, then no other DIG can be assigned as Slave Freq In (There is no need to check DIGIO5, as error 95 would be returned if DIGIO5 isn't configured as SERIAL\_SLAVE\_IN)

- If RVOutput is configured in Resistance Mode, then RV Pullup is is disabled

- If MasterFreqMode is configured for FreqTableMode, then verify MasterFreqTable is correct

    - There must be at least 2 entries in table, and index of First Entry must be 4.00 and last Entry must be 1023.00. Indexes must be increasing.

    - Frequencies must be >= 10 Hz and <= 50KHz

- If RVOutput is enabled (voltage or resistance), then verify SetpointTable

    -There must be at least 2 entries in table, and index of First Entry must be 4.00 and last Entry must be 1023.00. Indexes must be increasing.

- If RV Output Mode is Voltage, then Setpoint must be from 0 to 9.999 volts

- If RV Output Mode is Resistance, then Setpoint must be from 1.000 to 999.999 ohms

- Only one DIGIO can be configured as a TEST\_IN

- Only one DIGIO can be configured as a KINGAIR\_CLOCK

- Only one DIGIO can be configured as a KINGAIR\_DATA

- If Low Fuel is Disabled, then DIGIO's can not be assigned to LOW\_LEVEL\_ALARM

- There must be the same number of KingAirClock assigned as KingAirData. In this case, either both must be 0 or both must be 1.

-Accelerometer scaling must be 0, 1 or 2

![](data:image/png;base64;base64,)

**Inputs:**    stFlashConfiguration \*NewData

**Outputs:**  int32\_t

VerifyFactoryData routine is used to verify all Factory Parameter Data Items are within range. This routine is called when Write FactoryConfiguration Data serial command is received.

- Verify Calibration33

- Verify PowerInputFilteringGain

- Verify PowerRail5FilteringGain

- Verify TemperatureFilteringGain

- Verify InternalReferenceFilteringGain

- Verify RVOutputVoltageFilteringGain

- Verify RVOutputCurrentFilteringGain

- Verify KmaKFactor

- Verify PowerInputKFactor

- Verify PowerRail5KFactor

- Verify PwmOutputKFactor

- Verify RVOutputCurrentKFactor

- Verify RVOutputVoltageKFactor

![](data:image/png;base64;base64,)

Inputs:     stFactoryConfiguration \*NewData

Outputs:  int32\_t

VerifyPitchData routine is used to verify Pitch Compensation.

The following actions are performed in this routine:

- Only check remaining Parameter Data if ControlAxis is 1, 2, or 3

- AxisOffset must be from -1.0000 to +1.0000

- AxisCalibration TableLength must be >=2 and <=32

- Point1 (g's) of AxisCalibration Table must increase

- Point2 (angle) of AxisCalibration Table must be >= -90.00 and <= +90.00 degs

- Index Table Length must be >= 2 and <= 16

- First Index value must be >= 4.00 and last index value must be <= 1023.00, and index values must increase

- Angle Table Length must be >=2 and <= 16

- Angle values must be >= -90.00 and <= 90.00

![](data:image/png;base64;base64,)

Inputs:     stPitchCompensation \*NewData

Outputs:  int32\_t

ProcessSerialMessage routine is called when a valid serial message has been received.

The following actions are performed in this routine:

- If a message is received, then reset RS485Timer to prevent messages from being sent

- If a message is received, then reset RXMessageTimeout. This will prevent DIGIO1 from switching to NormalMode

- Request Slave Information Data if configured as SERIAL\_MASTER\_IN

- Save Slave Information Data if configured as SERIAL\_MASTER\_IN

- If Digital I/O #5 is configured for SERIAL\_SLAVE\_IN, thenadd to the message then retransmit message

- Save Slave Information

- Move updated Slave Information back into Serial Buffer for transmission

- Write FlashConfiguration Data

- Write FactoryConfiguration Data

- Write PitchCompensation Data

- Clear Event Log

- Read FlashConfiguration Data

- Read FactoryConfiguration Data

- Read PitchCompensation Data

- Read Event Log Partition 32k-128 blocks of 256 bytes each

- Read Signal Status Information

- Read System Variables

- Read Execution Timing Report

- Read CMark Array (Serial Data Buffer is 530 bytes, allowing up to 4240 Bit Marks)

- Return Firmware CRC Information

- Flash new Firmware

![](data:image/png;base64;base64,)

**Inputs:** stSerial \*SerialStruct

**Outputs:**None

ProcessSerialSlave routine is called from Main Loop � 50 times per second (average).

The following  are performed:

- If Digital I/O #2 is configured for Serial Master Out, then send Request Slave information command every 300ms

- As UART2\_TX is shared by Digital I/O #1 and #2, Don�t output any Serial Slave message when Digital I/O #1 is in SERIAL\_INOUT mode

-Set Area where Slave Information would be stored in Serial Buffer to zero

![](data:image/png;base64;base64,)

**Inputs:**None

**Outputs:** None

**Low-Level Requirements:**

|  |  |
| --- | --- |
| CD284022-LLR-0150 | The CD Series Liquid Level Sender **SHALL** be able to interface with an aircraft instrument system via any of the following interfaces:  � Six (6) Configurable Digital I/O signals � Resistance/Voltage (R/V) Analog Output � RS485 Serial � CAN Bus |

**File Names:**

|  |  |
| --- | --- |
| src/serial.c |  |

**Revision:**

**Draft Revision:**

1.1

**Revision Date:**

**Version:**

11

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0148

**Description:**

usart.c - UART Functions.

UART 1 is connected to RS485.

HAL\_UART\_TxCpltCallback routine is called on TX Complete by ISR and it performs the following:

-Turn off RS422/485 Driver

-Switch Digital I/o #1 Buffer back to an input to start receiving serial data

![](data:image/png;base64;base64,)

**Inputs:**   UART\_HandleTypeDef \*huart

**Outputs:** None

UART\_DMATransmitCplt routine disables DMA controller and then enables UART transmit complete interrupt.

![](data:image/png;base64;base64,)

Inputs:    DMA\_HandleTypeDef \* hdma

Outputs: None

STM32\_UART\_Transmit\_DMA routine initializes DMA controller to transfer data to UART. It checks if UART is ready for data, will get data from memory and transfer to UART. Once this is done CPU can do anything it needs, DMA controller does everything. No CPU time is used with UART. When DMA controller has transferred all data, then UART\_DMATransmitCplt gets called.

![](data:image/png;base64;base64,)

Inputs:    UART\_HandleTypeDef \*huart, uint8\_t \*pData, uint16\_t Size

Outputs: None

UART\_EndTransmit\_IT routine calls the HAL\_UART\_TxCpltCallback at end of UART transmit.

![](data:image/png;base64;base64,)

Inputs:    UART\_HandleTypeDef \*huart

Outputs: None

STM32\_UART\_IRQHandler is the interrupt handler, so every time UART generates an interrupt this routine gets called. There are � 2 items cause interrupts � 1) everytime a char is received, 2)at end of transmission (transmission complete). The routine clear Errors, even if there aren't any.

Inputs:    UART\_HandleTypeDef \*huart

Outputs: None

![](data:image/png;base64;base64,)

UART\_Start\_Receive\_DMA routine performs the following:

- Enable the DMA channel

![](data:image/png;base64;base64,)

Inputs:    UART\_HandleTypeDef \*huart, uint8\_t \*pData, uint16\_t Size

Outputs: HAL\_OK

STM32\_UART\_Receive\_DMA routine performs the following:

- Check that USART RT0EN bit is set

- Enable the UART Receiver Timeout Interrupt

![](data:image/png;base64;base64,)

Inputs:    UART\_HandleTypeDef \*huart, uint8\_t \*pData, uint16\_t Size

Outputs: UART\_Start\_Receive\_DMA(huart, pData, Size)

MX\_USART1\_UART\_Init routine is  used for initializing the UART1. UART1 is for RS485 TX/RX.

![](data:image/png;base64;base64,)

Inputs:    None

Outputs: None

MX\_USART2\_UART\_Init routine

- UART2\_RX - Connected to Digital I/O #1

- UART2\_TX - Connected to Digital I/O #1 through 4.7K resistor and to Digital I/O #2 through 4.7K resistor.

- UART2\_RX is used when Digital I/O #1 is configured in SERIAL INOUT mode.  This allows commands to be received through Digital I/O #1.

- UART2\_TX is used for two purposes.  When Digital I/O #1 is configured as SERIAL\_INOUT, after receiving a command, the Output Buffer Driver is switched from input to output mode.  Then a response message is sent via UART2\_TX through the 4.7K resistor and out the output Buffer chip onto Digital I/O #1.  This allows Digital I/O #1 to function in half duplex mode.

-  UART2\_TX is also used to drive Digital I/O #2 when it's configured as SERIAL\_SLAVE\_OUT or SERIAL\_MASTER\_OUT.

![](data:image/png;base64;base64,)

Inputs:    None

Outputs: None

MX\_USART4\_UART\_Init routine is used to initialize UART. DIG I/O #5 is RX.

![](data:image/png;base64;base64,)

Inputs:    None

Outputs: None

**Low-Level Requirements:**

|  |  |
| --- | --- |
| CD284022-LLR-0150 | The CD Series Liquid Level Sender **SHALL** be able to interface with an aircraft instrument system via any of the following interfaces:  � Six (6) Configurable Digital I/O signals � Resistance/Voltage (R/V) Analog Output � RS485 Serial � CAN Bus |

**File Names:**

|  |  |
| --- | --- |
| src/usart.c |  |

**Revision:**

**Draft Revision:**

1.2

**Revision Date:**

**Version:**

12

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0149

**Description:**

adc.c - This module contains functions that configure ADC, Execute ADC Self-Calibration, and read A/D inputs.

Start\_DMA routine is called once. This routine starts the ADC in a DMA circular mode.

![](data:image/png;base64;base64,)

**Inputs:**ADC\_HandleTypeDef \*hadc, uint32\_t \*pData, uint32\_t Length

**Outputs:** None

ProcessReadAdc routine is called from Main Loop 1000 times per second (on average) and performs the following functions:

- Waits approx 0.6 seconds after power=up, then executes the ADC Self Calibration

- Waits approx  1.0 seconds after power-up, then starts ADC conversions in continuous circular DMA mode

- DMA stores ADC counts into AdcCounts array, this code then moves the data in the SysVar structure for use by other logic.

- For Testing, the actual ADC counts can be replaced/overwritten by data received from a Host via serial

- ADC counts are stored in SysVars

![](data:image/png;base64;base64,)

**Inputs:**    None

**Outputs:** None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/adc.c |  |

**Revision:**

**Draft Revision:**

0.7

**Revision Date:**

**Version:**

7

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0150

**Description:**

### accel.c � Read Accelerometer

This module contains functions that initialize the accelerometer and read the accelerometer.

InitializeAccel routine  is called once during power-up and is used to initialize the accelerometer chip to the correct full-scale based on the AccelerometerScaling PDI.

It will only initialize if FactoryConfiguration is valid, and Accel is Present.

NOTE:  THIS ROUTINE IS BLOCKING.  The routine does not exist until the I2C communications is finished.  As this is only called once during initialization, blocking is acceptable.

![](data:image/png;base64;base64,)

**Inputs:**   None

**Outputs:**  None

ProcessAccelerometer routine is called every 1ms(on average) and reads the Accelerometer X/Y/Z data registers.

The Accelerometer is only read if:

 - I2C bus is idle (not being used by EEPROM)

 - Factory Configuration is Valid

 - Accelerometer is flagged as being present

 - There are no messages waiting to be written to EEPROM

 - Accelerometer data is stored into Sysvar data structure

![](data:image/png;base64;base64,) 

**Inputs:**  None

**Outputs:** None

**Low-Level Requirements:**

|  |  |
| --- | --- |
| CD284022-LLR-0050 | Before reading the accelerometer's X, Y, and Z registers, the software shall check that:   * The I2C bus is idle (not being used by EEPROM) * The Factory Configuration is valid * The Accelerometer is flagged as being present * There are no messages waiting to be written to EEPROM |
| CD284022-LLR-0060 | When reading the data from the accelerometer the software shall convert data so 0 Counts == -Full Scale, and 4095 counts == +Full Scale. |

**File Names:**

|  |  |
| --- | --- |
| src/accel.c |  |

**Revision:**

**Draft Revision:**

0.6

**Revision Date:**

**Version:**

6

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0151

**Description:**

### capacitive.c � Virtual capacitor box.

This code is for driving an external Capacitive Interface Module (CIM), which serially clocks in 8-bits of data that drives a capacitor.  The module is basically a 8-bit Capacitor Box. The capacitor box simulates a Capacitive Fuel Sender.

The CIM is in sync with the output from the Fuel Sender. It is done through timing of the Clock and Data lines.

Initial Testing using King-Air Gauge:

    0 Lbs   =   28

   100 Lbs   =   46

   200 Lbs   =   62

   300 Lbs   =   75

   400 Lbs   =   88

   500 Lbs   =   99

   600 Lbs   =   111

   700 Lbs   =   123

   800 Lbs   =   132

   900 Lbs   =   143

  1000 Lbs   =   155

  1100 Lbs   =   169

  1200 Lbs   =   184

  1300 Lbs   =   201

  1400 Lbs   =   219

The Capacitor Box simulates a Capacitive Fuel Sender.

   The CIM as an initial capacitance of xxx pf

   Setting Bit 0 adds x pf

   Setting Bit 1 adds x pf

   Setting Bit 2 adds x pf

   Setting Bit 3 adds x pf

   Setting Bit 4 adds x pf

   Setting Bit 5 adds x pf

   Setting Bit 6 adds x pf

   Setting Bit 7 adds x pf

The following are the routines in capacitive.c.

SetClock routine sets capacitive clock

![](data:image/png;base64;base64,)

**Inputs:**    None

**Outputs:**  None

ClearClock resets the capacitive clock.

![](data:image/png;base64;base64,)

**Inputs:**  None

**Outputs:** None

SetData sets pin.

![](data:image/png;base64;base64,)

**Inputs:**   None

**Outputs:**  None

ClearData resets pin.

![](data:image/png;base64;base64,)

Inputs:    None

Outputs:  None

ProcessCapacitive? routine is called every 5ms(on average) to synchronize the Capacitive Interface Board with the Sender, we have to allow the Data Watchdog Timer on the Sync Board to time-out with the Clock input Low.

The following actions are performed in this routine:

Check if manual (maintenance) mode has been enabled

If enabled store ManualCapacitive value in SysVars

Otherwise do a table lookup, SetpointIndex to CapacitiveValue

Find the I/O�s assigned to the Clock and Data Interface Module

Toggle Data line. This will re-trigger the Data Watchdog on Sync Board by doing following sequence:

     -Allow the Data Watchdog to timeout in 1.6 seconds

     - After re-triggering the Data Watchdog, toggle the clock after 1.3 seconds, then wait another 700ms

     - During the 700ms, the Data Watchdog will timeout, causing a reset pulse of 140ms.

     - After the sequence, 8-bit counter will be reset to 0 and will be in sync with Sender

![](data:image/png;base64;base64,)

Inputs:    None

Outputs:  None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/capacitive.c |  |

**Revision:**

**Draft Revision:**

0.5

**Revision Date:**

**Version:**

5

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0152

**Description:**

### indexes.c � Index Calculation Logic

This module calculates indexes.

CalculateSlaveIndex

This routine is used to calculate the Slave Index based on frequency and the bReverse flag.

Index = 65536/Slave Frequency. Slave Frequency = 1636400 (16364.00) to 6406 (64.06), resulting in an index of 4.00 to 1023.00.

![](data:image/png;base64;base64,)

**Inputs:**  int32\_t Freq, int8\_t bReverse

**Outputs:**Index, 4.00 to 1023.00 or 1023.00 to 4.00

ProcessSlaveIndexes

This subroutine calculates the Index value for each Slave Input.

![](data:image/png;base64;base64,)

Inputs:    None

Outputs: None

ProcessResistanceOutputControl routine is called when the R/V Output is configured for resistance. This routine adjusts the R/V Output voltage to maintain the correct output resistance.

The following are performed in this routine:

- Calculate resistivesetpoint for ohms

- Measure output voltage and current, calculate output resistance

- Setpoint value

- PwmDutyCycleFineTune = PID loop

- Sanity checks

- End result of PI is duty cycle � logic limits it between 0-100

![](data:image/png;base64;base64,)

Inputs:    None

Outputs: None

ProcessVoltageOutputControl routine is called when the R/V Output is configured for voltage. It adjusts the R/V Output voltage to maintain the correct output voltage.

Hardware w/ PWM � pulse with modulation �

NewDutyCycle takes Setpoint value and calculates a dutycycle value

Dutycycle goes to network and amplifier  (finetune is in case of calibration error)

If setpoint is greater than feedback our output isn�t high enough, then increment or decrease finetunevalue

Dutycycle + finetuning and make sure it�s not less than 0 or greater than 100

Gets used in pwm.c (pwmDutyCycle goes from being variable to hw)

Feedback is actual measured voltage that is being output

![](data:image/png;base64;base64,)

Inputs:    None

Outputs: None

ProcessMasterIndex routine is used to calculate SetPointIndex. This subroutine performs the following:

- ToggleTimer � in case of error toggle output every 30 seconds

- Calculate SlaveCount based on number of slave inputs

- Calculate Master index which is function of KMA reading within the sender and also function Slave Inputs (masterIndex = KMAIndex + all slave indexes)

- Calculate PitchCompensatedMasterIndex

- PitchCompensation logic calculates PitchKFactor

- If the DIG\_IO assigned as a Test input is active, then set SetPointIndex to full scale

- Lookup Setpoint from table based on SetpointIndex

- Error Checking � if error detected set voltage to 0V (setpoint = 0) to alert pilot

- If resistive mode call ProcessResistanceOutputControl

- If voltage mode call ProcessVoltageOutputControl

![](data:image/png;base64;base64,)

Inputs:    None

Outputs: None

**Low-Level Requirements:**

|  |  |
| --- | --- |
| CD284022-LLR-0150 | The CD Series Liquid Level Sender **SHALL** be able to interface with an aircraft instrument system via any of the following interfaces:  � Six (6) Configurable Digital I/O signals � Resistance/Voltage (R/V) Analog Output � RS485 Serial � CAN Bus |

**File Names:**

|  |  |
| --- | --- |
| src/indexes.c |  |

**Revision:**

**Draft Revision:**

0.6

**Revision Date:**

**Version:**

6

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0153

**Description:**

### freqin.c � Capture of Frequency Input and Calculation of Frequency

Originally, the input frequency was determined by using a timer.  The timer was configured to divide the Input Frequency by 8, and then to provide an interrupt on each rising edge of input.

 In the interrupt routine, the Execution Timer (TIMER2) was read and difference (delta) from the last interrupt as accumulated.  When the accumulated value reached 200000 (or 0.5 seconds), the input frequency was calculated.

This logic worked well until a slow rising input was used.  As the Bi-Directional Buffers have no phthisis, the output from the buffers would oscillate, causing the input frequency to be incorrect.

To solve this problem, the logic was changed to cause an interrupt using the GPIO module, not the TIMER module.  The GPIO was configured to cause an interrupt on each rising edge of the input frequency.

While there are still multiple rising edges on a slow rising input, the program will only see the first one.  By the time the interrupt routine starts executing, the remaining edges will have passed.

The downside is the interrupt rate is now 8 times higher than when the Timer was being used.  If multiple high frequency inputs are being measured, at some point the interrupt rate will become too high and prevent the program from executing correctly.

HAL\_GPIO\_EXTI\_Callback interrupt callback routine is called on the rising edge of an externally enabled interrupt, which would be any DIG\_IO assigned as a Slave Freqency Input.

 If the signal has a slow rise time, then there is the possibly that there will be glitches (multiple edges on both rising and falling edge of input signal.  (wasn't able to find a bi-directional level-shifting buffer with hysteresis)

The multiple edges are handled as follows:

On the first rising edge (the real one), an interrupt is generated and the ISR is executed.  While the ISR is being executed, the glitches are coming into the port pin.  But, as the interrupts are still disabled, the glitches aren't seen by the CPU � So life is good.

The interrupts are setup for rising edge only, but, on the slow falling edge, the glitches will create a rising edge.  As the glitches will have passed by the time this code is executed, we can check if the port pin is high or low.  If low, then it was a glitch on the falling edge that created the interrupt, so exit this routine without any further processing of the edge.  If high, then continue processing the edge, as it is a real rising edge.

If interrupt was from a real leading edge, then pin should be high.  If pin is low, then interrupt was from a falling edge with glitches. By the time this logic is executed, any glitches on the input frequency should have passed by this time this code is executed.

![](data:image/png;base64;base64,)

**Inputs:**   uint16\_t GPIO\_Pin

**Outputs:**  None

CalculuateFrequencyInput routine is called every time through main loop. At minimum input frequency of 20Hz, rise-to-rise time will be 50ms. At maximum input frequency of 20Khz, rise-to-rise time will be 50uS.

CapturePeriodCounter, CaptureAccumCounter, and CaptureAccum are updated in HAL\_GPIO\_EXTI\_Callback function

The following actions are performed in this routine:

Execution Start

If DIG\_I05 is configured for Serial Slave Input, then copy Slave Frequency Information from Serial Message to Sysvars data structure

If DIG\_I01 is not configured for Serial Slave Input, then calculate Slave Frequency of DIG\_IO that is configured for DOIG\_FREQ\_IN

If DIG I/O isn�t configured as Frequency Input, or rise-to-rise > 500ms, then set CaptureFrequency to 0

A change in CapturePeriodCounter means Accum has reached 4000000, so time to calculate a new input frequency

![](data:image/png;base64;base64,)

**Inputs**:     None

**Outputs**:  None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/freqin.c |  |

**Revision:**

**Draft Revision:**

0.4

**Revision Date:**

**Version:**

4

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0154

**Description:**

analog.c - Filters A/D input

This module contains functions that filters A/D inputs, converts filtered A/D counts into voltage, converts Voltage into Engineering Units.

The digital filter equation is a discrete Infinite Impulse Response (IIR) Low-Pass type Filter:

      X = (X \* (1 - G)) + (Y \* G)                     Eq. 1

    Where:

      X  - Filtered Output

      G - Gain, 0 to 1

      Y = Input Value to be Filtered

   NOTE: In a real system, G would always be greater than 0.   And with G==1, no filtering is    done

  The above equation works very well when using floating point. But, as the STM32F072 doesn't include a Floating Point Co-processor, Floating Point is not allowed in this project.  Only integers are allowed. As integers are only allowed, having the Gain be from 0 to 1 would result in only 0 and 1 being allowed. And as stated above, Gain would never be set to 0, this leaves only 1, which, as stated above, would provide no filtering.  So, the first change to the above equation is changing the Gain from 0 to 1, to 0 to 16384

      X = (((X \* (16384 - G)) + (Y \* G)) / 16384          Eq. 2

  Now, a Gain of 8192 in Eq. 2 would result in the same filtering as a gain of 0.5 from Eq. 1

  But, there is still some work that needs to be done.  As all values are integers, the division will always round down preventing the filtered output from every reaching the input value.    Example:  Given X starts at 0, G=8000, and Y=100

      T1:  X = (( 0 \* (16384-8000)) + (100 \* 8000)) / 16384 = 48

      T2:  X = ((48 \* (16384-8000)) + (100 \* 8000)) / 16384 = 73

      T3:  X = ((73 \* (16384-8000)) + (100 \* 8000)) / 16384 = 86

      T4:  X = ((86 \* (16384-8000)) + (100 \* 8000)) / 16384 = 92

      T5:  X = ((92 \* (16384-8000)) + (100 \* 8000)) / 16384 = 95

      T6:  X = ((95 \* (16384-8000)) + (100 \* 8000)) / 16384 = 97

      T7:  X = ((97 \* (16384-8000)) + (100 \* 8000)) / 16384 = 98

      T8:  X = ((98 \* (16384-8000)) + (100 \* 8000)) / 16384 = 98

      T9:  X = ((98 \* (16384-8000)) + (100 \* 8000)) / 16384 = 98

 As can be seen, while the input value is 100, the output only goes to 98.  To solve this problem, the input value is multiplied by 100:

      X = (((X \* (16384 - G)) + ((Y \* 100) \* G)) / 16384

  This results in X (filter output) having a fixed-point value of xxx.xx and now the equation will return 9998, or a value of 99.98.  Closer to 100 (input value), but we can still make some improvements.  By multiplying the input value even more, the output value can be made to approach the input value even more.  But this time, we don't want to make the filtered output value any bigger, so we will divide by the same amount:

      X = (((X \* (16384 - G)) + ((Y \* 100 \* 8192) \* G)) / 16384

      return X / 8192

  This gets us to 9999 or 99.99, but still not 100.00. So, the final push is the provide "rounding".  This is done by modifying the return value as follows:

      X = (((X \* (16384 - G)) + ((Y \* 100 \* 8192) \* G)) / 16384

      return (X + 4095) / 8192

  This will result in the typical rounding where a value of less than 0.5 would be rounded down, and a value greater than 0.5 is rounded-up. But, this will round positive values up, but negative values down.  So, the final equation used for the filtering is:

      X = (((X \* (16384 - G)) + ((Y \* 100 \* 8192) \* G)) / 16384

      if (X > 0)

        return (X + 4095) / 8192

      else

        return (X - 4095) / 8192

Why was the value of 8192 and 16384 used?  8192  = 2^^13 and 16384 = 2^^14.  These values can be done with shift operations.  And most modern compilers will replace the multiply/divide operations and use shift operations, which are way faster.

DigitalFilter routine implements a Digital Filter. For each signal being filtered, this routine needs to be called every 5ms to provide the correct filtering response. The filtered value is 100 times greater than the input value.

![](data:image/png;base64;base64,)

**Inputs**:  stFilterData \*FilterData, int32\_t NewData, uint32\_t Gain

**Outputs**: None

ProcessAdcFilter routine is called every 5ms (on average) from Main Loop.  This routine provides digital filtering of the KMA (Angular Sensor), Accelerometer, Temperature Sensor, Power supplies, and RV Output voltage and current. This routine updates SysVars data structure. Only run filtering if hardware is present and FactoryConfiguration data is valid.

![](data:image/png;base64;base64,)

**Inputs**:    None

**Outputs**: None

ProcessAnalogReadings:

This routine is called from main loop every 10ms (on average). This routine converts the filtered analog readings into Engineering Units. The results of the conversion are stored in SysVars data structure.

At the STmicro factory, they connect a precision 3.3VDC power supply to the chip and then do an A/D conversion on the internal Voltage Reference.  This value is then stored into VREFINT\_CAL\_ADDR in FLASH. - And can not be changed.

In a user system, an A/D conversion on Internal Voltage Reference is done.  The A/D value read in the user system should be a ratio of the value stored at VREFINT\_CAL\_ADDR, multiplied by 3.3.  This should result in the "actual" 3.3VDC power supply voltage.

The following actions are performed in this routine:

Read factory calibration value from FLASH

Calculate Power Input Voltage and +5VDC Power Supply Voltage

FactoryConfiguration->Calibration33 = Measured value of 3.3 Power Supply when Sender was tested at CiES

The 100/342 (or 100/2046) ratio is used so the KFactor value is as close as possible to 10000 during factory testing at CiES.

Run temperature calculations if present and Factory Configuration data is valid

Convert Temperature Sensor Filtered ADC counts into volts (Temperature Input Voltage) (x.xxxx V)

Convert Temperature Input Voltage into  DegC (xx.xx C)

Convert Temperature (Deg C) into Frequency:

     Freq = 87252.07 / (123 � T)

     -55=490.18Hz        0=709.37Hz      +120=29084.02Hz

Run KMA calculations only if present

Convert KMA filtered ADC counts into KMA voltage (x.xxxx V)

The output from the KMA is approx 0.20 to 4.8V - However, the A/D input range is 0V to 3.3V -  Therefore, to read the KMA output voltage, the signal goes through a voltage divider (22K/39K), which reduces the input from 0.2 to 4.8V to 0.128 to 3.069V.  To eliminate errors due to the tolerances of the voltage divider, a K-Factor is added.  The K-Factor is set during testing of the board. The 100/261 ratio is used so the KFactor is as close as possible to 10000 during factory testing at CiES.

Calculate KMA Percentage (xxx.xxx %).  Voltage output from KMA is a function of magnetic field and Vcc.

Vcc is connected to +5VDC.

The KMA Percentage is:

             Percentage = KMA Voltage / 5VDC Voltage

Calculate KMA Angle (xxx.xxx Deg) based on KMA Percentage.  4% = 0 Deg    96% = 180   Deg. Calculate KMA Index based on KMA Angle     0 Deg=MIN\_INDEX,  180 Deg=MAX\_INDEX

If reverse flag is set, then 0 to 180 Deg = MAX\_INDEX to MIN\_INDEX

Calculate KMA Frequency.

         KMA Index:

         400-102300 (xxxx.xx)            KMA Frequency =  1638400 Hz to 6406 Hz (xxxxx.xx)

 Run Accelerometer calculations if present and FactoryConfiguration data is valid

 Convert Filtered Accelerometer Counts into g's based on accelerometer scaling (x.xxxx g)

    AccelerometerScaling:  0=2000, 1=4000, 2=8000

 Run R/V Output calculations if present and FactoryConfiguration data is valid

Convert Filtered RV Output Voltage counts into voltage (xx.xxxx V)  To eliminate voltage  divider tolerance errors, conversion includes a factory calibrated K-Factor.

Convert Filtered RV Output Current (voltage across currrent sense resistor) into miliamps (xxx.xxx mA) Then add-in current used by 187K/91K feedback and 187K/39K voltage divider for A/D input  (278 || 226 = 124.7K) To eliminate voltage divider tolerance errors, conversion includes a K-Factor.

Calculate RV Output Ohms (R = V/I).  (xxxx.xx Ohms)

Calculate Power being dissipated by Current Sense Resistor

            P = (I\*I)/R

            Where: I = Current (xx.xxx mA)      R = Resistance of Current Sense Resistor (which is 1 Ohm)

             P = xxx.xxx mW

 Calculate power being dissipated by MOSFET. (xx.xxx W)

            P = I \* V

            I = R/V Current

            V = R/V Output Voltage - Current Sense Resistor Voltage

            Current Sense Resistor Voltage = R/V Output Current \* Current Sense Resistance

 Current Sense Resistance = 1 Ohm

            Current = xxx.xxx mA

            Voltage = xx.xxxx V

 Power = xx.xxx W

            Current Sense Voltage = xx.xxxx V

            V (x.xxx) = I (xx.xxx mA) \* R (x.x ohms) / 10000 (Scaling Factor)

 Check if Internal Current Source should be turned on or off

![](data:image/png;base64;base64,)

**Inputs**:   None

**Outputs**: None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/analog.c |  |

**Revision:**

**Draft Revision:**

0.4

**Revision Date:**

**Version:**

4

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0155

**Description:**

### pwm.c � Process PWM subsystem

This module contains code for the initialization of PWM subsystem. It initializes Timer Base, Timer Channel.

![](data:image/png;base64;base64,)

**Inputs**:    PWM\_InitTypeDef PWM\_InitStruct

**Outputs**: None

PWM\_SetDutyCycle

On rising edge of DutyCycle == 0, then configure GPIO as an Low Output

On falling edge of DutyCycle == 0, then configure GPIO as a Timer Output

![](data:image/png;base64;base64,)

**Inputs**:    int32\_t NewDutyCycle

**Outputs**: None

InitializePWM

This routine is called once during power-up before main-loop.

![](data:image/png;base64;base64,)

Inputs:    None

Outputs: None

ProcessPwmOutput

If factory configuration is valid and RV output is present then set PwmDutyCycle to 0 if <0. If > PWM\_MAX\_DUTY\_CYCLE, then set to PWM\_MAX\_DUTY\_CYCLE. Otherwise call PWM\_SetDutyCycle.

![](data:image/png;base64;base64,)

**Inputs**:    None

**Outputs**: None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/pwm.c |  |

**Revision:**

**Draft Revision:**

0.3

**Revision Date:**

**Version:**

3

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0156

**Description:**

watchdog.c - This module controls the IWDG module?

This module contains code that controls the IWDG module. This routine is called every 10ms. The Timeout is set to 2 seconds, and the Window at 1 second.

![](data:image/png;base64;base64,)

**Inputs:**    None

**Outputs:**   None

**Low-Level Requirements:**

|  |  |
| --- | --- |
| CD284022-LLR-0140 | The Software SHALL ensure the proper timing of operations. |

**File Names:**

|  |  |
| --- | --- |
| src/watchdog.c |  |

**Revision:**

**Draft Revision:**

0.4

**Revision Date:**

**Version:**

4

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0157

**Description:**

freqout.c - Frequency Output?

This module contains code to handle frequency output.

ProcessFrequencyOutputs

This routine is called every 50ms (on average). Set the output frequency of any Digital I/O that is configured as a Temperature Frequency Output

![](data:image/png;base64;base64,)

**Inputs**:     None

**Outputs**:  None

ProcessMasterFrequencyOutput routine is called every 50ms.

The following actions are performed in this routine:

If in ErrorMode, toggle the Master Frequency Output between 10.00Hz and 20.00khz.

Otherwise, Calculate Master Frequency

            In compatibly mode, Master Frequency = 65536 / Master Index

            In Frequency Table Mode, Master Frequency is based on table

![](data:image/png;base64;base64,)

**Inputs:**None

**Outputs:** None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/freqout.c |  |

**Revision:**

**Draft Revision:**

0.3

**Revision Date:**

**Version:**

3

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0158

**Description:**

lowfuelalarm.c - Low Fuel Alarm Output Logic

This module contains the Low Fuel Alarm Output Logic.

ProcessLowFuelAlarm routine is called every 20ms from Main Loop.  This logic calculates two variables, LowFuelLevelOutputActive and LowFuelLevelOutputState. LowFuelLevelOutputActive is based on the SetPointIndex, LowFuelLevelTrippoint and the CompareMode setting.

When CompareMode = 0, then OutputActive = Setpoint < Trippoint)      Low Fuel is when Setpoint is less than Trippoint

When CompareMode = 1, then OutputActive = Setpoint > Trippoint)      Low Fuel is when Setpoint is greater than Trippoint

LowFuelLevelOutputActive:  0=Fuel level is not Low,  1=Fuel Level is Low

LowFuelLevelOutputState is based on LowFuelLevelOutputActive.

0=Output Low,  1=Output High

![](data:image/png;base64;base64,)

**Inputs:**   None

**Outputs:** None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/lowfuelalarm.c |  |

**Revision:**

**Draft Revision:**

0.5

**Revision Date:**

**Version:**

5

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0159

**Description:**

### pitch.c � Pitch Compensation Logic

### This module contains the Pitch Compensation Logic.

The following actions are done in this routine:

Exit if Pitch Compensation is disabled otherwise:

    If ControlAxis is 1, the AccelValue is AccelGravityX     x.xxxx g

    If ControlAxis is 2, the AccelValue is AccelGravityY.    x.xxxx g

    If ControlAxis is 3, the AccelValue is AccelGravityZ.    x.xxxx g

    Convert Accel (g) to Degs based on Lookup table

    Find offset (index) within Angle and Index tables

    Find KRange using ControlKTable lookup Table

    Find KFactor1 as function of KRange

    Find KFactor2 as function of KRange

    KRange = KFactor2 - KFactor1;

    Find PitchKFactor

![](data:image/png;base64;base64,)

**Inputs**:    None

**Outputs**: None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/pitch.c |  |

**Revision:**

**Draft Revision:**

0.4

**Revision Date:**

**Version:**

4

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0160

**Description:**

testinput.c - Test Input Function?

This module contains the test input logic. It reads the test inputs and stores in Sysvars structure.

![](data:image/png;base64;base64,)

**Inputs**:     None

**Outputs**:  None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/testinput.c |  |

**Revision:**

**Draft Revision:**

0.3

**Revision Date:**

**Version:**

3

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0161

**Description:**

errorcondition.c - Handle Error Conditions?

This module sets/resets any Digital I/O configured for Error Condition Output.

ProcessErrorCondition

This routine is called every 20ms on average.

If the bErrorPresent flag is set, then there is currently an error being detected.

![](data:image/png;base64;base64,)

**Inputs:**None

**Outputs:** None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/errorcondition.c |  |

**Revision:**

**Draft Revision:**

0.3

**Revision Date:**

**Version:**

3

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0162

**Description:**

verify.c - Verification of Signals

This module contains the code for the verification of signals

ProcessVerifyStack routine is called every 1 second from Main loop.

Log message, then wait to make sure message has been saved to EEPROM, then go into infinite loop and only exit from loop is when Watchdog times-out.

![](data:image/png;base64;base64,)

**Inputs**:      None

**Outputs**:   None

SetGeneralError routine does the following:

If no current error, then reset ErrorTimer back to 0. ErrorTimer is basically a debounce timer. The error must be present for the debounce time before the error is logged to EEPROM.

![](data:image/png;base64;base64,)

**Inputs**:    int16\_t ErrorId, uint8\_t bError, int16\_t DebounceTime

**Outputs**:  None

ProcessSignalStatus routine is called every 20ms from Main Loop.  The following actions are performed:

delay for 10 seconds before looking for errors

Verify +5VDC Power Supply is within Limits

Verify Internal Reference Counts are within limits

Verify MOSFET Power

Verify Power Input Voltage is within limits

Verify KMA Percentage is within limits

 Verify Temperature is within Limits

Verify Slave Frequency is within limits if Digital I/O is configured for Slave Frequency Input

Check that Setpoint and Feedback are within limits

![](data:image/png;base64;base64,) 

**Inputs:**    None

**Outputs:**   None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/verify.c |  |

**Revision:**

**Draft Revision:**

0.4

**Revision Date:**

**Version:**

4

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0163

**Description:**

dig.c - Digital I/O Routines

DIG\_Init routine is used to initialize a Digital I/O Output.

This routine sets up and saves DIG\_HandleStruct data structure based on selected Channel (0-5). Digital I/O #5 has not timer available. Then GPIO is initialized base on selected mode.

If DIG\_IN mode selected then configure GPIO as an input without internal pull-up enabled. Configure Buffer Chip as an input also.

If DIG\_OUT mode selected configure Buffer Chip as an output. Configure GPIO as an output

If DIG\_SERIAL\_IN mode selected, it is only valid on DIG\_IO5. Configure GPIO as a UART4 RX input. Configure Buffer Chip as an input. RX connected to DIGIO1, TX connected to DIGIO1 and DIGIO2

If DIG\_SERIAL\_INOUT mode is selected, it is only valid on DIG\_IO1. Configure GPIO as a UART2 RX input. Configure Buffer Chip as an input. RX connected to DIGIO1, TX connected to DIGIO1 and DIGIO2

If DIG\_SERIAL\_OUT mode is selected, DIG\_SERIAL\_OUT is only valid on DIG\_IO2.  The DIG\_IO2 port pin (PB8) must be set as an input. The Buffer Chip is then set as an output.  The DIG\_IO2 signal wire is then driven by the USART2\_TX (PA2) signal through a 4.7K resistor (R6). Finally, configure Buffer (IC14) as an output

If DIG\_FREQ\_IN mode is selected, configure GPIO as an input in Interrupt mode on Rising Edge. Configure Buffer Chip as an input. Enable the interrupt.  NOTE:  As both Digital I/O #2 and #6 share the same interrupt pin, only one of them can be configured as a Frequency Input at any one time.

If DIG\_FREQ\_OUT mode is selected, set Buffer Chip direction to Output. Calculate Prescaler and Period.  The Prescaler is set to give the largest possible Period value that is less than 65000. This will result in the best possible frequency accuracy.  Initialize Timer Base, then  initialize Timer Channel.

If DIG\_PWM\_OUT mode is selected set Buffer Direction to Output. Initialize Timer Base. Initialize Timer Channel

![](data:image/png;base64;base64,)

**Inputs:**int32\_t Ch, DIG\_InitTypeDef DIG\_InitStruct

**Outputs:** None

DIG\_SetFrequency routine is Digital I/O Support routine used to set the Output Frequency of the selected Digital I/O.

![](data:image/png;base64;base64,)

Inputs:     int32\_t Ch (Channel number 0 to 5, of the Digital I/O being changed, int32\_t NewFreq (New Output Frequency, xxxxx.xx Hz)

Outputs:  None

DIG\_SetDutyCycle routine is the Digital I/O Support routine used to set the OUtput Duty Cycle of the selected digital I/O

![](data:image/png;base64;base64,)

Inputs:     int32\_t Ch (Channel number 0 to 5, of the Digital I/O being changed, int32\_t NewDutyCycle (New Output Duty Cycle, xxx.xx %)

Outputs:  None

Dig\_SetPin routine is Digital I/O support routine used to set a digital I/O output.

![](data:image/png;base64;base64,)

Inputs:     int32\_t Ch (Channel number 0 to 5, of the Digital I/O being changed)

Outputs:  None

Dig\_ResetPin routine is Digital I/O support routine used to reset a digital I/O output.

![](data:image/png;base64;base64,)

Inputs:     int32\_t Ch (Channel number 0 to 5, of the Digital I/O being reset)

Outputs:  None

Dig\_ReadPin routine is Digital I/O support routine used to read a digital I/O input.

![](data:image/png;base64;base64,)

Inputs:     int32\_t Ch (Channel number 0 to 5, of the Digital I/O being reset)

Outputs:  STM32\_GPIO\_ReadPin(DIG\_HandleStruct[Ch].Dig\_IO\_Port, DIG\_HandleStruct[Ch].Dig\_IO\_Pin)

GetDigIo routine is Digital I/O support routine used to read digital I/O input.

![](data:image/png;base64;base64,)

Inputs:     int32\_t Ch (Channel number 0 to 5, of the Digital I/O being reset)

Outputs:  FlashConfiguration->Digio(N)

InitializeDigitalIO routine is Digital I/O support routine used to initialize all Digital I/O. If mode is DIG\_MODE\_POWERUP then

      DIG\_InitStruct.Config = CONFIG\_DIG\_DISABLED;

      DIG\_InitStruct.Mode = DIG\_IN;

      DIG\_Init(i, DIG\_InitStruct);

Otherwise disable all digital I/O interrupts. Disable All Digital I/O Timers. Disable All Digital I/O Data Port Pins (basically turns all Data Ports into inputs. Set all Digital I/O External Bi-Directional Buffers as inputs.

![](data:image/png;base64;base64,)

Inputs:     int8\_t Mode (Mode - DIG\_MODE\_POWERUP or DIG\_MODE\_NORMAL)

Outputs:  None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/dig.c |  |

**Revision:**

**Draft Revision:**

1.2

**Revision Date:**

**Version:**

12

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0164

**Description:**

error\_handler.c

This module places STM32 into an infinite loop. The only exit from the loop is when Watchdog Timer resets STM32 processor.

![](data:image/png;base64;base64,)

**Inputs**:     None

**Outputs**:   None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/error\_handler.c |  |

**Revision:**

**Draft Revision:**

0.4

**Revision Date:**

**Version:**

4

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0165

**Description:**

execution.c module Monitors Min, Max, and Average Execution Times.

InitializeExecutionTimers routine initializes TIMER2 to 16Mhz providing a resolution of 0.0625 uS. TIMER2 is also used to measure Slave Frequency inputs.

![](data:image/png;base64;base64,)

**Inputs:** None

**Outputs:** None

ExecutionStart routine is called at the beginning of each major function/procedure. The routine records the value of TIMER2.

![](data:image/png;base64;base64,)

Inputs:    int16\_t Process (Process ID Number)

Outputs:  None

ExecutionEnd routine is called at the end of each major function/procedure. The routine reads the value of TIMER2. The routine then subtracts the beginning time from the current timer value, resulting in the execution time of the routine. The routine then checks if current execution time is less than Min or greater than Max. Routine also adds current time to accumulator, which is used later to calculate the average execution time.

![](data:image/png;base64;base64,)

Inputs:    int16\_t Process (Process ID Number)

Outputs:  None

ExecutionAverage routine is s called every 1 second (on average).  It calculates the average time based on accumulated time, and it also saves the Min/Max times, then resets Min/Max and accumulator. Average time is xx.xxx uS.

![](data:image/png;base64;base64,)

Inputs:    None

Outputs:  None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/execution.c |  |

**Revision:**

**Draft Revision:**

0.7

**Revision Date:**

**Version:**

7

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0166

**Description:**

global.c - this include file contains common declarations and definitions for the program

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/global.c |  |

**Revision:**

**Draft Revision:**

0.2

**Revision Date:**

**Version:**

2

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0167

**Description:**

hardware.c - This module contains the low-level hardware functions that used by several other modules. This module provides Hardware Support Routines for the STM32. These routines enable and disable interrupts, set priority, and sleep.

EnableInterrupts routine performs a master enable of interrupt that goes into the processor from NVIC.

![](data:image/png;base64;base64,)

**Inputs**:     None

**Outputs**:  None

DisableInterrupts routine performs a master disable of interrupt that goes into the processor from NVIC.

![](data:image/png;base64;base64,)

**Inputs**:    None

**Outputs**:  None

\_\_DSB routine is a process synchronization routine.

![](data:image/png;base64;base64,)

**Inputs**:    None

**Outputs**:  None

STM32\_GetTick routine returns uTicks which gets incremented every 1ms.

![](data:image/png;base64;base64,)

**Inputs:**None

**Outputs:** uTicks

STM32\_Delay is used to create a wait time.

![](data:image/png;base64;base64,)

Inputs:     volatile uint32\_t Delay

Outputs:  None

STM32\_NVIC\_SetPriority routine is used to set priority for nested vector interrupt control (NVIC). Setting priority is used to figure out which interrupt to execute first.

![](data:image/png;base64;base64,)

Inputs:     IRQn\_Type IRQn, uint32\_t PreemptPriority, uint32\_t SubPriority

Outputs:  None

STM32\_NVIC\_EnableIRQ NVIC is like a hardware module within the chip, inputs is from different things that cause interrupts (multiple inputs and one output) � all inputs are controlled (enable/disable) individual interrupts that go into NVIC � the output of NVIC goes into the processor. Processor Only has one interrupt input. When an interrupt occurs, the program has to figure from NVIC where interrupt came from.

![](data:image/png;base64;base64,)

Inputs:     IRQn\_Type IRQn

Outputs:  None

STM32\_NVIC\_DisableIRQ is used to disable interrupts.

![](data:image/png;base64;base64,)

Inputs:     IRQn\_Type IRQn

Outputs:  None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/hardware.c |  |

**Revision:**

**Draft Revision:**

0.6

**Revision Date:**

**Version:**

6

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0168

**Description:**

interrupts.c contains System Interrupt routines.

The following routines are used for detecting a change in digital I/O lines. Also used when Digital I/O is in slave/frequency input mode.

![](data:image/png;base64;base64,)

Inputs:     None

Outputs:  None

EXTI2\_3\_IRQHandler routine

![](data:image/png;base64;base64,)

Inputs:     None

Outputs:  None

EXTI4\_15\_IRQHandler

![](data:image/png;base64;base64,)

Inputs:     None

Outputs:  None

SysTick\_Handler routine gets called every 1ms to increment uticks.

![](data:image/png;base64;base64,)

Inputs:     None

Outputs:  None

The following are USART interrupt handler routines.

USART1\_IRQHandler routine

![](data:image/png;base64;base64,)

Inputs:     None

Outputs:  None

USART2\_IRQHandler routine

![](data:image/png;base64;base64,)

Inputs:     None

Outputs:  None

USART3\_4\_IRQHandler routine

![](data:image/png;base64;base64,)

Inputs:     None

Outputs:  None

The following are DMA interrupt handler routines.

DMA1\_Channel1\_IRQHandler routine

![](data:image/png;base64;base64,)

Inputs:     None

Outputs:  None

DMA1\_Channel2\_3\_IRQHandler routine

![](data:image/png;base64;base64,)

Inputs:     None

Outputs:  None

DMA1\_Channel4\_5\_6\_7\_IRQHandler routine

![](data:image/png;base64;base64,)

Inputs:     None

Outputs:  None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/interrupts.c |  |

**Revision:**

**Draft Revision:**

0.6

**Revision Date:**

**Version:**

6

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0169

**Description:**

logevent.c

The event log uses 32,768 bytes of the EEPROM, which is 2048 entries of 16 bytes each.

InitializeEventLog routine is called during the initialization phase of main loop.  The code checks the Index (first 4 bytes) of each record, looking for one of two things:

1. Either a -1, which would indicate the end of the event log, or

2. Looks for the record with the lowest index number.

 If a -1 isn't found, then this indicates that the Event Log is full and has rolled-over back to the beginning (ring buffer). In this case, the next entry into the log is the record with the lowest index number.

There is one specical case:  Last message was stored at the end of the Event Log.  In this case, the Entry Number doesn't drop.

There are two possiblies -  One, the table hasn't been completely filled. Two, the table has been completely filled. If the table hasn't been completely filled, then we will find an EntryNumber of -1. This would indicate the end of the Event Log and would be where the next message should be stored. If the Event Log has been completely filled and rolled-over, then the location where the next message should be stored will be where the Index value drops in value.

![](data:image/png;base64;base64,)

**Inputs:**None

**Outputs:**None

ProcessClearEventLog routine is called from every 10ms from Main Loop  The log is cleared by writing 0xFF to all 32,768 bytes.  This is done 128 bytes at a time, as that is the page size of the EEPROM.

![](data:image/png;base64;base64,)

**Inputs:**None

**Outputs:** None

LogEvent routine - Each Event Message is 64 bytes. The Event Log stores the last 512 messages (ring buffer).  With each messages being 64 bytes, that uses 32,768 bytes of EEPROM. When the end of the log buffer is reached, loop back to the beginning (ring buffer).

![](data:image/png;base64;base64,)

**Inputs:**uint32\_t EventId, int32\_t EventValue1, int32\_t EventValue2

**Outputs:** None

ProcessEventMessageQueue routine is used to process event messages.

![](data:image/png;base64;base64,)

Inputs:     None

Outputs:  None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/logevent.c |  |

**Revision:**

**Draft Revision:**

0.5

**Revision Date:**

**Version:**

5

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0170

**Description:**

math64.c provides 64 bit math support routines.

DivideWithRounding64 routine provides rounding. Result = (Dividend + (Divisor / 2)) / Divisor.

![](data:image/png;base64;base64,)

**Inputs:**int64\_t Dividend, int32\_t Divisor, enumMultDivideResult MultDivideResult

**Outputs:** Result

MultDivideWithRounding64 routine provides rounding.

Result = (Dividend + (Divisor / 2)) / Divisor

![](data:image/png;base64;base64,)

Inputs:   int32\_t Mult1, int32\_t Mult2, int32\_t Divisor, enumMultDivideResult MultDivideResult

Outputs:  Result

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/math64.c |  |

**Revision:**

**Draft Revision:**

0.2

**Revision Date:**

**Version:**

2

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0171

**Description:**

??reflash\_page.c module is used for writing pages 61, 62, and 63 into Flash. Flash memory is broken into 64 pages and each page has 2k bytes of data. Pages 61, 62, and 63 are used to store PDI information.

WaitForLastOperation routine is used to when an operation has been initiated in flash routine and there must be a wait time so that operation finishes before another operation can be started.

![](data:image/png;base64;base64,)

**Inputs:**None

**Outputs:** None

Flash\_Program routine writes data to flash.

![](data:image/png;base64;base64,)

Inputs:     uint32\_t Address, uint16\_t \*WordPtr

Outputs:  None

ReflashTuningParameters adds one layer on top of previous routine. This routine erases previous page so a new one can be written. Also unlocks flash control registers so page can be erased and rewritten.

![](data:image/png;base64;base64,)

Inputs:     uint32\_t PageNumber, uint16\_t \*DataPtr

Outputs:  None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/reflash\_page.c |  |

**Revision:**

**Draft Revision:**

0.6

**Revision Date:**

**Version:**

6

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0172

**Description:**

reflash.c is used to reflash firmware. Firmware can go from page 0 to 60. This routine gets programmed to page 55 of flash. When serial command is received to reload the firmware, control of program is executed from page 55 of flash.

WaitForLastOperation routine is used to wait until flash operation is done.

![](data:image/png;base64;base64,)

**Inputs:**None

**Outputs:** HAL\_Status

CalculateCrcArray routine takes a buffer and the length and returns CRC16.

![](data:image/png;base64;base64,)

Inputs:     uint8\_t \*Buffer, int32\_t Length

Outputs:  CRC->DR & 0xFFFF

ReflashSerialSend routine is used to transmit serial response back to host after a serial command is received.

The routine performs the following:

- Sync Serial Buffers, then calculate CRC.

-Send Message via selected SerialMode Mode

-Turn on RS422/485 Driver if that's mode selected

-Set  DIG\_IO as output

-Wait for Byte to be sent (Transmission Complete)

- Send Data

-Turn on RS422/485 Driver

- Set DIG\_IO as output

![](data:image/png;base64;base64,)

Inputs:     stSerial \*SerialStruct

Outputs:  None

GetSerialMessage routine is used to retrieve serial message from host.

![](data:image/png;base64;base64,)

Inputs:     stSerial \*SerialStruct

Outputs:  TRUE or FALSE

reflash routine disables the watchdog timer, and interrupts. Serial send and then get serial message, then tells host when it's done and then locks flash and issues a system reset. It executes new program.

![](data:image/png;base64;base64,)

Inputs:     stSerial \*SerialStruct

Outputs:  None

Flash\_Program routine erases current page. Waits for last operation to be completed. Wait for last operation to be completed. Proceed to program the new data. Write data in the address. Wait for last operation to be completed. If the program operation is completed, disable the PG bit.

![](data:image/png;base64;base64,)

Inputs:     uint32\_t Address, uint16\_t \*WordPtr

Outputs:  status

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/reflash.c |  |

**Revision:**

**Draft Revision:**

0.5

**Revision Date:**

**Version:**

5

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0173

**Description:**

timer.c - This module contains the STM32 timer routines. Most are original STM HAL code with some modifications by CiES.

STM32\_TIM\_Base\_Init routine is used to initialize timer base.

![](data:image/png;base64;base64,)

**Inputs:**TIM\_TypeDef \* TIMx, uint32\_t Prescaler, uint32\_t Period, uint32\_t AutoReloadPreload

**Outputs:** None

STM32\_TIM\_OC1\_SetConfig initializes channel within the timer. (for channel 1 and 2).

![](data:image/png;base64;base64,)

Inputs:     TIM\_TypeDef \* TIMx, TIM\_OC\_InitTypeDef \* OC\_Config

Outputs:  None

STM32\_TIM\_OC2\_SetConfig

![](data:image/png;base64;base64,)

Inputs:     TIM\_TypeDef \* TIMx, TIM\_OC\_InitTypeDef \* OC\_Config

Outputs:  None

STM32\_TIM\_PWM\_ConfigChannel initializes timer in PWM mode.

![](data:image/png;base64;base64,)

Inputs:     TIM\_TypeDef \* TIMx, TIM\_OC\_InitTypeDef \* sConfig, uint32\_t Channel

Outputs:  None

STM32\_TIM\_MspPostInit initializes port pin, connects timer to a port pin.

![](data:image/png;base64;base64,)

Inputs:     TIM\_TypeDef \* TIMx

Outputs:  None

STM32\_TIM\_PWM\_Start routine starts PWM timer.

![](data:image/png;base64;base64,)

Inputs:     TIM\_TypeDef \* TIMx, uint32\_t Channel

Outputs:  None

STM32\_TIM\_Base\_DeInit  routine uninitializes a timer.

![](data:image/png;base64;base64,)

Inputs:     TIM\_TypeDef \* TIMx

Outputs:  None

STM32\_TIM\_Base\_Start routine starts base timers.

![](data:image/png;base64;base64,)

Inputs:     TIM\_TypeDef \* TIMx

Outputs:  None

MX\_TIM2\_Init routine initializes timer 2.

![](data:image/png;base64;base64,)

Inputs:     None

Outputs:  None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/timer.c |  |

**Revision:**

**Draft Revision:**

0.5

**Revision Date:**

**Version:**

5

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0174

**Description:**

can.c - this module handles the CAN I/O.

STM32\_CAN\_Init routine initializes CAN peripheral device.

![](data:image/png;base64;base64,)

**Inputs:**CAN\_HandleTypeDef \* hcan

**Outputs:** HAL\_Status

MX\_CAN\_Init routine is part of CAN initialization.

![](data:image/png;base64;base64,)

Inputs:     None

Outputs:  None

HAL\_CAN\_MspInit routine is part of port pins initialization for CAN bus.

![](data:image/png;base64;base64,)

Inputs:     CAN\_HandleTypeDef \* canHandle

Outputs:  None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/can.c |  |

**Revision:**

**Draft Revision:**

0.4

**Revision Date:**

**Version:**

4

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-0175

**Description:**

dma.c? - DMA I/O routines

This module contains code for DMA I/O.

STM32\_DMA\_Init routine initializes DMA controller.

![](data:image/png;base64;base64,)

**Inputs:**DMA\_HandleTypeDef \*hdma

**Outputs:** None

STM32\_DMA\_Start\_IT routine starts interrupt. It starts DMA transfer.

![](data:image/png;base64;base64,)

Inputs:    DMA\_HandleTypeDef \*hdma, uint32\_t SrcAddress, uint32\_t DstAddress, uint32\_t DataLength

Outputs: None

STM32\_DMA\_IRQHandler routine handles DMA\_IRQ. It transfers complete interrupt management. Disables the transfer complete & transfer error interrupts, if the DMA mode is not circular. Clears the transfer complete flag. Transfers Error Interrupt management. When DMA transfer error occurs a hardware clear of its EN bits is performed. Then disable all DMA interrupts. Finally clear all flags.

![](data:image/png;base64;base64,)

Inputs:    DMA\_HandleTypeDef \*hdma

Outputs: None

**Low-Level Requirements:**

**File Names:**

|  |  |
| --- | --- |
| src/dma.c |  |

**Revision:**

**Draft Revision:**

0.4

**Revision Date:**

**Version:**

4

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender

1. Module Description ID: MD-10

**Description:**

This module controls initialization

**Low-Level Requirements:**

|  |  |
| --- | --- |
| CD284022-LLR-0020 | The software shall check to see that the factory configuration is valid. |
| CD284022-LLR-0030 | The software shall check to see that an accelerator is present. |
| CD284022-LLR-0040 | The software shall set the accelerometor scaling.   1. Send '0' to CTRL\_REG1 (0x2A) register of Accelerometer. 2. Send Accelerometer Scaling Value to XYZ\_DATA\_CFG (0x0E) register of Accelerometer. 3. Send '1' to CTRL\_REG1 (0x2A) of Accelerometer. |
| CD284022-LLR-0070 | Upon initialization of the ADC, the software shall set the priority and subpriority of the ADC DMA Channel 1 to 0. |
| CD284022-LLR-0080 | Upon initialization of the ADC, the software shall enable I/O Requests on DMA Channel 1. |
| CD284022-LLR-0090 | Upon Initialization of the ADC, the software shall assign Processor A Pin 1 to the ADC IN 0 alternate. |
| CD284022-LLR-0100 | Upon Initialization of the ADC, the software SHALL assign Processor A Pin 5 to the ADC IN 5 alternate. |
| CD284022-LLR-0110 | Upon Initialization of the ADC, the software SHALL assign Processor A Pin 6 to the ADC IN 6 alternate. |
| CD284022-LLR-0120 | Upon Initialization of the ADC, the software SHALL assign Processor A Pin 7 to the ADC IN 7 alternate. |
| CD284022-LLR-0010 | 1. LLR Description  Software shallhave boot code written in Assembly Language that performs the following:  * Move Initialized variable data from FLASH to SRAM * Sets uninitialized variables to zero * Set all unused memory to 0x12345678 * Set Stack Pointer to end of SRAM * Call main() * Interrupt Vectors will also be stored in the boot code |

**File Names:**

|  |  |
| --- | --- |
| src/accel.c |  |
| src/adc.c |  |
| src/can.c |  |
| src/dig.c |  |
| src/dma.c |  |
| src/execution.c |  |
| src/gpio.c |  |
| src/initsystem.c |  |
| src/logevent.c |  |
| src/mcp3421.c |  |
| src/pwm.c |  |

**Revision:**

**Draft Revision:**

1.2

**Revision Date:**

**Version:**

12

**Item:**

CD284022 Series Liquid Level Sender

**Project:**

CD284022 Series Liquid Level Sender