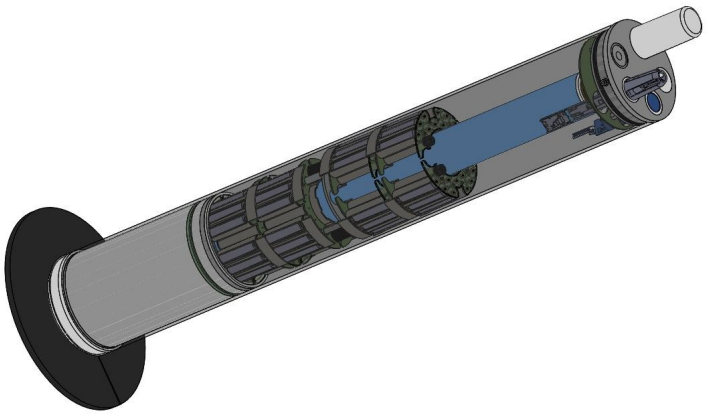
Low Cost Profiler (LCP)



User Manual
Control-board

Author : Basharat Martin
Basharat.Martin@noaa.gov
dated : 06/19/2024

Supervised by:
Noah Lawrence-Slavas, Matthew Casari

Engineering Development Division
NOAA Pacific Marine Environmental Laboratory
7600 Sand Point Way NE
Building 3/EDD
Seattle, WA 98115

Table of Contents

List of Figures

List of Tables

1. Introduction

Control-board is one of the key components of the Low Cost Profiler (LCP) which schedules the tasks and controls the whole flow of LCP using a StateMachine with FreeRTOS. It is interfaced with the piston-board for the power-lines as well as the I2C bus for controlling the actuator.

2. Hardware

The PCB of control-board is designed of rectangular shape with minimum dimensions approximately 18x5.1x1.5 (LxWxH) in cm. The artemis module which has Apollo3 blue SoC (MCU – ARM Cortex-M4 FPU) is the central piece on the control-board. It supports many different communication protocols which provide interfaces to internal and external peripherals. These interfaces, internal and external peripherals are listed below.

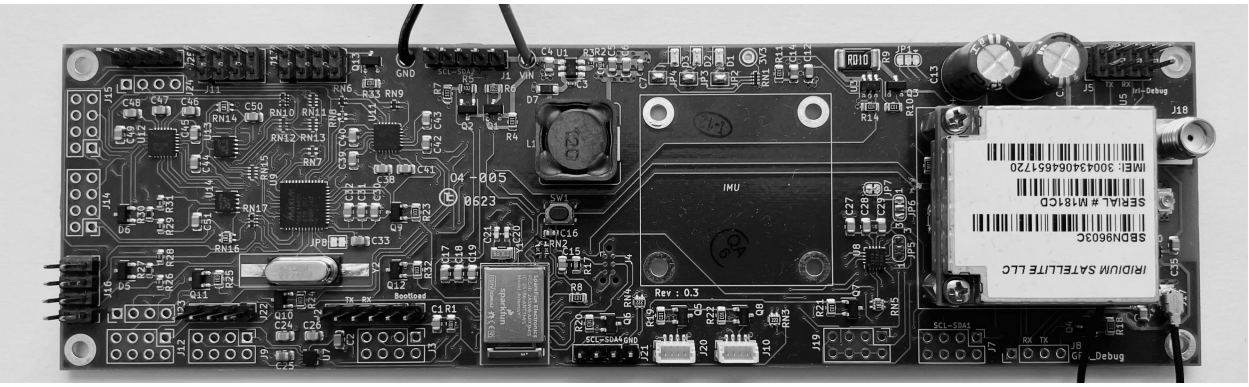


Figure 1: Control-board PCBA

2.1. Power Specification

Control-board supports the input voltage range from 6V to 42V as required by the DC-DC buck converter BD9G102G-LB (ROHM semiconductors). The maximum output current is 0.5A which is enough to run all the attached peripherals. The maximum output voltage is 3.3V except the LTC3225 Supercapacitor charger which outputs 5.3V to charge the capacitors used in the iridum module circuitry.

2.2. LEDs

There are three LEDs (Red, Green and Blue) namely LED1, LED2 and LED3 for debugging purpose. During the power consumption phase, these can be turned off or power saving mode.

2.3. Peripherals

The red chip on the pcb is the artemis module manufactured by sparkfun. It has the Apollo3 Blue SoC with MCU (ARM – Cortex-M4F) 48MHz main clock with burst mode max up to 96MHz, 1MB Flash and 384KB RAM. The following table describes the attached peripherals' connection to artemis module and based on the schematic naming conventions provided by artemis module pin out.

Table 1: Peripherals

Interface	Description
UART0 – (RX0/TX0)	UART0 is used to flash the firmware as well as listening on Debug outputs.
UART1 – (RX1/TX1)	Iridium 9603n module is connected to UART1 of artemis module.
I2C_1 – (SCL1/SDA1)	GPS – ublox is connected and functional on this interface. There are other peripherals connected on this bus as well but non-functional at the moment. 1. ADC 2. Qwiic connector (IMU not available on the pcb yet) 3. Qwiic connector (Spare) available for attaching an additional peripheral.
I2C_2 – (SCL2/SDA2)	Piston-board is connected on this bus along with power-lines.
I2C_4 – (SCL4/SCL4)	Openlog Datalogger (sparkfun) is connected on this interface.
SPI_0 – (MISO0/MOSI0)	Accelerometer is attached to this interface but non-functional at the moment.
SPI_3 – (MISO3/MOSI3)	MAX14830 (SPI to UART) is connect on the SPI_3 interface of artemis module and provides 4 UARTs to be served. Additional RS-232 and RS-485 converters are connected to these UARTs to provide interfaces to those peripherals which support RS-232 or RS-485. 1. Temperature Sensor S9 – is connected to UART COM0 through RS-232. 2. Pressure Sensor K9LX – is connected to UART COM3 through RS-485.

The existing peripherals which are being used for first prototype LCP test listed below:

2.3.1. Pressure Sensor

The pressure sensor from Keller (K9LX) with pcb number (9L 140) is connected at COM3 of MAX14830 through RS-485. Its defualt baud-rate is 9600 bps and runs also at 9600 bps. It uses the RS-485 driver with some tailored characteristics described by the keller communication protocol documentation.

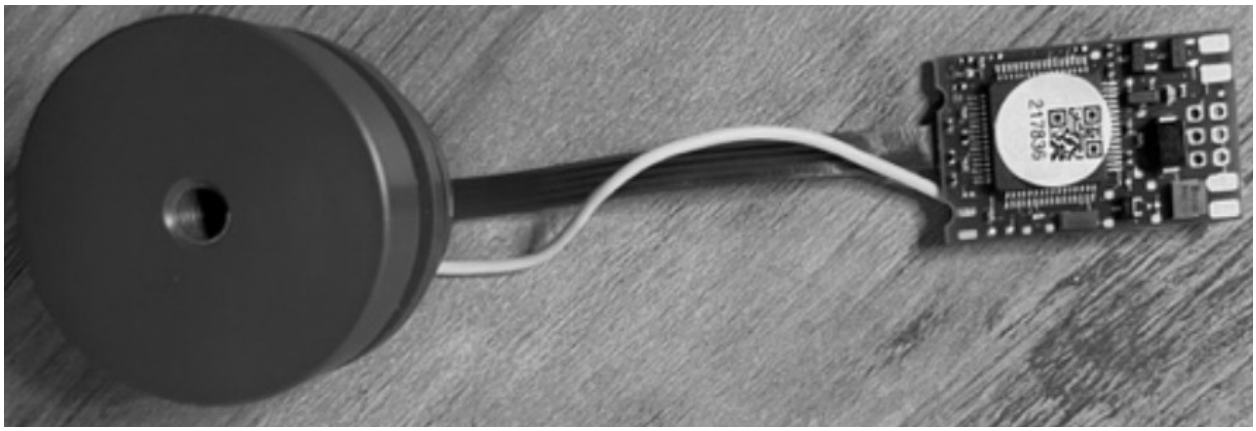


Figure 2: Keller K9LX

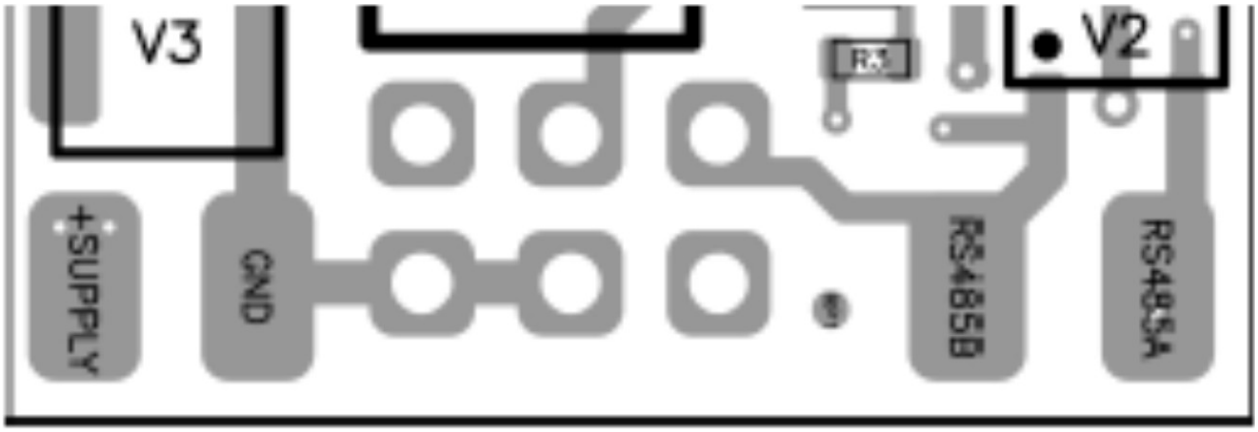


Figure 3: Keller K9LX – board (9L 140) pin-out

2.3.2. Temperature Sensor

The OEM digital temperature sensor S9 (Sound Nine) is connected at COM0 of MAX14830 through RS-232 serial interface. It is configured to operate at 9600 8N1 baud-rate. Its default sampling rate is 2Hz. S9 temperature sensor starts sampling and outputing temperature values right away it gets powered on. Auto-sampling is stopped by sending a “stop” command. Measurements are taken by sending a “sample” command explicitly.

Figure 4: OEM S9 Temperature Sensor

3. Schematic Design Changes

The schematic of LCP control-board initially was designed on KiCAD version 5 starting with revision 0.1 and up to revision 0.2. Later the schematic was upgraded to KiCAD version 6 with design change.

As per schematic design revision 0.3, there are a few modifications need to be made manually on the pcb in order for Max14830 to be functional as well as for iridium module. Those changes are mentioned below.

The schematic files are available from the following github server (NOAA – PMEL), branch **convert/kicad6**

git clone https://github.com/NOAA-PMEL/EDD-LCP_Control.git
cd EDD-LCP_Control
git checkout convert/kicad6

3.1. MAX14830

Max14830 from maximintegrated is a quard serial uart which is connected via SPI bus to the artemis module. There are four UARTs provided by the Max14830 chip and are configured in a way that either four of these can be used plain as UARTs or four RS-232 via TI-TRS3122E transceiver or two RS-232 and two RS-485 by removing a few pair of resistors from the PCBA. The following configuration of COMs is used to provide interfaces to pressure sensor and temperature sensor.

Table 2: Resistor Configuration

COM	Description
COM0 – RS-232	Removing RN7 and RN8 would enable RS-232 through COM0 for temperature sensor
COM1 – UART1	UART1 is enabled but not used at the moment
COM2 – UART2	UART2 is enabled but not used at the moment
COM3 – RS485	Removing RN11, RN12, RN13, and RN15 would enable the RS-485 through COM3 for Pressure sensor.

Note: Currently above mentioned resistor configuration is used and so in the firmware. In case, if there is any change in the configuration to use a different COM for an additional or existing UART, the changes need to be made in the firmware as well.

The following figures show the resistors network configuration in the schematic.

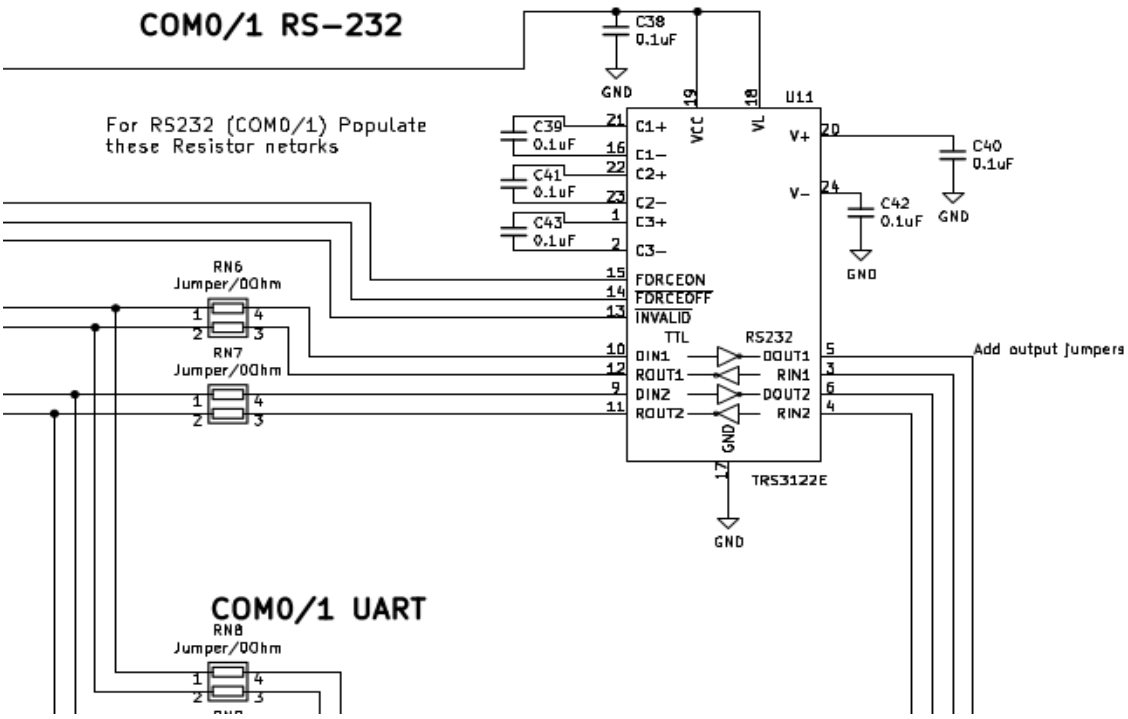


Figure 5: RS-232 Resistor Configuration

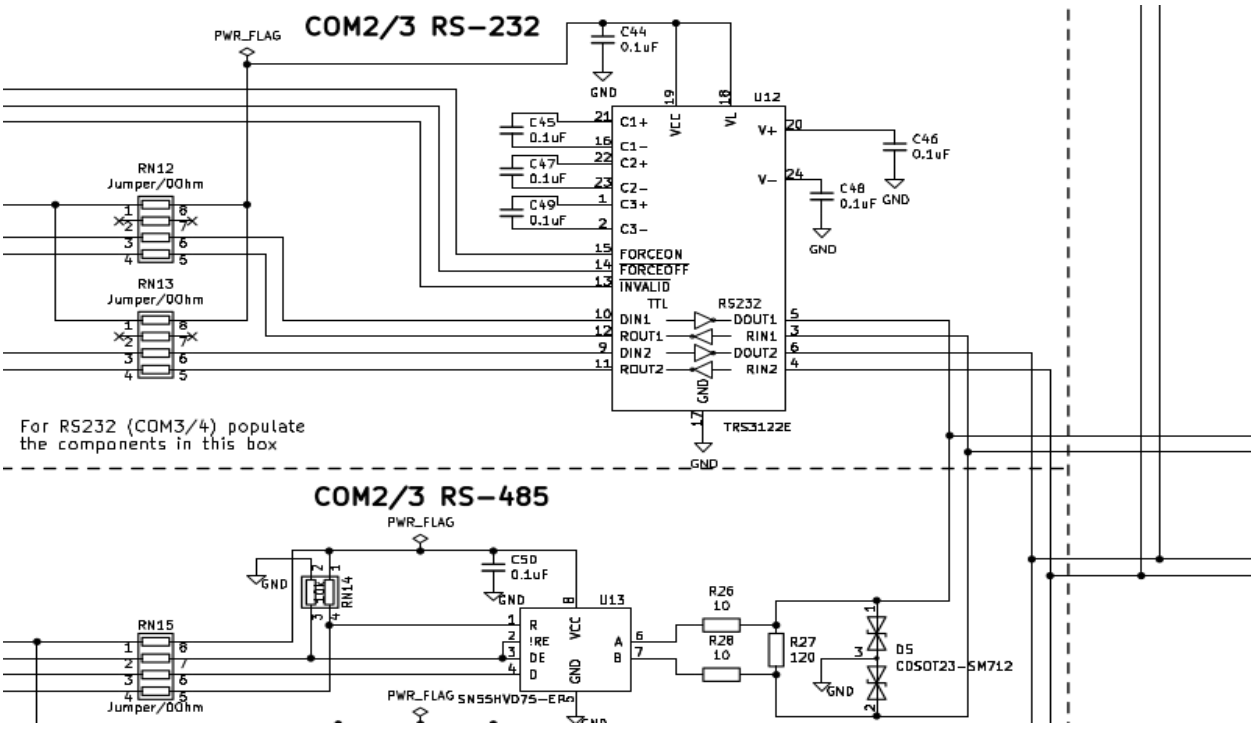


Figure 6: RS-485 Resistor Configuration

3.2. Iridium – 9603n

Iridium 9603n is a satellite modem to transmits and receives short burst data (SBD) messages through iridium satellites. It is connected to artemis module via UART1 with a baud-rate 19200 bps. The module's pin number 14 (DTR) needs to be grounded in order for Iridium – 9603n to work.

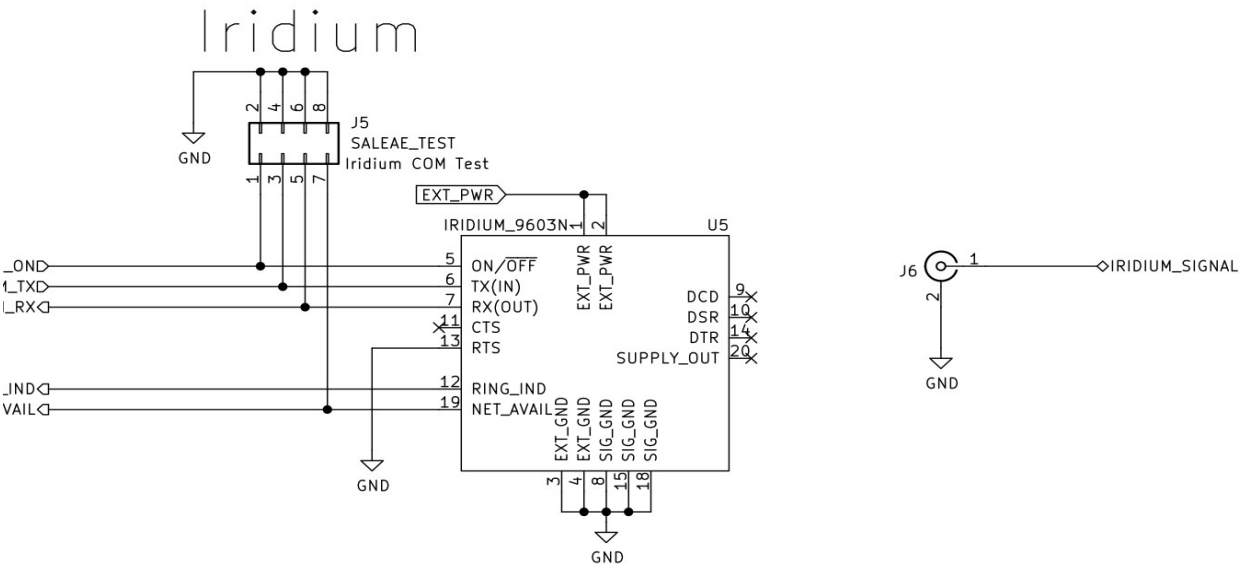


Figure 7: Iridium – 9603n

3.3. Antenna Switch

GPS and Iridium are using the same the RF Signal through Iridium_GPS_Signal as mentioned in the figure below. V1 (GPS_3V3) and V2(EXT_PWR) need to be swapped to be fully functional. These could also be swapped at J2 and J3 by switching the C34, C36 and C37 capacitors instead of swapping GPS_3V3 and EXT_PWR. Iridium and GPS could also be using a separate antennas and rule out the antenna switch.

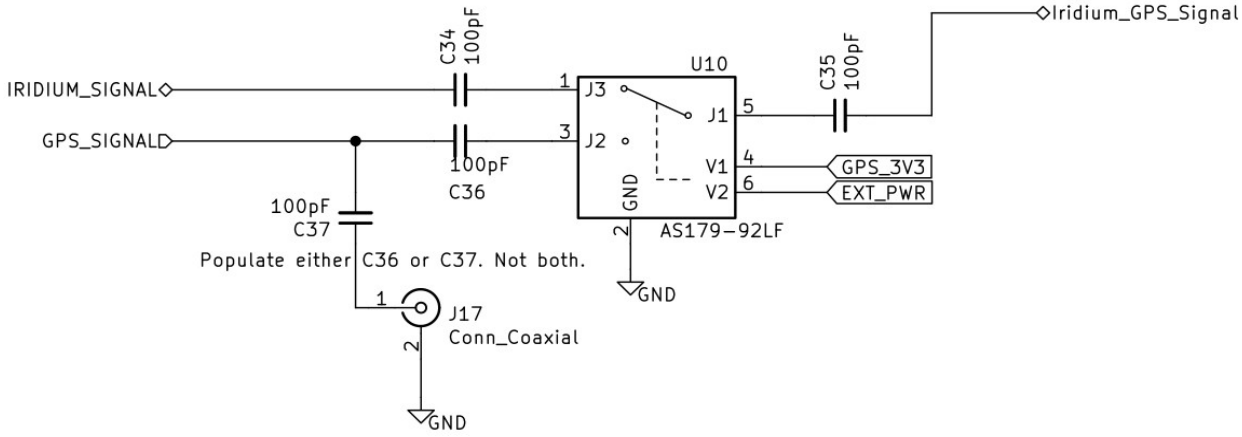


Figure 8: Antenna Switch

3.4. I2C Pull-up Resistors

RN2 2k2 resistors need to be replaced with 10k for the I2C_2 bus, since these should be close to the master (control-board) and on the other end (piston-board) the same I2C bus pull-up resistors must be removed completely.

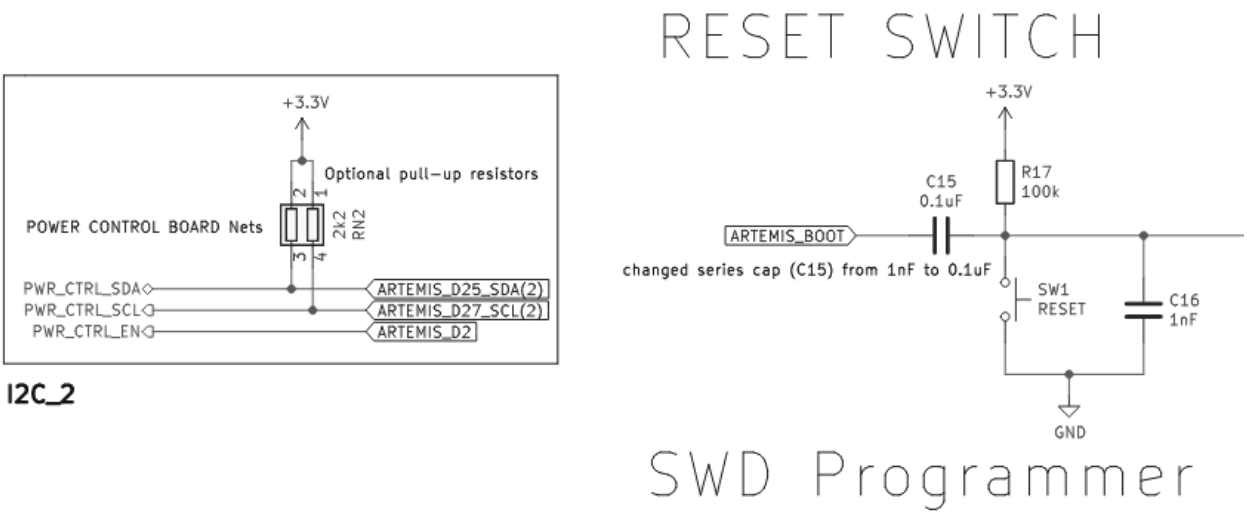


Figure 9: I2C_2 pull-up resistors

4. Software

Software for the control-board is open-source and available on the NOAA – PMEL github server. Currently git branch (master_gcc) is up-to-date and able to be compiled with gcc using make (Makefile).

4.1. Installation

The minimum requirement for compiling the firmware is Linux Ubuntu (22.04LTS) 64 bit version. Please follow the commands below which are tested on Linux Ubuntu (version 22.04 LTS). It could also work on Ubuntu 18 or 20 version.

git clone https://github.com/NOAA-PMEL/EDD-LCP_Control.git
cd EDD-LCP_Control
git checkout master_gcc

Note: make sure that you have “sudo” permission.
sudo ./run_script.sh

The above command will pull the necessary applications and ubuntu packages as required. If “run_script.sh” ran successfully then you must see the following output on the screen.

<< Exporting GCC PATH into .bashrc file ...
Checking if a user in the dialout group ...
User is not in the dialout group, Adding into the dialout group ...
completed

Please run the command: "source ~/.bashrc" >>

source ~/.bashrc
cd Firmware/LCP_Control/
make

“make” command will compile the firmware and output the binary file which will be later flashed onto the control-board. If “make” command ran all successfully then you must see the following output.

Linking gcc bin/LCP_Control_svl.axf with script ./linker/lcpcontrol_lowpower_svl.ld
Copying gcc bin/LCP_Control_svl.bin...

Please log out and log in again to activate user to dialout group

In order to flash the firmware into the control-board, a USB to TTL serial converter with is needed as shown in the following picture.

4.2. Firmware Version

Currently firmware version is kept to 0.0.0-dev due to LCP is under test, once the firmware release is made by a developer it can be set as mentioned in the LCP data format section.

4.3. Firmware Flash

There are two ways to flash the firmware to the control-board. First is through a Debugger namely (SEGGER J-link) using an IDE (Eclipse etc.) or a command-line but these are not configured yet. Secondly using a USB to TTL serial adapter which is currenlty being functional and used.

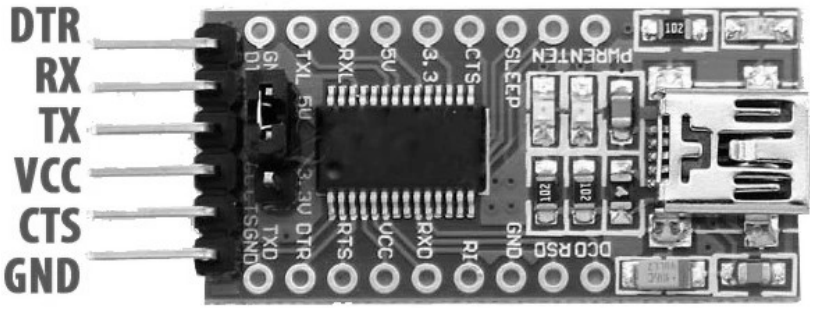


Figure 10: USB to TTL serial adapter

There are only **four** pins from this adapter required to attach to the control-board console/bootloader connector (J2). The pins connection must be followed as below.

DTR → CONSOLE_BOOT (Pin 6)
RX → CONSOLE_TX (Pin 5)
TX → CONSOLE_RX (Pin 4)
GND → GND (Pin 1)

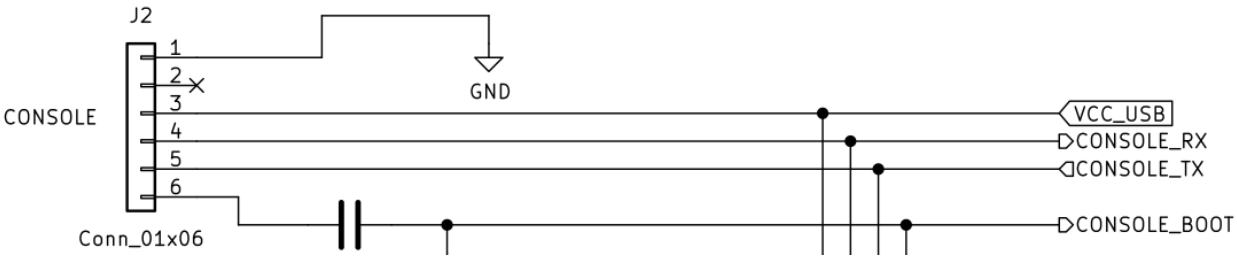


Figure 11: Control-board Console/Bootloader

Once the connection is made, power on the control-board. It is highly suggested during the firmware flashing process if it is the first time bringing-up the pcb and using external DC power-supply then set the minimum current with lower voltage e.g (50mA, 8V), otherwise use the voltage as mentioned in the Power Specification section.

Please follow the commands below to flash the firmware. Previsouly, the firmware was already compiled as mentioned in the installation section but still make sure that cursor is on right path on the terminal. Run the command pwd (print working directory) this will output the current path on the terminal.

```
# pwd
../EDD-LCP_Control/Firmware/LCP_Control
```

Before flashing the firmware, it is noted that artemis modules are shipped with two versions of bootloader, asb and svl.

ASB (Ambiq Secure Bootloader)	resides from 0x00 to 0xC000.
SVL (SparkFun Variable Bootloader)	resides from 0xC000

usually it is recommended to flash the firmware using SVL bootloader, but if for some reason it is not working, flashing with asb will always work since It is a part of Ambiq IC.

SVL bootloader flashing

```
# make bootload_svl COM_PORT=/dev/ttyUSB0
```

ASB bootloader flashing

```
# make bootload_asb COM_PORT=/dev/ttyUSB0
```

Note:

/dev/ttyUSB0 is a device (USB to TTL serial adapter) attached to Linux Ubuntu. It could also appear as ttyUSB1, or ttyUSB2 and so on. It depends on how many devices are connected.

Once the flashing firmware is done, utility named minicom or putty can be used to see the DEBUG outputs of the control-board.

4.4. Console Output

Terminal emulator or serial consoles like minicom or putty can be installed with the following command.

sudo apt install minicom
OR
sudo apt install putty

control-board UART0 is used to flash the firmware as well as for DEBUG outputs. For DEBUG outputs, UART0 is configure to run at 921600 8N1 baud-rate and using minicom command below print outputs can be seen. Please make sure that ttyUSB is picked with the correct number.

minicom -D /dev/ttyUSB0 -b 921600

The output should be following :

```
UART ENABLED

<<< TEST_PROFILE_2 Profile selected >>>

DATALOGGER :: Reading test profile pressure wait please
DATALOGGER :: file size = 5515
DATALOGGER :: Reading test profile pressure DONE

*****************************
DEBUG :: LCP Controlboard
*****************************

LCP Profiler Information
***************************************
        System ID           : 77
        LCP Firmware        : 0.0.0-dev
        Firmware Date       : 6.24.24
        Firmware Time       : 16:19:39
        LCP Variant         : 0
        LCP Serial          : L00
```

Note :
In case if there is no debug outputs, then either it is disabled in the artemis_debug.h file or press Enter button or control-board needs a reset by its reset button. The reason for this, is the DTR signal that takes the board into a bootloader mode.

4.5. FreeRTOS

The firmware of the control-board is designed and implemented of FreeRTOS as well as non-FreeRTOS fashion. Although the non-FreeRTOS functions for any specific peripheral are tested and functional but these are not scheduled to run in an endless loop to perform desired LCP tasks.

FreeRTOS scheduler is used to schedule the complete LCP tasks flow. FreeRTOS runs with the following configuration.

Table 3: FreeRTOS Configuration

Variable	Description
configCPU_CLOCK_HZ	FreeRTOS clock set to ARM Cortex-M SysTick Timer which is 48MHz
configUSE_TICKLESS_IDLE	Set to 2 (Low power Tickless mode) specified by Ambiq implementation.
configTICK_RATE_HZ	The frequency of FreeRTOS tick interrupt at 1000Hz
configMAX_PRIORITIES	Maximum task priority number set to 8
configMINIMAL_STACK_SIZE	Minimum STACK size used by idle-task to 256 bytes
configTOTAL_HEAP_SIZE	Total HEAP side used by FreeRTOS during its execution set to 32K.

There are many other useful configuration variables which be set and changed according to the requirement in the following file

include/RTOS/FreeRTOSConfig.h

4.6. StateMachine

LCP StateMachine which takes care of the complete LCP flow is scheduled with the FreeRTOS scheduler using different tasks starting from sensors initialization (non-FreeRTOS) up to the last task. Currently there are three modes are active and functional.

Note: The naming convention in the code is used for Mode as State.

The following table describes the list of active modes and its states.

Table 4: LCP modes and states

Mode	States
Pre-Deploy State/Mode	**Pre-Deploy States** 1. pds_systemcheck 2. pds_idle
Simple Profile State/Mode	**Simple Profile States** 1. sps_idle 2. sps_move_to_park 3. sps_park 4. sps_move_to_profile 5. sps_profile 6. sps_move_to_surface 7. sps_tx 8. sps_rx (non-functional for now)
Pop-Up State/Mode	**Pop-Up States** 1. pus_idle 2. pus_surface_float

4.6.1. Sensors Initialization

The control-board firmware starts off with initializing the UART Debug output as well as openlog datalogger for recording every Debug output in the SD-card. These can be enabled or disabled by modifying the following lines in the file below.

include/artemis/artemis_debug.h

```
/** put false OR true to log every DEBUG_PRINTF into the SD-Card */
#define DATALOG_DEBUG true
/** comment to disable the DEBUG_PRINTF */
#define ARTEMIS_DEBUG
```

Once the debug output configuration is passed, the StateMachine takes over and initializes remaining peripherals with non-FreeRTOS functions and outputs their specifications in detail such as LCP information, pressure sensor, temperature sensor, gps and piston-board. The output looks like the following.

```
LCP Profiler Information
**************************************
System ID               : 77
LCP Firmware            : 0.0.0-dev
Firmware Date           : 6.24.24
Firmware Time           : 13:38:30
LCP Variant             : 0
LCP Serial              : L00

**************************************
LCP Physical parameters
**************************************
Estimated Mass          : 11.139 kg, 24.557 lbs
Minimum Volume          : 0.011 m³, 654.633 in³
Maximum Density         : 1038.350 kg/m³, 0.038 lbs/in³

RTC Timer
**************************************
Clock started on 06.24.2024 at 13:28:23 (local)

K9LX Pressure Sensor
**************************************
Firmware Ver            : 5.20.12.28.
Serial Number           : 217836
Pressure                : 0.01407 bar
Temperature             : 26.259 °C, 79.266 °F
Pressure                : 1414 pascal
K9lx Pressure Sensor is initialized

S9 : Sampling Stopped
S9 Temperature Sensor
**************************************
MID                     : T2UV
C0                      : 0.0012662
C1                      : 0.0002752
C2                      : -0.0000007
C3                      : 0.0000001
R0                      : 3300.0
Average                 : 1
UID                     : 000000000302902CAF099D0358BF94F6
FW Ver                  : S9T0 V0.6
Status                  : OK
S9 Temperature Sensor is initialized

Piston Board Information
**************************************
System Identification   : LCPPST01
Firmware build year     : 2023
Firmware version        : v0.1.0

SENSORS :: Sensors are initialized

DATA :: SET, Maixmum Measurements (8000)
DATA :: SET, Maixmum Measurements (23000)

******************************
FreeRTOS here
schedular is going to start
******************************
```

After the “Sensors are initialized”, StateMachine sets the static memory for park_state and profile_state measurements data. These are usually setable constants in the **StateMachine.h** and can be modified accordingly.

The maximum number of park_state measurements are set to 80 and the maximum number of profile_state measurements are set to 230 measurements per 1 profile.

Setting these measurements for 100 profile would set the following values as shown before in the above output.

```
DATA :: SET, Maixmum Measurements (8000)
DATA :: SET, Maixmum Measurements (23000)
```

Note: static memory for park_state and profile_state measurements is linear and can be different in length of measurements for each profile. There are LCP constants defined in the following file, some of them are settable and some of them are kept non-settable. **“./include/config.h”**

On successful return from sensors initialization, StateMachines sets the freeRTOS tasks for currently three active modes namely (predeploy, simple profile and pop-up) and runs the Scheduler. The following sections will explain about each mode and its states.

4.6.2. Pre-Deploy Mode

Pre-Deploy mode or state is a manual hand deployment mode. It has two states pds_systemcheck and pds_idle. Pre-Deploy mode/state is the first state to run right after the sensors initialization function.

1. pds_systemcheck

If the sensors initialization was successful then pds_systemcheck state starts running GPS with 1Hz for GPS_Timer (default one minute) otherwise on sensors intialization failure, state will go into deep sleep by a debug-output systemcheck ERROR and turn on the red LED. Within GPS_Timer if GPS gets fixed then it will store the Latitude and Longitude coordinates and update the RTC with GPS time-stamp then it will exit and go to the pds_idel state. If in case there was no GPS fix within the GPS_Timer then the timer will run out, exit the state and go to pds_idle state without updating the RTC.

2. pds_idle

pds_idle state starts moving the piston for the calculated length based on initially assumed PARK_DENSITY and endlessly wait for the set BALLAST_DEPTH with BALLAST_DEPTH_SAMPLE_RATE. If the LCP (Low Cost Profile) is thrown into the water and pressure sensor measures the depth which is higher than BALLAST_DEPTH then Pre-Deploy Mode/State is done then it will pass on to the next Mode (Simple Profiler State).

4.6.3. Simple Profile Mode

There are eight states belong to the simple profile mode. One of them is non-functional at the moment (sps_rx). The functional states are briefly described below.

1. sps_idle

The sps_idle is a transitional state which takes over from pds_idle state, waits for one second, prints out statement mentiong starting the Profile Nr. 1 and then moves over to the next sps state which is move_to_park state.

2. sps_move_to_park

move_to_park state starts with setting the piston length based on the initially assumed PARK_DENSITY, volume compressibility, linear expansion coefficient and temperature. Currently volume compressibility (gamma) and linear expansion coefficient (alpha) are set to zero (0) for simplicity.
Once the piston length is achieved, pressure sensor starts sampling at MOVE_TO_PARK_SAMPLE_RATE and measures the depth. During this journey LCP moves towards the set PARK_DEPTH and adjusts the piston length according to the depth rate and stores the last updated length for the next profile. There are certain critical scenarios that are watched during this state namely CRUSH_DEPTH, PISTON_MOVEMENT_ON_BOTTOM, PISTON_POSITION_MINIMUM for 50 meter depth and so. The behavior of these scenarios can be analyzed and studied by looking into the sps_move_to_park state code. Once the PARK_DEPTH value is reached then move_to_park goes on to next sps_park state.

3. sps_park

sps_park state starts sampling the temperature sensor and pressure sensor with PARK_RATE_FAST in the first profile and with PARK_RATE in the next profiles. LCP will stay in the sps_park state for the PARK_TIME and the piston movement and adjustment happens only for keeping the LCP within PARK_DEPTH_ERR (deadband). For falling into the critical scenarios sps_park state acts as mentioned in the move_to_park state. During the set PARK_TIME, temperature and pressure values are averaged for at least 10 samples and are stored in the static memory. Once the sps_park state reaches to PARK_TIME, it exists and moves over to the next state which is sps_move_to_profile state.

4. sps_move_to_profile

move_to_profile state starts moving piston to the calculated length based on initially assumed TO_PROFILE_DENSITY. After the piston length is reached to this length, the move_to_profile state logic tries its best to take the LCP towards PROFILE_DEPTH by adjusting the piston length based on averaged depth rate. The move_to_profile state also takes care of the critical scenarioes as mentioned earlier.
move_to_profile passes on to the next state named sps_profile on hitting to the PROFILE_DEPTH successfully. The piston length gets updated during the length adjustments and used in the next profiles instead of assume density initially.

5. sps_profile

sps_profile state sets the piston length based on SYSTEM_RISE_RATE_SETPOINT to move, this piston length is calculated including the buoyancy and assume density of water. During this state, LCP starts collecting measurements of pressure and temperature sensor and averages over 10 samples assuming that LCP is moving upward towards surface. Piston length adjustment happens based on the averaged depth rate and adjusted piston length is used in the next profiles. Some of critical scenarios are also taken care during this state as mentioned in the previous states. sps_profile stops sampling the readings once it hits the BALLAST_DEPTH_PROFILE and passes over to the next state namely sps_move_to_surface.

6. sps_move_to_surface

move_to_surface state moves the piston up to PISTON_MOVE_TO_SURFACE length which makes sure that LCP is above the water surface. Once this piston length has reached, sps_move_to_surface also starts GPS with 1Hz for GPS_Timer and stores the latitude and longitude to the current profile number. Whether GPS gets fixed or not withint GPS_Timer move_to_surface goes on to the next state sps_tx.

7. sps_tx

sps_tx state initializes the iridium module and turns its power on. It starts with satellite visibility timer SATELLITE_TIMER, and within this timer period if iridium satellites are visible then it stops the timer and starts preparing measurements for park_mode. In case of SATELLITE_TIMER runs out and there is no satellite visibility, then it starts over up to SATELLITE_VISIBILITY_TRIES. There are number of tries which are set for transmitting the sbd messages defined in the “**config.h**” file that iridium tries to communicate with the satellite and transmits the sbd data.
Whether park_mode measurements are transmitted successfully or not, sps_tx moves on transmitting the profile_mode measurements as well with the set number of satellite visibility and iridium tries.

The functions or APIs which deal with creating iridium sbd messages with a header and payload are defined in the “**src/app/data.c**” file and detail description of iridium sbd message is also explained in the iridium data format section.

Once the sps_tx state finishes its task, it moves to the move_to_park state with previous adjust piston length and runs in a complete loop for set number of profiles SYSTEM_PROFILE_NUMBER.

4.6.4. Pop-Up Mode

Pop-Up mode or State has two states namely, pus_idle and pus_surface_float. Pop-up mode is currently functional when LCP activates the CRUSH_DEPTH flag and pus_surface_float state tries to extende piston to its full length alongwith checking the other critical scenarios and secondly when set number of profiles are reached then pus_idle state uninitializes all the sensors, turns them off and stays forever into the deep sleep while assuming to be floating on the surface.

The StateMachine can also be viewed in the graphical representation in the following figure.

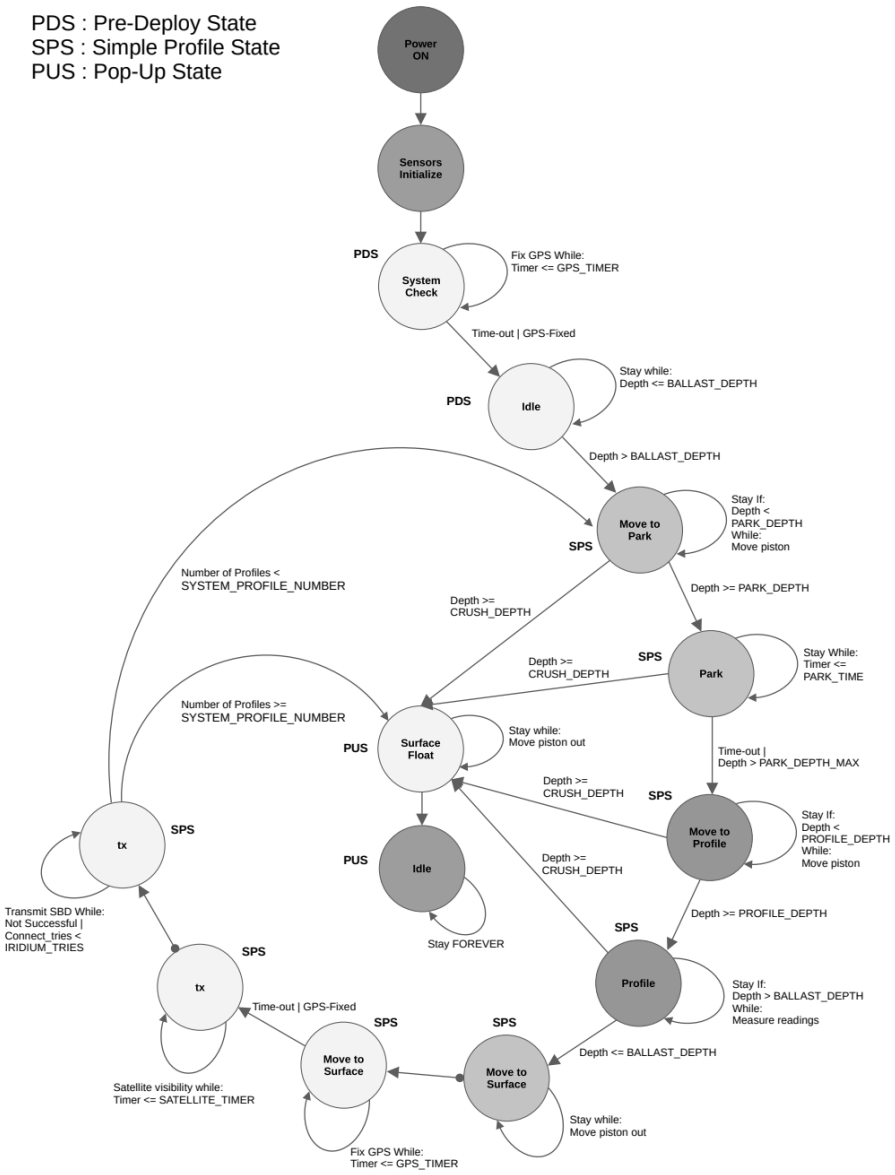


Figure 12: StateMachine graphical view

4.7. Config File

The Config file "**include/config.h**" has almost all the settable and non-settable constant values which LCP uses extensively during the StatMachine execution and it looks like the following.

```
#ifndef CONFIG_H
#define CONFIG_H

#include "artemis_debug.h"

#define PI                                  ( 3.14159265359 )
#define G_CONST                             ( 9.80665 )

#define SYS_SMALL_PISTON_DIAMETER           ( 2.25f * 1.0f ) //0.0254f )     /* in inches */
#define SYS_SMALL_PISTON_MAX_LENGTH         ( 6.0f  * 1.0f ) //0.0254f )
#define SYS_LARGE_PISTON_DIAMETER           ( 4.50f * 1.0f ) //0.0254f )
#define SYS_LARGE_PISTON_MAX_LENGTH         ( 6.0f  * 1.0f ) //0.0254f )
#define SYS_HOUSING_DIAMETER                ( 4.88f * 1.0f ) //0.0254f )
#define SYS_HOUSING_LENGTH                  ( 35.0f * 1.0f ) //0.0254f )

#define SMALL_PISTON_DIAMETER               ( SYS_SMALL_PISTON_DIAMETER )
#define SMALL_PISTON_RADIUS                 ( SYS_SMALL_PISTON_DIAMETER / 2.0f )
#define SMALL_PISTON_RADIUS_SQR             ( SMALL_PISTON_RADIUS * SMALL_PISTON_RADIUS )
#define SMALL_PISTON_MAX_LENGTH             ( SYS_SMALL_PISTON_MAX_LENGTH )
#define SMALL_PISTON_MAX_VOLUME             ( PI * SMALL_PISTON_RADIUS_SQR * SMALL_PISTON_MAX_LENGTH )

#define LARGE_PISTON_DIAMETER               ( SYS_LARGE_PISTON_DIAMETER )
#define LARGE_PISTON_RADIUS                 ( SYS_LARGE_PISTON_DIAMETER / 2.0f )
#define LARGE_PISTON_RADIUS_SQR             ( LARGE_PISTON_RADIUS * LARGE_PISTON_RADIUS )
#define LARGE_PISTON_MAX_LENGTH             ( SYS_LARGE_PISTON_MAX_LENGTH )
#define LARGE_PISTON_MAX_VOLUME             ( PI * LARGE_PISTON_RADIUS_SQR * LARGE_PISTON_MAX_LENGTH )

#define HOUSING_DIAMETER                    ( SYS_HOUSING_DIAMETER )
#define HOUSING_RADIUS                      ( SYS_HOUSING_DIAMETER / 2.0f )
#define HOUSING_RADIUS_SRQ                  ( HOUSING_RADIUS * HOUSING_RADIUS )
#define HOUSING_LENGTH                      ( SYS_HOUSING_LENGTH )
#define HOUSING_VOLUME                      ( PI * HOUSING_RADIUS_SRQ * HOUSING_LENGTH )

#define SYSTEM_MAX_VOLUME                   ( HOUSING_VOLUME + SMALL_PISTON_MAX_VOLUME +
                                              LARGE_PISTON_MAX_VOLUME + 0.01f )
#define SYSTEM_MIN_VOLUME                   ( HOUSING_VOLUME - 0.01f )
#define SYSTEM_MIN_LENGTH                   ( 0.0f )
#define SYSTEM_MAX_LENGTH                   ( SMALL_PISTON_MAX_LENGTH + LARGE_PISTON_MAX_LENGTH )

#define DRAG_PLATE_DIAMETER                 ( 8.0f * 1.0 ) //0.0254f )
#define DRAG_PLATE_RADIUS                   ( DRAG_PLATE_DIAMETER / 2.0f )
#define DRAG_PLATE_RADIUS_SRQ               ( DRAG_PLATE_RADIUS * DRAG_PLATE_RADIUS )
#define DRAG_PLATE_AREA                     ( PI * DRAG_PLATE_RADIUS_SRQ )

//#define SYSTEM_MASS_EST                   ( 24.17f * 1.0 ) //0.453592 )   /* in kg */
#define SYSTEM_MASS_EST                     ( 24.557f  * 1.0 ) //0.453592 )   /* in lbs */
#define SYSTEM_CROSSSECTION_AREA            ( DRAG_PLATE_AREA )

#define CYLINDER_DRAG_COEFF                 ( 0.81f )

#define SYSTEM_VOLUME_MIN                   ( HOUSING_VOLUME )
#define SYSTEM_VOLUME_MAX                   ( HOUSING_VOLUME + SMALL_PISTON_MAX_VOLUME )
#define SYSTEM_VOLUME_RESERVE               ( LARGE_PISTON_MAX_VOLUME )
.....
.....
.....
```

4.8. Test Profiles

There are different test profiles are defined in the “artemis_debug.h” and “config.h” files and at least one profile has to be selected in order to execute the StatMachine.

“**include/artemis/artemis_debug.h**”

```
/** defined test profiles, tank, lake etc, uncomment one at a time */
//#define __TEST_PROFILE_1__
//#define __TEST_PROFILE_2__
//#define __TEST_TANK__
//#define __TEST_LAKE__
//#define __TEST_OCEAN__
#define __TEST_PS__
```

By uncommenting one profile will automatically select the same test profile in the “config.h” file

“**include/config.h**”

```
....
....

#elif defined(__TEST_PS__)
/** Puget Sound Testing */
#define BALLAST_DEPTH                   ( 1.0f )
#define BALLAST_DEPTH_SAMPLE_RATE       ( 1.0f )
#define BALLAST_DEPTH_PROFILE           ( 1.0f )
#define MOVE_TO_PARK_SAMPLE_RATE        ( 0.10f )
#define PARK_DEPTH                      ( 150.0f ) //( 4.0f )
#define PARK_DEPTH_ERR                  ( 15.0f )
#define PARK_DEPTH_MAX                  ( 200.0f )
#define PARK_RATE                       ( SYSTEM_PROFILER_PARK_RATE )
#define PARK_RATE_FAST                  ( SYSTEM_PROFILER_PARK_RATE_FAST )
#define PARK_TIME_FIRST                 ( 60.0f * 60.0f )    /* 1 hr */
#define PARK_TIME                       ( 180.0f * 60.0f )    /* 3 hrs */
#define PARK_DENSITY                    ( 1023.8f )
#define MOVE_TO_PROFILE_SAMPLE_RATE     ( 0.10f )
#define PROFILE_DEPTH                   ( 165.0f ) //( 4.5f )
#define PROFILE_DEPTH_ERR               ( 1.0f )
#define PROFILE_RATE                    ( SYSTEM_PROFILER_PROFILE_RATE )
#define TO_PROFILE_DENSITY              ( 1023.9f )
#define PROFILE_DENSITY                 ( 1022.0f )
#define CRUSH_DEPTH                     ( 210.0f )
#define PARK_POSITION_INCREMENT         ( PISTON_POSITION_INCREMENT )
#define PARK_POSITION_INCREMENT2        ( PISTON_POSITION_INCREMENT2 )
#define PISTON_MOVEMENT_ON_BOTTOM       ( 0.70f )    /* 0.70 inch piston length if we hit the bottom */
....
....
....

#else
   #error "<<< ERROR:: Please select at least one TEST Profile in the artemis_debug.h !!! >>>"
#endif

#endif // CONFIG_H
```

4.9. Lowest Power mode

By commenting the following defined (LCP_FREE_RTOS) line as well as **artemis_debug_initialize()** function in the main.c file, will put the LCP (control-board) in the lowest possible power mode which is far less than 1mA in current consumption.

"**main.c**"

```
#include "main.h"
#include "sensors.h"
#include "StateMachine.h"

//#define LCP_FREE_RTOS
//#define SENSORS_Test
```

and

```
/** initialize mcu features */
artemis_mcu_initialize();

/** initialize debug features */
//artemis_debug_initialize();
```

after commenting above (bold) lines, LCP can be flashed by compiling the firmware as mentioned in the firmware flash section.

Note: Please do not forget to uncomment above two (bold) lines if intention is to execute the LCP with normal operation.

5. Iridium Data Format

Iridium SBD i9603n allows up to maximum size of 340 bytes in originated message and up to maximum size of 270 bytes in terminated message. The message must be in the binary form.

The SBD message must be in the following format:

Table 5: LCP Iridium SBD Data Length

Message Type	Maximum Length	Checksum	Bytes containing length	Total
Originated Message (Out)	340 Bytes	2 Bytes	0 Bytes	340 + 2
Terminated Message (In)	270 bytes	2 Bytes	2 Bytes	2 + 270 + 2

Originated message : {binary SBD message} + {2-bytes checksum}

Terminated message: {2 bytes message length} + {binary SBD message} + {2-bytes checksum}

5.1. Message Breakdown

5.1.1. Message Header

Introduced in the IoTAS project, the PMEL SBD message header will be used to help identify where the data message is coming from.

The header is a 28 bytes message, which is formatted as follows.

Table 6: LCP Iridium SBD Data Format

Primary Header:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
SystemID	Firmware Version		Year and Date		Latitude		
Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13		
Latitude	Longitude				LCP Variant		

* LCP Variant will detail the type of sensors in the transmission.
e.g temperature + pressure, temperature + pressure + conductivity

Serial Number:
The LCP serial number is transmitted in the following 3 bytes: serial number is in alphanumeric.

Byte 14	Byte 15	Byte 16
Serial Number		

Profile Number and Length of measurements:
The bytes 17 and 18 represent the profile number and total length of measurements for one profile.

Byte 17	Byte 18
Profile Nr.	Length of measurements

Mode and Page Number:
The mode and page number are to differentiate the profile mode (Park/Profile) and iteration of measurement that is being transmitted.

Byte 19	
Mode	Page Nr.

TimeStamp:
Timestamp bytes contain the start and end time (unix epoch time) for profiling.

Byte 20	Byte21	Byte22	Byte23	Byte 24	Byte 25	Byte 26	Byte 27
Start Time				End Time			

Payload:
The LCP messages use a payload of measurements into 312 (bytes). Currently payload contains measurements data of pressure and temperature sensor with its up to defined resolution.

Byte28	....	Byte340
Measurements		

Temperature sensor resolution : 12 bits (-5 to 35 °C) , precision: 0.01
Pressure sensor resolution : 8 bits (0 to 20 bar) , precision: 0.1
Conductivity sensor resolution : 12 bits (25 to 65 mS/cm) , precision: 0.01
DO sensor resolution : 12 bits (?) , precision: 0.01

The message will be packed in binary form with the specified bit resolution of measurements with no extra checksum.

* SBD allows 2-bytes summation of all bytes as a checksum in addition to 340 bytes.

5.1.2. Data Format Description:

1. System ID:
System ID is single byte variable which differentiates the LCP Profiler among other devices at NOAA. System ID will be fixed once and will be used in all other LCP variants.

* this still needs to be discussed with Noah
Dummy ID = 0x77

2. Firmware Version:
Firmware Version consists of 2 bytes comprising with Firmware Major, Minor and Patch numbers.

Development Release = 0.0.0 (use this until the first stable release)
Initial Release = 0.0.1 (First Official stable release)

4 bits for Major, 4 bits for Minor and 8 bits for patch
maximum number = 15.15.255

every 256th patch will increase the minor, and every 16th minor number will increase the Major number.

*** Note** : 15x15x255 = 57375 changes are allowed after the first stable release.

3. Year and Date:
Two bytes are reserved for year (XX), month (XX) and date (XX)

Total 16 bits

Example:
Year = 99 (7 bits)
Month = 12 (4 bits)
Date = 31 (5 bits)

4. Latitude:
There are four bytes for latitude with up to 5 digits precision (+- 1.11m)
(+- 90°)

5. Longitude:
There are four bytes for latitude with up to 5 digits precision (+- 1.11m)
(+- 180°)
6. LCP Variant:
LCP Variant has a one byte to differentiate among different variants of LCP Profiles based on type and number of sensors.

Value = 0x00 (for first release , temperature and pressure sensors)

7. Serial Number:
Serial number is an alphanumeric value. There are three bytes allotted for a serial number. The first byte will always be letter “L” which represents LCP, there other two bytes can be numbered accordingly.

e.g = L01 (in alphanumeric) , 0x4C0001 (in hex)

8. Type of Measurement:
One byte consists the information for which type/mode of measurements it is.

Value = 0x00 (Park mode)
Value = 0x01 (Profile mode)
Value = 0xXX (etc)

9. Page Number:
One byte page number consists the information for number of measurements belongs to the same type/mode of measurements.

Maximum = 255 , but it can be increased by using the bits from “type of measurement”.

10. Start Time:
Four bytes are for start epoch time for any mode of measurements, e.g Profile or Park mode. The epoch-time is the number of seconds that have elapsed since Jan 1, 1970. The time is converted into seconds and subtract from Jan, 1 1970. Four bytes of unsigned 32_int are enough to store the time stamps.

11. End Time:
End time is also of four bytes to store the time-stamp when the measurements end.

12. Payload:
Payload consists the actual measurements of sensors reading. 313 bytes are allocated for sensors reading which breakdown at the level of bit resolution.

Example :

Temperature sensor resolution : 12 bits (-5 to 35 °C) , precision: 0.01
Pressure sensor resolution : 8 bits (0 to 20 bar) , precision: 0.1
Conductivity sensor resolution : 12 bits (25 to 65 mS/cm) , precision: 0.01
DO sensor resolution : 12 bits (?) , precision: 0.01

Pressure + Temperature sensor:
total bits = 12 + 8 = 20 bits per measurement.
312 bytes = 125 measurements

Pressure + Temperature + Conductivity sensor:
total bits = 12 + 8 + 12 = 32 bits per measurement.
312 bytes = 78 measurements

Pressure + Temperature + Conductivity + DO:
total bits = 12 + 8 + 12 + 12 = 44 bits per measurement.
312 bytes = 56 measurements

adding more sensors would reduce the number of measurements in a single Iridium SBD message.

5.2. Message Decoder

Once the LCP is on the surface after collecting park mode and profile mode measurements and able to transmit successfully in its binary form, then on the receiving end these sbd messages need to be decoded and extracted the information which were encoded during profiling.

The decoder binary file and its source files can be found in the following path.

```
# cd EDD-LCP_Control/LCP_sbd_decoder
# tree

├── lcpdecoder
├── lcpdecoder.c
└── README.txt

0 directories, 3 files
```

OR

```
# ls -lh
```

Compile lcpdecoder.c source and copy the lcpdecoder (binary file) to the path where the sbd messages are residing.

```
# gcc lcpdecoder.c -o lcpdecoder
```

In order to decode either a single or several sbd messages, decoder must be in the same directory-path, takes as an argument (./), extracts information and separates the park mode and profile mode measurements in the directories with (park and profile) accordingly. The decoded message would be named exactly the same as sbd message (input) but with the .txt extension.

For example :

```
# cp lcpdecoder sbd_messages_06_05_2024/
# cd sbd_messages_06_05_2024/
# tree

...
...
├── 300434064658730_000963.sbd
├── 300434064658730_000964.sbd
├── 300434064658730_000965.sbd
└── lcpdecoder

0 directories, 201 files
```

Decode the messages

./lcpdecoder ./

...
...
LCP :: sbdfile = 300434064658730_000826.sbd
LCP :: txtfile = 300434064658730_000826.txt

LCP :: sbdfile = 300434064658730_000877.sbd
LCP :: txtfile = 300434064658730_000877.txt

tree
...
...
├── 300434064658730_000964.sbd
├── 300434064658730_000965.sbd
├── lcpdecoder
├── park
│   ├── 300434064658730_000766.txt
│   ├── 300434064658730_000768.txt
...
...
│   ├── 300434064658730_000959.txt
│   ├── 300434064658730_000962.txt
│   └── 300434064658730_000964.txt
└── profile
    ├── 300434064658730_000767.txt
    ├── 300434064658730_000769.txt
    ├── 300434064658730_000771.txt
...
...

The output of the decoded message should look like as follow:

cat park/300434064658730_000766.txt

```
        LCP Profiler Information
========================================
ID                  :  77
Firmware            :  0.0.0-dev
Firmware Date       :  6.3.24
Latitude            :  47.6864480
Longitude           :  -122.2544640
LCP Variant         :  0
LCP Serial          :  L00
Profile Nr          :  0
Measurements        :  19
Mode                :  0, Park Mode
Page Nr             :  0
Start Time          :  1717521464
Stop Time           :  1717521640

Sr.No.        Pressure(bar)      Temperature(°C)
========================================
1                   1.800        19.140
2                   1.800        19.140
3                   1.900        19.140
4                   2.000        19.140
5                   2.000        19.160
6                   2.000        19.170
7                   2.000        19.170
8                   2.000        19.190
9                   2.000        19.200
10                  1.900        19.220
11                  1.900        19.230
12                  1.900        19.240
13                  1.900        19.260
14                  1.900        19.280
15                  1.900        19.290
16                  2.000        19.310
17                  2.000        19.310
18                  2.000        19.320
19                  2.100        19.340

total measurements=19
```

6. Changelog

1. 06/19/2024 – Basharat Martin
    - Created initially