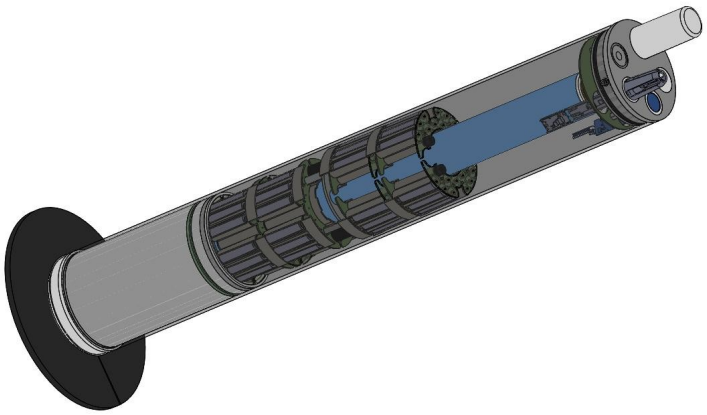
# Low Cost Profiler (LCP)



# User Manual
## Piston-board

Author : Basharat Martin
Basharat.Martin@noaa.gov
dated : 07/01/2024

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

Piston-board is a round shaped pcb that serves the Low Cost Profiler (LCP) a backbone for moving the linear actuator. It stays in the lowest power mode possible and only wakes up when an I2C command comes from the control-board to either inquire the information or move the actuator. There is a 12-inches stroke-length linear actuator attached and controlled by the piston-board.

# 2. Hardware

The PCB of piston-board is designed of almost round shape with dimensions approximately 10.8cm diameter and 3cm in height approximately (considering a bulk capacitor being soldered). The MSP430FR5989 (16-bit RISC architecture up to 16-MHz main clock) is the main chip from TI on the piston-board. It is meant to be battery-powered and provides the power to the whole LCP including the control-board. The piston-board has a fuel gauge chip MAX17205 from maximintegrated which monitors and keeps track of battery power consumption but currently this chip is non-functional. MCU is set to be running on 8MHz main clock instead of 16MHz.

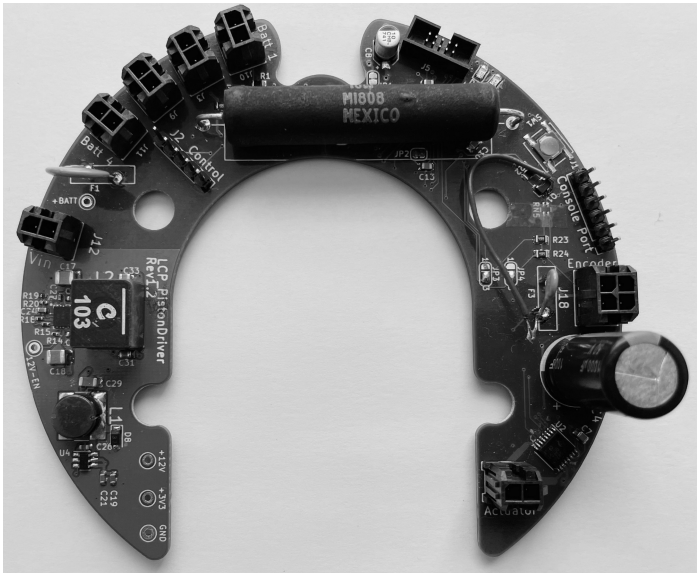



*Figure 1: Piston-board PCBA*

## 2.1. Power Specification

Piston-board has two DC-DC buck converters BD9G102G-LB (ROHM semiconductors) input voltage range from 6V to 42V which provides the 3.3V to MCU and LM61460 (TI) input voltage range from 3V to 36V which proves 12V input to the motor driver (DRV8874 H-Bridge). So the input voltage range can be considered as from 6V to 36V and maximum output current from the H-Bridge for the actuator is up to 6A.

## 2.2. LEDs

There are two LEDs (Green and Blue) for visual debugging purpose. These leds are mostly kept turned off for power saving purpose.

## 2.3. Pinout

There are couple of peripherals are being used on the piston-board from TI MCU internally and externally and these are listed below.

*Table 1: Pinout*

| Interface | Description |
|---|---|
| UART0 – (UCA0RXD/TXD) | J1 connector is used for Debug outputs namely CONSOLE_RX and CONSOLE_TX lines. |
| I2C_0 – (UCB0SDA/SCL) | I2C_0 lines appeared as UCB0SDA and UCB0SCL in the schematic and attached to the MAX17205 (fuel gauge) chip which is currently non-functional. |
| I2C_1 – (USB1SDA/SCL) | I2C_1 lines appeared as UCB1SDA and UCB1SCL and provide an interface externally to the control-board. |
| I2C_4 – (SCL4/SCL4) | Openlog Datalogger (sparkfun) is connected on this interface. |
| 12V_Enable (P2.5) | This pin enables the 12V DC-DC buck converter |
| H-Bridge Pins<br>MD_EN (P2.4)<br>MD_PH (P2.6)<br>MD_SLEEP (P4.3)<br>MD_FAULT (P5.0)<br>MD_IPROPI (P1.2)<br>MD_PMODE (P5.1) | These IO pins are attached to the H-Bridge DRV8874 motor driver for the linear actuator. Pin (P2.4) is a PWM, Pin (P1.2) is an ADC and rest of them are general purpose IO. |
| ENCODER_CH_A (P1.3) | Pin (P1.3) serves as an interrupt driven input to the encoder_A (Hall Sensor) of the actuator. |
| ENCODER_CH_B (P1.4) | Pin (P1.4) serves as an interrupt driven input to the encoder_B (Hall Sensor) of the actuator. |

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

## 2.4. Peripherals

The linear actuator from Firgelli Automation is a bullet series 50 cal. with 12 inches stroke length and have built-in hall effect sensors which measure the rotation of the motor. The detail technical specification can be found on this link below.
https://www.firgelliauto.com/products/bullet-series-50-cal-linear-actuators?variant=16273291771975

*Figure 2: Linear Actuator*

### 2.4.1. Peripheral Pinout

| | | |
|---|---|---|
| Black (a) | → | ACTUATOR_POWER- |
| Red (b) | → | ACTUATOR_POWER+ |
| Yellow (c) | → | HALL_EFFECT_V+ |
| White (d) | → | HALL_EFFECT_V- |
| Brown (Ch.A) | → | HALL_EFFECT_CH_A |
| Green (Ch.B) | → | HALL_EFFECT_CH_B |

# 3. Schematic Design Changes

The schematic of LCP piston-board initially was designed on KiCAD version 5. It still needs to be transformed to the latest version of KiCAD. Current piston-board pcb version is Rev1.2.

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

# git clone https://github.com/NOAA-PMEL/EDD-LCP_PistonDriver.git
# cd EDD-LCP_PistonDriver
# git check master_gcc
# cd Electronics/LCP_PowerControl

The above path keeps the schematic design files which can be opened on KiCAD-5.

## 3.1. I2C Pull-up Resistors

RN2 R_Pack02 resistors need to be removed for the I2C_1 bus, since there have been already 10K resistors for the I2C lines on the master (control-board) side.

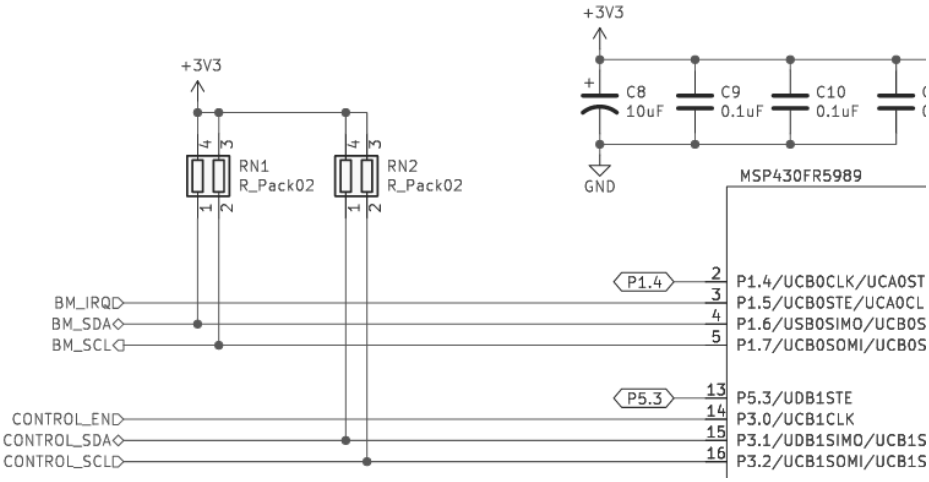



*Figure 3: I2C_1 pull-up resistors*

## 3.2. Power save on Encoders

Currently encoders (ENCODER_CH_A and ENCODER_CH_B) are connected to the MCU IO interrupt driven pins through (RN5 – 4.7K) pull-up resistors and controlled by the switch (Q1).

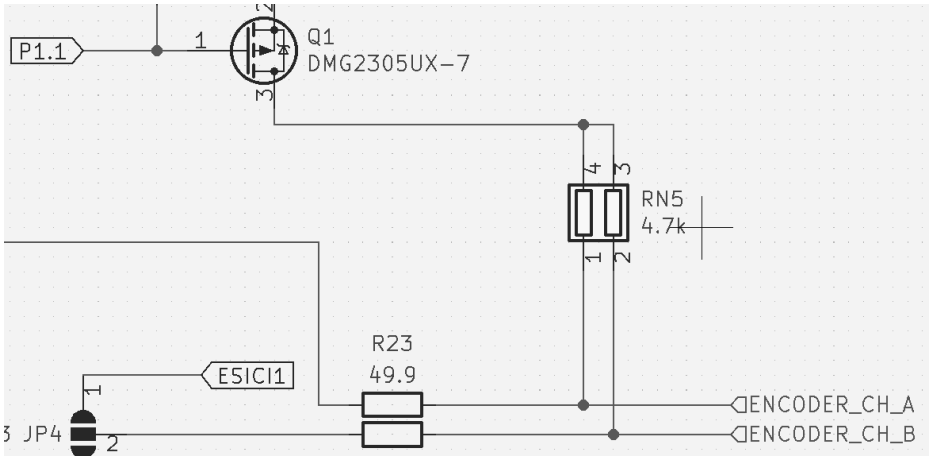



*Figure 4: Encoder pull-up resistors*

**Note:**
These resistors (RN5) need to be removed and pin number 3 of the switch (Q1) must be connected to the fuse (F3). The power line (+3V3) of the Fuse (F3) needs to be routed to the switch (Q1) pin number 3.

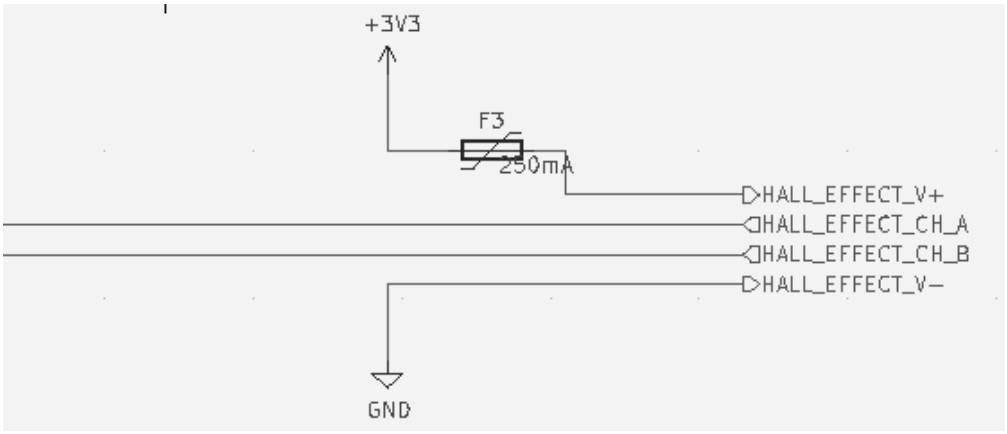



*Figure 5: Encoder power through Fuse (F3)*

The above configuration or design change will control the encoder's power through the switch (Q1) and this can save the power when piston-board is inactive and not moving the actuator.

# 4. Software

Software for the piston-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_PistonDriver.git
# cd EDD-LCP_PistonDriver
# 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.

```
msp430-gcc.tar.xz found
Extracting msp430-gcc.tar.xz file ...
Copying libmsp430.so file to /usr/local/lib ...
...
...
...
...
Checking if a user in the dialout group ...
User in already in the dialout group ...
completed

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

# source ~/.bashrc
# cd Firmware/LCP_PistonDriver
# make

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

```
b/tlv.o driverlib/uups.o driverlib/wdt_a.o -o LCPPistonDriver.elf

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

## 4.2. Firmware Version

Current firmware version is v0.1.0 and it is in its development phase due to LCP is under test, once the firmware release is made by a developer it can be set accordingly.

## 4.3. Firmware Flash

In order to flash the firmware to the piston-board, a MSP-FET Flash Emulation Tool from TI is required as shown in the following picture.

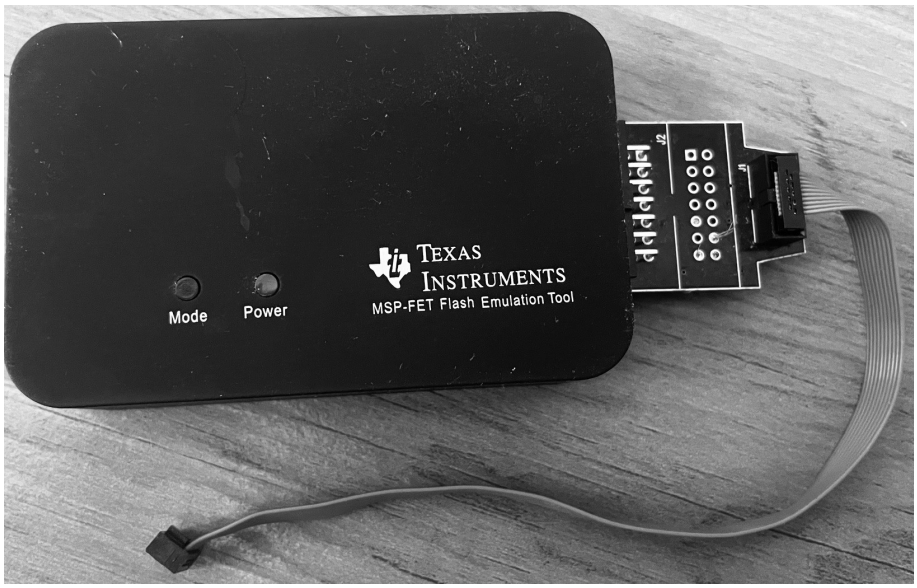



*Figure 6: MSP-FET Emulation Tool*

There is a 10 pins female connector at the end of this emulation tool that connects to the 10 pins male header on the piston-board (J5).

# make flash

The above command will flash the firmware to the piston-board and output should look like the following.

```
Writing 4096 bytes at 10000 [section: .upper.rodata]...
Writing 1885 bytes at 11000 [section: .upper.rodata]...
Done, 62666 bytes total
MSP430_Run
MSP430_Close
```

## 4.4. Console Output

Using a USB to TTL serial adapter as shown in the figure below can be used to see the DEBUG outputs from the piston-board.

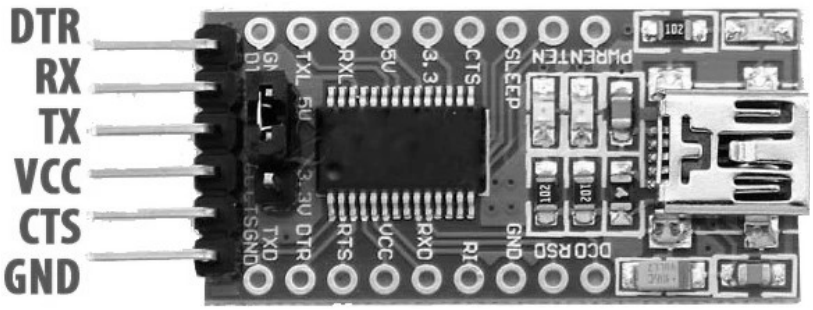



*Figure 7: USB to TTL serial adapter*

There are only **three** pins required from this adapter required to attach to the piston-board console connector (J1). The pins connection must be followed as below.

**RX → CONSOLE_TX (Pin 4)**
**TX → CONSOLE_RX (Pin 5)**
**GND → GND (Pin 1)**

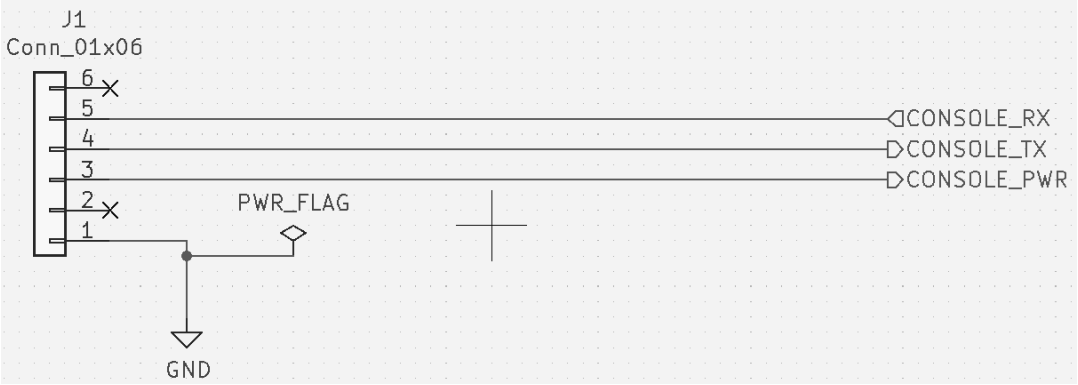



*Figure 8: Piston-board Debug Console*

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

piston-board UART0 is used for listening on DEBUG outputs. For DEBUG outputs, UART0 is configure to run at 115200 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 115200

Piston-board has a very convenient utility (shell prompt/command-line) developed by Matthew Casari. By using this utility, piston-board can take shell commands for moving the actuator , calibrating the actuator and other useful commands. So the output from either minicom or putty must look like the following

shell> **type help command, press Enter and see the output below.**

```
shell>help
get: Get value of register/variable
set: Set value of register/variable
: ***System Information***
ser: Get Serial Number
id: Return system id
ver: Return firmware version
report: Return System Report
dump: Control Memory Dump
: *** Other ***
debug: Set Debug state
rev: Retract piston
fwd: Extend piston
stop: Stop piston
LED: Set LED State
hello: Say hello
help: Lists all commands
shell>
```

Only known commands are accepted and run and output the useful information. These commands are explained in the shell command section.

**Note:** sometimes, minicom does not allow to take input from the keyboard, in that case use “putty” with LF and CR enabled.

## 4.5. Shell Commands

Although by typing help does not show or list all the known commands at the output yet, but these can be found in the header and source files of shell_commands.h and shell_commands.c.

There are a few very useful commands which often needed in the course of setting up the actuator for LCP.

### 4.5.1. User admin

There are certain commands that require an admin user and can be set as follows:

```
shell>set username admin
DEBUG: set user called
DEBUG: value= admin
> OK
shell>
```

### 4.5.2. Calibration

First time use of actuator with the piston-board must need a calibration which counts the rotations through hall-effect sensors in the actuator and sets encoder values in the memory for better length precision.

```
shell>set cal 1
```

The above command will start moving the actuator in the reverse direction, once completed it will start in the forward direction and starts counting the rotation via encoders. By achieving the maximum length forward it sets the maximum encoder values and starts moving in the reverse direction and sets the minimum encoder value to 0 and finishes the calibration. To see the maximum and minimum encode values, type the following command

```
shell> get maxencode
DEBUG: get maxencode called
max encode= 141841, min encode= 0
> OK
shell>
```

To see the current encoder value

```
shell>get encode
DEBUG: get encode called
encode= 0
> OK
shell>
```

### 4.5.3. Forward direction

The moving the actuator in the forward direction can be achieved by the following command.

```
shell>fwd
DEBUG: fwd command
DEBUG: PIS_Extend called
DEBUG: ENC_SetDir
DEBUG: Enabling DRV8874
DEBUG: DRV8874_reverse entered
DEBUG: DRV8874_reverse After sleep 10ms
DEBUG: reverse pwm 1,0 out
DEBUG: Setting DRV8874 REVERSED at 100%, pwm = 1024
shell>
```

**Note:** Please be noted, that using this command “fwd”, the actuator does not count on the encoder values but rather by using the 12V and maximum allowed current, it will keep moving forward until the mechanical switch in the actuator stops it. So basically “fwd” command will fully extend the piston all the way out, unless it is stopped by the “stop” command.

```
shell>stop
DEBUG: stop command
DEBUG: PIS_Stop called
DEBUG: DRV8874_stop After sleep 10ms cycles
DEBUG: DRV8874 Stopped
DEBUG: Disabling DRV8874
shell>
```

For controlled piston length movement the following command can be used, either by moving the length of LCP calculated volume.

```
shell>set lset 1.0
DEBUG: set lset called
DEBUG: value= 1.0
DEBUG: PIS_Run_to_length called with length = 1.000000
DEBUG: PIS_Extend called
```

lset stands for (length set) in inches. And the same can be set for volume vset (volume set).

```
shell>set vset 670.0
```

**Note:** make sure that vset does not exceed the minimum and maximum volume values.

### 4.5.4. Reverse direction

Moving the actuator in the reverse direction by the command “rev” would keep running in backward until the mechanical switch stops it on fully retracted or it can also be stopped by using a “stop” command explicitly.

```
shell>rev
DEBUG: rev command
DEBUG: PIS_Retract called
DEBUG: Stopping movement before Retracting
DEBUG: PIS_Stop called
DEBUG: DRV8874_stop After sleep 10ms cycles
DEBUG: DRV8874 Stopped
DEBUG: Disabling DRV8874
DEBUG: ENC_SetDir
DEBUG: Enabling DRV8874
DEBUG: DRV8874_forward entered
DEBUG: DRV8874_forward After sleep 10ms cycles
DEBUG: Setting DRV8874 FORWARD at 100%, pwm = 1024
shell>

shell>stop
DEBUG: stop command
DEBUG: PIS_Stop called
DEBUG: DRV8874_stop After sleep 10ms cycles
DEBUG: DRV8874 Stopped
DEBUG: Disabling DRV8874

shell>
```

Moving the actuator in a controlled length position, use the following same command

`shell>set lset value` (value must be in inches and within actuator’s length limit)

### 4.5.5. get commands

Get commands are useful to get the piston-board and actuator information including the memory map. set and get commands defined in the shell_commands.c file.

```
shell> get encode
```

```
shell> get maxencode
```

```
shell> get lset
```

```
shell> get vset
```

```
shell> get encode
```

“report” command will output the piston-board information

```
shell>report
********* SYSTEM REPORT *********
System ID              : LCPPST01
Serial Number          :
Firmware Version       : v0.1
Year Built             : 2023
Log Level              : 1
Encoder Min            : 0
Encoder Max            : 141841
shell>
```

“dump” command will output the memory map, explained in the memory map section.

```
shell>dump
 f4 a6 24 44 00 00 00 00 7c a8 23 44 7c a8 23 44
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 80 3f 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 40 41 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 ff 01 00 00 00 00 00 01 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 4c 43 50 50 53 54 30 31
 00 00 00 00 00 00 00 00 e7 07 00 01 00 00 00 00
shell>
```

## 4.6. Memory Map

Piston-board has a 256-bit memory map starting address from 0x00 to 0xFF defined in the memory.h and memory.c files. This memory map retains many constants and variables values persistently. These values are retained even on power-cycles the piston-board. Only flashing a new firmware would reset and replace these values or setting these values explicitly by a user (either with shell or a control-board commands).

**User Settable System Variables**

0x00 to 0x07
0xF0 to 0xF7

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0x00 | VOL_SETPOINT_IN3 | | | | | | | VAR_WRITE |
| 0x10 | | | | | | | | |
| 0x20 | LEN_PISTON_IN | | | | | | | |
| 0x30 | | | | | | | | |
| 0x40 | PST_POSITION_MIN | | | | PST_POSITION_MAX | | | |
| 0x50 | PST_ENC_COUNTS | | | | PST_ENC_COUNTS_MAX | | | |
| 0x60 | TRV_DIR | TRV_ENG | | USER_OVERRIDE | MOV_ZERO | MOV_FULL | | PST_CAL |
| 0x70 | | | | | | | | SYS_RST |
| 0x80 | PID_COEFF_P | | | | PID_COEFF_I | | | |
| 0x90 | PID_COEFF_D | | | | PID_USED | | | |
| 0xA0 | | | | | | | | |
| 0xB0 | | | | | | | | |
| 0xC0 | | | | | | | | |
| 0xD0 | | | | | | | | |
| 0xE0 | | | | | | | | |
| 0xF0 | SER_NUM | | | | | | | |
| | | | | | | | | |
| | User Settable System Variables | | | | | | | |

*Figure 9: Settable Memory-Map*

The above memory-map can be set explicitly either by a user manually on the shell> by giving defined commands or the control-board via I2C commands

**Note:**
VAR_WRITE (0x07) needs to be set to 0xA5 in order to write to any settable system variables

**Read-Only System Variables/Constants**

0x08 to 0x0F
0xF8 to 0xFF

| | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|
| **0x00** | VOL_TOTAL_IN3 | | | | VOL_HOUSING_IN3 | | | |
| **0x10** | VOL_SMALL_PISTON_IN3 | | | | VOL_LARGE_PISTON_IN3 | | | |
| **0x20** | LEN_TOTAL_IN | | | | | | | |
| **0x30** | LEN_SMALL_PISTON_IN | | | | LEN_LARGE_PISTON_IN | | | |
| **0x40** | AREA_SMALL_PISTON_IN2 | | | | AREA_LARGE_PISTON_IN2 | | | |
| **0x50** | PST_RATE | | | | PST_POSITION_IN | | | |
| **0x60** | TRV_ZERO | TRV_FULL | TRV_MIN | TRV_MAX | | | | |
| **0x70** | | | | | | | | |
| **0x80** | BAT_RETCAP | | | | | | | |
| **0x90** | BAT_REPSOC | | | | | | | |
| **0xA0** | BAT_VCELL | | | | | | | |
| **0xB0** | BAT_CURRENT | | | | | | | |
| **0xC0** | BAT_TTE | | | | | | | |
| **0xD0** | BAT_STATUS | | | | | | | |
| **0xE0** | SYS_ID | | | | | | | |
| **0xF0** | YEAR_BUILT | | FIRM_MAJ | FIRM_MIN | FIRM_BUILD | | | |
| | | | | | | | | |
| | Read-only System Variables/Constants | | | | | | | |

*Figure 10: Read-only Memory-Map*

The above part of the memory-map is a read-only which can be set upon flashing a new firmware.

## 4.7. PID Controller

In order to have a precision control on the actuator length, PID controller can be implemented to control the actuator length via feedback-loop. Currently, a PID controller is not implemented in the firmware to achieve a precise actuator length but there are a few suggestions below for implementing one.

PID controller is a classic control theory mechanism to achieve a precise set-point in a robust manner based on the PID namely (Proportional, Integral and Derivative) parameters. These can be tuned either manually or Ziegler-Nicholas method.

The output of the PID controller is the PWM which must set its maximum limits from 100 to -100 in case of integral windup. Following is the equation for PID Controller.

$$u(t) = K_p\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{d}{dt}e(t)\right)$$

*Figure 11: PID Controller Equation*

Where e(t) is the error between the set-point and current-point in time(t)

The above equation can be implemented in its digital form with sample time at least 1ms.

The control actuator length is set by the command “set lset value” which uses encoder values and runs the actuator length towards the value. In the following file and its function, PID controller can be implemented

**file → “src/piston.c”**
**function → PIS_Run_to_length()**

# 5. Changelog

1. 07/01/2024 – Basharat Martin
    - Created initially