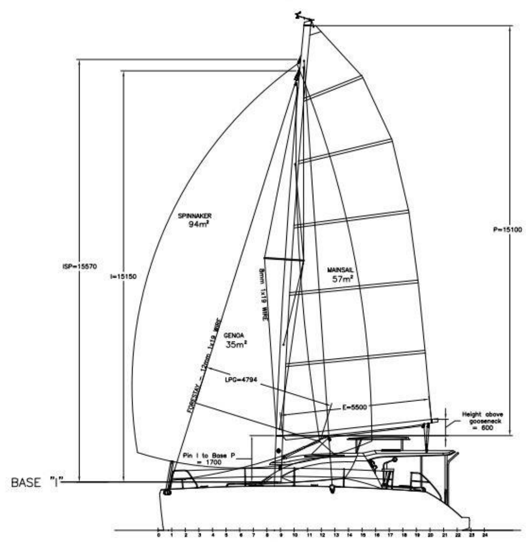
Catamaran Electronics



Specification

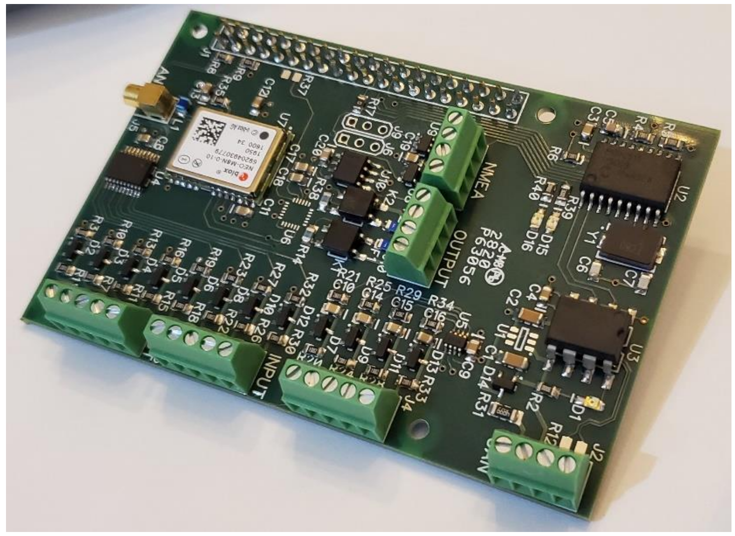
Boat IO

(Rev A)

Contents

1 Introduction

The Boat IO or Boat Input/Output, is a single circuit board that can be plugged into a Raspberry Pi. The Boat IO design is centered on monitoring the electronics and electrical systems on a modern boat. Many features have been embedded into the design to allow as much flexibility as possible:



- GPS module
- CAN Bus/NMEA 2000 interface
- Eight Digital Inputs
- Two Digital Outputs (0.5A)
- Four Analog Inputs
- NMEA0183 Interface
- Two programmable LEDs
- Real Time Clock

Each of the interfaces can be programmed using Python or other similar languages on the Pi.

2 Power

Power for the Boat IO is derived from the PI itself

3 Interfaces

BOAT IO has numerous interfaces to make it as flexible as possible. The applications for BOAT IO are endless and limited only by the amount of code the user is willing to write.

3.1 GPS

Embedded on the card is a very sophisticated uBlox NEO-M8N GPS module. This can be programmed to provide a GPS position based on not only the USA GPS system, but also the French, the Chinese and the Russian system. Four simultaneous receivers in one allow more reliable GPS position information regardless of potential jamming or interference. **Note** the GPS can be configured to use either the Pi serial port or the I2C interface. If the serial interface is used for GPS, the NMEA 0813 is not available. The configuration is selected with jumpers J9 and J10.

Refer to the data sheet for the uBlox device https://www.u-blox.com/en/product/neo-m8-series

If gpsd and NTP are installed on the Pi, the GPS is managed by the Linux system. The GPS can be tested using GPS tools like cgps.

3.2 CAN/NMEA 2000

Boat IO contains a single channel CAN interface that is compatible with the common marine buses for example NMEA 2000. Common applications for the CAN bus, could include using the Boat IO to read tank sensors on the analog interface and provide that information on the NMEA 2000 bus for display on the chart plotter.

Boat IO card uses the Pi standard MCP2515 CAN device and therefore has support within the Pi Linux distribution. Using the CAN bus can be done at a script level or programmed in Python. There are many examples of doing this on the web and I recommend looking at sites like:

https://copperhilltech.com/pican2-controller-area-network-can-interface-for-raspberry-pi

3.3 Analog and Digital Input/Outputs

To be able to monitor conditions on the boat, like the bilge pumps, the battery voltages. Boat IO has several input and outputs that can be used.

Firstly, there are eight input that are digital, so they are either on or off. These can be connected to the Bilge pumps and used to monitor the number of times that pumps turn on. They could also be used to trigger a text message to the owner every time the pumps turn on. These digital inputs are protected up to 30V and can be used to monitor both 12V and 24V systems. Inputs have a lower level cutoff around 6V. Below approximately 6V the input will read a zero. Above this limit it will read a one.

Secondly there are four analog inputs. These inputs would normally be connected to batteries or other analog sensors and can be used to read voltages. The inputs are protected up to 30V. Accuracy of the input is 8 bits or 255 possible values. The Analog to Digital converter has an accuracy specification that should be referenced. Typically, the input will be +/-0.2V.

Finally Boat IO also includes two channels of output. These outputs can control 0.5A of power and could be used to turn on the Bilge pumps (thru a relay) or they could be used to turn on a light to make it look like someone is onboard. The opto-relay is protected by a fuse F1 and F2 which are rated for 0.5mA.

The device used for the Analog to digital is the MAX11601. This is an I2C device, address 0xC8. Refer to the data sheet at https://www.maximintegrated.com/en/products/analog/data-converters/analog-to-digital-converters/MAX11601.html.

The Digital Inputs are read using an I2C IO Expander PCF8574, address 0x40. Refer to the data sheet at https://www.nxp.com/products/interfaces/ic-spi-serial-interface-devices/ic-general-purpose-i-o/remote-8-bit-i-o-expander-for-icbus-with-interrupt:PCF8574_74A.

Digital outputs are controlled using the NXP PCA9536, address 0x82. Refer to the data sheet at https://www.nxp.com/products/interfaces/ic-spi-serial-interface-devices/ic-general-purpose-i-o/4-bit-i2c-bus-and-smbus-i-o-port:PCA9536. The PCA9536 controls two isolated opto relays on IO0 and IO1. There are two LEDs that can be controlled on IO2 and IO3. The LEDs are there for user feedback and can be controlled in Python code.

3.4 NMEA 0183

Many boats still utilize the NMEA 0183 interface which is a much slower serial interface compared to the newer NMEA 2000 interface. Boat IO allow the Pi to both send and receive information from the NMEA 0183. This may include AIS information, or GPS information. Once read by the Pi, this information could be written to the NMEA 2000 interface. This allows the Pi to act as a gateway between the older NMEA 0183 and the newer NMEA 2000. **Note**: The NMEA 0183 interface shares the Pi Serial port with the GPS. The GPS can be configured to use the I2C interface allowing the NMEA0183 to use the Pi serial. The configuration is selected with jumpers J9 and J10.

3.5 Real Time Clock

Now the Pi will have access to a real time clock and GPS synchronized time. The BOAT IO design incorporates the Pi aware PCF8523 Real Time Clock (RTC). The clock has a 10mm Lithium backup battery for when the Pi is powered down. Battery life should be greater than three years depending on the temperature it has been used in.

4 Board Details

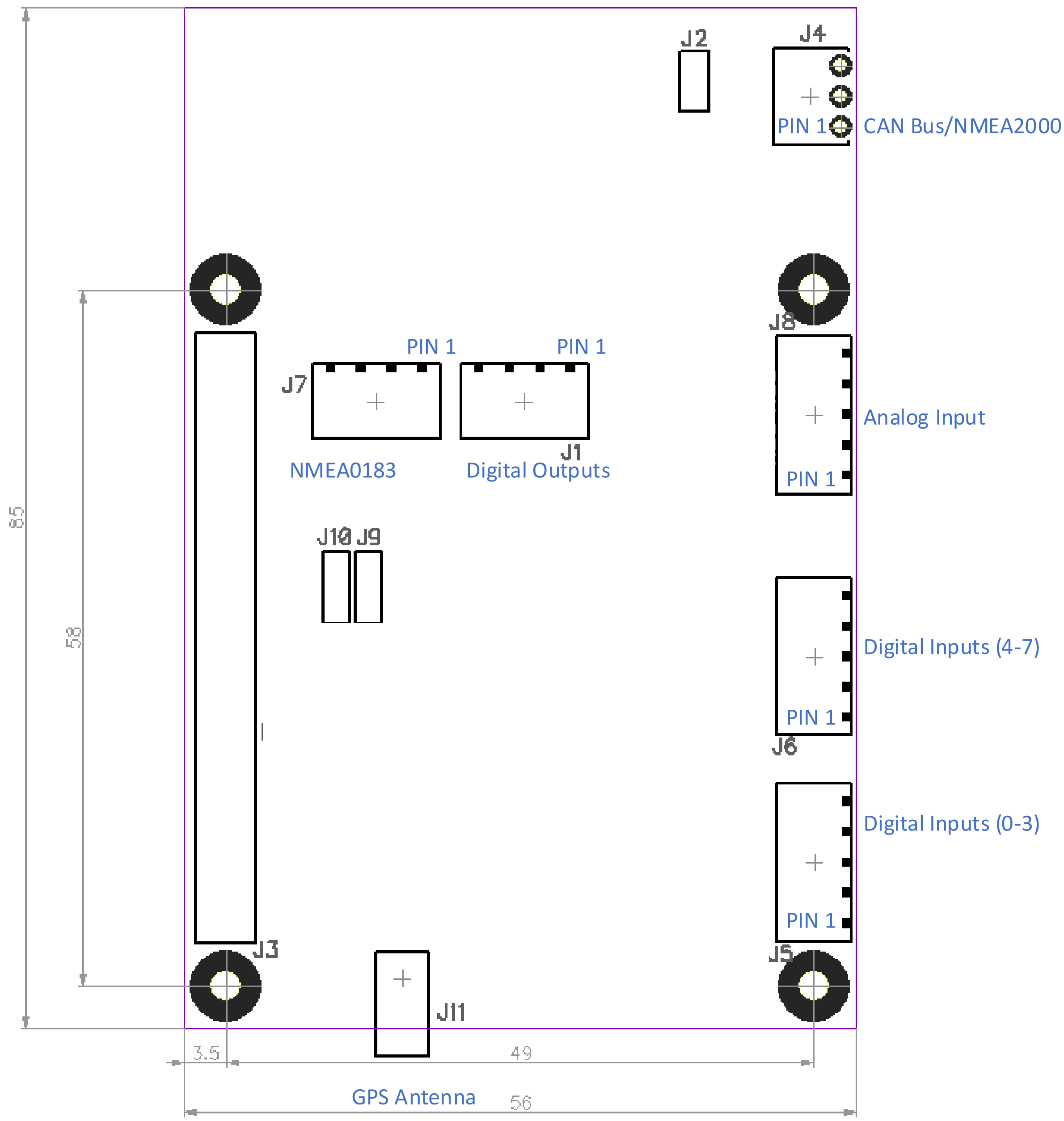


Figure 1 Connector Locations and Pin 1

5 Connector Pinouts

The tables that follow identify the pinout for each connector. Please note the location of pin 1 per the diagram in Figure 1. Pin 1 is typically System Ground when applicable.

J1 – Outputs	Signal	Notes
1	Output1	Isolated, Limited to 500mA
2	Output1 Return	
3	Output2 Return	
4	Output2	Isolated, Limited to 500mA

J4 – CAN/NMEA2000	Signal	Notes
1	System Ground	
2	CAN_L	CAN Bus signals
3	CAN_H	CAN Bus signals

Note the CAN bus needs termination. Normally this is included on the bus, but by inserting jumper J2, this will add 120 ohms to the bus. It is not normally needed.

J5 – Digital Inputs	Signal	Notes
1	System Ground	
2	Digital Input 0	Maximum input 30V
3	Digital Input 1	
4	Digital Input 2	
5	Digital Input 3	

J6 – Digital Inputs	Signal	Notes
1	System Ground	
2	Digital Input 4	Maximum input 30V
3	Digital Input 5	
4	Digital Input 6	
5	Digital Input 7	

J7 – NMEA 0183	Signal	Notes
1	NMEA0183 Rx-	
2	NMEA0183 Rx+	
3	NMEA0183 Tx-	
4	NMEA0183 Tx+	

J8 – Analog Inputs	Signal	Notes
1	System Ground	
2	Analog In 1	0 – 16V corresponds to 0-255 when reading Maximum input 30V
3	Analog In 2	
4	Analog In 3	
5	Analog In 4	

J9 and J10	Signal	Notes

1	NMEA0183	Connect jumper 1-2 on J9 and J10 for NMEA0183
2	Pi Serial	Use Pi serial for NMEA0183 or GPS
3	GPS	Connect jumper 2-3 on J9 and J10 for GPS

J11 – GPS Antenna	Signal	Notes
1 (Center)	GPS Signal	3.3V @ 10mA for active antenna
2	System Ground	

Notes:

NMEA0183 and NMEA2000 refer to industry standard marine busses. BOAT IO is not certified as NMEA0183 or NMEA2000 but is know to work.

6 Programming

Programming the Boat IO in Python is very simple. Refer to the small program below to see examples of reading the Analog and Digital IO.

```
import smbus

import time

from os import system

bus=smbus.SMBus(1)

**** Digital Output

PCA9536=0x41  #7 bit address (will be shifted to add the read/write bit)

PCF8574=0x20

LED1 = 0x04

LED2 = 0x08

OUT1 = 0x01

OUT2 = 0x02

***** Analog Input

MAX11600_ADDRESS = 0x64      #shifted left

MAX11600_CHANNEL0 = 0x00

MAX11600_CHANNEL1 = 0x02

MAX11600_CHANNEL2 = 0x04

MAX11600_CHANNEL3 = 0x06

MAX11600_SINGLE = 0x01

MAX11600_SCAN_SINGLE =0x60

MAX11600_SETUP = 0x80

MAX11600_REF = 0x50

MAX11600_RST = 0x02


def MAX11600_init():

        out = MAX11600_SETUP+MAX11600_REF+MAX11600_RST
```

```
        print "MAX11600 Setup:",hex(out)
        bus.write_byte(MAX11600_ADDRESS,out)

def MAX11600_read(channel):
        out = MAX11600_SCAN_SINGLE+channel+MAX11600_SINGLE
#       print "MAX11600-Conf:",hex(out)
        bus.write_byte(MAX11600_ADDRESS,out)
        return bus.read_byte(MAX11600_ADDRESS)
***** Digital Input
def PCA9536_init():
        # set register 3 to all outputs
        bus.write_byte_data(PCA9536,3,0x00)
        #turn off all outputs
        bus.write_byte_data(PCA9536,1,0xFF)

def PCA9536_output(out):
        # flip the bits
        out = (0xFF ^ out) & 0x0F
        #print "out:",out
        bus.write_byte_data(PCA9536,1,out)

def PCF8574_input():
        val =  bus.read_byte(PCF8574)
        return val

***** Initialize the devices
PCA9536_init()
MAX11600_init()
```

```
while True:
#	print "LED1"
#	PCA9536_output(LED1)
#	time.sleep(2)
#	print "LED1+LED2"
#	PCA9536_output(LED2+LED1)
#	time.sleep(2)
#	print "LED1+LED2+OUT1"
#	PCA9536_output(OUT1+LED2+LED1)
#	time.sleep(2)
#	PCA9536_output(OUT1|LED2|LED1)
#	print "Scale", ANALOG_SCALE
#	time.sleep(1)
	print "Input {0:b} ".format(PCF8574_input())
	#time.sleep(1)
	print ("An1:" MAX11600_read(MAX11600_CHANNEL0)
	time.sleep(1)
	print "An2",MAX11600_read(MAX11600_CHANNEL1)
	time.sleep(1)
	print "An3",MAX11600_read(MAX11600_CHANNEL2)
	time.sleep(1)
	print "An4",MAX11600_read(MAX11600_CHANNEL3)
	_ = system('clear')
	time.sleep(1)
```