

Popoto Modem

POPOTO MINI USER'S GUIDE

Software Version 2.06

Document Versioning Information				
Version	Purpose	Author	Sw Ver	Date
1.01	Initial draft	Jim DellaMorte		9/28/2018
1.02	Updated Document SSB	John DellaMorte		2/2/2019
1.03	Added many new commands	John DellaMorte		1/15/2020
1.04	Fixed page numbers	John DellaMorte		2/26/2020
1.05	Added commands	John DellaMorte	2.06	5/28/2020

Table of Contents

1	Getting Started	6
1.1	In the box	6
1.2	Required equipment	10
1.3	Hardware Configuration	10
1.4	Bench Test	11
1.4.1	Airtest with Transducers	11
1.5	Communicating with Popoto	12
1.5.1	RS-232 UART Connection	12
1.5.2	Ethernet Connection	12
1.6	Running the application	13
1.6.1	Checking the Version Number	13
1.6.2	Listing the help	13
1.6.3	Running a ping	13
1.6.4	Ranging	14
1.6.5	Telnet operation	14
1.6.6	Chat Operation	15
1.6.7	Setting the data rate for Payload operation	16
1.7	Single Sideband Operation	18
1.7.1	SSB Transmitter	18
1.7.2	SSB Receiver	19
1.7.3	Return to data mode	20
1.7.4	SSB Controllable parameter	20
	System Overview	21
1.8	Introduction	21
1.9	Architecture	21
1.10	System connections	23
1.10.1	RS-422 4 wire serial	23
1.10.2	RS-232 Uart	23
1.10.3	Network: 10/100BaseT	23
1.11	Modes of operations	24
1.11.1	Local pshell	24
1.11.2	Remote pshell	24
1.11.3	Matlab™	25
1.11.4	Custom interfaces	25
2	Hardware components	26
2.1	A brief tour of the Digital Board	26
2.2	A brief tour of the Analog Board	28
2.3	Mounting tray	29

2.3.1	Thermal putty.....	30
2.4	Transducer.....	30
3	<i>The pshell.....</i>	31
3.1	Modes of operation	31
3.2	Requirements for running	31
3.3	Invoking pshell.....	31
3.3.1	The pshell.init file	31
3.3.2	Invoking pshell from a linux prompt.....	32
3.4	Invoking commands	32
3.4.1	Help.....	32
3.4.2	Tab Completion.....	32
3.4.3	Commands.....	32
3.5	Extending the pshell	53
3.6	Set-able and Get-able Parameters of Popoto Modem	53
3.6.1	System Level Variables	53
3.6.2	Receiver Oriented Variables.....	59
3.6.3	Transmitter Oriented Variables.....	70
4	<i>Updating the firmware.....</i>	82
4.1	Introduction.....	82
4.2	Details on how to update the firmware.....	82
4.2.1	Upload Procedure:.....	82
5	<i>Diagnostics/Logs.....</i>	85
5.1	Introduction.....	85
5.2	Popoto log.....	85
5.2.1	Introduction	85
5.2.2	Location.....	85
5.2.3	Logging Levels.....	86
5.2.4	MSM Logs.....	86
5.3	PCM Logging.....	86
5.3.1	Introduction	86
5.3.2	Socket based PCMLogs.....	88
5.3.3	Target File based PCM Logs.....	88
5.3.4	Notes:.....	88
5.4	pshell Logging	89
6	<i>Appendix.....</i>	90
6.1	The Acoustic Message Header.....	90

List of Figures

Figure 1 25 Khz Popoto Transducer	6
Figure 2. Popoto Analog Board	7
Figure 3 Popoto digital board	8
Figure 4 Popoto Mounting Tray	9
Figure 5 High Level Popoto Block Diagram	11
Figure 6 Popoto System	21
Figure 7 Overview of the Popoto Software architecture. Circles represent task-level processing. Arrows represent data flow between these modules.....	22
Figure 8: There are 3 main interfaces to the Popoto Modem application. 4 Wire RS422 UART, 3 Wire RS232 UART and 10/100 BaseT Ethernet. The location of each is highlighted in this figure.	23
Figure 9: Local pshell processing. The pshell runs on the ARM processor, and connects to the sockets via localhost. The only requirement for interface is a text-based connection either via one of the serial ports or over the ethernet (ssh).....	24
Figure 10: Operating in remote-pshell mode. In this use-case the remote processor opens the sockets to the Popoto_app directly and communicates via the network.....	25
Figure 11 Operating in Matlab™ mode. In this use-case the remote processor is a PC running Matlab, using the Popoto.m class to interface to the Popoto_app. This use-case allows for rapid access to the PCM stream of the Popoto system.....	25
Figure 12 Annotated Digital Board	26
Figure 13 Analog Top Side	29
Figure 14 Transducer impedance vs frequency air vs water	30
Figure 15: Format of a single PCM Log Packet. These packets are transmitted on the TCP PCM Recording socket.	87
Figure 16: The PCM Packets are sent one after the other to the TCP Socket or to the Target log file.	87

1 Getting Started

In this section we will explore how to configure, cable, and try the Popoto hardware and software system.

1.1 In the box

A complete Popoto system consists of the following hardware components:

1. Transducer
2. Analog board
3. Digital board
4. SD card
5. Mounting tray

Transducer: The Popoto transducer consists of a potted ceramic piezo ring. It is designed to efficiently convert mechanic signal energy to and from electrical analog signals in the 25 KHz region. A picture of the transducer is shown below:



Figure 1 25 Khz Popoto Transducer

Analog Board: The Popoto analog board provides signal conditioning to and from the transducer and provides conversion of the analog signals to the digital domain. The signal conditioning of the receiver includes amplification, high pass filtering of the data, and analog to digital conversion. The signal conditioning for transmitter includes digital to analog conversion, and high power transmit amplification. The analog board also includes a line level analog path to and from SMA connectors for debug purposes.



The analog board directly connects to the digital board by way of a 30 position connector at the bottom of the board.

A picture of the analog board is shown below:



Figure 2. Popoto Analog Board

Digital Board: The Popoto digital board provides for all signal processing, interface to analog board, interface digital communication interfaces including:

- RS-232
- RS-422
- Ethernet

It also hosts all non-volatile and volatile memory, performs power conditioning, gpio interface, and real-time clock functionality.

The heart of this board is an OMAP L138 device made by Texas Instruments. This device includes an ARM 9 host processor which runs Arago Linux, and a TMS320C647x DSP floating point DSP device which performs the computationally intensive signal processing tasks.

A picture of the digital board is shown below:

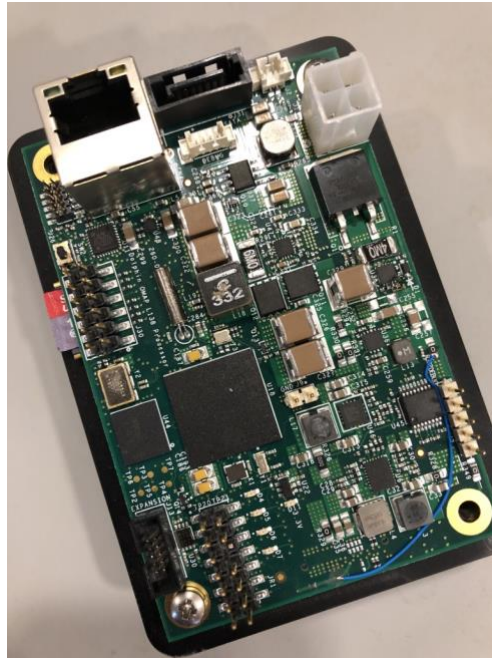


Figure 3 Popoto digital board

micro SD Card: The enclosed Popoto micro SD card has been formatted Ext4 and includes all of the operating system files, the Popoto application, the DSP application. It additionally provides room for several GB of diagnostic storage if desired.

Mounting Tray: The Popoto mounting tray is used to mechanically host both the analog and digital boards. The two board connectors mate together through a slot in the mounting tray. Another very important function of the mounting tray is to act as a heat sink for the power amplifier on the underside of the analog board. This heat conductive interface is critical to achieving the 100 Watt transmit capability of Popoto. The thermal junction between the mounting tray and power amplifier on the analog board requires conductive thermal compound at this interface.

There are six tapped M4 mounting locations on each end for the user to mount the boardset.

A picture of the mounting tray is shown below:

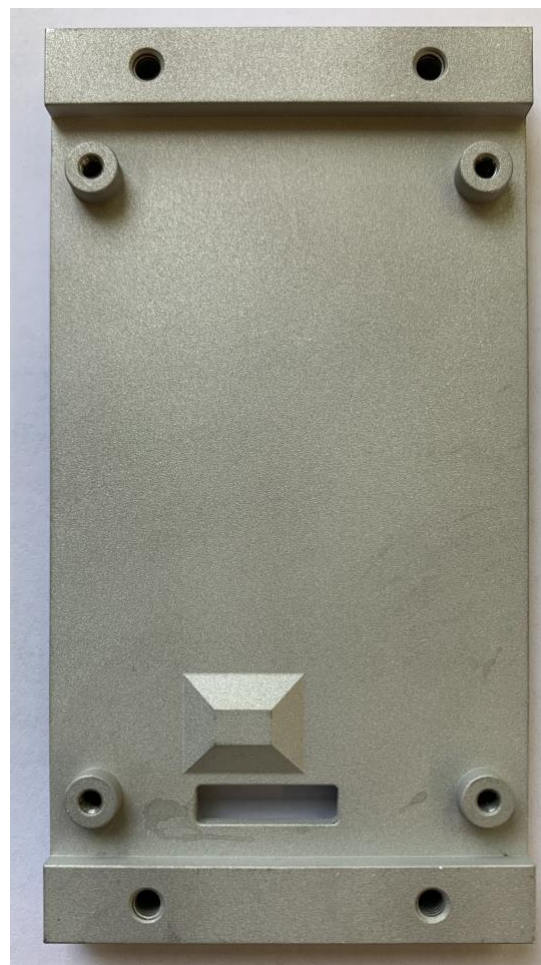


Figure 4 Popoto Mounting Tray

1.2 Required equipment

Along with the hardware and software that comprise Popoto, it is necessary have the following equipment to facilitate the “Getting Started” procedure of this chapter.

- 12-18 Volt 5 Amp DC Power Source
- Ethernet cable
- PC running Ubuntu with ethernet capability
- RS-422 to USB connector
- SMA connectors
- Configuration Jumpers

Other helpful PC software to have at the ready includes

- MATLAB
- Audacity Audio Software
- Python 2.7
- Serial Port software for RS-232 or RS-422 connections

1.3 Hardware Configuration

There are three jumpers on the analog board that should be configured. These jumpers are:

Jumper	Purpose	Setting
J1	When inserted this jumper connects the SMA amplifier to positive power. If the SMA is intended for use, this jumper should be inserted	Jumper inserted for SMA
J2	This jumper selects the input to Popoto as either from the Transducer or the SMA input	1-2 Transducer Input 2-3 SMA input
J5	When inserted this jumper connects the SMA amplifier to negative power. If the SMA is intended for use, this jumper should be inserted	Jumper inserted for SMA
J8	Connects the Analog high gain path to the transducer. This should be inserted	Inserted
J10	Connects an external Transducer signal to the low-gain path. Normally not inserted	Open



1.4 Bench Test

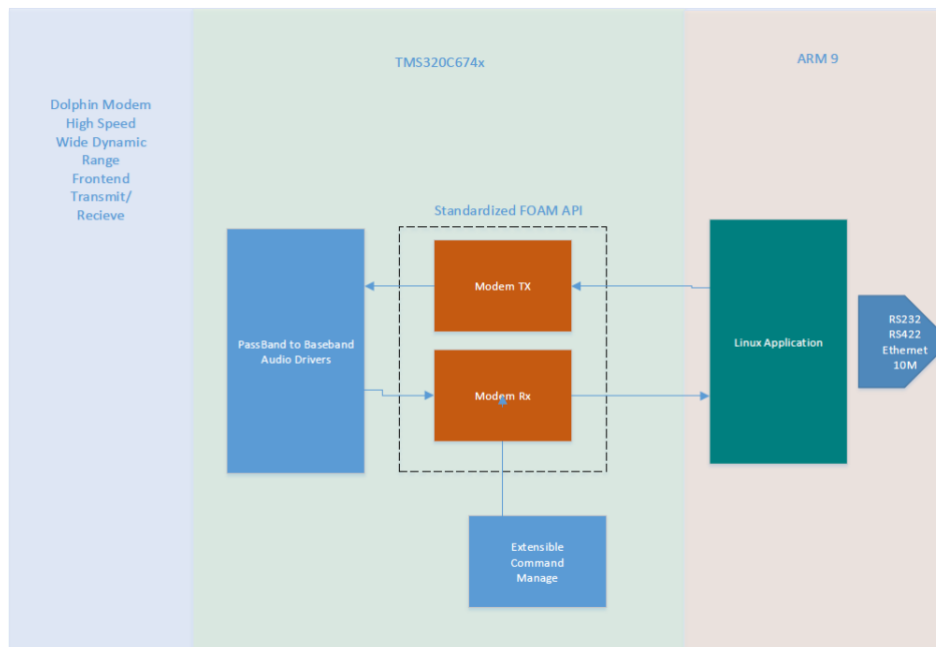


Figure 5 High Level Popoto Block Diagram

1.4.1 Airtest with Transducers

Although the Popoto modem is designed to operate acoustically in an ocean environment, it can communicate (although somewhat less reliably) in air. The acoustic energy transmitted from Modem 1 can indeed be propagated through the air for short distances and received by Modem 2. Assuming the multipath energy from sound reflection of the walls is not too damaging, this signal can be detected and demodulated. If the multipath of the room prevents detection, some careful placement of sound absorbing materials such as foam or cloth, and repositioning either the transmitter or receiver transducer until reliable communication is usually possible.

Running an air test is a good way to validate operation prior to water operation. Once reliable communication is achieved, various commands such as ranging can be exercised effectively. It should be noted that the range command will not yield accurate range estimates in air because the speed of sound in air is more than 5 times slower than the speed of sound in water. However, ranging in air is still useful for basic system checkout prior to fielding the modem in the water.

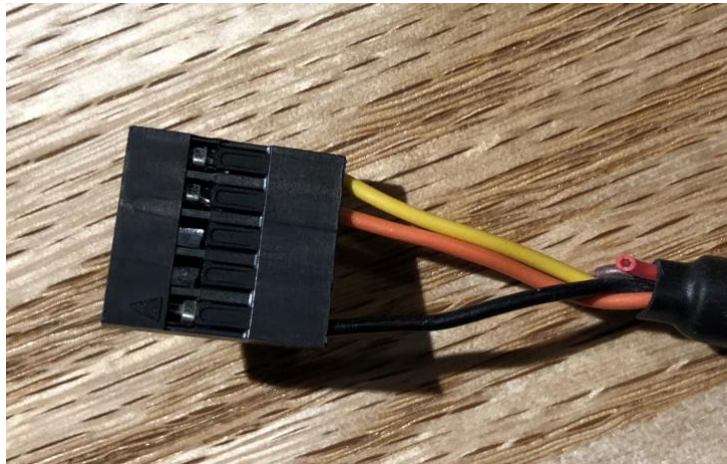
Configuration of the analog section for an air test is quite simple. First ensure that the Jumper J2 is inserted between positions 1-2. This sets the analog input to driven by the acoustic signals picked up by the transducer.

Once J2 is set, all that is necessary is plugging in the transducer into the J9 connector of the analog board.

1.5 Communicating with Popoto

1.5.1 RS-232 UART Connection

For the purpose of “Getting Started”, it is recommended that a RS-232 Level UART to USB cable be used. This is a standard off the shelf FTDI Cable with one end pinned to a 5 pin connector that fits onto JP3 of the digital board, the other end of the cable is a standard USB connector and will connect to a Windows or Linux PC.



When the UART USB cable is inserted the OS will discover the new communication device. At that time open any standard serial terminal program configured for communication at 115200 bps, with no parity, 8 data bits, and 1 stop bit. When Popoto is powered up, the boot process of the Popoto will be visible on this serial terminal. The boot up is finished when a login prompt is available. Though login is not necessary for this “Getting Started” section, it is certainly possible to login at this prompt. The user is root and there is no password.

1.5.2 Ethernet Connection

The ethernet connection is the most high-speed and versatile way to communicate to the Popoto.

1.5.2.1 Cabling

To connect to the Popoto using the ethernet simply attach an ethernet connector to the CN4 of the digital board. The other end can either connect directly to the PC or through a switch or router.

If a direct connect to the PC is utilized, it is required that the PC adapter is configured for this case. To configure the ethernet network adapter the settings of TCP IPV4 must set to be in the proper IP domain of Popotos static IP address, namely a 10.0.0.xx domain. Usually it is helpful not to be connected to other network devices such as wifi when in this mode.



If a connection through a router is desired, it is necessary for the router IP address domain be in the same domain as Popoto, namely 10.0.0.xx,

1.5.2.1 Static IP

Note the Popoto modems are configured by way of the SD image to be at either 10.0.0.232 fixed IP. This address can be changed using `setIP` command in pshell.

1.6 Running the application

For this section it is assumed that we will be running air tests.

Once the modem has been completed the boot process it is possible to connect to the Popoto by way of pshell. To invoke the pshell to connect to Popoto at address 10.0.0.232 simply open a command shell in your operating system and type

```
python pshell
```

You should then see a popoto command prompt. Connect your pshell to your hardware using the `connect` command as follows (assuming your hardware is at 10.0.0.232)

```
connect 10.0.0.232 17000
```

Once connected, it is possible to invoke a wide variety commands from the pshell.

1.6.1 Checking the Version Number

From the pshell type the `version` command. Popoto will respond with current software version number.

1.6.2 Listing the help

To list the commands supported by the pshell, simply type `help`. Popoto will respond with a list of supported pshell commands. Note that tab completion for these commands is supported.

1.6.3 Running a ping

When both modems are online and connected to their pshells, it is possible to send an acoustic ping from one modem to be received by the other.

At the pshell type

```
ping 4
```



This command initiates the transmission of a test packet at about 4 watts of acoustic power. This level of power is appropriate for an air test where the transducers of units are spaced 1-2 meters apart.

While the transmission is executing you will notice a red “transmitting” led illuminate on the transmitters analog board. Once the transmission completes (3-4 seconds) the led will turn off. On the receiver pshell, there should be indication of a packet received and the both the packet and the header data should be displayed.

A message indicating ‘CRC error’ may occurs at the time of transmission on the receiving Popoto instead of a ‘CRC check’ message. This occurs if the multipath of the room is adding so much interference that the demodulator cannot successfully demodulate the test packet. In such a case, reposition the transducers or pad any reflective surfaces to minimize acoustic reflection.

1.6.4 Ranging

Once a successful ping has been achieved, it is instructive to try a range command. The syntax of the range command is as follows:

```
range 4
```

This command instructs the initiating modem (Modem A) to send a range request at the power associated with about 4 watts. You will immediately see the transmitter red LED of Modem A illuminate for about a second. This request should be received by the receiving modem Modem B. Upon successful demodulation of the range request by Modem B, it schedules a transmission back to the receiver of Modem A. This new transmission will illuminate the red LED of Modem B. When that transmission is complete, Modem A will measure the time required to receive the response to its request, account for turnaround time, and calculate the round trip time. This is mapped to a distance using the speed of sound in water and range is calculated.

Upon successful completion of the whole ranging cycle, a range report will be displayed on the Modem A pshell.

1.6.5 Telnet operation

Lastly, it is possible to open up a chat window between both modems. From a linux prompt on the terminal, type

```
telnet 10.0.0.232 17001
```

this will open a telnet window connected to Modem A (at the 10.0.0.232 address).

Assuming Modem B has been setup with and IP address of 10.0.0.223.

Next open another linux prompt on the terminal and type

```
telnet 10.0.0.223 17001
```

This will start another telnet session connected to Modem B (at the 10.0.0.223 address)



1.6.6 Chat Operation

Chat operation is a mode of the modem where two modems can communicate keyboard to keyboard in a normal text configuration in a half duplex mode. To enter chat mode the

chat

command is entered at the pshell prompt. It is necessary to enter chat mode at both the receiver and transmitter for chat mode to work.

While in chat mode characters that are entered in the keyboard are grouped into packets and transmitted through the water, received by the receiver and presented to the user.

The start and stop of a packet is determined by 3 factors. The first method is to enter a carriage return after a string of characters. This return signals the end of a string of characters to be sent out of the modem. The second method to signal the end of a string is to timeout. After period of no typing that exceeds the user configurable timeout parameter, the transmitter console will take the user input gathered up until the timeout interval, group them into a packet and send them. The last way to terminate a sequence of characters for transmission is to exceed the user configured number of bytes per packet. For example if the parameter ConsolePacketBytes is set to 32, then input characters are bundled into groups of 32 and sent out automatically.

The user configurable parameters for console operation are shown below:

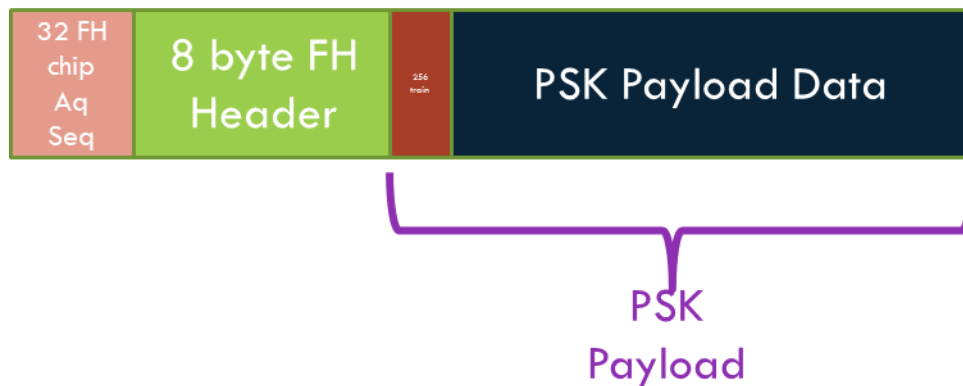
Parameter	type	Description
ConsolePacketBytes	int	Sets the number of bytes in the data console, after which they automatically are sent to the transmitter
ConsoleTimeoutSec	int	Sets the timeout in seconds for the data console, after which they automatically are sent to the transmitter

To exit chat mode type *ctrl-]*, followed by *e* for exit. Exit from both the transmitter and receiver to resume normal operation.

1.6.7 Setting the data rate for Payload operation

All packets comprised of three parts, the acquisition, the header, and an optional payload. The acquisition sequence does not change for different data rates. The header is always sent at 80bps frequency hopping mode. The header contains information such as the transmit ID, intended receiver ID (broadcast ID is 255), tx power, if there is a payload of information following, what the payload length is, and what the modulation scheme for sending the payload is.

An example of a PSK payload packet is shown below:



The modulation rate of the payload portion of the waveform is configured using the *PayloadMode* variable. The various modulation schemes are:

- | | |
|---|--------------------------|
| 0 | 80bps Frequency Hop mode |
| 1 | 5120 bps PSK |
| 2 | 2560 bps PSK |
| 3 | 1280 bps PSK |
| 4 | 640 bps PSK |
| 5 | 10240 bps PSK |

The PSK receiver includes user configurable parameters that can be adjusted for optimal reception as a function of the channel. These include the number of taps for the equalizer and the location of the first tap. Under normal operation these parameters are set for typical operation with the number of forward taps (FIR) = 44, the number of backward (IIR) taps=6, and the location of the first tap=16. Note the computational load of the receiver increases with the square of the number of taps and the maximum number of taps (Forward + Backward) should not exceed 70. Also note that additional taps often increase noise and as such more taps does not always mean better performance.

Parameter	Type	Description
PayloadMode	int	Sets the modulation of Transmitter Payload 0- Frequency Hop 1- PSK 5.12 kbps 2- PSK 2.56 kbps 3- PSK 1.28 kbps 4- PSK 640 bps
PSK_FnTaps	int	Sets the number of taps for FIR portion of filter (default 44)
PSK_BnTaps	int	Sets the number of taps for IIR portion of filter (default 6) <i>Note: PSK_FnTaps+PSK_BnTaps must be less than 70</i>
PSK_StartOffset	int	Sets the location of first tap in FIR delay line (default 16)

1.7 Single Sideband Operation

Utilizing Popoto's single sideband transmitter (SSB) and receiver allow for half duplex voice communication through the water. The SSB signal is inherently an analog signal being through the ocean at a carrier frequency. As an analog signal, this means that the reception of the analog waveform includes the analog impairments of the channel. So if the channel is noisy, the receiver will hear the noise. If the channel has echo, the resulting speech will include echo. If there is no noise and no echo, and analog levels are set properly, there will be no distortion of speech aside of the normal band limiting associated with telecom speech.

To utilize the SSB functionality of Popoto, it is necessary to ensure that the voice path electronics are powered up. This is done by ensuring jumpers J1 and J5 are populated.

1.7.1 SSB Transmitter

The transmitter consists of a single sideband modulator which receives speech from the microphone input J3 and modulates it up to carrier for transmission out of the transducer and through the water.

There are 3 ways to place the SSB transmitter in transmit mode.

1. A Popoto ssbtx command
2. A hardware PTT signal
3. Using the properly adjusted VOX

1.7.1.1 The ssbtx command

Issuing the ssbtx command places Popoto in transmit mode. This can be clearly seen by the transmit LED glowing red on the analog board. Once in transmit mode, audio that is input on SMA J3 will be modulated, shifted up to carrier, power amplified, and delivered to the transducer.

1.7.1.2 Adjustment of transmit power

Proper adjustment of the transmit power is critical for good operation of the SSB transmitter. Setting this power properly is a function of 2 variables

1. Microphone sensitivity
2. Desired transmit power

Both of these variables are controlled by the SSB_Txpower variable. This variable should be set such that the desired PEP power is achieved while speaking at a normal level in the microphone.

1.7.1.3 Peak Envelope Power

The proper adjustment of power for voice operation revolves around properly setting the Peak Envelope Power. PEP is the value of power that is output by the transmitter when the speech is at



peaks in its overall envelop. Typically average power of speech is between 10%-20% of the peak envelop power.

These adjustments should be made while the transducer is in water. Also these setting can be approximated by careful monitoring of the input power in these peak regions and setting the SSB_Txpower constant appropriately.

Choosing an appropriate PEP level is a function of the distance that one wishes to transmit, the SNR of the channel, along with the reflectivity of the channel. These settings can be experimentally derived in the water and presets can be made in the pshell for optimum speech quality.

1.7.1.4 PTT keying of the transmitter

The Popoto hardware presently includes two GPIOs that are used for PTT and also headset volume control. The truth table shown below illustrates the various modes associated with the GPIOs. When the two GPIOs are zero, the transmitter is keyed, when they are 1,1 the receiver operates, and the other two states will raise or lower the headset volume by 1 dB per click.

Gpio8[6]	Gpio7[14]	State
0	0	PTT depressed (Transmit Mode)
0	1	Headphone Volume UP
1	0	Headphone Volume DOWN
1	1	Receive Mode

1.7.1.5 Transmitter Vox

The SSB transmitter can be switched on using the speech signal itself. To utilize this feature, the SSB_Vxmode should be set to 1. Next the SSB_Vxlevel should be increase from zero slowly while speaking to arrive at the trigger point for the VOX. Proper setting of this level will ensure that constant level audio background will not trigger the transmitter, but onsets of speech will trigger the transmitter. Note that once the transmitter is keyed, the transmitter remains on for a period of 2 seconds.

1.7.2 SSB Receiver

Voice mode reception is enabled by issuing the *ssb* command from the pshell. At this command the modem will transition from data modem mode to single side band receiver. Demodulated



audio will be present on the SMA connector J4. The audio level present on J4 is controllable by setting the `SSB_Volume` parameter to the user desired level.

Additionally, the receiver incorporates a squelch for eliminating background noise between segments of received speech. To utilize the squelch it is important to set the `SSB_SqLevel` parameter in the pshell. An `SSB_SqLevel` of zero reflects no squelch and the receiver will be continuously in the receive state with demodulated audio being presented to the headphones. The user can gradually increase the squelch level until the interspeech segments are muted.

1.7.3 Return to data mode

To return to data mode simply enter `datamode` at the pshell prompt.

1.7.4 SSB Controllable parameter

The table below show all of the settable/gettable parameters available through the pshell for the purpose of controlling SSB operation.

Parameter	Type	Description
<code>SSB_Volume</code>	float	Sets volume level of headset
<code>SSB_Txpower</code>	float	Sets microphone gain; for SSB, this controls the Tx power
<code>SSB_VxMode</code>	int	Normal PTT set to 0; VOX mode PTT set to 1
<code>SSB_VxLevel</code>	float	Sets the trigger level for PTT VOX
<code>SSB_SqLevel</code>	float	Sets the background noise level for squelch trigger (0.-always on)



System Overview

1.8 Introduction

The Popoto system consists of several components working together to create an acoustic digital communication system. At the lowest level, a Transducer provides the physical interface between the Modem and the water. This transducer is connected to the Analog board which can both drive the transducer as an output, and receive from the transducer as an input. The analog board digitizes input and converts the analog signal in the water to digital data which is sent to the Digital board. The digital board demodulates the data on the DSP, and sends the bitstream to the ARM9 which determines what to do with the data based on the current processing state.

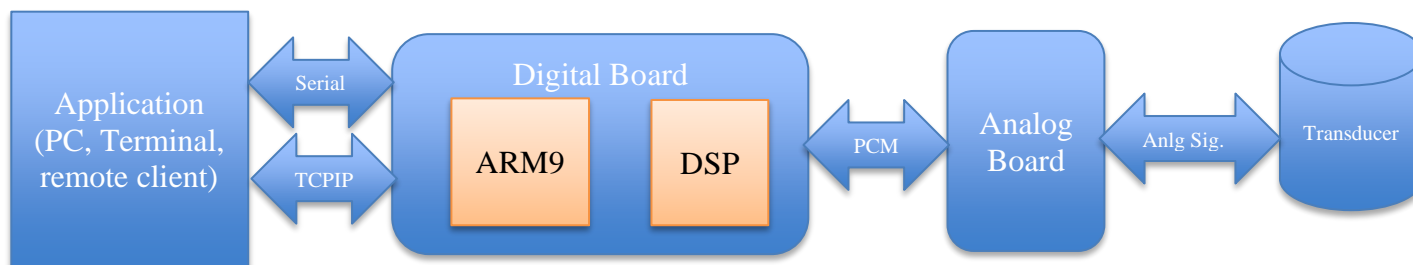


Figure 6 Popoto System

1.9 Architecture

The Popoto modem system consists of a modem *Algorithm* operating within a framework. The modem algorithm is responsible for modulating and demodulating PCM data to and from bits. The Framework is responsible for bringing data to and from the algorithm, and controlling the hardware and user interface to the modem system. In the Popoto system the modem used is a Frequency Hopping modem, and the Framework is what we call FOAM.

FOAM is an acronym for a Flexible Open Acoustic Modem Framework. This framework is written in C++ and is engineered for the development and support of underwater acoustic modems.

FOAM consists of 7 main features:

1. A processing statemachine (MSM) to control the interaction of the modem through States such as Listening, Receiving, Transmitting and Ranging.
2. A base class (ModuleBase) which provides Message Queues and processing threads which process text-based commands and requests from the Host or other threads
3. A Controller module which provides a single point of access to all settable and gettable parameters in the system, each through a unique text string.
4. A socket-based interface to the Host. FOAM has 4 separate interfaces:
 - a. Command and Status (TCPCommand)
 - b. Modulated data (TCPData)
 - c. PCM Logging (TCPPcmRecorder)
 - d. Virtual A/D D/A Socket (TCPPcm)

5. A System Controller (SysCtl) Which provides functions to interact with Popoto Hardware.
6. JSON Parseable status for ease of interface, and human-readability.
7. Complete interoperability with a PC based version of the code.

The figure below shows a task-level view of the FOAM system. Commands are received on the TCPCommand socket, and are sent to the Modem State Machine (MSM) where they are parsed, and processed, or forwarded to the correct entity for processing.

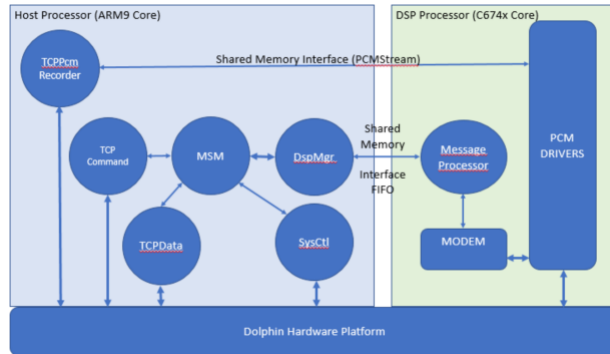


Figure 7 Overview of the Popoto Software architecture. Circles represent task-level processing. Arrows represent data flow between these modules.

1.10 System connections

The Popoto modem has 3 primary interfaces for an external CPU or computer to connect to:

- RS422 4 Wire serial
- RS-232 Uart
- 10/100 BaseT networking

Each of these connections has properties that make it attractive in different situations.

1.10.1 RS-422 4 wire serial

1.10.1.1 Reasons to use it

Good for long distance connections, up to 1200 meters. Simple serial interface. Robust to noise and interference.

1.10.1.2 Reasons to avoid it

Remote unit needs drivers. Only good for up to 115200 bits per second which is not adequate for PCM Streaming.

1.10.2 RS-232 Uart

1.10.2.1 Reasons to use it.

Figure 8: There are 3 main interfaces to the Popoto Modem application. 4 Wire RS422 UART, 3 Wire RS232 UART and 10/100 BaseT Ethernet. The location of each is highlighted in this figure.

The RS-232 uart is a 3 wire serial interface. It consists of 3 signals, Transmit data, Receive Data, and ground. The signal lines run at 3.3Volts This interface is particularly attractive if the user is interfacing the modem to a local device, such as a micro controller on a UAV. All that is required is a tx, rx and gnd signal.

For PC or laptop lab use, the pinout for this connector is a 5 pin 10mil header configured exactly as the standard FTDI USB cables, which makes for a simple USB to serial interface available off the shelf.

1.10.2.2 Reasons to avoid it

This interface is only good for very short distances, such as within the same enclosure. The bandwidth for this interface is limited to 115200 Bits per second which is not adequate for PCM streaming.

1.10.3 Network: 10/100BaseT

1.10.3.1 Reasons to Use it

The 10-100BaseT Ethernet networking provides the highest speed and most flexible connection to the Popoto system. Using TCP sockets over the ethernet provides upto 100Mbits/S of full-duplex throughput to the Popoto from a remote computer located up to 100 meters away. This



bandwidth can be used for real-time PCM capture, or rapidly updating software. Additionally, the flexibility of the TCP sockets allows for 3

1.10.3.2 Reasons to avoid it

The additional speed and flexibility of the ethernet comes at a cost of ~250 milliwatts. In addition, the range of the ethernet is limited to ~ 100 meters.

1.11 Modes of operations

The flexibility of the Popoto system provides for several use-cases. Each of these use-cases applies to a different product scenario, so it is important when deciding which to employ, that the requirements of the end product are carefully considered.

1.11.1 Local pshell

The pshell is a python program that connects with the Popoto application and provides a shell interface to the modem and its command, status and data interfaces. This shell provides simple commands such as send ranging, or setTxPower level so that either under human or computer control the modem can be utilized. In this use-case, the interface to the modem can be any one of:

- Serial RS-232
- Wire RS-422
- Or Ethernet over SSH

The pshell program runs co-resident with the Popoto_app on the OMAP's ARM core processor. Figure 8 shows local pshell processing.

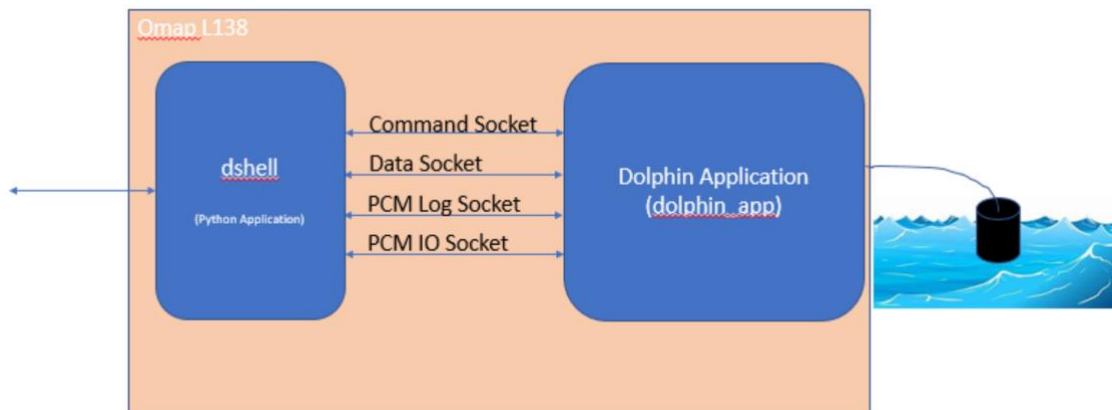


Figure 9: Local pshell processing. The pshell runs on the ARM processor, and connects to the sockets via localhost. The only requirement for interface is a text-based connection either via one of the serial ports or over the ethernet (ssh)

1.11.2 Remote pshell

The remote pshell operates in the same way as local pshell, however the pshell python program runs on a remote processor, and the connection to the Popoto_app is over TCP Sockets and networks. Using a remote pshell is advantageous for streaming PCM directly to the PC's



harddrive. Additionally the remote pshell is a good choice for running regression tests, as the regression suites can live on the remote pc, which can also log results.

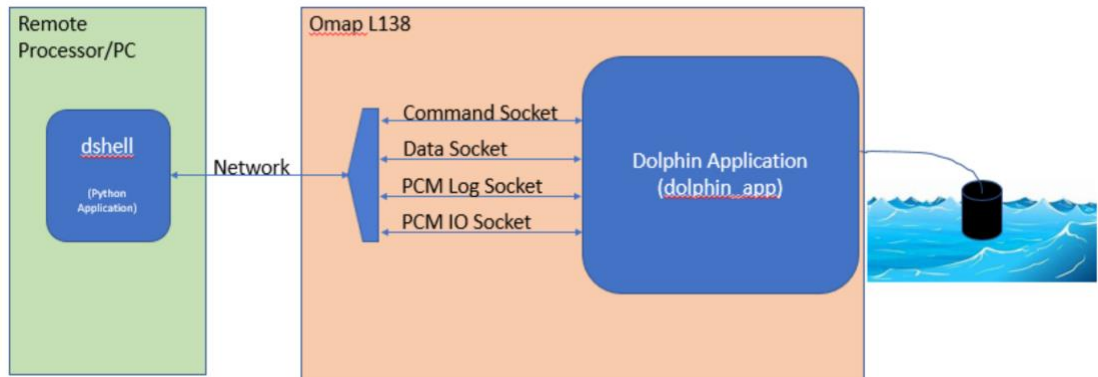


Figure 10: Operating in remote-pshell mode. In this use-case the remote processor opens the sockets to the Popoto_app directly and communicates via the network.

1.11.3 Matlab™

Matlab™ mode is very similar to remote-pshell mode, except that the connection to the Popoto_app does not use a Python program, rather it connects using Matlab™. Matlab™ is an excellent choice for running lab tests as it is a powerful language that is easy to use. Given Matlab™'s expense, and need for a full PC to run, it is not likely to be deployed in a customer's end product.

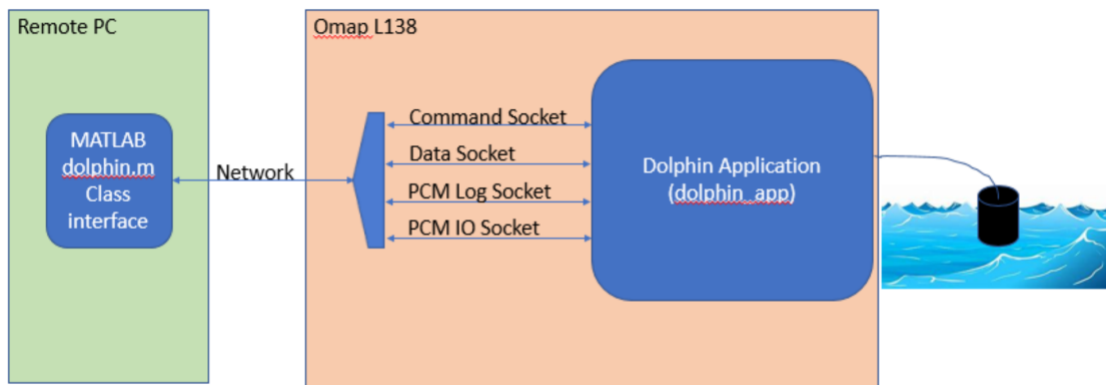


Figure 11 Operating in Matlab™ mode. In this use-case the remote processor is a PC running Matlab, using the Popoto.m class to interface to the Popoto_app. This use-case allows for rapid access to the PCM stream of the Popoto system.

1.11.4 Custom interfaces

The Popoto system uses standard sockets for communications, so it is entirely possible for a customer to generate a custom interface written in the language of his choice. Please see the Popoto.py and Popoto.m files for ideas on how to implement such an interface.

2 Hardware components

2.1 A brief tour of the Digital Board

The Popoto digital board contains many useful and flexible interfaces and connections. Whether or not some of these interfaces are utilized is application specific and will likely vary with customer needs. It is useful to take a macro view of the digital board and understand the function of the major items.

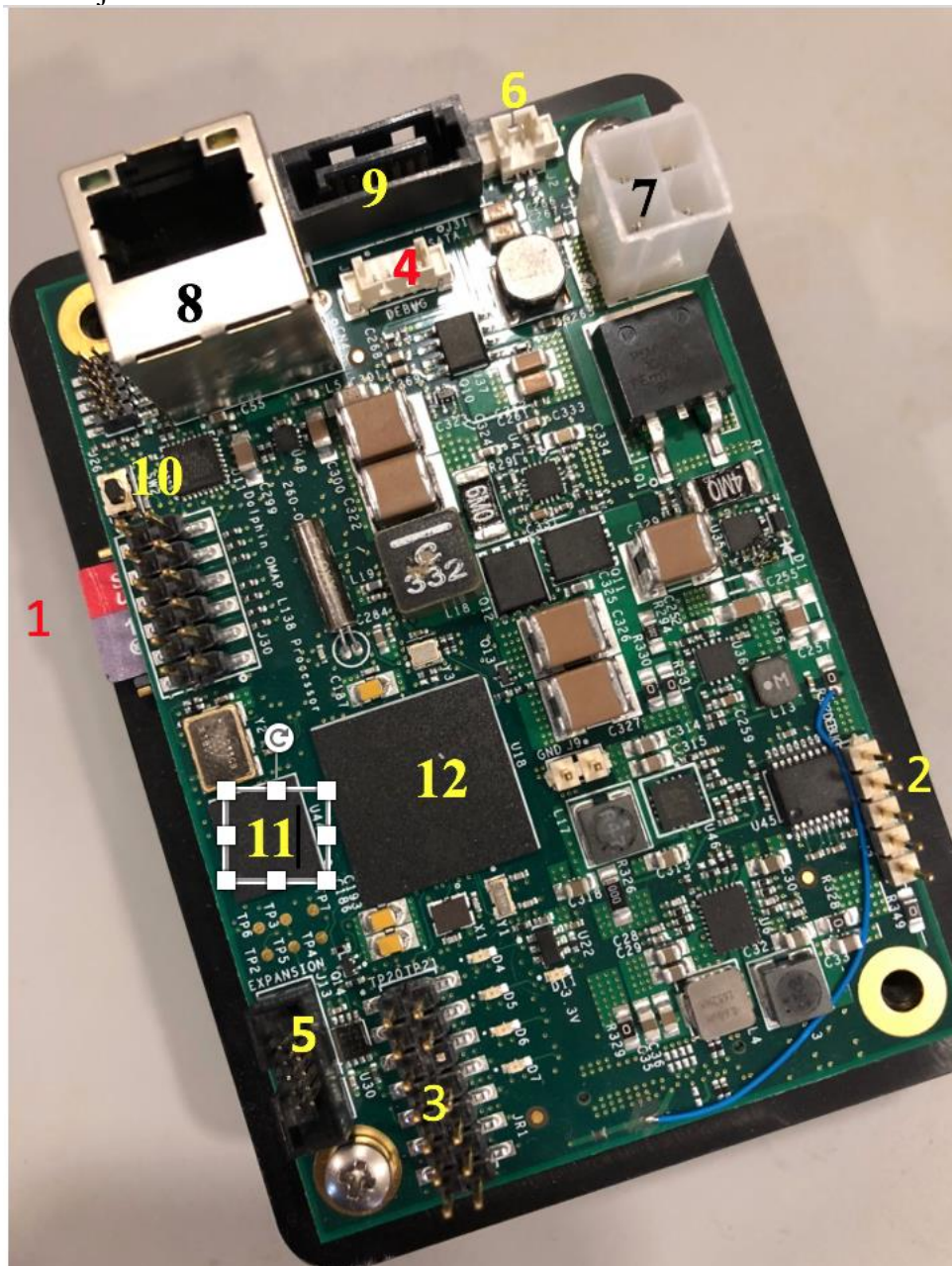

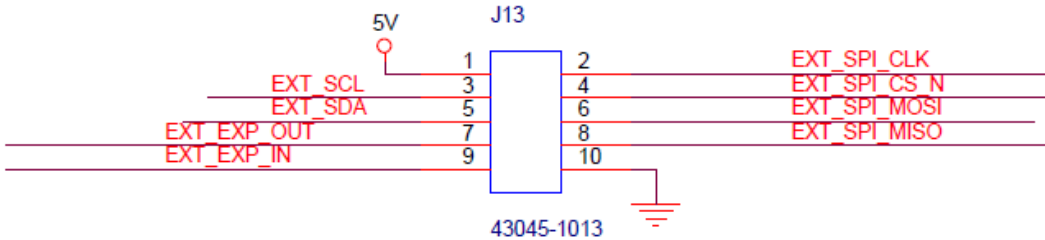


Figure 12Annotated Digital Board

Key																															
1	Micro SD slot. All Popoto software resides on this device. Although the system software will fit on a device as small as 1Gb, larger devices can be employed and are useful for storing logs and pcm recordings. Popoto can host microSD devices as large as 32 Gb.																														
2	<p>This row of 5 pins is for a standard RS-232 level UART connection. This connection is setup for 115200, n81</p> <p>The pinout for the standard 5 pin UART connector is as follows:</p> <table border="1"> <thead> <tr> <th>Pin Number</th> <th>Name</th> <th>Type</th> <th>Cable Color</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>GND</td> <td>Ground</td> <td>Black</td> <td>Device Ground Pin</td> </tr> <tr> <td>2</td> <td>NC</td> <td></td> <td></td> <td></td> </tr> <tr> <td>3</td> <td>NC</td> <td></td> <td></td> <td></td> </tr> <tr> <td>4</td> <td>RXD</td> <td>Signal IN</td> <td>Orange</td> <td>PC out Popoto IN</td> </tr> <tr> <td>5</td> <td>TXD</td> <td>Signal OUT</td> <td>Yellow</td> <td>PC In Popoto Out</td> </tr> </tbody> </table>	Pin Number	Name	Type	Cable Color	Description	1	GND	Ground	Black	Device Ground Pin	2	NC				3	NC				4	RXD	Signal IN	Orange	PC out Popoto IN	5	TXD	Signal OUT	Yellow	PC In Popoto Out
Pin Number	Name	Type	Cable Color	Description																											
1	GND	Ground	Black	Device Ground Pin																											
2	NC																														
3	NC																														
4	RXD	Signal IN	Orange	PC out Popoto IN																											
5	TXD	Signal OUT	Yellow	PC In Popoto Out																											
3	JTAG emulator connection. This connector is useful for software debug on the Popoto.																														
4	<p>This connector is an RS422 serial connector that is typically used for serial communication with Popoto. These lines are wired and brought outside the bottle through the water tight connectors.</p> <p>The pinout of this connector is shown below:</p> 																														
5	<p>This connector is an expansion board that includes a SPI interface and an I2C interface. These interfaces are industry standard interfaces the details of which can be found on the web. This connector is useful for interface with scientific equipment or peripherals that could gather data and either store locally or transmit back using the modem.</p> <p>The pinout for this connection is:</p> 																														



6	Remote power off. This two pin connector can be connected to a magnetic reed switch, or simply brought out from the bottle on spare connector terminals. When these two terminals are shorted, the board is powered down. In the powered down state all processing and interfaces are stopped and the board is 'off' in very low current state >1ma. . When these two terminals are 'open', the board is powered up and runs. It is important to note that the transition from 'off' to 'on' begins the boot cycle for Popoto and this takes about 45 seconds until Popoto is up and running.
7	DC System supply. This is the primary power source for Popoto and it can safely span from 12-18 Volts. In fielded applications this power source is typically a battery pack, however it can be safely powered with a quality DC power supply. The current consumption of Popoto from the power supply varies substantially between receive and transmit mode. While in transmit mode Popoto requires more current as a function of the transmitted power requested. Gross current requirements for an external power supply follows: Receive only < 300 ma Tx less than 1 watts <1 amp Tx up to 100 watts 10 amps
8	Ethernet PHY connection 100 Mb/s
9	SATA connection. For possible connection of external SATA storage
10	Push button system reset. Forces a reboot.
11	DDR volatile memory
12	OMAP L138 processor composed of ARM & DSP

In addition to the items depicted in the table above, it is useful to know of 2 temperature sensors that exist on the digital board. These two sensors are readable by the I2C interface have been strategically located. The sensor at address 1001000x is located near the OMAP processor and gives a quick information of the thermal status in the proximity of the processing unit. The sensor at address 1001001x is located on the periphery of the board near the heatsink away from heat producing components and gives status on the general ambient temperature of the bottle.

2.2 A brief tour of the Analog Board

The analog board performs all the receive signal conditioning from the transducer along with all of the high power transmit capability. It interfaces with the digital board by passing the digitized analog signals over a high speed digital audio bus.

The complete frequency response of the analog board largely driven by the transducer response. The receiver was designed to cover a bandwidth from 20 KHz. through 48 KHz. when paired



with an appropriate transducer. The transmitter and its matching network is more sensitive to frequency variations. A substantial move in center frequency will likely change achievable power unless a carefully selected transducer is available and or the matching network components are adjusted.

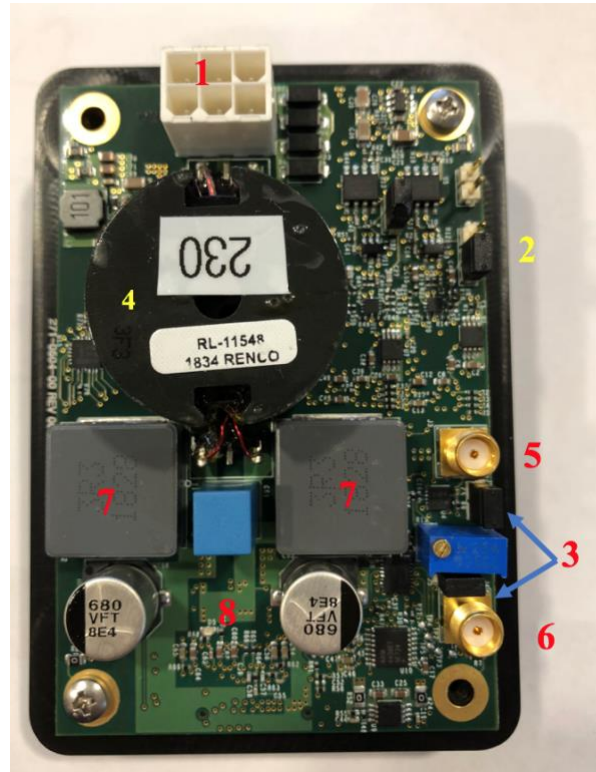


Figure 13 Analog Top Side

Key	Feature
1	Plug for acoustic transducer
2	Jumper for input select {Transducer 1-2; SMA input 2-3}
3	Jumpers to enable SMA amplifier power
4	Transformer for transducer
5	SMA input line level analog
6	SMA output line level analog
7	Power amplifier inductors
8	Transmit LED, active when transmitter power amplifier is enabled

2.3 Mounting tray

The mounting tray shown in **Error! Reference source not found.**, has a three fold purpose:



1. Provide a secure physical mounting for the analog and digital boards that can also securely mount to the bottle.
2. Provide EMI shielding between the active digital board and the sensitive analog board.
3. Provide thermal heat sink capability for the power amplifier during high power transmissions.

2.3.1 Thermal putty

The thermal putty provides a strong thermal interface between the power amplifier and the mounting tray. This interface is critical for high power operation and a high quality thermal compound was selected. Use of Fujipoly XR-Um putty is recommended.

Anytime the analog board is unmounted and remounted it is recommended to change the thermal putty at the Power Amplifier/Mounting tray junction.

2.4 Transducer

The Popoto transducer shown is a potted piezo ceramic ring. This transducer is capable of both receive and transmit operation. The transmit operation is capable of operation in water at powers of 100 Watts {lower powers are suggested for air testing}.

The frequency response of a typical transducer is depicted below:

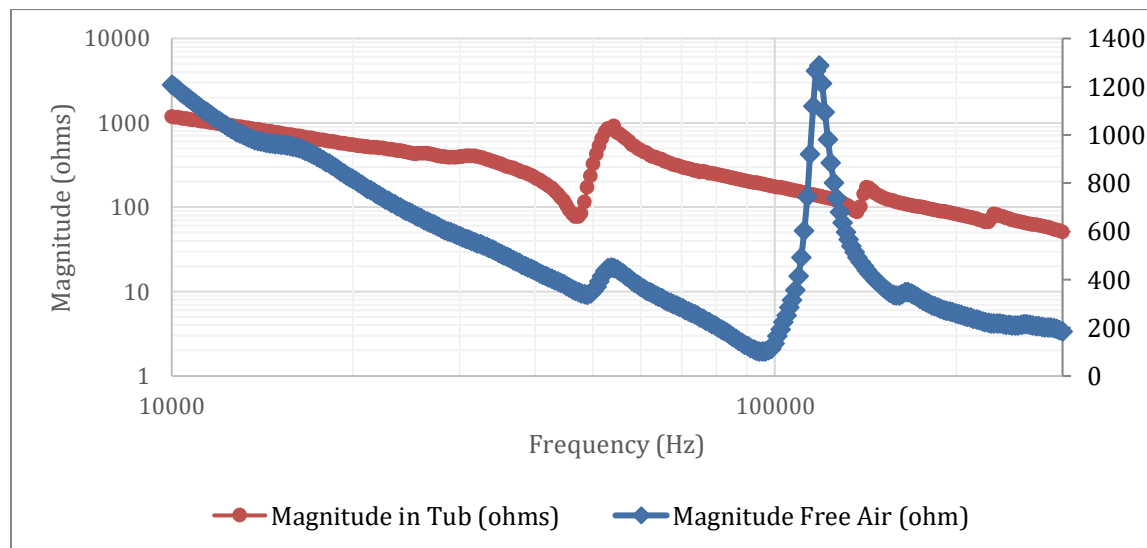


Figure 14 Transducer impedance vs frequency air vs water

Here are some operating guidelines as they relate to transducer operation:



1. When transmitting, always have the transducer plugged in
2. Keep careful control of the ping and range amplitude when operating.
3. When transmitting in air keep powers below 10 watts.
4. Start at low powers (<10 watts) to verify proper operation before jumping to high powers in water.
5. A good low power ping and range command is `ping .1`
6. Receiver performance in water greatly differs from receiver performance in air. Indoor air reflections, multipath are very challenging and will require effort to minimize.
7. Indoor environments such as a laboratory typically have other sources of ultrasound present that will tend to interfere with detections or cause false detections. These can include monitors, LED lighting, proximity detectors used in alarm systems etc. To verify if these are present in your lab environment it is recommended to do a 30 second PCM recording without transmission and analyze the power spectrum of the recording in MATLAB using the `pwelch` function.
8. When setting the transducer directly on a solid surface, i.e. lab bench, the bench can act as a large mechanical pickup device and add to interference. As such it is a good idea to set the transducers on a foam pad or equivalent.

3 The pshell

The pshell is a python command line shell utilizes commands defined in `Popoto.py` to provide a python scriptable command shell containing all of the most useful commands from `Popoto.py`. In addition the command shell provides for help and tab completion for ease of use. Responses from commands are echoed to the command shell along with asynchronous alerts from Popoto.

3.1 Modes of operation

There are two fundamental modes of operation of the pshell, it can be run on the user PC under the PC's local python or it can be run on the python that exists on Popoto OMAP platform. Because communication from pshell to the Popoto is done through IP sockets, this gives the flexibility of running pshell locally on the target or remotely on any PC on the network.

3.2 Requirements for running

python 2.7 (it is already installed on the Popoto hardware)
CMD command shell installed (it is already installed on the Popoto hardware)
CMD2 command shell installed (this gives some added features)

3.3 Invoking pshell

As delivered, the Popoto will invoke the pshell automatically and present a command prompt to the user on the RS-422 port.

3.3.1 The `pshell.init` file

The `pshell.init` file is located in the root directory `/`. This file is a collection of pshell commands that get executed on power up of Popoto.

This file is intended for the user to customize this file and set bootable parameters such as localID, carrier frequency etc. The syntax is normal pshell syntax where line comment character (# first position) and whitespace are ignored.

3.3.2 Invoking pshell from a linux prompt

Although pshell runs automatically at boot. It is possible to terminate the local running pshell process and run the pshell from any python with an IP connection to Popoto.

From the linux command prompt

```
python pshell
```

This will start the up the pshell and you are ready to being typing commands.

3.4 Invoking commands

3.4.1 Help

To gain a complete list of commands at any time simply

3.4.2 Tab Completion

The pshell supports tab completion. Tab completion will also show a list of various options for a particular command.

3.4.3 Commands

This section presents a list of the currently implemented commands. A brief description is presented along with typical invocations.

connect	34
getvaluef	34
quit	34
setvaluei	35
disablemsmlog	35
getvaluei	35
ping	35
range	35
setgainmode	36
shell	36
edit	37
help	37
py	37
recordstart	37
shortcuts	38
enablemsmlog	38
history	38
pyscript	38
recordstop	38



setRate80	39
setRate640	39
setRate1280	39
setRate2560	39
setRate5120	40
setRate10240	40
setclock	40
getclock	40
setcarrier25	41
setcarrier30	41
setcarrier	41
disconnect	41
version	41
netplay	42
netrec	42
playstart	42
playstop	42
Rx	43
chat	43
mips	43
powerdown	43
deepsleep	44
remote	44
upload	44
download	44
multiping	45
transmit	45
transmitJSON	45
transmitJSONFiles	45
ssb	46
setIP	46
getIP	47
ssbtx	48
datamode	48
getPEP	48
startrx	48
getIP	48
load	49
q	49
set	49
setverbosity	49
setvaluef	50
unq	50
sleep	50
remote <on/off>	50



connect

Description:

The connect command is used to connect the pshell with the command socket. This is typically the first command executed in the session of a pshell. A successful connection responds with the list of available parameters.

Invocation:

connect ipaddress port

Examples:

connect localhost 17000

connect 10.0.0.232 17000

getvaluef

Retrieve a floating point value from Popoto. A list of available parameters can be shown by hitting a tab after the command.

Invocation:

getvaluef parameter

Examples:

getvaluef batteryvoltage

getvaluef detectthreshold

quit

Description:

Exit the pshell

Invocation:

quit



setvaluei

Description:

Set an integer parameter value in Popoto. . Tab completion after setvaluei will show the settable parameters.

Invocation:

setvaluei parameter value

Examples:

setvaluei MODEM_Enable 1

setvaluei UPCONVERT_Carrier 30000

disablemsmlog

Description:

Disable the modem statemachine logging.

Invocation:

disablemsmlog

getvaluei

Retrieve an integer value from Popoto. A list of available parameters can be shown by hitting a tab after the command. Tab completion after getvaluei will show the settable parameters.

Invocation:

getvaluei parameter

Examples:

getvaluei MODEM_Enable

getvaluei DOWNCONVERT_Carrier

ping

Send a frequency hopped test message at a prescribed power level.

Invocation:

ping txpowerlevel

{txpower level approximates transmit power in watts}

Examples:

ping 10

ping .3

range



Send a frequency hopped range request message at a prescribed power level. The receiving modem will respond at the same level.

Invocation:

range txpowerlevel

{txpower level approximates transmit power in watts}

Examples:

range .2

range 10

setgainmode

Set the frontend gain control mode of the receiver.

Mode values are {0,1,2} where:

0- low gain channel mode

1- high gain channel mode

2- Autoselect the best channel (default)

Invocation:

setgainmode mode

Examples:

setgainmode 1

setgainmode 2

shell

Execute an OS command from the pshell.

Invocation:

shell oscommand

Examples:

shell ls

{spills a directory}

shell pwd

{shows the current working directory}



edit

Bring up a VI editor from the pshell.

Invocation:

edit <filename>

Examples:

edit

edit myfile.txt

help

Show a list of available pshell commands

Invocation:

help

py

Execute a python command.

Invocation:

py pythoncommand

Examples:

py run("myscript.py")

py print("hello world")

recordstart

Start a pcm recording to disk. Note that the recording destination has two options, the recording can be stored on the Popoto disk, or if a high bandwidth ethernet connection to the Popoto socket exists, it is possible to record directly onto the local PC hardrive.

When recording to the Popoto disk, it is possible to automatically segment files into files of [duration] seconds. For example to record pcm into 60 second files, auto named using time, specify duration as 60.

Invocation:

recordstart filename [duration]

Examples:

```
recordstart mypcmfile.pcm  
recordstart mypcmfile.pcm 60
```

shortcuts

Show a list of available pshell shortcut keys

Invocation:

```
shortcuts
```

enablemsmlog

Description:

Enable the modem statemachine logging in the Popoto.log file on the target hardware.

Invocation:

```
enablemsmlog
```

history

Description:

Display a history of all the pshell commands entered in a session.

Invocation:

```
history
```

pyscript

Description:

Run a python script file.

Invocation:

```
pyscript <script_path> [script_arguments]
```

Examples:

```
pyscript myscript.py 12 8 7
```

recordstop

Stop a pcm currently recording to disk.

Invocation:

```
recordstop
```



setRate80

Description:

Set the local modem to use the 80 bit per second modulation scheme

Invocation:

setRate80

Examples:

setRate80

setRate640

Description:

Set the local modem to use the 640 bit per second PSK modulation scheme

Invocation:

setRate640

Examples:

setRate640

setRate1280

Description:

Set the local modem to use the 1280 bit per second PSK modulation scheme

Invocation:

setRate1280

Examples:

setRate1280

setRate2560

Description:

Set the local modem to use the 2560 bit per second PSK modulation scheme

Invocation:

setRate2560

Examples:

setRate2560

setRate5120

Description:

Set the local modem to use the 5120 bit per second PSK modulation scheme

Invocation:

setRate5120

Examples:

setRate5120

setRate10240

Description:

Set the local modem to use the 10240 bit per second PSK modulation scheme. This has no error correction so be advised.

Invocation:

setRate10240

Examples:

setRate10240

setclock

Description:

set the Realtime clock in the format YYYY.MM.DD-HH:MM;SS

Invocation:

setclock YYYY.MM.DD-HH:MM;SS

Examples:

setclock 2020.01.12-12:00:00

getclock

Description:

get the Realtime clock in the format YYYY.MM.DD-HH:MM;SS

Invocation:

getclock

Examples:

getclock



setcarrier25

Description:

set the modem carrier to 25 kHz

Invocation:

setcarrier25

Examples:

setcarrier25

setcarrier30

Description:

set the modem carrier to 30 kHz

Invocation:

setcarrier30

Examples:

setcarrier30

setcarrier

Description:

set the modem carrier to user specified frequency

Invocation:

setcarrier <hertz>

Examples:

setcarrier. 26000

disconnect

Description:

Disconnect the Popoto modem from the IP socket

Invocation:

disconnect

Examples:

Disconnect

version

Description:

Display the software version and hardware serial number of local modem

Invocation:

version

Examples:

version



netplay

Description:

Plays a baseband or passband file out the transducer using the network sockets

Invocation:

netplay <delresearch file> <BB/PB>

Where BB/PB 1= BaseBand Samples 0 = PassBand Samples

Examples:

netplay mysound.rec 0

netrec

Description:

Records file using the network sockets

use:

Invocation:

netrec <delresearch File> <time in seconds> <BB/PB>

where file BB/PB=1 -> Baseband Recording 0->

Examples:

netrec mysound.rec 60 0

playstart

Description:

Starts a playback of audio from a file on the the local modem's filesystem out the transducer.

Invocation:

playstart <filename> <scale factor>

where filename is the name of the file to play

where scale factor is a floating point gain to apply to the file

Examples:

playstart mysound.pcm 1

playstop

Description:

stop and close an in-process playback

Invocation:

playstop

Examples:

playstop



Rx

Description:

Receive packets and format the output for test purpose

Invocation:

Rx [Verbose Flag]

Verbose Flag = 1 Output SNR and Doppler info

Press any Key to stop Rx mode

Examples:

Rx

chat

Description:

puts Popoto into a character chat mode
ctrl-] to exit

Invocation:

chat

Examples:

chat

mips

Description:

Query the popoto modem to determine internal cycle counts for algorithm .

This is an internal tool used by the developers

powerdown

Description:

puts the omap in low power mode

Invocation:

powerdown

Examples:

powerdown



deepsleep

Description:

Place Popoto into Deep Sleep mode to be awakened by a wake up tone on the acoustic interface.

Invocation:

deepsleep

Examples:

deepsleep

remote

Description:

Sets the pshell in remote mode.

In remote mode a command entered on shell prompt will be executed on the remote modem

NOTE: You cannot issue a transmit command remotely

Invocation:

remote <off>

NOTE: 'remote off' exits from remote mode

Examples:

remote

remote off

upload

Description:

Uploads a file in streaming mode to remote modem. Remote modem should be in download mode to capture the upload to a file.

Invocation:

upload [filename] [power level]

Examples:

upload /home/root/myfile.zip 10

download

Description:

Downloads a file in streaming mode from a remote modem.

Invocation:

upload [filename] [power level]

The transmit power level is not included when invoking, it is assumed that the remote modem will start its upload process imminently.



However, when the power level parameter is included, the command begins by instructing the remote modem to begin the upload process of the specified file automatically. The protocol completes with the download of the file. The file is saved on the local disk with the same path as the specified remote file, however a `.download` extension is appended to that filename signifying an acoustically downloaded file.

Examples:

`download /home/root/myfile.zip 10`

This downloads a file from the remote modem and saves locally to `/home/root/myfile.zip.download`

multiping

Description:

Sends multiple test messages to a remote modem

Invocation:

`multiping <power Watts> <number of pings> <delay in seconds>`

Examples:

`multiping 10 100 60`

transmit

Description:

Transmit a string to the remote modem

Invocation:

`transmit <string>`

Examples:

`transmit hello world`

transmitJSON

Description:

Transmit a JSON encoded message to the remote host.

This is used for sending binary data to the remote.

the structure of the message is

```
{"Payload":{"Data":[<COMMA SEPARATED 8 BIT VALUES>]}}
```

```
{"Payload":{"Data":[1,2,3,4,5]}}
```

sends the binary sequence 0x01 0x02 0x03 0x04 0x05

Invocation:

`transmitJSON`

Examples:

`transmitJSON {"Payload":{"Data":[1,2,3,4,5]}}`

transmitJSONFiles

Description:

transmit a series of JSON messages out of a file.



`transmitJSONFiles <filename> <power> <delay between transmissions> <num
transmissions per packet>
transmitJSONFiles`

Examples:

transmitJSONFiles. Myfiles.txt 10 100 2

`ssb`

Description:

Sets the modem in SSB voice receiver mode

Invocation:

ssb

Examples:

ssb

datamode returns user to data mode from *ssb*

`setIP`

Description:

Sets the modem IP address

Invocation:

setIP IP address

Examples:

setIP 10.0.0.232

datamode returns user to data mode from *ssb*



getIP

Description:

Gets the modem IP address

Invocation:

getIP

Examples:

getIP

Returns:

IPv4 Address: eth0 Link encap:Ethernet HWaddr 22:83:5C:E7:3A:60

inet addr:10.0.0.232 Bcast:10.255.255.255 Mask:255.0.0.0

UP BROADCAST MULTICAST MTU:1500 Metric:1

RX packets:0 errors:0 dropped:0 overruns:0 frame:0

TX packets:0 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1000

RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

Interrupt:33

lo Link encap:Local Loopback

inet addr:127.0.0.1 Mask:255.0.0.0

inet6 addr: ::1%71/128 Scope:Host

UP LOOPBACK RUNNING MTU:65536 Metric:1

RX packets:101 errors:0 dropped:0 overruns:0 frame:0

TX packets:101 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1

RX bytes:11501 (11.2 KiB) TX bytes:11501 (11.2 KiB)

sit0 Link encap:IPv6-in-IPv4

NOARP MTU:1480 Metric:1

RX packets:0 errors:0 dropped:0 overruns:0 frame:0

TX packets:0 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1

RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

ssbtx

Description:

Sets the modem in SSB voice transmit mode PTT depressd

Invocation:

ssbtx

Examples:

ssbtx

datamode returns user to data mode from ssb

datamode

Description:

Exit voice mode and return to data mode

Invocation:

datamode

getPEP

Description:

Get the peak envelop voice power since the last invocation of getPEP. Note that the first getPEP returns zero always. Also note that for a meaningful measure the user should be speaking at a normal volume for several seconds.

Invocation:

getPEP

startrx

Description:

Start up the receiver.

Invocation:

startrx

getIP

Description:

display the Ip address of the popoto modem.

Invocation:

getIP



load

Description:

Runs commands in script file that is encoded as either ASCII or UTF-8 text.

Invocation:

load <file_path>

* file_path - a file path pointing to a script

Script should contain one command per line, just like command would be typed in console.

q

Halt characters being echoed onto the pshell window. This is useful if you need to type something when being bombarded with alerts from Popoto. You can simply quiet the output until you 'unq' which will unquiet it.

Invocation:

q

set

Description:

Set a pshell command shell behavior parameter. These parameters can be listed by issuing a set command with no parameters.

Invocation:

set param value

Examples:

set

set editor vi

set timing true

setverbosity

Description:

Set the verbosity level <0-5> of messages in the pshell. A higher verbosity number yields more messages that may be useful to gain understanding in a debug scenario.

Invocation:

setverbosity <0-6>

Examples:

setverbosity 3

set timing true

setvaluef

Description:

Set a floating point parameter value in Popoto. Tab completion after setvaluef will show the settable parameters.

Invocation:

setvaluef parameter value

Examples:

setvaluef FHMODEM_DetectThresh 18.5

unq

Resume normal pshell reporting

Invocation:

unq

exit

exit the pshell

Invocation:

exit

sleep

Description:

General purpose delay. Wait for a specified delay in seconds

Invocation:

sleep delayinseconds

Examples:

sleep 10

sleep 3

remote <on/off>

Description:

Sets the pshell in remote mode.

In remote mode the command entered on shell prompt will be executed on the remote modem via acoustic communication.

NOTE: You cannot issue a transmit command remotely



Invocation:

Example of changing remote tx power:

remote on

setvalue TxPower 20.

remote off



Popoto **Modem**

3.5 Extending the pshell

One of the best parts of pshell is that it is easy to extend with simple python. For example if you want to make a command that does five ranges spaced by 30 seconds, it is as simple as adding these lines:

```
def do_nranges(self,line):
    for x in range(1,5):
        self.dol.range(.1)
        time.sleep(30.)
```

Note the command name in the pshell would be nranges. With the pshell, you have the power of the python language to create complex commands or specific syntaxes, mappings, command checking etc very quickly and efficiently.

3.6 Set-able and Get-able Parameters of Popoto Modem

3.6.1 System Level Variables

Variable Name:	<u>APP_MODEMSMAOUT</u>
Description:	This value sets whether to send signal data out the SMA port.
Genre:	System
Data Type:	int
Permissions:	Read/Write
Min :	0- Not out SMA
Max :	1- Sent out SMA
Syntax :	{ "Command": "GetValue", "Arguments": "APP_ModemSMAOut int 0"} { "Command": "SetValue", "Arguments": "APP_ModemSMAOut int 1 0"}
Return :	{"APP_ModemSMAOut":value}

Variable Name: [APP_SOCKETBASEDPCM](#)

Description:

This value sets whether to enable socket based PCM or default of A/D D/A PCM

Genre: System

Data Type: int

Permissions: Read/Write

Min : 0- A/D D/A based PCM

Max : 1- Socket based PCM

Syntax :

```
{ "Command": "GetValue", "Arguments": "APP_SocketBasedPCM int 0"}
{ "Command": "SetValue", "Arguments": "APP_SocketBasedPCM int 1 0"}
```

Return :

```
{"APP_SocketBasedPCM":value}
```

Variable Name: [APP_SYSTEMMODE](#)

Description:

This value sets whether the modem is in 0-data mode, 1-SSB Tx, 2- SSB Rx

Genre: System

Data Type: int

Permissions: Read/Write

Min : 0- data mode

Max : 1- SSB Tx mode

2- SSB Rx mode

Syntax :

```
{ "Command": "GetValue", "Arguments": "APP_SystemMode int 0"}
{ "Command": "SetValue", "Arguments": "APP_SystemMode int 1 0"}
```

Return :

```
{"APP_SystemMode":value}
```

Variable Name: [BATTERYVOLTAGE](#)

Description:

This command queries the system battery voltage.



Genre: System

Data Type: float

Permissions: Read

Min : 0.0 Volts

Max : 40.0 Volts

Syntax :

{ "Command": "GetValue", "Arguments": "BatteryVoltage float 0" }

Return :

{ "BatteryVoltage": voltage }



Variable Name: [LEDENABLE](#)

Description:

This value determines if onboard LEDs are 0-disabled, or 1-enabled. It can be useful for power reduction to summarily disable board LEDs.

Genre: System

Data Type: int

Permissions: Read/Write

Min : 0- LEDs disabled

Max : 1- LEDs enabled

Syntax :

```
{ "Command": "GetValue", "Arguments": "LedEnable int 0"}  
{ "Command": "SetValue", "Arguments": "LedEnable int 1 0"}
```

Return :

```
{"LedEnable":value}
```

Variable Name: [LOGGINGLEVEL](#)

Description:

This value sets the level of verbosity for the message **logging**

Genre: System

Data Type: int

Permissions: Read/Write

Min : 0- minimum logging

Max : 5- maximum logging

Syntax :

```
{ "Command": "GetValue", "Arguments": "LoggingLevel int 0"}  
{ "Command": "SetValue", "Arguments": "LoggingLevel int 1 0"}
```

Return :

```
{"LoggingLevel":value}
```




Variable Name: [RNG_SPEEDOFSOUND](#)

Description:

This value sets the speed of sound in meters per second. An accurate measure of the speed of sound is necessary for accurate ranging functionality. Please set this value as a function of the local environment. The default is 1500 meters per second.

Genre: System

Data Type: float

Permissions: Read/Write

Min : 1400 mps

Max : 1600 mps

Syntax :

```
{ "Command": "GetValue", "Arguments": " RNG_SpeedOfSound float 0"}
```

```
{ "Command": "SetValue", "Arguments": " RNG_SpeedOfSound float 1500 0"}
```

Return :

```
{ " RNG_SpeedOfSound":value}
```

Variable Name: [TCPECHO](#)

Description:

This value determines if TCP characters are echoed on the telnet Tx stream

Genre: System

Data Type: int

Permissions: Read/Write

Min : 0- no echo

Max : 1- telnet echo

Syntax :

```
{ "Command": "GetValue", "Arguments": "TCPEcho int 0"}
```

```
{ "Command": "SetValue", "Arguments": "TCPEcho int 1 0"}
```

Return :

```
{ "TCPEcho":value}
```



Variable Name: [TEMP AMBIENT](#)

Description:
This command queries the system local ambient temperature in degrees Celsius.

Genre: System

Data Type: float

Permissions: Read

Val : Degrees C

Syntax :
{ "Command": "GetValue", "Arguments": "Temp_Ambient float 0" }

Return :
{ "Temp_Ambient": voltage }

3.6.2 Receiver Oriented Variables

Variable Name:	<u>DOWNCONVERT CARRIER</u>
Description:	This value sets the receiver downconverter carrier in Hz.
Genre:	Receiver
Data Type:	int
Permissions:	Read/Write
Min :	20000
Max :	59750
Syntax :	{ "Command": "GetValue", "Arguments": "DOWNCONVERT_Carrier int 0" }
	{ "Command": "SetValue", "Arguments": "DOWNCONVERT_Carrier int 25000 0" }
Return :	{ "DOWNCONVERT_Carrier":value }

Variable Name:	<u>GAINADJUSTMODE</u>
Description:	This value sets the mode of the input gain channels. 0-low gain, 1- high gain, 2- auto gain
Genre:	Receiver
Data Type:	int
Permissions:	Read/Write
Val :	0-low ; 1-high ;
	2-automatic mode
Syntax :	{ "Command": "GetValue", "Arguments": "GainAdjustMode int 0" }
	{ "Command": "SetValue", "Arguments": " GainAdjustMode int 2 0" }
Return :	{ " GainAdjustMode":value }

Variable Name:	<u>INBANDENERGY</u>
Description:	This parameter is the smoothed averaged sum of squares of the input passband energy.

Genre:	Receiver
Data Type:	float
Permissions:	Read
Min :	0.
Max :	Max float
Syntax :	{ "Command": "GetValue", "Arguments": "InbandEnergy float 0" }
Return :	{ "InbandEnergy":value }

Variable Name:	<u>INBANDNOISEENERGY</u>
Description:	This parameter is the smoothed averaged sum of squares of the input baseband energy sampled immediately prior to reception of a valid hopped acquisition signal.
Genre:	Receiver
Data Type:	float
Permissions:	Read
Min :	0.
Max :	Max float
Syntax :	{ "Command": "GetValue", "Arguments": "InBandNoiseEnergy float 0" }
Return :	{ "InBandNoiseEnergy":value }

Variable Name:	<u>LOCALID</u>
Description:	This value contains the ID number of the modem. Valid numbers are 0-255 (note 255 is designated as a broadcast ID)
Genre:	Receiver
Data Type:	int
Permissions:	Read/Write
Val :	0 - 254
	255 -broadcast
Syntax :	
	{ "Command": "GetValue", "Arguments": "LocalID int 0" }
	{ "Command": "SetValue", "Arguments": "LocalID int 127 0" }
Return :	
	{ "GainAdjustMode": value }

Variable Name: [MODEM_ENABLE](#)

Description:
This value enables modem processing. When disabled, transmitter and receiver will not process, but recording and playout still operate.

Genre: Receiver/Transmit

Data Type: int

Permissions: Read/Write

Val :
0- modem disable
1-modem enable

Syntax :
{ "Command": "GetValue", "Arguments": "MODEM_Enable int 0"}
{ "Command": "SetValue", "Arguments": "MODEM_Enable int 1 0"}

Return :
{ "MODEM_Enable":value}

Variable Name: [PSK_BNTAPS](#)

Description:
This value contains the number of backwards taps for the PSK equalizer. Note that the number of backwards taps + forward_taps is constrained to be less than 70 taps. The default value is 6

Genre: Receiver/Transmit

Data Type: int

Permissions: Read/Write

Min : 0
Max : <70
NumFwd+NumBwd

Syntax :
{ "Command": "GetValue", "Arguments": "PSK_BnTaps int 0"}
{ "Command": "SetValue", "Arguments": "PSK_BnTaps int 8 0"}

Return :
{ "PSK_BnTaps":value}

Variable Name: [PSK CONSTELLATION](#)

Description:

This value retrieves the 64 latest PSK constellation points

Genre: Receiver

Data Type: Float

Permissions: Read

Min: Min float

Max : Max float

Syntax :

```
{ "Command": "GetValue", "Arguments": "PSK_Constellation float 0"}
```

Return :

```
{ "PSK_Constellation": [v0,v1,v2,v3,...v63]}
```

Variable Name: [PSK FNTAPS](#)

Description:

This value contains the number of forward taps for the PSK equalizer. Note that the number of backwards taps + forward_taps is constrained to be less than 70 taps. The default value is 44

Genre: Receiver/Transmit

Data Type: int

Permissions: Read/Write

Min : 0

Max : NumFwd+NumBwd < 70

Syntax :

```
{ "Command": "GetValue", "Arguments": "PSK_FnTaps int 0"}
```

```
{ "Command": "SetValue", "Arguments": "PSK_FnTaps int 32 0"}
```

Return :

```
{ "PSK_FnTaps": value }
```

Variable Name: [PSK_PDSNR](#)

Description:
This parameter post detection SNR when in PSK modem

Genre: Receiver

Data Type: float

Permissions: Read

Min : 0.

Max : Max float

Syntax :
{ "Command": "GetValue", "Arguments": "PSK_PDSNR float 0" }

Return :
{ "PSK_PDSNR": value }

Variable Name: [PSK_PLL](#)

Description:
This value contains error measured within the PLL when in PSK mode.

Genre: Receiver

Data Type: float

Permissions: Read

Min: 0

Max : MAX float

Syntax :
{ "Command": "GetValue", "Arguments": "PSK_PLL float 0" }

Return :
{ "PSK_PLL": value }

Variable Name: [PSK TAPS](#)

Description:

This value retrieves the backward and forward equalizer taps. Note they are concatenated backward + forward.

Genre: Receiver

Data Type: Float

Permissions: Read

Min: Min float

Max : Max float

Syntax :

{ "Command": "GetValue", "Arguments": "PSK_Taps float 0" }

Return :

{ "PSK_Taps": [b0, b1, b2, bn, f0, f1, ...fm] }

n backward taps, m forward taps

Variable Name: [RANGETIMEOUT_MS](#)

Description:

This value contains the fixed turnaround delay from received range request to transmit range response. This delay is measured in mS. The default value is 15000

Genre: Receiver/Transmit

Data Type: int

Permissions: Read/Write

Min : 0

Max : 60000

Syntax :

{ "Command": "GetValue", "Arguments": "PSK_FnTaps int 0" }

{ "Command": "SetValue", "Arguments": "PSK_FnTaps int 32 0" }

Return :

{ "RangeTimeout_mS": value }

Variable Name: [RXENABLE](#)

Description:

This value is used to enable/disable the receiver processing.

Genre: Receiver

Data Type: int

Permissions: Read/Write

Min : 0- receiver disable

Max : 1- receiver enable

Syntax :

{ "Command": "GetValue", "Arguments": "RxEnable int 0" }

{ "Command": "SetValue", "Arguments": "RxEnable int 1 0" }

Return :

{ " RxEnable":value }

Variable Name: [RXSCRAMBLERMODE](#)

Description:

This value enables or disables the data scrambler for PSK payloads

Genre: Receiver/Transmit

Data Type: int

Permissions: Read/Write

Min : 0- disable PSK payload scrambler

Max : 1- enable PSK payload scrambler

Syntax :

{ "Command": "GetValue", "Arguments": "RxScramblerMode int 0" }

{ "Command": "SetValue", "Arguments": "RxScramblerMode int 1 0" }

Return :

{ " RxScramblerMode":value }

Variable Name: [RXSTATE](#)

Description:

This value is used to enable/disable the analog board receiver board hardware. It is not recommended for users to modify.

Genre: Receiver

Data Type: int

Permissions: Read/Write

Min : 0- receiver hardware disable

Max : 1- receiver hardware enable

Syntax :

```
{ "Command": "GetValue", "Arguments": "rxState int 0" }
{ "Command": "SetValue", "Arguments": "rxState int 1 0" }
```

Return :

```
{ "rxState": value }
```

Variable Name: [SIGNALENERGY](#)

Description:

This parameter is the smoothed averaged sum of squares of the input baseband energy sampled during reception of a valid hopped acquisition signal.

Genre: Receiver

Data Type: float

Permissions: Read

Min : 0.

Max : Max float

Syntax :

```
{ "Command": "GetValue", "Arguments": "SignalEnergy float 0" }
```

Return :

```
{ "SignalEnergy": value }
```

Variable Name: [SNR](#)

Description:

This parameter is the ratio of signal to noise expressed in Db and captured during a valid acquisition.

Genre: Receiver

Data Type: float

Permissions: Read

Min : 0.

Max : Max float

Syntax :
 { "Command": "GetValue", "Arguments": "SNR float 0" }

Return :
 {"SNR":value}

Variable Name: [SSB SQLEVEL](#)

Description:
 This value sets the voice received squelch level. A level of zero corresponds to no squelch. This value is set by a user to just above the background noise to mute the receiver until such time as a signal arrives above the noise.

Genre: Receiver

Data Type: float

Permissions: Read/Write

Min : 0.

Max : 1.

Syntax :
 { "Command": "GetValue", "Arguments": "SSB_SqLevel float 0" }
 { "Command": "SetValue", "Arguments": "SSB_SqLevel float .5 0" }

Return :
 {"SSB_SqLevel":value}

Variable Name:	<u>SSB VOLUME</u>
Description:	This value sets the voice received volume level
Genre:	Receiver
Data Type:	float
Permissions:	Read/Write
Min :	0.
Max :	100.
Syntax :	{ "Command": "GetValue", "Arguments": "SSB_Volume float 0" } { "Command": "SetValue", "Arguments": "SSB_Volume float 10 0" }
Return :	{ "SSB_Volume": value }

3.6.3 Transmitter Oriented Variables

Variable Name: CARRIERTXMODE

Description:
This value enables or disables the carrier only transmit mode. In this mode, regardless of the data sent to the modem, a single tone at the carrier frequency is transmitted. This mode is useful for debug and experimentation.

Genre: Transmitter

Data Type: int

Permissions: Read/Write

Min : 0- (default) modem mode

Max : 1- tone mode

Syntax :
{ "Command": "GetValue", "Arguments": "CarrierTxMode int 0"}
{ "Command": "SetValue", "Arguments": "CarrierTxMode int 1 0"}

Return :
{ "CarrierTxMode":value}

Variable Name: CONSOLEPACKETBYTES

Description:

This value corresponds to the number of bytes received on the telnet console to trigger an autosend of the telnet data. This parameter is intended to make console to console operation easier to use.

Genre: Transmitter

Data Type: int

Permissions: Read/Write

Min : 1

Max : 8192

Syntax :

```
{ "Command": "GetValue", "Arguments": "ConsolePacketBytes int 0"}  
{ "Command": "SetValue", "Arguments": "ConsolePacketBytes int 256 0"}
```

Return :

```
{ "ConsolePacketBytes": value }
```

Variable Name: CONSOLETIMEOUTMS

Description:

This value corresponds to time in milliseconds for the telnet console to trigger an autosend of the telnet data when data is available. This parameter is intended to make console to console operation easier to use.

Genre: Transmitter

Data Type: int

Permissions: Read/Write

Min : 0

Max : 60000

Syntax :

```
{ "Command": "GetValue", "Arguments": "ConsoleTimeoutMS int 0" }  
{ "Command": "SetValue", "Arguments": "ConsoleTimeoutMS int 10000 0" }
```

Return :

```
{ "ConsoleTimeoutMS": value }
```


Variable Name: PAYLOADMODE

Description:

This value corresponds modulation data rate of the Payload portion of the waveform.

Genre: Transmitter

Data Type: int

Permissions: Read/Write

Values : 0- 80bps Frequency Hop

1- 5120 bps PSK

2- 2560 bps PSK

3- 1280 bps PSK

4- 640 bps PSK

5- 10240 bps (uncoded)
PSK

Syntax :

```
{ "Command": "GetValue", "Arguments": "PayloadMode int 0" }
```

```
{ "Command": "SetValue", "Arguments": "PayloadMode int 1 0" }
```

Return :

```
{ "PayloadMode": value }
```

Variable Name:	<u>PEAKENVELOPEPOWER</u>
Description:	This value contains the peak envelope power of the previous SSB transmission. Reading this variable clears the value.
Genre:	Transmitter
Data Type:	float
Permissions:	Read
Min :	0.
Max :	MAX float
Syntax :	{ "Command": "GetValue", "Arguments": "PeakEnvelopePower float 0" }
Return :	{ "PeakEnvelopePower": value }

Variable Name:	<u>PLAYMODE</u>
Description:	This value corresponds places the transmitter waveform player in either passband mode or baseband mode.
Genre:	Transmitter
Data Type:	int
Permissions:	Read/Write
Min :	0- passband mode
Max :	1- baseband mode

Syntax :

```
{ "Command": "GetValue", "Arguments": "PlayMode int 0"}  
{ "Command": "SetValue", "Arguments": "PlayMode int 1 0"}
```

Return :

```
{"PlayMode":value}
```

Variable Name:	<u>REMOTEID</u>
Description:	This value contains the intended receiver ID for the transmission.
Genre:	Transmitter
Data Type:	int
Permissions:	Read/Write
Min :	0
Max :	254 255 (broadcast)
Syntax :	{ "Command": "GetValue", "Arguments": "RemoteID int 0" } { "Command": "SetValue", "Arguments": "RemoteID int 12 0" }
Return :	{ "RemoteID": value }

Variable Name:	<u>SSB_TXPOWER</u>
Description:	This value contains a scalar factor that scales the transmit output prior to transmission.
Genre:	Transmitter
Data Type:	float
Permissions:	Read/Write
Min :	0.
Max :	100.
Syntax :	{ "Command": "GetValue", "Arguments": "SSB_Txpower float 0" } { "Command": "SetValue", "Arguments": "SSB_Txpower float 0.3 0" }
Return :	{ "SSB_Txpower": value }

Variable Name:	<u>SSB_VXLEVEL</u>
Description:	This value contains a scalar that sets the level for a voice activated transmit switch when in voice mode and SSB_VxMode is enabled. This input voice level must be exceeded for the transmitter to self enable, once enabled the transmitter remains on until a prescribed hangover period of silence expires. Note the default value is .005.
Genre:	Transmitter
Data Type:	float
Permissions:	Read/Write
Min :	0.
Max :	1.
Syntax :	{ "Command": "GetValue", "Arguments": "SSB_VxLevel float 0" } { "Command": "SetValue", "Arguments": "SSB_VxLevel float 0.003 0" }
Return :	{ "SSB_VxLevel": value }

Variable Name:	<u>STREAMINGTXLEN</u>
Description:	This value contains the number of bytes to be set when uploading a file.
Genre:	Transmitter
Data Type:	int
Permissions:	Read/Write
Min :	1
Max :	8192
Syntax :	{ "Command": "GetValue", "Arguments": "StreamingTxLen int 0" } { "Command": "SetValue", "Arguments": "StreamingTxLen int 1045 0" }
Return :	{ "StreamingTxLen": value }

Variable Name:	<u>TPASTATE</u>
Description:	This value is an enable variable that is used to powerup or powerdown the transmit final amplifier. It is not recommended for typical users to modify this variable.
Genre:	Transmitter
Data Type:	int
Permissions:	Read/Write
Min :	0
Max :	1
Syntax :	{ "Command": "GetValue", "Arguments": "tpaState int 0" }
	{ "Command": "SetValue", "Arguments": "tpaState int 0 0" }
Return :	{ "tpaState":value }

Variable Name:	<u>TXCHIRPMODE</u>
Description:	This value is an enable flag for setting the transmitter into 'chirp mode.' In this mode the transmitter sends an lfm chirp followed by silence prior to sending the acquisition sequence. This mode is used for characterizing channels by transmitting a known signal and receiving the signal shaped by the channel.
Genre:	Transmitter
Data Type:	int
Permissions:	Read/Write
Min :	0- disabled (default)
Max :	1- enabled
Syntax :	{ "Command": "GetValue", "Arguments": "TxChirpMode int 0" }
	{ "Command": "SetValue", "Arguments": "TxChirpMode int 1 0" }
Return :	{ "TxChirpMode":value }

Variable Name:	<u>TXPOWERWATTS</u>
Description:	This value contains a parameter for the desired power in watts for the transmission. It presumes the transducer has gone through a calibration phase for accurate operation.
Genre:	Transmitter
Data Type:	float
Permissions:	Read/Write
Min :	0.
Max :	100.
Syntax :	{ "Command": "GetValue", "Arguments": "TxPowerWatts float 0" } { "Command": "SetValue", "Arguments": "TxPowerWatts float 30 0" }
Return :	{ "TxPowerWatts":value }

Variable Name:	<u>TXTIMEOUT_MS</u>
Description:	This value is the number of milliseconds that the transmitter can remain transmitting a packet before a timeout will occur. The timeout terminates the transmission state.
Genre:	Transmitter
Data Type:	int
Permissions:	Read/Write
Min :	0
Max :	6000000 {default}
Syntax :	{ "Command": "GetValue", "Arguments": "TxTimeout_mS int 0" } { "Command": "SetValue", "Arguments": "TxTimeout_mS int 15000 0" }
Return :	{ "TxTimeout_mS":value }

Variable Name:	<u>UPCONVERT CARRIER</u>
Description:	This value sets the transmitter upconverter carrier in Hz.
Genre:	Transmitter
Data Type:	int
Permissions:	Read/Write
Min :	20000
Max :	59750
Syntax :	{ "Command": "GetValue", "Arguments": "UPCONVERT _Carrier int 0" } { "Command": "SetValue", "Arguments": "UPCONVERT _Carrier int 25000 0" }
Return :	{ "UPCONVERT _Carrier":value }

Variable Name:	<u>UPCONVERT OUTPUTSCALE</u>
Description:	This value sets the transmitter upconverter carrier in Hz. Note setting the transmitter power via the API will be overwritten if this message is sent.
Genre:	Transmitter
Data Type:	float
Permissions:	Read/Write
Min :	0.
Max :	10.
Syntax :	{ "Command": "GetValue", "Arguments": "UPCONVERT _ OutputScale float 0" } { "Command": "SetValue", "Arguments": "UPCONVERT _ OutputScale float 1. 0" }
Return :	{ "UPCONVERT _ OutputScale":value }

Variable Name:	<u>RNG TA DELAYMS</u>
-----------------------	---------------------------------------

Description:

This value is the number of milliseconds that the transmitter waits from reception of a range request. The default value is 5000

Genre: Ranging

Data Type: int

Permissions: Read/Write

Min : 3000

Max : 8000

Syntax :

```
{ "Command": "GetValue", "Arguments": "RNG_TA_DelayMs int 0" }
```

```
{ "Command": "SetValue", "Arguments": "RNG_TA_DelayMs 7000 0" }
```

Return :

```
{ "RNG_TA_DelayMs": value }
```

4 Updating the firmware

4.1 Introduction

Firmware updates are accomplished through the ethernet port. An overview of the process involves three steps:

1. SCP an update tar file
2. Extract the tar file
3. Run the update shell script

4.2 Details on how to update the firmware

Requirements:

- Network Connection
- Pshell connection (Rs 422)
- [Secure Shell/Copy utility](#)
- Update_<version_num>.tar

4.2.1 Upload Procedure:

Note: By default the Popoto boardset is shipped without a root password. In each of the examples below all remote operations happen without password entry. If you have added a root password to your system, please enter the password when prompted by the SSH and SCP utilities in the steps below.

Step 1:

Connect Popoto to ethernet connection.

Step 2:

Determine or Set the Popoto IP Address

From the pshell issue [getIP](#)

```
Popoto-> getIP
IPv4 Address: eth0      Link encap:Ethernet  HWaddr 2E:A4:4D:D2:40:82
      inet addr:10.0.0.232  Bcast:10.255.255.255  Mask:255.0.0.0
      inet6 addr:
2603:3005:82a:8000:2ca4:4dff:fed2:4082%71/64 Scope:Global
      inet6 addr: fe80::2ca4:4dff:fed2:4082%71/64 Scope:Link
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:639 errors:0 dropped:0 overruns:0 frame:0
TX packets:159 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:57888 (56.5 KiB)  TX bytes:26702 (26.0 KiB)
Interrupt:33
```

In this example the IP address is 10.0.0.232.

To change the IP address of the Popoto, issue the [setIP](#) command from pshell.

Step 3:

Confirm connection to the Popoto Mini's network connection using the ping command from your local computer's command window.

```
% ping 10.0.0.232
PING 10.0.0.232 (10.0.0.232): 56 data bytes
64 bytes from 10.0.0.232: icmp_seq=0 ttl=64 time=0.853 ms
64 bytes from 10.0.0.232: icmp_seq=1 ttl=64 time=1.198 ms
64 bytes from 10.0.0.232: icmp_seq=2 ttl=64 time=0.969 ms
^C
```

Step 4:

Using a secure copy utility, such as OpenSSH's `scp`, located on your local computer, copy the update file to the Popoto's root directory

```
% scp Update_2.6.0.tar root@10.0.0.232:/
```

Step 5

Shell into the Popoto, using an ssh utility

```
ssh root@10.0.0.232
```

You should receive a prompt like:

```
root@popoto:~#
```

Step 6

From the `root@popoto:` prompt, change directories to the root directory, and untar the update file previously uploaded.

```
root@popoto:~# cd /
root@popoto:/# tar xvf Update_2.6.0.tar
```

This will create (or overwrite) the following 2 files

```
Update.sh
Update.tgz
```

Step 7

Execute the Update shell command to install the newest version.

```
root@popoto:/# ./Update.sh
```

This will generate the following output.

```
Update.sh
Version.txt
boot/
boot/dolphin.dtb
home/
home/root/
home/root/popoto.py
home/root/pshell
home/root/popoto_app
lib/
lib/firmware/
lib/firmware/platform.out
pshell.init
version.txt
Connection to 10.0.0.232 closed by remote host.
Connection to 10.0.0.232 closed.
```

At this point the Popoto unit will reboot, and come up with the new firmware version installed.

In the pshell window (RS-422) you should end up at a prompt that says

```
Welcome to the Popoto modem Shell!
```

```
Communicating Naturally
```

```
Popoto-> {"Info ":"Popoto Modem Version <New Version Number and informational tag> "}
```

5 Diagnostics/Logs

5.1 Introduction

The FOAM architecture has built in logging support to enable diagnostics and debug of any in field problems. The logging consists of a rolling file based log file, along with options for saving the passband PCM data. The log file is useful for determining message flow and state transitions, and the PCM passband logging is useful for diagnosing signal processing and signal quality issues.

5.2 Popoto log

5.2.1 Introduction

The `Popoto.log` is a diagnostic logfile which is updated as the `Popoto_app` runs, keeping track of message and logic flows within the system. This logfile has the following properties.

- The Log file is Leveled: All logs are assigned a severity level in the code, and by changing the output filter, only logs greater than a set severity level are displayed.
- The log file is Timestamped: Each log message is tagged with a millisecond accurate realtime clock stamp, as well as a PCM Count timestamp. The Realtime clock is useful for comparing transmit to receive times between units, and the PCM clock gives an indication of when a message is displayed with respect to reception or transmission of acoustic messages.
- The Logfile is Rolling: Each time the Popoto app is started, the previous log file is added to a list of 10 preceeding log files. So that in the `Popoto_app` directory we have `Popoto.log`, `Popoto.log.1`, `Popoto.log.2`.... `Popoto.log.10` where `Popoto.log` is the current logfile, and `Popoto.log.1` is the most recent log file preceding this logfile.

5.2.2 Location

On the target hardware the `Popoto.log` file is found in the `/home/root` directory. On the PC-Based Linux simulation, the `Popoto.log` is found in the `/tmp` directory. In order to allow more than one Popoto image to run on a pc, the base-port number is appended to the `Popoto.log` filename. For example:

```
/tmp/Popoto.log.17000
```

Corresponds to a Popoto image run at a base port of 17000

Or

```
/tmp/Popoto.log.18000
```

Corresponds to a Popoto image run at a base port of 18000.

5.2.3 Logging Levels

Each log message is assigned a logging level from 0 to 7. Lower log levels are more severe, and higher log levels are increasing details. The log levels are defined as follows:

0. logERROR
1. logWARNING
2. logINFO
3. logDEBUG
4. logDEBUG1
5. logDEBUG2
6. logDEBUG3
7. logDEBUG4

By default all log messages with a logging level of logINFO or lower are written to the log. To increase or decrease the log level issue the `SetValue LoggingLevel int <Level> 0` command

Or from the pshell:

```
setvaluei LoggingLevel <level>
```

To get the current logging level, issue the `GetValue LoggingLevel int 0` command,

Or from the pshell:

```
getvaluei LoggingLevel
```

5.2.4 MSM Logs

The Modem State Machine has a built in logging mechanism that can be connected to the Popoto.log file. This allows the user to see events, and state transitions as the modem state machine operates. To enable the MSM logs, send the command `EnableMSMLogs`. Or, from the pshell:

```
enablemsmlogs
```

To disable logs, send the `DisableMSMLogs` command.

Or from the pshell:

```
disablemsmlogs
```

5.3 PCM Logging

5.3.1 Introduction

The Popoto system incorporates a means for logging the inbound PCM signals as seen on the A/D. This logging mechanism is useful for diagnosing system problems. Since the

PCM signals that are logged are exactly what is presented to the Demodulator, it possible to “re-run” a test condition, to determine the signal parameters or noise environment. Two methods of logging are provided to the user:

1. TCP Socket Based Logging
2. Target File Logging

Each of these methods produces a data stream of packets that are formatted as follows:

Count	Data type	Description
1	32 Bit unsigned int	PCM Counter. Gives the current PCM counter. Should increase by 640 each frame. A skip in this count indicates lost data
1	32 Bit unsigned int	Status Word. Currently 0 indicates High Gain Channel, and 1 indicates low gain channel
640	32 Bit Floating point	PCM Samples. All normalized to High Gain Receive Level.

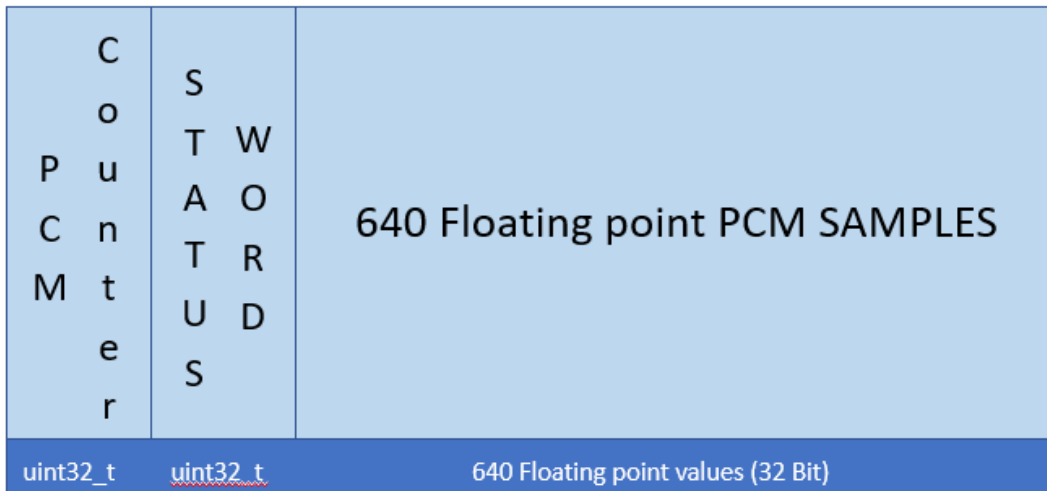


Figure 15: Format of a single PCM Log Packet. These packets are transmitted on the TCP PCM Recording socket.



Figure 16: The PCM Packets are sent one after the other to the TCP Socket or to the Target log file.

5.3.2 Socket based PCMLogs

The Popoto system opens a TCP Server at baseport+2 (17002 default) which continually streams PCM Packets as described above. Both the Popoto.py and Popoto.m interface classes have functions to read that socket and log the data to the local pc.

From the pshell

```
recordstart <Filename> local
```

will start the recording in the current working directory.

To stop the recording:

From the pshell:

```
recordstop
```

5.3.3 Target File based PCM Logs

The Popoto system provides a command to store the received pcm locally. By sending the RecordFileStart <FileName> command, the user can start logging data to the local SD card.

If the filename is specified without a path, it will be recorded in /home/root. All other paths should be complete paths. Wild cards are not parsed.

From the pshell:

```
recordstart WaterTestCapeCodCanal2_20.pcm
```

will begin a recording on the Popoto unit in the /home/root directory

To stop the recording, a RecordFileStop command can be sent on the command socket.

Or, from the pshell:

```
recordstop
```

A Matlab™ utility: rPCMDData() is provided in the test/MATLAB GUI directory. This utility can read a file logged by the pshell or by the Target recording, and returns 3 arrays, the PCM data, the PCM Counter(sequence number) and the status word.

5.3.4 Notes:

It is important to realize that PCM recording generates data very quickly. Each packet is 642 * 4 Bytes long, and 160 packets are generated per second. This results in a file that grows at 410,810 bytes per second, or roughly 1.5 G Bytes per hour.

5.4 pshell Logging

The pshell provides a log of all commands and status responses for a pshell session. This is useful for capturing the results of tests, or to evaluate the responses and commands that were run. pshell logs are size-limited, and rotate. These logs can be found in the directory that the pshell was run in.

6 Appendix

6.1 The Acoustic Message Header

Every acoustic packet contains an eight byte header packet. Some types of acoustic packets are only a header, while others contain a subsequent payload packet.

bits	Field	Purpose	Format
0-7	Message Type	Identify between Packet, Packet with payload, ranging etc. MessageIDs 0 - Data 128 -Range Response 129 -Range Request 130 -Status	8 Bit MessageID
8-15	SenderID	ID of the transmitting modem	0x0 – 0xfe - ID
16-23	ReceiverID	The intended ID of the destination receiver.	0x0 – 0xfe - ID 0xff = Broadcast message
24-31	TxPower	The transmitted scale factor as entered by the transmitting modem	Transmit power level as a Q8 scale value
32-47	PayloadInfo	If a the present message does not contains a payload, then this field is 0. If a payload follows, the bits are assembled according to the payloadinfo fields described below.	Number of bytes
48-55	Reserved	Reserved for future expansion	
56-64	CRC	Parity check 8 bit CRC of the header	

6.2 Payload Structure

If header bytes 4 and 5 are not zero, modulated payload data will immediately follow the modulated header data. The payload is described by the 16 bits in the payload info field of the header as follows:

Header Byte 4							
7	6	5	4	3	2	1	0
Plen7	Plen6	Plen5	Plen4	Plen3	Plen2	Plen1	Plen0

Header Byte 5							
15	14	13	12	11	10	9	8
Mod4	Mod3	Mod2	Mod1	Mod0	Stream	Plen9	Plen8

The length of the payload in bytes is set by the 10 bits of the Plen field. Although the field contains 10 bits, the payload size is capped by the software to a maximum of 256 bytes.

Bits 11-15 of byte 5 of the header contain the modulation employed for the payload. Popoto uses the following enumerated modulations:

Mod Bitfield Value	Modulation	Rate
0	Frequency Hopped FSK	80 bps
1	Phase Shift Keying	5120 bps
2	Phase Shift Keying	2560 bps
3	Phase Shift Keying	1280 bps
4	Phase Shift Keying	640 bps
5	Phase Shift Keying	10240 bps (uncoded)

When large files are transmitted, it is more efficient to transfer the file in streaming mode. When streaming mode is invoked, the bit 10, of header byte 5 is set to indicate streaming mode. In streaming mode, the payload length Plen indicates the number of 255 byte frames which follow before another header transmission. All 255 byte packet remainders are handled by the software automatically.