PEPyCO:
PETALO Python Console for DAQ control

Version 1.1
15th March 2019

PEPyCO is a linux terminal based control software for L1 development. It is fully implemented in Python 2.7 inspired in previous Dr. Aliaga Matlab functions. This console makes use of ethernet sockets in semi duplex mode for data transfer and basically allows reading and writing of L1 internal Hardware and Software configuration registers. Some extra functionality has been added for command batch execution and command history.

1. Configuration

PEPyCO configuration parameters are stored in *"PETALO_COMM.json"* using JSON standard. Feel free to modify this parameters as required in your environment using your favorite text editor.

PARAMETER	INFORMATION	DEFAULT VALUE
buffer_size	Buffer size of TCP socket transactions	1024
port	Port used for TCP socket	9116
localhost	Local Host IP (not required)	158.42.34.154
ext_ip	L1 DAQ board IP	158.42.34.143

2. Communication Protocol

2.1 Command Identifier

Communication is always carried out in 32-bit words. The general format for a data frame is the following:

Word #1	Command ID	Destination
Word #2	Number of parameters (N)	
Word #3	Parameter #1	
Word #4	Parameter #2	
...		

Command ID – Table Section 3

Table 1. General Command Frame Structure

Commands sent from PC to DAQ have an even number (bit 0 = 0), and responses to commands from DAQ to PC have the same number plus 1 (bit 0 = 1). Responses may also contain a Command ID from the following list if an error occurred:

Error flag	Numeric Value
ERR_BAD_PACKET	-1
ERR_INVALID_DESTINATION	-2
ERR_INVALID_COMMAND	-3

Table 2. Error flags in Communication Establishment

The “Destination” field indicates which of the DAQ boards the command should be relayed to, or the response originates from. If only one board is used, it will be 0.

2.3 Frame Length

The number of parameters (N) indicates the length of the frame: it is equal to N+2 words, 32-bit each. The minimum frame length is 2 words (8 bytes). Often, response frames will have at least one parameter, and if they only have one then it will indicate status: 0 if successful, or a negative value if an error occurred.

2.2 Connection Establishment

The DAQ currently only accepts connection to one host at the same time. On connection attempt, the DAQ will return a response packet with Command ID = 0001 and either 1 or 2 parameters:

Parameter #1	Status
Parameter #2	IP address of connected host

Table 3. DAQ response during connection establishment

The first parameter will be 0 (STA_CONNECTION_ACCEPT) if connection was successful, or 1 (STA_CONNECTION_REJECT) if another host is already connected. In that case, the second parameter will contain the IP address of the host PC that is connected to the DAQ.

Once the connection is established TCP socket remains opened until any side closes it. This fact implies a semiduplex communication process, since PC side will act always as master element. A Full Duplex communication will be considered for future versions.

3. List of Commands

COMMAND ID	INFO	PARAM #1	PARAM #2	RESPONSE	PARAM #1	PARAM #2
"SOFT_REG_W" CODE:[2]	Modifies the value of a software register.	Register ID	Value	**"SOFT_REG_W_r" CODE:[3]**	Register ID Error Flag *	--
"SOFT_REG_R" CODE:[4]	Reads the value of a software register.	Register ID	--	**"SOFT_REG_R_r" CODE:[5]**	Register ID Error Flag *	Value
"HARD_REG_W" CODE:[6]	Modifies the value of a hardware register.	Register ID	Value	**"HARD_REG_W_r" CODE:[7]**	Register ID --- ERR_INVALID_REGISTER [-4]	--
"HARD_REG_R" CODE:[8]	Reads the value of a hardware register.	Register ID	--	**"HARD_REG_R_r" CODE:[9]**	Register ID --- ERR_INVALID_REGISTER [-4]	Value
"PLL_REG_W" CODE:[10]	Modifies the value of a LMK04828 PLL register.	Register ID	Value	**"PLL_REG_W_r" CODE:[11]**	Register ID --- ERR_NOT_AVAILABLE [-7]	--
"PLL_REG_R" CODE:[12]	Reads the value of a LMK04828 PLL register.	Register ID	--	**"PLL_REG_R_r" CODE:[13]**	Register ID --- ERR_NOT_AVAILABLE [-7]	Value
"I2C" CODE:[14]	Generic I2C command on the I2C bus connected to the power regulators on the mezzanine. Variable size supported.	**n_PARAMS (n bytes written to I2C) (see format of PARAMS)**		**"I2C_r" CODE:[15]**	**n_PARAMS (n bytes read from I2C)**	

ERROR FLAG	VALUE
ERR_INVALID_REGISTER	-4
ERR_READ_ONLY	-5
ERR_WRITE_ONLY	-6
ERR_NOT_AVAILABLE	-7

*Table 1. * Error Flags*

BITS	INFO
10	Issue Sr after this byte
9	Do not issue ACK after this byte (last byte to be read)
8	=1 means this byte is to be read, not written
7-0	Data

Table 4. I2C PARAMS format

4. List of Software Registers

GROUP	REG	ACCESS	DESCRIPTION
0 System Diagnostics and Status	000	RO	Board ID
	001	RO	Hardware version
	002	RO	Firmware version
	003	RW	Console verbosity level
	004	RO	Session counter
	005	RO	Command counter
	006	RW	Default timeout (ms)
1 Peripheral Control	000	RW	LED 0 Status
	001	RW	LED 1 Status
	002	RW	LED 2 Status
	003	RW	LED 3 Status
	004	RW	LED 4 Status
	005	RW	LED 5 Status
	006	RW	LED 6 Status
	007	RW	LED 7 Status
	008	RO	DIP Switch 0 Position
	009	RO	DIP Switch 1 Position
	010	RO	DIP Switch 2 Position
	011	RO	DIP Switch 3 Position
	012	RO	DIP Switch 4 Position
	013	RO	DIP Switch 5 Position
	014	RO	DIP Switch 6 Position
	015	RO	DIP Switch 7 Position
	016	RO	Push button 1 Status
1 Peripheral Control	017	RO	Push button 2 Status
	018	RO	Push button 3 Status
	019	RO	Push button 4 Status
	020	RW	Freq divisor for PLL SPI bus
	021	RW	Freq divisor for ADC SPI bus
	022	RW	Freq divisor for I2C SPI bus
2 Temp Sensor	000 -- 015	RO	Raw data word read from corresponding LTC2498 ADC channel 0 to 15
	006 -- 031	RO	Voltage value read from corresponding LTC2498 ADC channel 0 to 15, in mV

5. PEPyCO Command Line Console

Installation

Copy all Gitlab link contents to a directory in your file system.

Check execution permission for *DAQ_control.py*.

Write *./DAQ_control.py* in your favorite shell terminal and enjoy

Close session with ‘CRT+c’

Description and Features

PEPyCO allows single and batch mode command execution and also makes use of a Linux style history system that will make your life a lot easier. For single command execution follow this syntax (please notice brackets and quotation marks):

Command Format:	**[“COMMAND_ID”, DAQ_ID, [ARG1, ARG2,...]]**
Example:	["SOFT_REG_R", 0, [0x00010006]]
	This will read LED 6 Status
Example:	["SOFT_REG_W", 0, [0x00010006, 1]]
	This will switch LED 6 on

History function will store last 1000 commands in a hidden file (*“.petalo_hist”*) in program installation directory. The control of the history feature is similar to other Linux shells like *bash* or *cshell*. Use UP & DOWN arrow keys to navigate through history commands and LEFT & RIGHT arrow keys to edit and modify your command.

Last session commands (send and received) will be stored in *“petalo_log.txt”* in JSON format.

For batch mode execution, first create your batches using a standard text editor. Commands must be introduced in .JSON format (including number of parameters etc.).

HINT: You can use “petalo_log.txt” to extract the commands you need for your batches.

PROBLEM: JSON standard doesn’t support hexadecimal format so addresses and data must be stored as integers

Follow this syntax in order to invoke batch execution:

Batch execution:	**"run *batch_name.txt*"**
Example:	“run enciende_LEDS.txt”
	This batch will switch on all your LEDs