|  |
| --- |
|  |
|  |
| Instructions for the author: Headings present in the document should not be deleted |
|  |
|  |
|  |
|  |
| EPICS tpg300 Module Technical Documentation |
|  |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
| Author | Affiliation | Editor | Approver |
| Klemen Strniša | Cosylab | Daniel Piso Fernandez | Garry Trahern |
|  |  |  |  |

Document revision history

|  |  |  |
| --- | --- | --- |
| Version | Reason for revision | Date |
| 1.0 | Initial version | 2013-09-04 |
| 1.3 | Updated for bugfixes | 2014-01-27 |

Table of Contents

1. Glossary 1

2. Introduction 2

2.1 Scope 2

2.2 References 2

3. Hardware Description 3

3.1 The TPG 300 3

4. Architecture Design 4

4.1 High-level Architecture Overview 4

4.2 TPG 300 Protocol File 4

4.3 EPICS Records 8

4.3.1 tpg300\_common.template 8

4.3.2 tpg300\_sensor.template 12

4.3.3 tpg300\_function.template 13

4.3.4 tpg300.db 16

4.4 Software Version 16

5. User Manual 17

5.1 Installation and Configuration 17

5.1.1 Required Software 17

5.1.2 RPMs 17

5.1.3 RPM Installation 17

5.1.4 Hardware Configuration 17

5.1.5 Using The tpg300 module in Your Application 18

5.2 Operation Manual 19

5.2.1 Demo Application 19

5.2.2 Opening the GUI 19

List of figures

[Figure 1: Main and pressure sensor configuration OPI 20](#_Toc410128083)

[Figure 2: Switching function OPI 21](#_Toc410128084)

[Figure 3: Pressure readback OPI 21](#_Toc410128085)

List of TABLES

[Table 1: Abbreviations 1](#_Toc410128086)

[Table 2: TPG 300 protocol file commands 8](#_Toc410128087)

[Table 3: tpg300\_common.template records 12](#_Toc410128088)

[Table 4: tpg300\_common.template macros 12](#_Toc410128089)

[Table 5: tpg300\_sensor.template records 13](#_Toc410128090)

[Table 6: tpg300\_sensor.template macros 13](#_Toc410128091)

[Table 7: tpg300\_function.template records 15](#_Toc410128092)

[Table 8: tpg300\_function.template macros 15](#_Toc410128093)

[Table 9: tpg300.db macros 16](#_Toc410128094)

# Glossary

|  |  |
| --- | --- |
| Abbreviation | Definition |
| CB | Control Box |
| CODAC | Control, Data Access and Communication |
| CSS | Control System Studio |
| EGU | Engineering Units |
| EPICS | Experimental Physics and Industrial Control System |
| ESS | European Spallation Source |
| GUI | Graphical User Interface |
| IOC | Input Output Controller |
| PV | EPICS Process Variables |

Table 1: Abbreviations

# Introduction

## Scope

This document provides technical documentation for the EPICS module that offers support for the Pfeiffer Vacuum TPG 300 vacuum gauge controller. Architecture design, database templates with required macro definitions and installation instructions are described in detail.

## References

List of references:

1. Total pressure gauge controller TPG 300 Operating manual
2. StreamDevice, <http://epics.web.psi.ch/software/streamdevice/>
3. AsynDriver, <http://www.aps.anl.gov/epics/modules/soft/asyn/>
4. EPICS R3.14 Channel Access Reference Manual

# Hardware Description

## The TPG 300

The main properties of the TPG 300 vacuum gauge controller are described in the device operating manual (1).

# Architecture Design

## High-level Architecture Overview

The software support is built on top of the StreamDevice EPICS module (2) which is a framework for supporting communication with serial devices.

The general architecture consists of a StreamDevice protocol file that describes the structure of the serial protocol (commands and responses). The contents of the protocol file are referenced by EPICS records that send commands and receive responses.

The low level communication (serial over Ethernet, RS232 or similar) is handled by the AsynDriver framework (3).

## TPG 300 Protocol File

The following commands are specified in the protocol file. The definition and syntax of these commands is described in the device operating manual (1).

|  |  |  |  |
| --- | --- | --- | --- |
| Command | Description | Parameters | Return value |
| get\_puc | Query PE Underrange control status. | None. | Integer as record value. |
| set\_puc | Set PE Underrange control status. | Integer record value. | None. |
| get\_units | Query the units used for pressure readout. | None. | Integer as record value. |
| set\_units | Set the units used for pressure readout. | Integer record value. | None. |
| get\_save\_params | Query which set of parameters is currently stored in non-volatile memory. | None. | Integer as record value. |
| set\_save\_params | Save a set of parameters into non-volatile memory. | Integer record value. | None. |
| get\_baud | Query the baud rate of the serial connection. | None. | Integer as record value. |
| get\_version | Query the version of the serial connection protocol supported by the device. | None. | String as record value. |
| get\_slot\_1 | Query the first expansion board slot contents. | None. | String as record value. |
| get\_slot\_2 | Query the second expansion board slot contents | None. | String as record value. |
| get\_slot\_3 | Query the third expansion board slot contents | None. | String as record value. |
| get\_mode | Query measurement channel status.  Must be used by used by a record of type calcout. | None. | Four integer return values as fields of the record:   * A – A1 status * B – A2 status * C – B1 status * D – B2 status |
| set\_mode | Set measurement channel status.  Must be used by used by a record of type calcout. | Four fields of the record should contain:   * A – A1 status * B – A2 status * C – B1 status * D – B2 status | Integer as field E of the record, representing the status of command execution (success or reason for failure). |
| get\_filter | Query smoothing filter time constant.  Must be used by used by a record of type calcout. | None. | Four integer return values as fields of the record:   * A – A1 filter * B – A2 filter * C – B1 filter * D – B2 filter |
| set\_filter | Set smoothing filter time constant.  Must be used by used by a record of type calcout. | Four fields of the record should contain:   * A – A1 filter * B – A2 filter * C – B1 filter * D – B2 filter | Integer as field E of the record, representing the status of command execution (success or reason for failure). |
| get\_function | Query switching function assignment and switching thresholds.  Must be used by used by a record of type calcout. | Switching function as string. Options are:   * 1 * 2 * 3 * 4 * A * B | Floating point as field A of the record, representing the lower threshold.  Floating point as field B of the record, representing the upper threshold.  Integer as filed C of the record, representing the assignment of the switching function to a specific measurement channel. |
| set\_function | Set switching function assignment and switching thresholds.  Must be used by used by a record of type calcout. | Switching function as string. Options are:   * 1 * 2 * 3 * 4 * A * B   Floating point as field A of the record, representing the lower threshold.  Floating point as field B of the record, representing the upper threshold.  Integer as filed C of the record, representing the assignment of the switching function to a specific measurement channel. | Integer as field D of the record, representing the status of command execution (success or reason for failure). |
| get\_func\_stat | Query status of switching functions. | None. | Four integer return values as fields of the record:   * A – function 1 * B – function 2 * C – function 3 * D – function 4 * E – function A * F – function B |
| get\_pressure | Readback pressure measured by a specific measurement channel. | Measurement channel as string. Options are:   * A1 * A2 * B1 * B2   Name of the record that will contain the status of measurement channel status after response is received. | Floating point as record value, representing the pressure reading in chosen units.  The record that was passed as the second parameter contains an integer value, representing the status of the measurement channel |

Table 2: TPG 300 protocol file commands

## EPICS Records

### tpg300\_common.template

This template defines the database with records used to control and monitor the global parameter of the device.

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| $(PREFIX):PUC | bo | Set PE Underrange control status. Executes protocol command *get\_puc*, see Table 2. |
| $(PREFIX):PUC-RBV | bi | Query PE Underrange control status. Executes protocol command *get\_puc*, see Table 2. |
| $(PREFIX):UNITS | mbbo | Set the units used for pressure readout Executes protocol command *set\_units*, see Table 2. |
| $(PREFIX):UNITS-RBV | mbbi | Query the units used for pressure readout. Executes protocol command *get\_units*, see Table 2. |
| $(PREFIX):BAUD-RBV | stringin | Query the baud rate of the serial connection. Executes protocol command *get\_baud*, see Table 2. |
| $(PREFIX):VERSION-RBV | stringin | Query version of the protocol file. Executes protocol command *get\_version*, see Table 2. |
| $(PREFIX):SLOT1-RBV | stringin | Query the first expansion board slot contents. Executes protocol command *get\_slot\_1*, see Table 2. |
| $(PREFIX):SLOT2-RBV | stringin | Query the second expansion board slot contents. Executes protocol command *get\_slot\_2*, see Table 2. |
| $(PREFIX):SLOT3-RBV | stringin | Query the third expansion board slot contents. Executes protocol command *get\_slot\_3*, see Table 2. |
| $(PREFIX):SAVE-PARAMS | mbbo | Save the current set of parameters or the default ones to non-volatile memory. Executes protocol command *set\_save\_params*, see Table 2. |
| $(PREFIX):SAVE-PARAMS-RBV | mbbi | Query which set of parameters is stored in non-volatile memory. Executes protocol command *get\_save\_params*, see Table 2. |
| $(PREFIX):MODE | calcout | Set the state of all measurement channels. Executes protocol command *set\_mode*, see Table 2.  This record pulls values that are currently set in records that hold state settings for each measurement channel separately. It is processed when any of those record processes. |
| $(PREFIX):MODE-STAT | mbbi | This record reads the information about the command execution status from the $(PREFIX):MODE record. At all times it contains the latest command status for the command *set\_mode*, see Table 2. |
| $(PREFIX):MODE-RBV | calcout | Query the state of all measurement channels. Executes protocol command *get\_mode*, see Table 2.  This record triggers the processing of $(PREFIX):MODE-FO. |
| $(PREFIX):MODE-FO | seq | Processes the records that hold readbacks of each measurement channel status separately. |
| $(PREFIX):FILT | calcout | Set the smoothing filter time constant for all measurement channels. Executes protocol command *set\_filter*, see Table 2.  This record pulls values that are currently set in records that hold filter constants for each measurement channel separately. It is processed when any of those record processes. |
| $(PREFIX):FILT-STAT | mbbi | This record reads the information about the command execution status from the $(PREFIX):FILT record. At all times it contains the latest command status for the command *set\_filter*, see Table 2. |
| $(PREFIX):FILT-RBV | calcout | Query the smoothing filter time constant of all measurement channels Executes protocol command *get\_filter*, see Table 2.  This record triggers the processing of $(PREFIX):FILT-FO. |
| $(PREFIX):FILT-FO | seq | Processes the records that hold readbacks of each measurement channel filter constant separately. |
| $(PREFIX):FUNC-STAT-RBV | calcout | Query status of all switching functions. Executes protocol command *get\_func\_stat*, see Table 2.  This record triggers the processing of $(PREFIX):FUNC-STAT-FO. |
| $(PREFIX):FUNC-STAT-FO | seq | Processes the records that hold readbacks of each switching function status separately. |
| $(PREFIX):SCAN-RATE | seq | Processes pressure readback records for all measurement channels and switching function state records effectively setting a global rate for pressure and switching function state readbacks. |
| $(PREFIX):ASYN | asyn | Asyn record whose only purpose is to monitor connection status in the database. |
| $(PREFIX):RECNCT | calcout | Record that processes its output link each time the connection is established. |
| $(PREFIX):CNCT-FO | seq | Processed each time the connection to the device is established.  It initializes all setting output records to the values currently set on the device. It also triggers processing of all setting readback records. |
| $(PREFIX):RBV-FO | seq | Processes all setting readbacks except switching function parameters. |
| $(PREFIX):SP-RBV-FO | seq | Processes settings related to switching function parameters. |
| $(PREFIX):FILT-CNCT-FO | seq | Decides when to initialize filter time constant setting records with readback values. |
| $(PREFIX):FILT-INIT | calcout | Decides when to initialize filter time constant setting records with readback values. |
| $(PREFIX):FILT-INIT-FO | seq | Initializes filter time constant setting records with readback values. |
| $(PREFIX):MODE-CNCT-FO | seq | Decides when to initialize measurement channel state setting records with readback values. |
| $(PREFIX):MODE-INIT | calcout | Decides when to measurement channel state setting records with readback values. |
| $(PREFIX):MODE-INIT-FO | seq | Initializes measurement channel state setting records with readback values. |

Table 3: tpg300\_common.template records

The following macros must be defined to successfully load the tpg300\_common.template:

|  |  |
| --- | --- |
| Macro | Description |
| PREFIX | Name prefix. |
| TPG\_PORT | Asyn port name of the underlying Asyn port driver that handles the low level communication. |
| SCAN\_RATE | The rate at which pressure readbacks for all measurement channels are queried from the device. |

Table 4: tpg300\_common.template macros

### tpg300\_sensor.template

This template defines the database with records used to control and monitor the parameters relevant for a single measurement channel. This also includes the actual pressure measurements.

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| $(PREFIX):$(SENSOR)-PRES-RBV | ai | Query the current pressure measurement for the measurement channel. Executes protocol command *get\_pressure*, see Table 2. |
| $(PREFIX):$(SENSOR)-PRES-STAT | mbbi | This record contains the measurement channel status. |
| $(PREFIX):$(SENSOR)-MODE | mbbo | Set the state for this measurement channel. |
| $(PREFIX):$(SENSOR)-MODE-RBV | mbbi | Status readback of this measurement channel. |
| $(PREFIX):$(SENSOR)-FILT | mbbo | Set the smoothing filter time constant for this measurement channel. |
| $(PREFIX):$(SENSOR)-FILT-RBV | mbbi | Smoothing filter time constant readback of this measurement channel. |

Table 5: tpg300\_sensor.template records

The following macros must be defined to successfully load the tpg300\_sensor.template:

|  |  |
| --- | --- |
| Macro | Description |
| PREFIX | Name prefix. |
| TPG\_PORT | Asyn port name of the underlying Asyn port driver that handles the low level communication. |
| SENSOR | Which measurement channel to control. Valid options are [A1, A2, B1, B2]. |
| SOURCE | The record field to connect to when reading and data from records that execute the *get\_mode* and *get\_filter* commands. Valid options are [A, B, C, D].  The value of this macro is completely determined by the value of the SENSOR macro. The mapping [SENSOR->SOURCE] must always be [A1->A, A2->B, B1->C, B2->D]. |

Table 6: tpg300\_sensor.template macros

### tpg300\_function.template

This template defines the database with records used to control and monitor the parameters relevant for a single switching function.

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| $(PREFIX):SP$(FUNCTION) | calcout | Set the thresholds and measurement channel assignment for this switching function. Executes protocol command *set\_function*, see Table 2.  This record pulls values that are currently set in records that hold the thresholds and assignment settings separately. It is processed when the assignment setting record processes. |
| $(PREFIX):SP$(FUNCTION)-STAT | mbbi | This record reads the information about the command execution status from the $(PREFIX):SP$(FUNCTION) record. At all times it contains the latest command status for the command *set\_function*, see Table 2. |
| $(PREFIX):SP$(FUNCTION)-LOW | ao | Lower threshold setting for this switching function. |
| $(PREFIX):SP$(FUNCTION)-HIGH | ao | Higher threshold setting for this switching function. |
| $(PREFIX):SP$(FUNCTION)-ASSIGN | mbbo | Assignment setting for this switching function. Triggers protocol command execution. |
| $(PREFIX):SP$(FUNCTION)-RBV | calcout | Query the threshold and assignment settings for this switching function. Executes protocol command *get\_function*, see Table 2. |
| $(PREFIX):SP$(FUNCTION)-FO | seq | Processes the records that hold threshold and assignment settings for this switching function. |
| $(PREFIX):SP$(FUNCTION)-LOW-RBV | ai | Lower threshold setting for this switching function readback. |
| $(PREFIX):SP$(FUNCTION)-HIGH-RBV | ai | Higher threshold setting for this switching function readback. |
| $(PREFIX):SP$(FUNCTION)-ASSIGN-RBV | mbbi | Assignment setting for this switching function readbnack. |
| $(PREFIX):SP$(FUNCTION)-CNCT-FO |  | Decides when to initialize threshold and assignment setting records with readback values. |
| $(PREFIX):SP$(FUNCTION)-INIT | calcout | Copies the value from threshold and assignment setting readbacks into appropriate setter records. |
| $(PREFIX):SP$(FUNCTION)-INIT-FO | seq | Helper data fanout for above. |
| $(PREFIX):SP$(FUNCTION)-STAT-RBV | bi | Readback of the status for this switching function. |

Table 7: tpg300\_function.template records

The following macros must be defined to successfully load the tpg300\_function.template:

|  |  |
| --- | --- |
| Macro | Description |
| PREFIX | Name prefix. |
| TPG\_PORT | Asyn port name of the underlying Asyn port driver that handles the low level communication. |
| FUNCTION | Which measurement channel to control. Valid options are [1, 2, 3, 4, A, B]. |
| SOURCE | The record field to connect to when reading and data from the record that execute the *get\_function* command. Valid options are [A, B, C, D, E, F].  The value of this macro is completely determined by the value of the FUNCTION macro. The mapping [FUNCTION->SOURCE] must always be [1->A, 2->B, 3->C, 4->D, A->E, B->F]. |

Table 8: tpg300\_function.template macros

### tpg300.db

This is a convenience database that defines no new records but only instantiates all the available templates for a single TPG 300 controller. This means the following templates with the following macro values:

* tpg300\_common.template
  + Instantiated once with no macro expansion
* tpg300\_sensor.template
  + Instantiated 4 times with SENSOR=[A1, A2, B1, B2] and SOURCE=[A, B, C, D]
* tpg300\_function.template
  + instantiated 6 times with FUNCTION=[A, B, C, D, E, F]and SOURCE=[A, B, C, D, E, F]

The macros that are left undefined (and therefore have to be defined when this database is loaded in an IOC startup script) are:

|  |  |
| --- | --- |
| Macro | Description |
| PREFIX | Name prefix. |
| TPG\_PORT | Asyn port name of the underlying Asyn port driver that handles the low level communication. |
| SCAN\_RATE | The rate at which pressure readbacks for all measurement channels are queried from the device. |

Table 9: tpg300.db macros

## Software Version

The software support was developed using:

* EPICS base R3.14.12.3
* AsynDriver 4.21
* StreamDevice 2.6

If you are using a different version of the EPICS base or any of the modules check the release notes first for possible changes.

# User Manual

## Installation and Configuration

### Required Software

* Scientific Linux 6.3 64bit
* CODAC 4.1

### RPMs

* codac-core-4.1-epics-tpg300.rpm

Software support module.

* codac-core-4.1-epics-tpg300-doc.rpm

Documentation.

* codac-core-4.1-epics-tpg300-opi.rpm

Engineering GUI.

* codac-core-4.1-epics-tpg300-src.rpm

Software support module sources.

* codac-core-4.1-epics-tpg300-sample.rpm

Sample IOC application.

* codac-core-4.1-epics-tpg300-sample-src.rpm

Sample application sources.

### RPM Installation

1. Open Terminal: Applications -> System tools -> Terminal
2. Install the driver packages.

$ sudo yum install <rpm>

For example to install all packages type:

$ sudo yum install codac-core-4.1-epics-tpg300\*

### Hardware Configuration

The TPG 300 controller is equipped with a serial connection. The controller can therefore be connected to the IOC directly via a serial port or through a serial-to-Ethernet converter.

The type of the low level connection determines how the IOC startup script is configured and how the connection can be tested.

In case of a direct serial connection the connection can be tested with a serial terminal for example the Unix tool *minicom*. In case of an Ethernet connection the Unix *netcat* tool can be used.

### Using The tpg300 module in Your Application

1. In your IOC’s <top>/configure/RELEASE file set path to the required modules.

ASYN=${EPICS\_MODULES}/asyn

STREAMDEVICE=${EPICS\_MODULES}/streamdevice

TPG300=${EPICS\_MODULES}/tpg300

1. In your application’s src/Makefile add modules database definitions and support libraries.

tpg300-sample\_DBD += base.dbd

tpg300-sample\_DBD += asyn.dbd

tpg300-sample\_DBD += drvAsynIPPort.dbd

tpg300-sample\_DBD += drvAsynSerialPort.dbd

tpg300-sample\_DBD += stream.dbd

tpg300-sample\_LIBS += asyn

tpg300-sample\_LIBS += stream

1. Configure the path where the application searches for protocol files in your startup script. In CODAC all protocol files are installed in a standard location.

epicsEnvSet("STREAM\_PROTOCOL\_PATH", "${CODAC\_ROOT}/protocol")

1. Configure the Asyn port driver that handles the underlying low level connection to the controller. If the controller is connected to serial-to-Ethernet converter the underlying connection is handled by the AsynIPPort driver.

drvAsynIPPortConfigure(TPG, <ip\_address:ip\_port>)

If the controller is connected to the IOC directly via a serial port then the underlying connection is handled by the AsynSerialPort driver instead.

drvAsynSerialPortConfigure(…)

Consult the AysnDriver documentation (3) for details on how to configure the AsynSerialPort driver.

1. Configure macros and load the tpg300.db database (see chapter 4.3.4) in your startup script.

dbLoadRecords("${EPICS\_MODULES}/tpg300/db/tpg300.db", "PREFIX=TPG300,TPG\_PORT=TPG,SCAN\_RATE=1 second")

## Operation Manual

### Demo Application

Optionally a demo IOC and application that is using the epics-tpg300 module can be installed.

1. Open Terminal and install sample application.

$ sudo yum install codac-core-4.1-epics-tpg300-sample

1. Start the sample IOC.

$ tpg300-sample-ioc start

1. Connect to the sample IOC

$ tpg300-sample-ioc connect

1. Verify that the sample IOC has started successfully. The IOC output should be similar to:

#!../../bin/linux-x86\_64/tpg300-sample

## You may have to change tpg300-sample to something else

## everywhere it appears in this file

< envPaths

epicsEnvSet("ARCH","linux-x86\_64")

epicsEnvSet("IOC","ioctpg300-sample")

epicsEnvSet("TOP","/opt/codac-4.1/apps/tpg300-sample")

epicsEnvSet("EPICS\_BASE","/opt/codac-4.1/epics/base")

epicsEnvSet("STREAM\_PROTOCOL\_PATH", "/opt/codac-4.1/protocol")

cd /opt/codac-4.1/apps/epics-tpg300-sample

## Register all support components

dbLoadDatabase "dbd/tpg300-sample.dbd"

tpg300\_sample\_registerRecordDeviceDriver pdbbase

drvAsynIPPortConfigure(TPG, 192.168.127.254:4001)

## Load record instances

dbLoadRecords("/opt/codac-4.1/epics/modules/tpg300/db/tpg300.db", "PREFIX=TPG300,TPG\_PORT=TPG,SCAN\_RATE=1 second")

cd /opt/codac-4.1/apps/epics-tpg300-sample/iocBoot/ioctpg300-sample

iocInit

Starting iocInit

############################################################################

## EPICS R3.14.12.3 $Date: Mon 2012-12-17 14:11:47 -0600$

## EPICS Base built Aug 20 2013

############################################################################

iocRun: All initialization complete

epics>

### Opening the GUI

1. Open Terminal and install GUI.

$ sudo yum install codac-core-4.0-epics-tpg300-opi

1. Start CSS: Applications -> CODAC 4.1 Tools -> CSS
2. From CSS menu select Window – Open Perspective – Other and select OPI Runtime.
3. In the Navigator panel open: CSS -> opi -> boy -> epics-tpg300 –> tpg300.opi
4. Select “TPG300” in the Combo Box in the top right corner of the screen.

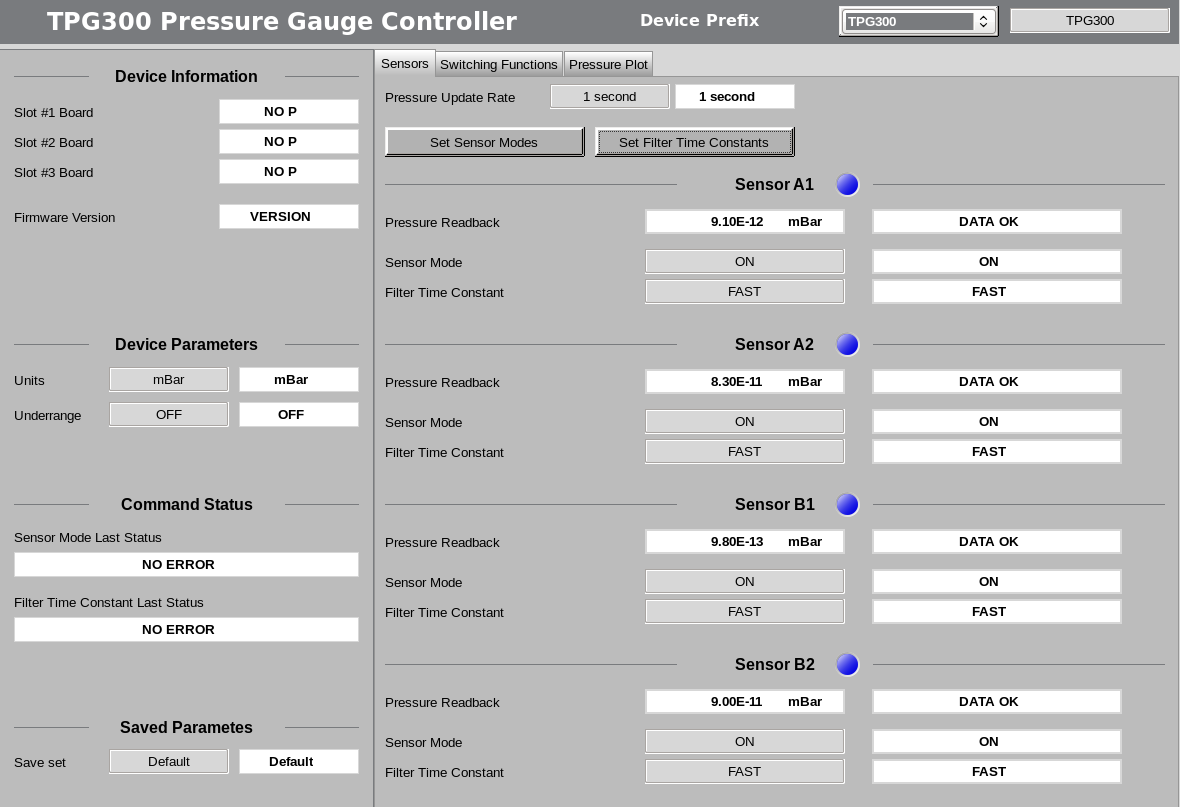


Figure 1: Main and pressure sensor configuration OPI

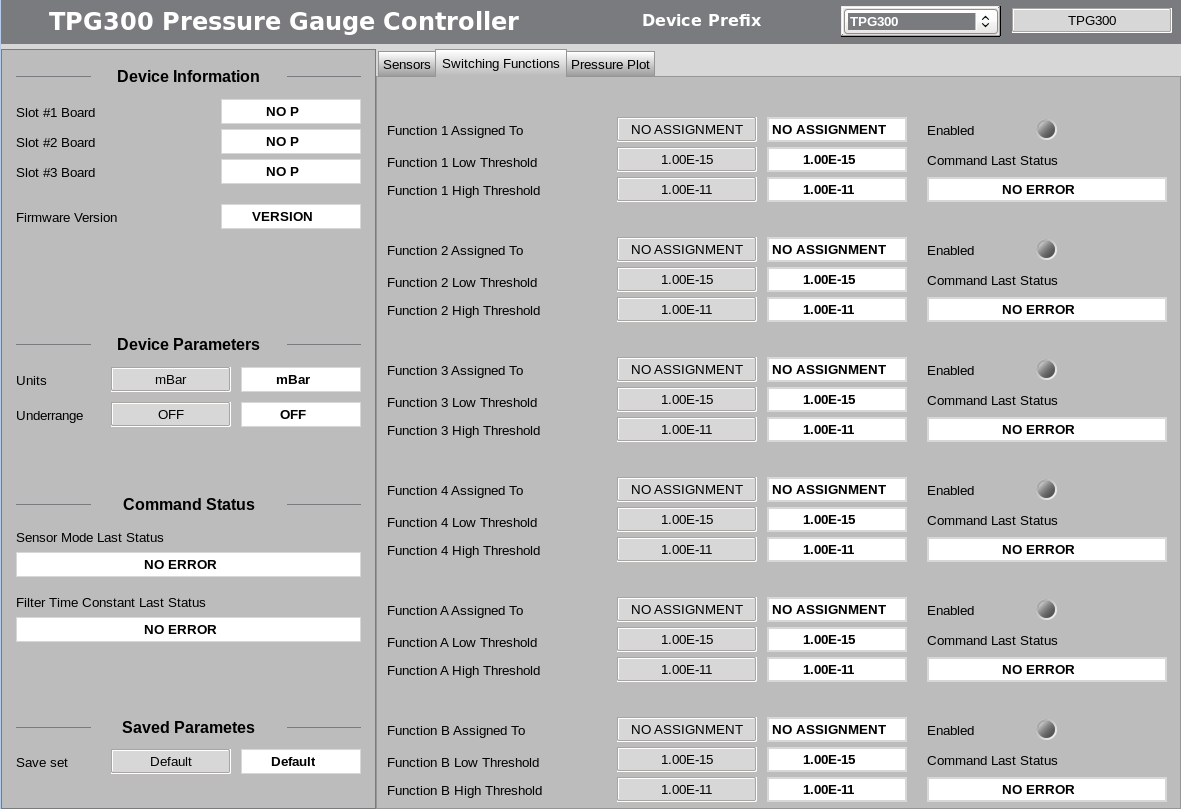


Figure 2: Switching function OPI

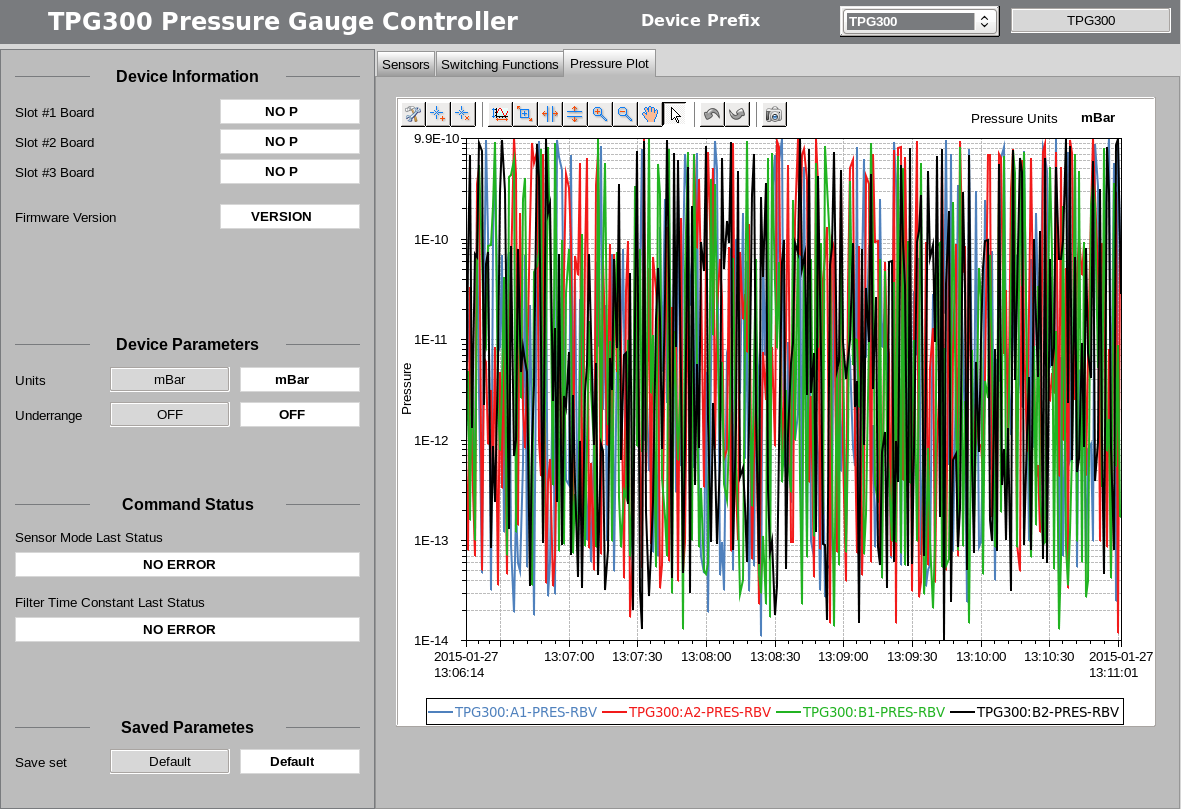


Figure 3: Pressure readback OPI