QphoX CryoSwitch Controller

Application Note

July 18, 2023

Version 1.0

Table of Contents

Connections and Ports information

The QphoX CryoSwitch Controller is specifically designed for driving inductive loads at room or cryogenic temperatures.

The controller supports a maximum of 4 ports (version dependent) with 7 pins per port. The following figure shows the back plate and pinout of the controller.

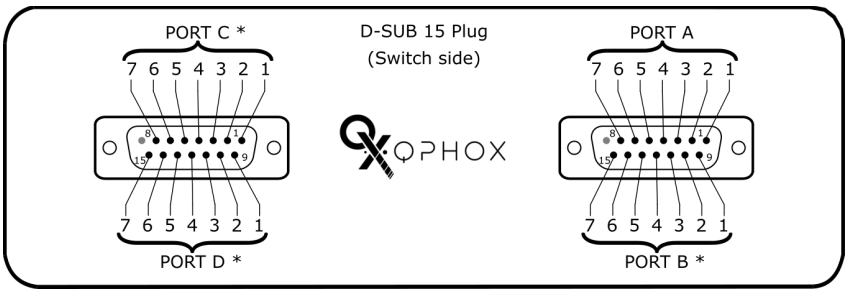

Figure 1: Controller output side

For example, if an SP6T cryogenic relay/actuator is used, the connections will depend on the specific model. Two different connection examples are shown in figure 2:

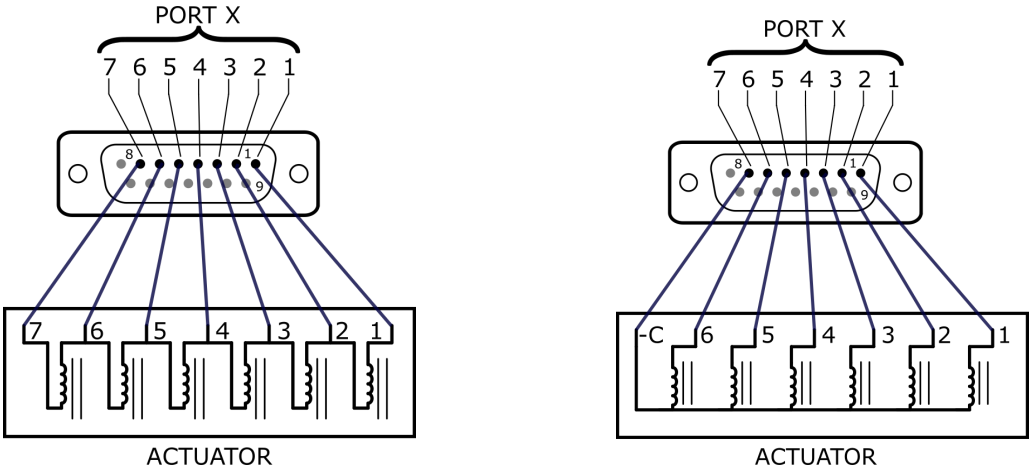

Figure 2: Application example

As seen in the left panel of figure 2, 7 pins are used for driving a 6 contact actuator. Each pair of pins (1-2, 2-3…, 6-7) is used to drive a single relay contact of the actuator. In the right panel, 6 contact pins and a common pin are used for driving the individual actuator contacts.

Internal Block Diagram

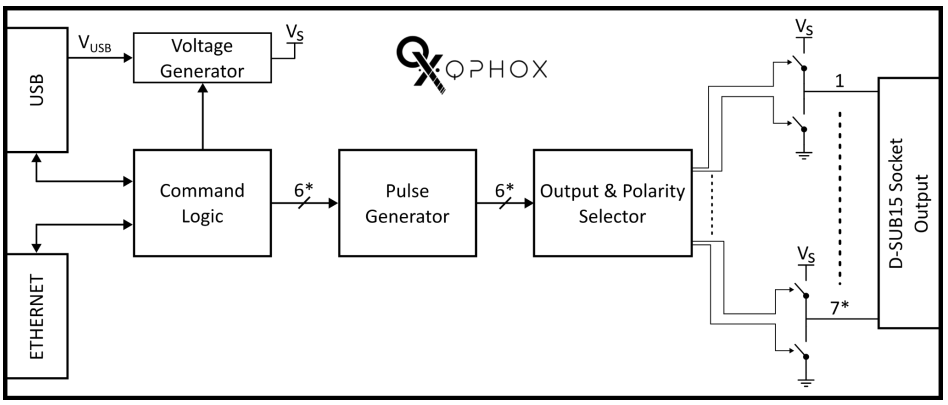

Figure 3: Internal block diagram

Typical application case

Background

The CryoSwitch Controller pulsing features use precise voltage and timing control to reduce heat dissipation. This way every user can optimize power dissipation while engaging the actuator.

Traditional electro-mechanical relays utilize an applied electromagnetic field in order to actuate a mechanical switch and offer high isolation and low cross-talk. These switches consist of an electromagnet in the form of a coil and a moving ferromagnetic plate that responds to the field in order to open or close the switch contact. The magnetic field is proportional to the coil drive current which must exceed a threshold in order to fully engage the actuator. Different temperature environments (e.g. Millikelvin or room temperature), connectors, and cabling resistance will influence the equivalent line resistance (R_{line}) between the Controller and the switch, hence limiting the current for a certain voltage pulse. Figure 4 shows the equivalent diagram of a typical application including the line resistance. While in this figure R_{line} is presented as a single resistor, it is important to note that R_{line} is an equivalent for the entire system, thus a pulse configuration that works well for a room temperature test setup may not be optimal for a cryogenic setup where minimal power dissipation is critical.

By changing the output voltage and pulse width, it is however possible to optimize the pulse current and compensate for the effective line resistance for either environment.

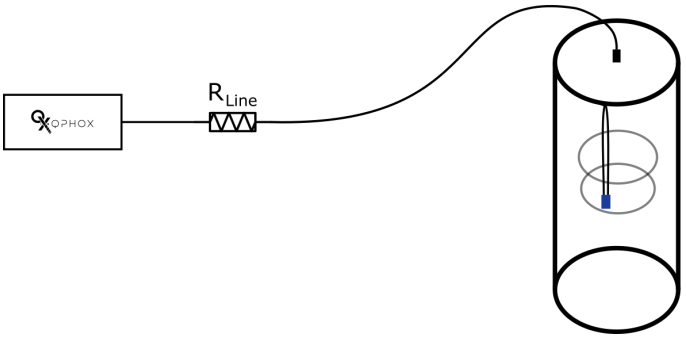

Figure 4: Typical cryogenic setup diagram

Power dissipation at Cryogenic Temperature

Estimating power dissipation or heating during switching can be challenging due to various factors. These factors include the refrigerator's cooling power and thermal mass, internal line resistance, temperature-dependent cooling power, etc. Consequently, the resulting heating can range from sub-mK to several mK per switching action. It is important to consider each switching action as energy dissipation within the fridge, rather than power, as power isonly delivered over a certain period of time.

One way of estimating the energy dissipated by the switching action is by continually actuating the switch at a constant rate and measuring the temperature once thermal equilibrium is reached. When constant switching is performed, the power delivered to the fridge will be on average the same, causing the temperature to rise until it matches the average cooling power at that temperature.

In a similar manner, employing one of the fridge heaters to execute the same action allows us to measure the heating power necessary to attain an identical temperature equilibrium. Dividing the measured power by the switching frequency provides us with an estimation of the energy per switching.

In our specific case, QphoX performed test measurements inside a Bluefors LD250 dilution refrigerator with the actuator (Radiall R583423141) mounted on the mixing chamber (MC) plate. The measured resistance between one of the actuator terminals was 560 Ohm at room temperature and 7 Ohm at cryogenics.

Using an output voltage of 5V, which is the lowest voltage the controller supports, we find that we need to apply each pulse for at least 10ms to reliably switch, resulting in a single switch temperature increase of 0.8mK at 69mK base temperature.

After performing the previously described test, we attained an equilibrium temperature of 107mK with a switching frequency of 0.2Hz. Using the MC heater, we achieved the same temperature with a heating power of 346 µW. Figure 5 shows the heating power vs temperature for our specific test.

Therefore, the estimated switching energy can be calculated as:

$$E_d = \frac{P_d}{f_s} = \frac{346\mu W}{0.2Hz} = 1.73\frac{mJ}{pulse}$$

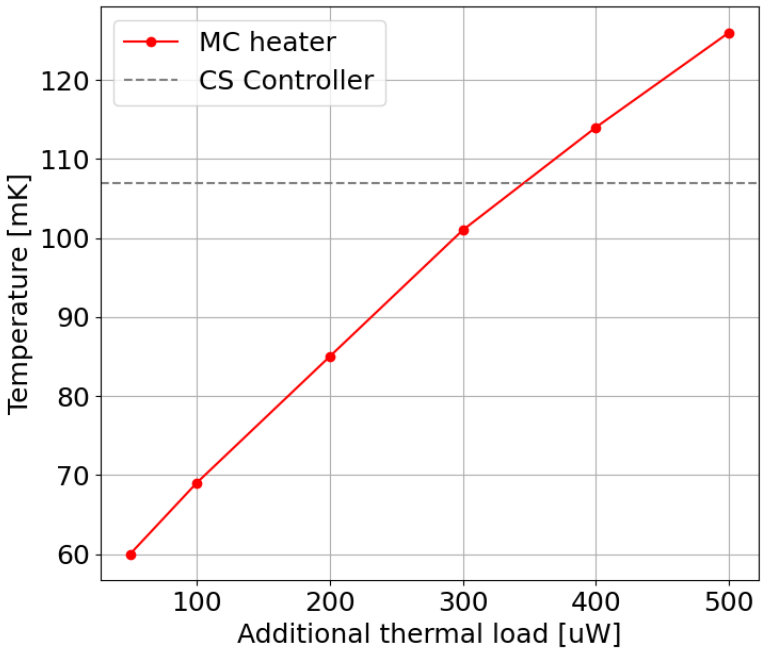

Figure 5: Heating power vs. temperature

Actuator current waveform

Figure 6 shows a typical current waveform at room temperature.

When the pulse starts, the coil is energized and the rising current exerts a force on the actuator. When the force on the actuator exceeds the retaining force, the actuator starts to move. The motion of the actuator (usually made from a ferromagnetic material) induces a back electromagnetic force in the coil, reducing the current between the controller pins. As the movement continues, the current continues to drop until the actuator reaches the final position.

Once the actuator is at rest, the current starts to rise, saturating the coil until it reaches its final DC current (given by the equivalent line resistance and pulse voltage). This part of the switching is not strictly necessary, therefore it is called the excess pulse. Although one might try to reduce the pulse width to avoid the excess pulse and therefore further power dissipation, it is difficult to know beforehand what the pulse duration should be. As discussed earlier, this depends on the fridge wiring and temperature. Having said that, pulse widths of 15-20ms are usually sufficient to reliably engage the actuator.

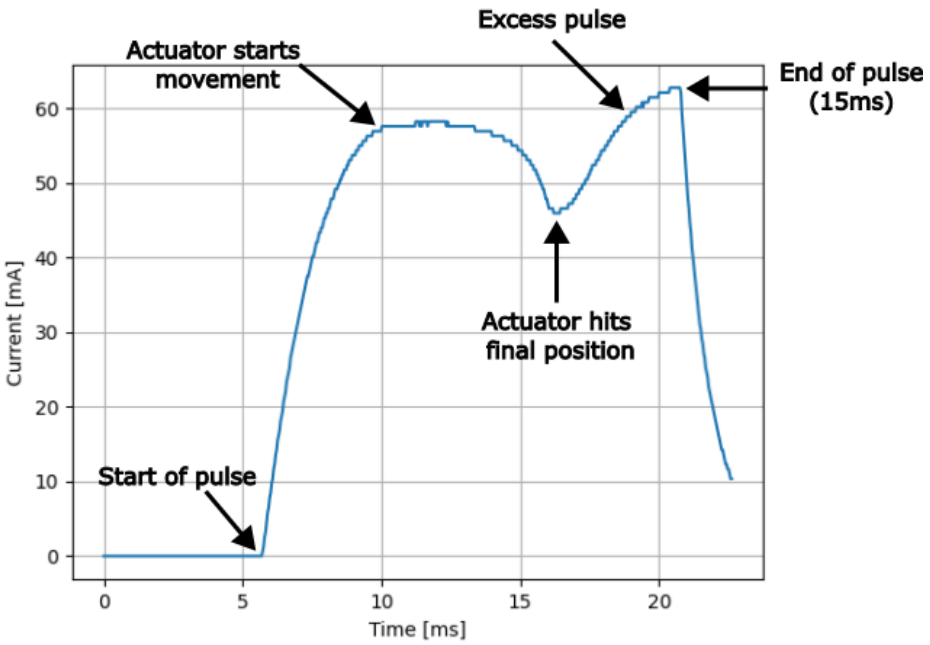

Figure 6: Normal operation waveform

Overcurrent protection

The CryoSwitch Controller features two overcurrent protection modes. The first one occurs when chopping is enabled. Chopping clips the current to the maximum set value by reducing the average pulse voltage. The second mode is when the chopping is disabled, in this case, as soon as the overcurrent condition is met, the output gets disabled in a latching way.

The following image shows the current waveforms when the chopping function is enabled and disabled.

The blue trace shows the case when the chopping feature is enabled. As soon as the current exceeds the preset threshold (60mA in this case), the controller reduces the voltage in order to reduce the current. After the current drops, the controller will increase the voltage in order to increase the current. This cycle continues until the end of the pulse.

On the other hand, the orange trace shows the case when the chopping feature is disabled, as soon as the current exceeds the preset threshold the output is disabled.

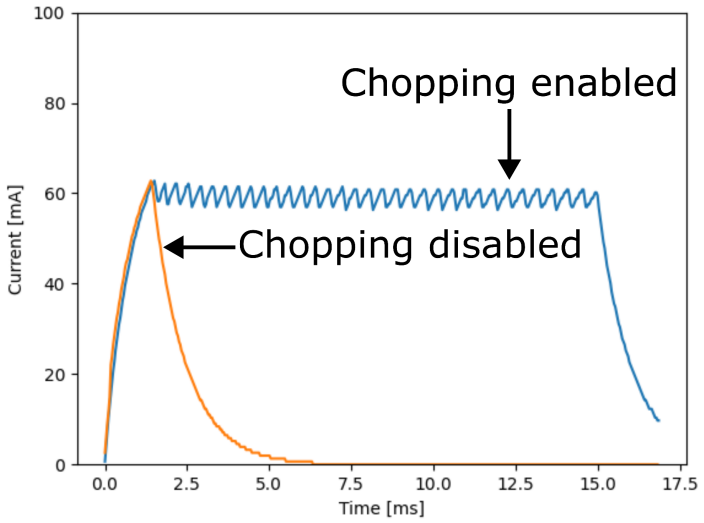

Figure 7: OCP modes of operation

SDK functions and methods

Introduction

The library contains the ‘CryoSwitch’ class which is a user-friendly way of interfacing between the QphoX CryoSwitch Controller and a computer or another USB/Ethernet capable device.

The Cryoswitch class initialization takes some optional arguments:

class Cryoswitch(debug=False, COM_port='', IP=None, SN=None)

A basic implementation of the CryoSwitchController class can be achieved with the following functions:

SDK installation

GIT repository

For user convenience, a small repository containing all the necessary files can be found here.

Inside the repository, you will find a copy of the Application Note (this file), the controller datasheet, and finally a library containing python code to interface with the CryoSwitch Controller, as we as a standalone, executable control GUI.

Python setup

The controller's library doesn't need any specific Python distribution, but some extra Python libraries must be installed.

The easiest way to do so is by using ‘pip’. Inside the repository folder and using your base or virtual environment run:

```
> pip install -r requirements.txt
```

Preparation Summary

- Clone the GIT repository.
- Install the necessary Python dependencies.

SDK implementation

A typical application would look like this:

First, we initialize the controller software by declaring an instance of the Cryoswitch class:

switch = Cryoswitch()

There are two ways the user can connect to it, the first one is via ethernet. If ethernet is used, the user needs to provide the appropriate IP address. By default the IP address is 192.168.1.101 and the subnet mask 255.255.255.0, they can be easily changed by running the functions described in the **Further functions** section.

If USB is used to establish the connection with the controller a COM port or serial number can be specified. Otherwise, the program will scan for available Cryoswitch controllers and connect to one.

Then we start the controller hardware by enabling the different voltage rails and setting the protections. By default, the output voltage is set to 5V and the pulse width to 15ms. This can be done by running the **start()** command:

switch.start()

At this point the controller is running, the last step before switching the actuator is setting the appropriate output voltage. As mentioned earlier, the appropriate pulse voltage depends on the user's setup configuration; however, as a starting point, we recommend a 5V pulse for cryogenic operation and a 20-25V pulse for room temperature operation.

An easy way to check if the actuator was engaged is by looking at the current waveform. **The plotting function is enabled by default.**

Each controller has up to 4 ports depending on the version, each one of them designed to control one 7-pin cryogenic actuator. figure 1 shows the output port configuration.

After the **start()** function is executed the controller is ready for operation. If one would like to connect the fifth contact connected to port "A"

switch.connect(port='A', contact=5)

The same applies to the disconnect function:

switch.disconnect(port='A', contact=5)

Basic functions

- start()

- Input: None
- Default: None
- Enables the voltage rails, voltage converter and output stage.

- set_output_voltage(*Vout*)

- Input: Desired output voltage (Vout)
- Default: 5V
- Sets the converter voltage to *Vout*. The output stage later utilizes the converter voltage to generate the positive/negative pulses.

- set_pulse_duration_ms(ms_duration)

- Input: Pulse width duration in milliseconds (ms_duration).
- Default: 15ms.
- Sets the output pulse (positive/negative) duration in milliseconds.

- connect(*port*, *contact*)

- Input: Corresponding port and contact to be connected. *Port*={A, B, C, D}, *contact*={1,...,6}
- Default: None.
- Connects the specified switch contact of the specified controller port (switch).
- Returns: Current waveform.

- disconnect(*port*, *contact*)

- Input: Corresponding port and contact to be disconnected. *Port*={A, B, C, D}, *contact*={1,...,6}
- Default: None.
- Disconnects the specified switch contact of the specified controller port (switch).
- Returns: Current waveform.

Overcurrent functions

- enable_OCP()

- Input: None
- Default: None.
- Enables the overcurrent protection.

- set_OCP_mA(OCP_value)

- Input: Overcurrent protection trigger value (OCP_value).

- Default: 100mA.
- Sets the overcurrent protection to the specified value.

- enable_chopping()

- Input: None.
- Default: None.
- Enables the chopping function. When an overcurrent condition occurs, the controller will 'chop' the excess current instead of disabling the output. Please refer to the installation guide for further information.

- disable_chopping()

- Input: None.
- Default: None.
- Disables the chopping function. When an overcurrent condition occurs, the controller will disable the output voltage. Please refer to the installation guide for further information.

- reset_OCP()

- Input: None
- Default: None.
- Resets the overcurrent protection (only necessary when the chopping is disabled).

Further functions

- set_sub_net_mask(mask='255.255.255.0')

- Input:
  - Subnet mask number as a string and separated by dots.
- Default: '255.255.255.0'.
- Sets the controller Subnet Mask.

- get_sub_net_mask()

- Input: None
- Returns: Subnet mask number as a string and separated by dots.
- Queries the current controller Subnet Mask.

- set_ip(add='192.168.1.101')

- Input:
  - IP address number as a string and separated by dots.
- Default: '192.168.1.101'.
- Sets the controller IP address.

- get_ip()

- Input: None
- Returns: IP address number as a string and separated by dots.
- Queries the current controller IP address.

- get_pulse_history(port=None, pulse_number=None)

- Inputs:
  - *port*: string, optional (A, B, C, D). Filter by the specified port.
  - *pulse_number:* int, optional. Number of pulses to display.
- Default: None.
- Prints the pulse history, if no pulse_numer is specified it'll show the last 5 pulses.
- Returns: None.

- get_switches_state(port=None)

- Inputs:
  - *port*: string (A, B, C, D). Filter by the specified port.
- Default: None.
- Returns the last known state of the port based on the tracking.
- Returns: Dictionary.

- select_switch_model(model='R583423141')

- Input:

    - Switch the model as a string. Either ‘R583423141’ or ‘R573423600’
- Returns:
    - Bool: True if the selected model is valid
    - Bool: False otherwise.
- Selects the switch model to be used.

- get_power_status()

- Input: None
- Returns: Power status as an integer, 1 enabled; 0 disabled.
- Queries the converter power status

- disconnect_all(port)

- Inputs:
    - *port*: string (A, B, C, D).
- Default: None.
- Disconnects all the contacts of the specified port.
- Returns: None.

- smart_connect(port, contact)

- Input:
    - *port*: string (A, B, C, D).
    - *contact:* (1,...,6)
- Default: None.
- Disconnects the previously connected contacts and connects the specified switch contact. Based on the tracking history.
- Returns: Current waveform of the specified contact.

- flash()

- Input: None
- Returns: None
- Loads the latest firmware to the controller

Recommended settings

In summary, the user should set the appropriate pulse voltage and pulse width depending on their fridge configuration. A good general starting point to find the right configuration is:

Setup	Voltage	Width	Line resistance
Room temperature	25V	15ms	≈440Ohm
Cryogenic	5V	10ms	≈20Ohm
Recommended settings			

Troubleshooting

Recognizing insufficient output voltage:

The traces in Figure 7 shows a fully engaged actuator waveform (blue) and a non-engaged actuator waveform (orange). Since the current is directly proportional to the pulse voltage, reducing the pulse voltage will also reduce the pulse current and vice versa. The actuator needs a minimum amount of current to overcome the actuator retaining force (around 60mA). A pulse that does not provide sufficient current to overcome the actuator retaining force will show a waveform characteristic of the orange trace. This can be simply solved by increasing the pulse voltage.

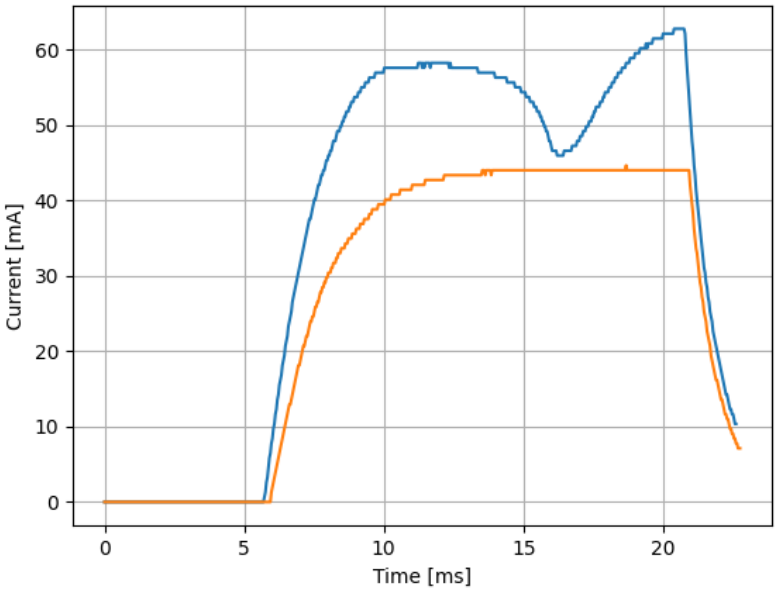

Figure 7: Low voltage waveform

Recognizing open-circuits:

Another common scenario is unconnected/broken cables, either inside the fridge or along the line. Internally, for every pulse, the controller uses some of the pulse current to bias the internal output stage. The bias current depends on the pulse voltage and is between 5-25mA. Figure 8 shows a typical waveform when the outputs are not connected. Although the plot shows there is some current flowing (bias current), the user can easily distinguish this from an actual pulse and infer that the output is probably not properly connected.

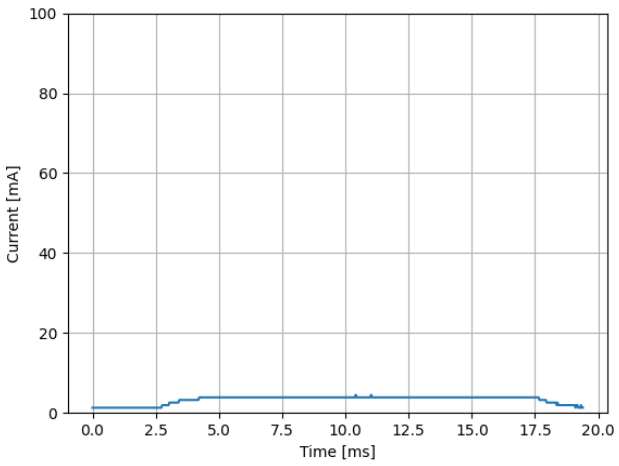

Figure 8: Unconnected waveform