

Digital isolated 4 channel temperature controller

Version: v4.3D

Charles Baynham

January 12, 2017

Contents

1	Overview	3
1.1	Microprocessor	4
1.2	Output	4
1.3	Input	4
1.4	Commands	4
2	Connector Pins	5
2.1	Power	6
2.2	Output	6
2.3	Thermistor input	7
2.4	Switches	8
2.5	LEDs	8
2.6	USB	8

3	Options and Hardware Configuration	9
3.1	OPA Choice	9
3.2	OPA gain	10
3.3	Instrumentation gain	11
3.4	Setpoint resistors	11
3.5	Sensing series resistors	12
3.6	Pins.h	12
4	Initial setup	13
5	Voltage Conversion Formulae	15
6	Hardware Design	16
6.1	Isolation	16
6.2	ADS1262 - Analog to Digital Converter	17
6.3	INA330 - Thermistor Amplifier	18
6.4	OPA548/9 - Output Amplifiers	19
6.5	DAC8564 - Digital to Analog Converter	19
6.6	ATMega328p - The micro controller	20
6.7	RS485	20
7	Software Structure	21
7.1	Doxygen	21
7.2	Code Overview	22
	Appendices	24
A	Available Commands	25
A.0.1	CommandDefinitions.h File Reference	25
B	Full Doxygen documentation	31

Section 1

Overview

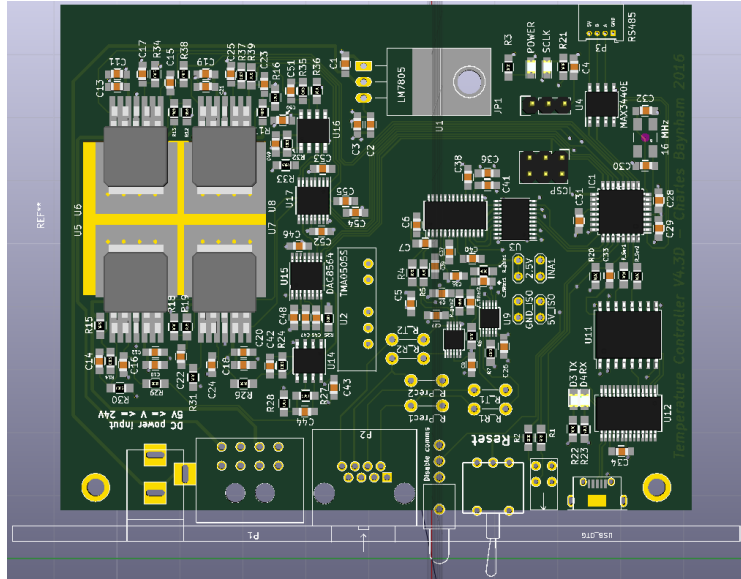


Figure 1.1: Picture / render of the board

This manual describes the operation and design processes behind the Yb⁺ digital temperature controller, version v4.3D. The controller is designed for taking high precision temperature measurements via a thermistor and applying low bandwidth PID feedback in response.

1.1 Microprocessor

The device is controlled by an on-board microprocessor running at 16 MHz. It operates in headless mode and can be configured to automatically recover from power failures. Setup and monitoring is by USB connection, baud rate 57600. This USB connection is electrically isolated from the rest of the board to prevent ground loops or digital noise.

1.2 Output

Output is by either OPA458 high power opamps. Output can be up to 1 A continuous (see Figure 3.1), at voltages up to 24 V depending on the power supply, V_S .

The board contains space for four output channels allowing for voltages between 0 V and V_S . Alternatively the channels can be combined to form two bipolar output channels, permitting outputs up to $\pm V_S$. See Section 2 for details. Note that these are not rail-to-rail opamps: the minimum possible output is around 300 mV, so applications that require lower voltages than these must use the bipolar configuration.

1.3 Input

Error signal acquisition is done by taking a balanced measurement between a thermistor and a set resistor. The board produces an excitation voltage, usually 1.0 V, and measures the difference in current between the thermistor and resistor to produce a temperature measurement. For less demanding applications the set resistor can be on-board. For the highest stability, the precision resistor should be placed next to its paired thermistor.

This measurement is performed by an INA330 chip, of which there are two present on the board. The entire analogue section is electrically isolated from the rest of the board to prevent noise on the electrical ground affecting the measurement.

There are therefore two possible input channels by balanced thermistor measurements. See Section 2 for details.

1.4 Commands

Control of the device is via serial connection over USB. It accepts commands, detailed in Appendix A. A Labview interface is also available which handles this communication.

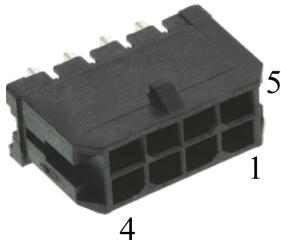
2.1 Power

Power supply is via a standard 2.1 mm, centre-positive barrel connector. The board expects a DC voltage up to 30 V and of at least 7 V, limited by the LM7805 power regulator that powers most of the electronics. Whatever voltage is supplied here will be the maximum possible output control voltage. When choosing what voltage to supply, the user should take note of considerations in Section 3.1 regarding opamp overheating.

2.2 Output

Pin	Purpose
1	GND
2	GND
3	GND
4	GND
5	Unipolar output 1 or bipolar 1 negative output (BPA)
6	Unipolar output 2 or bipolar 1 positive output (BPA)
7	Unipolar output 3 or bipolar 2 negative output (BPB)
8	Unipolar output 4 or bipolar 2 positive output (BPB)

Table 2.1: Pin connections for the output



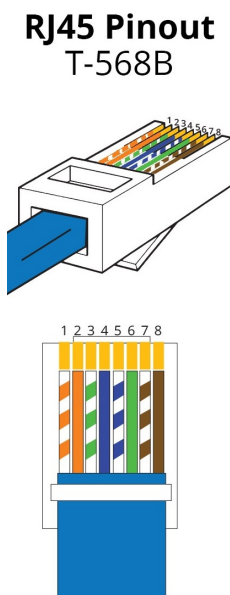
The output connector is a Molex MICRO_FIT 3.0 series 43045. It has 8 contacts, 4 of which are the OPA outputs and 4 are GND. In unipolar operation the device to be controlled (hereafter the plant) should be connected to one of the channels and GND. For bipolar operation connect to both the OPA output channels and leave the GND pins unconnected.

Figure 2.2: Molex MICRO_FIT 3.0 series 43045 output connector

2.3 Thermistor input

Pin	Common colour	Purpose
1	Orange and white	GND
2	Orange	Set resistor 1 +
3	Green and white	GND
4	Blue	Thermistor 1 +
5	Blue and white	GND
6	Green	Set resistor 2 +
7	Brown and white	GND
8	Brown	Thermistor 2 +

Table 2.2: Pin connections for the error signal input



The error signal is produced by an INA330 chip that places a 1.0 V excitation voltage (V_{excite}) across both a thermistor (R_{therm}) and a set resistor (R_{set}) and measures the difference in current. The signals in the RJ45 connector should therefore be connected to a thermistor and the set resistor. If an on-board set resistor is in use (see Section 3.4) then the setpoint resistor pin on the input connector should be left floating.

The RJ45 connector accepts standard Ethernet-style cat5 cable. This contains 4 twisted pairs, each of which is assigned to a channel. Table 2.2 assumes the standard T568-B colour scheme for RJ45 mounted CAT5 cable (orange, blue, green, brown: solid and dotted).

Figure 2.3: Pinout for a RJ45 T568-B connector ²

²Image from <http://blog.showmecables.com/rj45-pinout/>

2.4 Switches

There are two switches present on the board:

The vertical switch, `COMMS_DISABLE` can be used to disable / enable updating of the controller via USE connection. When this is set to disabled, all functions that alter the state of the controller will be prohibited (monitoring commands still work). When set to enabled, the controller functions as normal. This is intended to prevent accidental perturbation of a system under control.

The horizontal, momentary switch `RESET` causes a complete system reset.³

2.5 LEDs

The two front-mounted LEDs indicate the status of the lock(s) in progress. The LEDs are unlit if the device is currently outputting a constant voltage. If attempting to perform a lock, the two LEDs display the current distance from the setpoint of the lock controlling their respective control channels. A solid light means that the error signal is less than 0.01 V, a slow flash means between 0.01 V and 0.1 V and a quick flash means greater than 0.1 V. These threshold values are defaults but can be altered over the serial connection (see Appendix A for a full listing of available commands).

Locks are indexed in the order that they are started.

2.6 USB

The USB interface takes a micro-USB cable, allowing for monitoring and control over the system. The interface is isolated from the rest of the board to prevent ground loops.

Communications are handled by a FTDI USB - serial interface IC which presents itself as a virtual COM port. Communication is then done via serial commands at a baud rate of 57600. See Appendix A for more info on commands accepted.

³Pressing this switch is actually not strictly equivalent to a power cycle as it does not reset the ADC. However, in the initial setup section of the software the ADS is reset via its SPI interface, so these two methods should give identical results.

Section 3

Options and Hardware Configuration

There are various points at which the board can be customised for a particular use case. Each section indicates default choices that will result in sensible behavior and are pre-installed on the boards.

The file `Pins.h` contains the descriptions of these customizations. If changes are made to the default settings, the user must alter this file and re-flash the device to reflect the changes, as described in Section 3.6.

3.1 OPA Choice

The four channel version of this temperature controller only has the option for output via OPA548. These are compact devices, heatsunk to the ground plane and capable of monitoring for thermal overload and accurately limiting current down to 0 A.

Figure 3.1 shows the output currents that these devices are capable of. The power dissipated by the OPAs depends on the difference between the voltage supply and the operational voltage. For example, if the supply voltage is 15 V and 5 V of output is requested, the OPAs will have to dissipate a 10 V voltage drop across them. Depending on the current supplied, this can result in a considerable power.

The power supply voltage and opamp family should therefore be chosen bearing in mind the intended output voltage and current of the device, with reference to Figure 3.1.

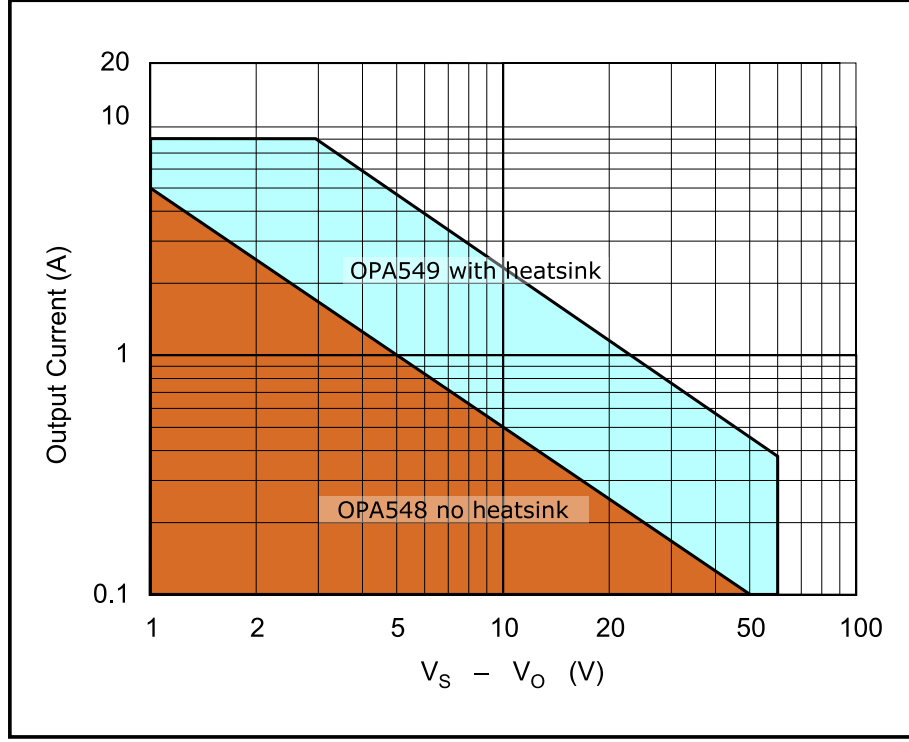


Figure 3.1: Safe Operating Area (SOA) for the OPA devices. V_S and V_O are the supply voltage and output voltage respectively. For the OPA549, this assumes the use of the heat-sink recommended. Diagram adapted from the OPA548 and OPA549 data-sheets. The OPA549 is not available with the 4 channel board option.

3.2 OPA gain

The DAC outputs voltages ranging from 0 V to 2.5 V. These are amplified by the OPA opamps to reach the desired output voltages. The choice of gain resistors R12 to R19 set the gain of this amplification and therefore the maximum possible output voltage using the formula:

$$G = \frac{R_x + R_y}{R_x} \implies V_{max} = \frac{R_x + R_y}{R_x} \times 2.5 \text{ V} \quad (3.1)$$

Default: 100 k Ω and 500 k Ω , gain = 6x, $V_{max} = 15 \text{ V}$

3.3 Instrumentation gain

The input current sensing chips work by current through a gain resistor to match the current difference between the thermistor and set resistor they are measuring, as described in Section 6.3. The choice of resistors R_{gain1} and R_{gain2} therefore set the range of inputs that are available.

For precise sensing near the setpoint, a large gain should be chosen. However, in order to make possible a wide range of inputs, a smaller gain is appropriate. If you choose to alter the default gains, you must ensure that both R_{gain1} and R_{gain2} are the same and their values are updated in `Pins.h` (see Section 3.6).

Default: 51 k Ω giving a 7 °C to 36 °C temperature sensing range

3.4 Setpoint resistors

For applications that do not require extreme stability, it is possible to place the set resistor on the board in footprints R_{prec1} and R_{prec2} . If this is done, the external precision resistor connections should not be made.

Although it is possible to lock to arbitrary resistance values, various advantages are gained by operating at the point where $R_{thermistor} = R_{set}$. In particular, operating at this point means that noise in the voltage reference or EMF noise from other sources will not affect the setpoint. Also, for small values of V_{out} the device will increase the gain of the input stage of the ADC, allowing greater sensitivity.

Again, for less demanding applications this is unnecessary: on-board precision resistors and digitally defined setpoints will provide sufficient resolution.

Default: 10 k Ω on-board resistors

3.5 Sensing series resistors

Unfortunately, the INA330 instrumentation amplifiers struggle to drive even modest capacitive loads, such as that presented by 30 cm of BNC cable. For this reason a 1 k Ω resistor is necessary in series with the thermistor and precision resistor to prevent bias voltage oscillations.

For the most demanding applications this series resistor can be removed and replaced with a piece of wire, reducing sensitivity to board temperature but necessitating short wires to the thermistor / set resistor.

Default: 1 k Ω precision resistor

3.6 Pins.h

Once customisation options have been chosen, the user must re-flash the device with a version of the source code reflecting these choices. To do this, the file `Pins.h` should be updated with the changes made and then the software should be compiled and uploaded using e.g. the Arduino IDE. Remember to temporarily re-enable flashing by using the vertical switch on the front panel.

Section 4

Initial setup

The procedure for setting up a board is therefore as follows:

1. *Optional: Choose what configuration you require in accordance with Section 3, or accept the defaults.*
2. *Optional: If you chose to customize your board in the previous step, make these modifications now.*
3. *Optional: Open the `Pins.h` file in the MC software folder. Edit it in accordance with your choices or leave it at defaults. (See the Doxygen documentation (Appendix B) for an in-depth description of each option)*
4. Solder the OPA drivers you chose to the board if using the single channel board. The dual channel board comes pre-populated.
5. Using Atmel Studio and an AVRISP mkII programmer, connect to the ISCP header on the board and upload the Optiboot boot-loader file `./CodeforMicrocontroller/optiboot_xplained328p.hex`. This step only has to be performed once per micro-processor.

The board must be powered via the barrel-jack connector during this step.

6. Using the Arduino IDE, compile and upload the code to the device. After this is done, remember to re-enable flashing using the `FLASH_DISABLE` jumper.

After this process is complete, the device will appear over USB as a COM port. In order to test that everything went well, send the command `*TST` (at baud rate 57600). If the response “OK” is received, the software is installed correctly and communication between the microprocessor and the ADC is working.



*TST not currently implemented

Section 5

Voltage Conversion Formulae

The readings output by the device will be in volts, whereas usually the user will be interested in determining the resistance of a thermistor. For more details about how this voltage is produced, see Section 6.3. The voltage output V_{out} is the voltage produced by the INA330 instrumentation amplifier according to its gain resistor R_{gain1} or R_{gain2} . To convert this to a resistance, you must know the resistance of the set resistor (default $R_{set} = 10\text{ k}\Omega$), the excitation voltage ($V_{excite} = 1.0\text{ V}$), the gain resistance ($R_{gain} = 51\text{ k}\Omega$) and the series resistance (default $R_{series} = 1\text{ k}\Omega$). Use the following formulae:

$$R_{therm} = \left[\frac{1}{R_{set} + R_{series}} - \frac{V_{out}}{V_{excite} \cdot R_{gain}} \right]^{-1} - R_{series} \quad (5.1)$$

or

$$V_{out} = V_{excite} R_{gain} \left[\frac{1}{R_{set} + R_{series}} - \frac{1}{R_{therm} + R_{series}} \right] \quad (5.2)$$

Thus, close to the set point, i.e. for small values of V_{out} , V_{out} is linearly related to resistance.

Section 6

Hardware Design

The board is designed to be rack-mounted with dimensions of $100\text{ mm} \times 80\text{ mm}$. It is compatible with IEC 60297-3/-4 specs regarding dimensions and can fit front panels such as e.g. RS 258-2920.

6.1 Isolation

Since the error signals read by this board are at DC, much care was taken to avoid ground loops and crosstalk that could introduce bias. The ADC described in Section 6.2 implements various filtering methods, but the board is also designed to prevent extraneous noise.

The USB input interface is opto-isolated from the rest of the board, with power for the FTDI USB chip provided by the computer host. Also the entire analog section; incorporating the ADC, instrumentation amplifiers and input opamps; is isolated from the rest of the board and powered by a DC-DC converter.

Although the output stages are not isolated from the rest of the board due to their high power requirements, care has been taken to manage high frequency digital return current in order to avoid signal crosstalk. The board has internal ground and power planes to reduce EMI and these both contain a bottleneck at the DAC: the point at which digital signals end and analog signals begin. This design is intended to keep digital return currents within the digital section of the board, avoiding pollution of the output stages.

row that would have benefited from increased gain, it automatically boosts the gain stage to increase sensitivity further. If the signal later becomes too large for the gain selected this is also automatically detected and corrected by the micro controller.

Because of this re-sampling and filtering, a single measurement can take up to 5 s to complete. Programs that interface with the temperature controller should therefore be aware that commands such as `ERR0? 1t` can take up to 5 s and should take appropriate steps to avoid timeout. The Labview interface handles this automatically.

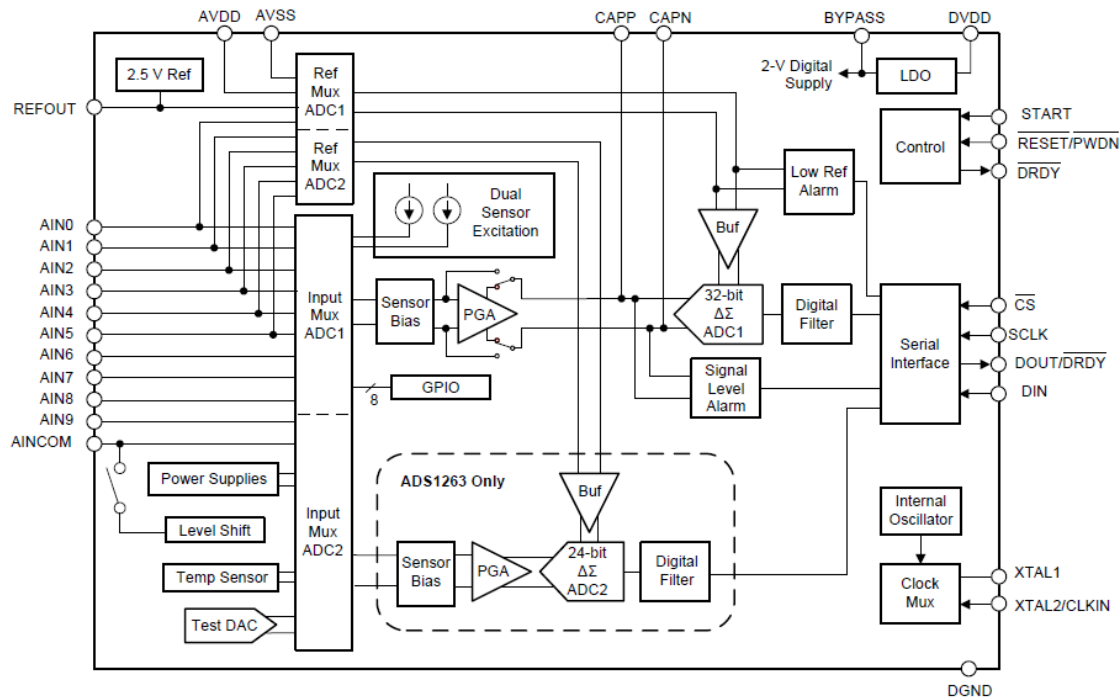


Figure 6.2: Flow diagram of the ADS1262. Reference: ADS1262 datasheet by Texas Instruments

6.3 INA330 - Thermistor Amplifier

The thermistor / setpoint precision resistor comparison is done by an INA330 chip. These chips take a reference voltage, 1.0 V on this board, ratiometrically derived from the stable 2.5 V reference of the ADS1262. This is buffered and used to drive both the thermistor and the resistor. The current difference between these two arms is then forced through a

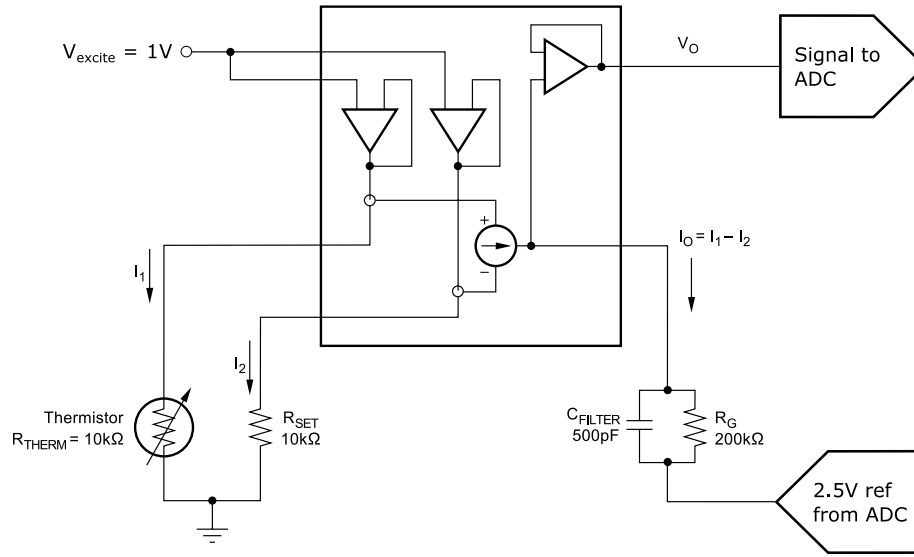


Figure 6.3: Flow diagram of the INA330. The 2.5 V reference comes from the ADC and the output voltage is refereed to this reference. The 1.0 V excitation reference is derived from the 2.5 V reference ratiometrically for maximum common-mode rejection. Diagram adapted from INA330 datasheet by Texas Instruments.

gain resistor, R_{gain1} or R_{gain2} (by default 51 kΩ), amplifying it with respect to the 2.5 V reference. This signal is then read by the ADC and compared to the 2.5 V reference in differential mode.

Figure 6.3 shows a schematic of the INA's operation. For a resistance to voltage conversion formula use Equation 5.1.

6.4 OPA548/9 - Output Amplifiers

The final output to the plant is handled by an OPA548. These amplify the 0 V \rightarrow 2.5 V signal produced by the DAC to the voltage required by the user, set by gain resistors as described in Section 3.2.

6.5 DAC8564 - Digital to Analog Converter

The production of the analog signal is done by the DAC8564: a 16-bit digital to analog converter, referenced to its own internal 2.5 V reference.

Its output ranges from 0 V to 2.5 V. This signal is then either amplified by the OPAs to become the output level, or doubled by a AD8639 auto-zeroing opamp to control the current limiting feature of the OPAs.

6.6 ATmega328p - The micro controller

The brain of the device is an ATmega328p MC. This is the same processor that is found on the Arduino Uno or Nano, and the board is configured (instructions in Section 4) to appear as an Arduino Nano to the computer. However, by breaking the Arduino into pieces several advantages are gained including reduced board space and the option to add on-board isolation to the USB interface.

The ATmega is clocked from a 16 MHz full-swing crystal. When the board is assembled for the first time the ATmega's flash memory will be blank and the device will not be programmable over the usual USB interface. For the first time only, programming must be done by the ISCP header provided, as described in Section 4.

6.7 RS485

The board is equipped with a 4 pin Molex PicoBlade 53048-0410 connector that enables RS485 communications. This is currently unimplemented, but will allow for control of up to 255 RS485 boards from a single USB connection.

Section 7

Software Structure

The code running on the micro-controller is written in C++. The ATMega328 is set up to appear the same as an Arduino Nano: it is therefore possible to compile and upload the code using the usual Arduino IDE. Note that re-flashing like this requires two things: a) the `FLASH_DISABLE` switch should be set correctly (see Section 2.4) and b) the boot-loader must have been installed using the ISCP header (see Section 4).

The code-base makes considerable use of the object-orientated programming style. While this can appear daunting at first, in the long run it simplifies modifications and results in more compartmentalized code that is easier to understand. This overview and the more in-depth Doxygen documentation both assume some familiarity with concepts such as virtualisation, inheritance, abstraction and polymorphism.

There are very many online and offline resources that can give you an overview of these ranging from 1 page to multiple-volume books. For a quick starting point, try <https://www.cs.bu.edu/teaching/cpp/inheritance/intro/> and <https://www.cs.bu.edu/teaching/cpp/polymorphism/intro/>.

7.1 Doxygen

The code is thoroughly commented throughout to aid comprehension. These comments have been written to comply with the requirements of the documentation program Doxygen <http://www.stack.nl/~dimitri/doxygen/>. It is therefore simple to compile all these comments into a manual by installing the Doxygen program on your computer, launching the wizard, selecting the `Doxyfile` file in the temperature controller's source code and

running the wizard. This will produce a neatly linked set of HTML files that can be opened in any browser. It will also produce \LaTeX output, the result of which is appended to this document in Appendix B.

The Doxygen output is detailed and should be considered the authority if modification to the code is required, but this section provides an overview of the code structure.

7.2 Code Overview

As per usual object-orientated design, the jobs performed by the temperature controller are split between objects. Two of the most important are `ErrorChannels` and `CtrlChannels`. These classes manage interactions with the world. They are abstract classes, and provide a template of functions which must be fulfilled by their derived classes.

For instance, `CtrlChannels` have a method `setCtrl()` which sets the output voltage available at the OPA. This method takes a double as a parameter, the new control level, which adheres to a convention used throughout the code: all error readings or control outputs are given as a number between -1 and +1 where -1 corresponds to the lowest possible level and +1 corresponds to the highest.

A specialisation of `CtrlChannel` is `V4_OPA_OutputChannel`. This overrides the “pure virtual” methods in `CtrlChannel` and writes a voltage to the output channel it manages. On initialisation this object is assigned to one of the OPAs. A value of for output -1 would correspond to 0V, a value of +1 would be `+MAX_VOLTAGE V`.

A different specialisation of `CtrlChannel` is `V4_OPA_OutputChannelBipolar`. This object also performs output, but rather than outputting a voltage on a single OPA, it uses 2 OPAs at the same time in order to output a bipolar differential level.

`ErrorChannels` have similar functionality but are used for input instead of output. However, since taking a reading can take several seconds (see Section 6.2) there is a bit more structure involved. Readings are initiated with `startReading()`. Before this happens, the calling code should check that no reading is already in progress by calling `globalReadInProgress()`. Once a reading has been started, its progress can be checked by calling `readingReady()` and then its result can be output using `getReading()`.

Other important objects in the code are:

Controllers Contain references to an `ErrorChannel`, a `CtrlChannel` and an `Algorithm`. Every time `doLoop()` is called, attempt to take a reading from the `ErrorChannel`, process it with the `Algorithm` and output it with the `CtrlChannel`.

Algorithms An abstract class that defines an object responsible for processing an input error signal and returning the new control level. This object should contain information about the set-point and any limits applied. An example implementation is `PIDAlgorithm`.

CommandHandler An independent object that can be included as an Arduino library in any project. This object manages the input and parsing of serial commands. It uses a hashing algorithm to minimize memory consumption and assigns all memory on the stack to avoid fragmentation. It is responsible for calling the appropriate function when a command is issued. See its own Doxygen documentation for much more detail.

A typical cycle in the code looks like this:

1. Check `CommandHandler` for new commands and execute them if present.
2. For all `Controllers`, see if a lock is in progress
3. If so:
 - If an ADC reading is in progress do nothing.
 - If no reading is in progress, start one.
 - If an ADC reading for this channel is complete, get the result and move to step 4.
4. Feed the result of the reading into the contained `Algorithm`
5. Get the new control level back from the `Algorithm` and pass this to a `CtrlChannel` for output.

Appendices

Appendix A

Available Commands

The commands recognised by the controller are defined in the file `./Code_for_microcontroller/YbTempCtrl/CommandDefinitions.h` and the Doxygen documentation (Appendix B) for this file contains a complete list of them.

This documentation is reproduced here for convenience. The following is automatically generated from the code's markup, so please excuse any odd formatting. All parameters should be separated by spaces. All commands must be followed by a newline (ASCII 10) character.

A.0.1 CommandDefinitions.h File Reference

Defines all the commands that the device listens for.

```
#include "CommandHandler\CommandHandler.h"
```

Functions

- `template<size_t size>`
`CommandHandlerReturn registerCommands (CommandHandler< size > *h)`
Register the commands.

Detailed Description

Defines all the commands that the device listens for.

This file contains definitions of all the commands that can be issued to the device.

Command processing is done via the `CommandHandler` class which has its own documentation.

All functions to be called by `CommandHandler` take exactly the same form: they return `void` and take a `const ParameterLookup` object as an argument. This object dispenses `c` strings as the params, e.g. `params[1]` is the first parameter, etc. (`params[0]` is the command itself). See the definition of `#commandFunction` for the required spec.

All commands are parsed and stored using a hash function for memory efficiency, and are case-insensitive.

The following is a summary of the available commands. If query form of a command with different functionality exists is it listed separately. If it's identical, this is shown as e.g. `*TST(?)`. If no query / non-query form exists, it is not shown in this table; calling it will result in an error.

Command	Description	Params	Output
<code>*TST</code>	Test comms	0	"Loud and clear!"
<code>*TST?</code>	Test comms	0	"Query received"
<code>*IDN(?)</code>	Identify	0	"ARDUINO PID"
<code>*VER(?)</code>	Output version string	0	e.g. "v4.1"
<code>*RST</code>	Reset the device (N.B. does not clear EEPROM)	0	-
<code>STOR</code>	Store a command in EEPROM to be executed on startup	"command to be called on startup"	"Done"
		Multiple commands can be stored by separating them with a <code>','</code>	"#StoreCommand error: command too long"
<code>RETR</code>	Retrieve any stored commands from EEPROM	0	Stored command or "None"

Command	Description	Params	Output
WIPE	Erase any stored commands in EEPROM	0	"Done"
ERRO(?)	Query the error signal on given channel.	[channel = 1t,2t,1v,2v] for signals 1 or 2 from thermistor input or voltage input	Error signal float
			If thermistor reading, voltage read from -2.5 -] +2.5
	If this channel is being controlled, return the previously measured value. Else, measure it now and then return.		If voltage reading, return voltage from -5 -] +5 V
			e.g. "-0.00151"
STAT(?)	Return a string describing the status of the controller	0	t.b.c.
CONT	Set the control signal.	[output channel = 1, 2, 3, 4, BPA, BPB] [voltage = 0 - max for single, -max - +max for BP]	e.g. "#SetControl BP 2.500"
	N.B. If this is called during a lock this will set the control signal to the given value, but this may subsequently be changed by the locking algorithm		
CONT?	Query the control signal	[output channel = 1, 2, 3, 4, BPA, BPB]	e.g. "2.500"
THER?	Check the thermal state of the given channel. For the bipolar channel, return "BAD" if either OPA is overheated	[output channel = 1, 2, 3, 4, BPA, BPB]	e.g. "GOOD" / "BAD"

Command	Description	Params	Output
LIMI	Set software limits on the output voltage	[output channel = 1, 2, 3, 4, BPA, BPB]	e.g. "#SetLimits 1 0.000 3.000"
		[min voltage]	
		[max voltage]	
LIMI?	Query limits	[output channel = 1, 2, 3, 4, BPA, BPB]	e.g. "0.000 3.000"
THRE	Set thresholds at which the LEDs will start flashing.	[thresholdHigh] [thresholdLow]	e.g. "#setThresholds 0.1 0.01"
	LEDs will flash fast when the absolute error signal is] thresholdHigh, slow when it's between threshold-High and threshold-Low and light solidly when it's [threshold-Low		
THRE?	Query current LED thresholds	0	e.g. "0.1, 0.01"
CLIM	Set hardware limits on the output current	[output channel = 1, 2, 3, 4, BPA, BPB]	e.g. "#SetCurrentLimit 1 1.000"
		[max current in amps]	
CLIM?	Get current hardware on output current	[output channel = 1, 2, 3, 4, BPA, BPB]	e.g. "1.000"
SETP	Set setpoint for lock	[output channel = 1, 2, 3, 4, BPA, BPB]	If no lock running on this channel: "#SetSetpoint error: no lock running on channel xxx"
			Else, e.g. "#SetSetpoint 1 0.000"
SETP?	Query lock setpoint	[output channel = 1, 2, 3, 4, BPA, BPB]	If no lock running on this channel: "#SetSetpoint error: no lock running on channel xxx"
			Else, e.g. "0.000"

Command	Description	Params	Output
LOCK	Start PID lock	[input channel = 1t,2t,1v,2v]	e.g. "#StartLock 1t BP 0.000 1 0.5 0 10"
		[output channel = 1, 2, 3, 4, BPA, BPB]	
		[setpoint]	
		[Kp] [Ki] [Kd] [N=10]	
VOLT	Output a constant voltage	[output channel = 1, 2, 3, 4, BPA, BPB] [voltage = 0 - max for single, -max - +max for BP]	e.g. "#ConstVoltage 1 2.500"
	N.B. Unlike "CONT", this function will output a constant voltage, disabling any PID lock that was previously running on this output channel		

Function Documentation

template<size_t size> CommandHandlerReturn registerCommands (CommandHandler<size> * h) Register the commands.

Register the commandFunction functions with the CommandHandler object, defining:

- The command used to call them
- The number of parameters they must be called with

Parameters

<i>h</i>	Pointer to a CommandHandler object
----------	------------------------------------

Template Parameters

<i>size</i>	Number of commands this <code>CommandHandler</code> can contain. This will be inferred by the number specified in its definition.
-------------	-----------------------------------------------------------------------------------------------------------------------------------

Returns

A `CommandHandlerReturn` enum detailing any errors that occurred during the execution of this method.

Appendix B

Full Doxygen documentation

The following pages are the full documentation of the code, produced automatically by Doxygen. Doxygen is a code commenting system that allows you to write structured comments throughout your code which can later be extracted into a code reference.

To get the latest version of this reference, incorporating any changes, you need to install Doxygen, run the Doxywizard, load the file `./Code_for_microcontroller/YbTempCtrl/Doxyfile` and click “Run doxygen”. This will produce HTML and L^AT_EX output.

It will also generate a file called `make.bat` in `./Code_for_microcontroller/YbTempCtrl/doc/latex/make.bat`. Execute this file to produce the updated L^AT_EX code reference. To incorporate this into the manual you are currently reading, just recompile the latex from `./Manual/TemperatureControllerManual.tex`.

Yb Temperature Controller

Generated by Doxygen 1.8.11

Contents

1	Yb+ temperature controller	1
1.1	Introduction	1
1.2	Documentation	2
1.3	Version Control	2
1.4	Available commands	2
1.5	Code outline	3
2	arduino-pin-toggler	3
2.1	Introduction	3
2.2	Usage	3
3	Todo List	4
4	Module Index	4
4.1	Modules	4
5	Namespace Index	4
5.1	Namespace List	4
6	Hierarchical Index	5
6.1	Class Hierarchy	5
7	Class Index	5
7.1	Class List	5
8	File Index	6
8.1	File List	6
9	Module Documentation	7
9.1	Algorithms	7
9.1.1	Detailed Description	7
9.2	Control signal output	8
9.2.1	Detailed Description	8
9.3	Error signal input	9
9.3.1	Detailed Description	9

10 Namespace Documentation	10
10.1 YbCtrl Namespace Reference	10
10.1.1 Detailed Description	10
10.1.2 Enumeration Type Documentation	10
11 Class Documentation	11
11.1 YbCtrl::Algorithm Class Reference	11
11.1.1 Detailed Description	12
11.1.2 Member Function Documentation	12
11.2 YbCtrl::Controller Class Reference	14
11.2.1 Detailed Description	15
11.2.2 Constructor & Destructor Documentation	15
11.2.3 Member Function Documentation	15
11.3 YbCtrl::CtrlChannel Class Reference	18
11.3.1 Detailed Description	19
11.3.2 Member Function Documentation	19
11.4 YbCtrl::ErrorChannel Class Reference	23
11.4.1 Detailed Description	24
11.4.2 Member Function Documentation	24
11.5 YbCtrl::PIDAlgorithm Class Reference	27
11.5.1 Detailed Description	28
11.5.2 Constructor & Destructor Documentation	28
11.5.3 Member Function Documentation	29
11.6 pinToggler< numPins > Class Template Reference	31
11.6.1 Detailed Description	31
11.6.2 Member Function Documentation	32
11.7 YbCtrl::TemporaryLooper Class Reference	33
11.7.1 Detailed Description	33
11.8 YbCtrl::V4_ADC_ChannelPair Class Reference	34
11.8.1 Detailed Description	34
11.8.2 Constructor & Destructor Documentation	35
11.8.3 Member Function Documentation	35
11.8.4 Member Data Documentation	38
11.9 YbCtrl::V4_OPA_OutputChannel Class Reference	38
11.9.1 Detailed Description	39
11.9.2 Constructor & Destructor Documentation	39
11.9.3 Member Function Documentation	40
11.10 YbCtrl::V4_OPA_OutputChannelBipolar Class Reference	44
11.10.1 Detailed Description	45
11.10.2 Member Function Documentation	45

12 File Documentation	50
12.1 arduino-pin-toggler/pinToggler.h File Reference	50
12.1.1 Detailed Description	50
12.1.2 Macro Definition Documentation	50
12.1.3 Enumeration Type Documentation	51
12.2 CommandDefinitions.h File Reference	51
12.2.1 Detailed Description	51
12.2.2 Function Documentation	53
12.3 CommandHandler/CommandHandler.h File Reference	54
12.3.1 Detailed Description	54
12.4 Pins.h File Reference	54
12.4.1 Detailed Description	54
12.5 Pins_2chan.h File Reference	54
12.5.1 Detailed Description	55
12.5.2 Variable Documentation	55
12.6 Pins_4chan.h File Reference	58
12.6.1 Detailed Description	59
12.6.2 Variable Documentation	60
Index	63

1 Yb+ temperature controller

1.1 Introduction

This is the source code for the Yb+ temperature controller, programmed and designed by [Charles Baynham](#).

It is intended for high precision control of cavities, however can be used to control laser diode temperature or other low-bandwidth lock processes that could benefit from digital control. Error signal input can be either by voltage (from -5V to +5V) or through a balanced thermistor reading, mediated by a built-in INA330 IC. For demanding temperature applications, the latter method is recommended.

This code is designed to be uploaded to an ATmega328 microprocessor. It is written in C++ and can be compiled and uploaded via the Arduino IDE. However, if you intend to work on it for any significant length of time, I *highly* recommend that you install a more capable IDE. I wrote this software in Visual Studio Community which is a free download from Microsoft and will make your life so much easier.

1.2 Documentation

If you are reading these comments in a pdf or html format then you are currently looking at the compiled documentation. If, however, you are reading this README.md file in a text editor then your first step before editing the code should be to get hold of the full code documentation.

The code is documented throughout using the Doxygen commenting system. These can be compiled easily into a pdf or a local html website by:

1. Installing Doxygen from <http://www.stack.nl/~dimitri/doxygen/download.html>
2. Using the Doxygen GUI to load the file in this folder, Doxyfile
3. Selecting Run doxygen on the Run tab.

This will produce the html documentation and, if you have pdflatex installed, source files for a pdf.

Alternatively, open the Documentation.pdf file that should be bundled with this repo.

1.3 Version Control

This folder is a GIT repository that maintains a full history of development including every change since this code was started. This history is contained in a hidden folder called `.git` which should not be accessed directly.

The upshot of this is that the code you see may not be the latest, and may instead be a "snapshot" of a previous state. To ensure you are looking at the latest iteration, and to make updates, install a git client of your choosing. <https://windows.github.com/> is very lightweight, <http://www.sourcetreeapp.com/> is more fully featured and <https://desktop.github.com/> strikes a balance between the two.

Once installed, "checkout" the master branch for the latest code version. See <http://rogerdudler.github.io/git-guide/> for a simple guide.

Note that this code makes use of several submodules. If you "clone" the code, you will by default not get these. In order to correctly get the code for these as well, run the command `git submodule update --init --recursive` after cloning the main code. If you use a more advanced GUI such as SourceTree then this should be handled for you automatically.

1.4 Available commands

The device listens for commands on a virtual serial COM port served via USB. The USB connection is electrically isolated to avoid ground pollution.

Baud rate should be set to 57600.

The board also contains footprints to enable RS485 communication via a backplane connection, however this is not yet implemented.

For a detailed description of all the commands available, see the documentation for [CommandDefinitions.h](#).

1.5 Code outline

This code is split into classes according to the object orientated paradigm. A typical lock could be implemented as follows:

1. Define a `YbCtrl::ErrorChannel` object (or rather, an implementation of `YbCtrl::ErrorChannel`) to handle input.
2. Define a `YbCtrl::CtrlChannel` object to handle output
3. Define a `YbCtrl::Algorithm` object to implement the locking transfer function required.
4. Pass these three objects to a new `YbCtrl::Controller` object.
5. Loop, calling `YbCtrl::Controller::doLoop()` on each iteration.

For more detail, every class and method is individually documented. See the Classes section or the Namespaces section to see more.

Todo Implement RS485 Modbus protocol

Copyright

Charles Baynham 2016

2 arduino-pin-toggler

2.1 Introduction

This Arduino library manages the toggling of arbitrary pins at controllable rates. For example, it can be used to flash an LED at different speeds according to the state of your device.

It works using interrupts and is designed to be lightweight when running, so will not interfere with existing code.

This library requires exclusive usage of TIMER1 so is incompatible with libraries that also use this timer.

2.2 Usage

Include the class by adding `#include <pinToggler.h>` to your sketch.

Init the class by passing it an array of pins that will be toggled. E.g.

```
int pins[3] = {13, A4, A5};
pinToggler<3>::init(pins);
```

The template parameter (`<3>` above) defines the total number of pins being controlled.

These pins will start LOW, not toggling.

To start the toggling, set their speed to OFF, SLOW, MEDIUM, FAST or MAX. E.g.

```
pinToggler<3>::setFlashRate(0, SLOW);
```

N.B. The template parameter (here `<3>`) must match the one used in `pinToggler::init()` or this will throw an error. Also the LED parameter in `pinToggler::setFlashRate` refers to the pins passed to `pinToggler::init()`, zero-indexed in the order that they appeared in in the array.

The speeds refer to fractions of the max speed, defined by `FLASH_FREQ_HZ`.

Warning

Note that all the function calls here are static members of the class. Although a class object is created, this happens internally. For memory management purposes, be aware that this class allocates `3 * numPins` bytes on the heap. Thus, to avoid memory fragmentation, `pinToggler::init()` should be called as early in your code as possible.

Copyright

Charles Baynham 2016

3 Todo List

File `CommandHandler.h`

Write the CommandHandler documentation

File `Pins_2chan.h`

Add option for user to configure different hardware gains on the two OPAs.

File `Pins_4chan.h`

Add option for user to configure different hardware gains on the two OPAs.

page `Yb+ temperature controller`

Implement RS485 Modbus protocol

Member `YbCtrl::CtrlChannelReturn`

Change to enum class

Class `YbCtrl::TemporaryLooper`

This class is currently unused: the code in `Controller` exists but is commented out. It could be used to implement e.g. an autotuning lock or a relocking routine

4 Module Index

4.1 Modules

Here is a list of all modules:

Algorithms	7
Control signal output	8
Error signal input	9

5 Namespace Index

5.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

YbCtrl	
Namespace to hold all the temperature controller classes	10

6 Hierarchical Index

6.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

YbCtrl::Algorithm	11
YbCtrl::PIDAlgorithm	27
YbCtrl::Controller	14
YbCtrl::CtrlChannel	18
YbCtrl::V4_OPA_OutputChannel	38
YbCtrl::V4_OPA_OutputChannelBipolar	44
YbCtrl::ErrorChannel	23
YbCtrl::V4_ADC_ChannelPair	34
pinToggler< numPins >	31
YbCtrl::TemporaryLooper	33

7 Class Index

7.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

YbCtrl::Algorithm	
Abstract class for algorithms	11
YbCtrl::Controller	
Object to manage the locking loop	14
YbCtrl::CtrlChannel	
Abstract object to allow outputting a control signal	18
YbCtrl::ErrorChannel	
Abstract object to allow measuring an error signal	23
YbCtrl::PIDAlgorithm	
Implementation of an Algorithm to perform a PID lock	27
pinToggler< numPins >	
Class for toggling pins	31

YbCtrl::TemporaryLooper	
Temporarily divert the lock, before returning to normal	33
YbCtrl::V4_ADC_ChannelPair	
Implementation of an ErrorChannel for the ADS1262	34
YbCtrl::V4_OPA_OutputChannel	
Implementation of a CtrlChannel for a single-ended output	38
YbCtrl::V4_OPA_OutputChannelBipolar	
Implementation of a CtrlChannel for a bipolar output	44

8 File Index

8.1 File List

Here is a list of all documented files with brief descriptions:

ADS1262_reg.h	??
Algorithm.h	??
CommandDefinitions.h	
Defines all the commands that the device listens for	51
Controller.h	??
CtrlChannel.h	??
ErrorChannel.h	??
MemoryFree.h	??
PIDAlgorithm.h	??
Pins.h	
Hardware limits and pins	54
Pins_2chan.h	
Hardware limits and pins	54
Pins_4chan.h	
Hardware limits and pins	58
TemporaryLooper.h	??
V4_ADC_ChannelPair.h	??
V4_OPA_OutputChannel.h	??
V4_OPA_OutputChannelBipolar.h	??
__vm/.YbTempCtrl.vsarduino.h	??
arduino-pin-toggler/pinToggler.h	
Contains all the code for the arduino-pin-toggler library	50
CommandHandler/CommandHandler.h	
Contains all the code for the Arduino CommandHandler library	54

CommandHandler/Microprocessor_Debugging/debugging_disable.h	??
CommandHandler/Microprocessor_Debugging/debugging_enable.h	??
CommandHandler/Microprocessor_Debugging/debugging_init.h	??
Microprocessor_Debugging/debugging_disable.h	??
Microprocessor_Debugging/debugging_enable.h	??
Microprocessor_Debugging/debugging_init.h	??

9 Module Documentation

9.1 Algorithms

Locking algorithms.

Classes

- class [YbCtrl::Algorithm](#)
Abstract class for algorithms.
- class [YbCtrl::PIDAlgorithm](#)
Implementation of an [Algorithm](#) to perform a PID lock.

9.1.1 Detailed Description

Locking algorithms.

The classes in this module are responsible for transforming error signals into control signals. They are based on the template base class [Algorithm](#) which defines various methods, most important of which is [Algorithm::output\(double\)](#). This method takes an error signal as a parameter (using the -1 -> +1 format that is universal in this codebase) and outputs a desired control signal (also using the same format).

9.2 Control signal output

Manage output of a control signal.

Classes

- class [YbCtrl::CtrlChannel](#)
Abstract object to allow outputting a control signal.
- class [YbCtrl::V4_OPA_OutputChannel](#)
Implementation of a [CtrlChannel](#) for a single-ended output.
- class [YbCtrl::V4_OPA_OutputChannelBipolar](#)
Implementation of a [CtrlChannel](#) for a bipolar output.

9.2.1 Detailed Description

Manage output of a control signal.

The classes in this group are collectively responsible for managing the output of control signals to the real world. Their template base class is [CtrlChannel](#).

9.3 Error signal input

Manage input of an error signal.

Classes

- class [YbCtrl::ErrorChannel](#)
Abstract object to allow measuring an error signal.
- class [YbCtrl::V4_ADC_ChannelPair](#)
Implementation of an [ErrorChannel](#) for the ADS1262.

9.3.1 Detailed Description

Manage input of an error signal.

The classes in this group are collectively responsible for managing the input of error signals from the real world. Their template base class is [ErrorChannel](#).

10 Namespace Documentation

10.1 YbCtrl Namespace Reference

Namespace to hold all the temperature controller classes.

Classes

- class [Algorithm](#)
Abstract class for algorithms.
- class [Controller](#)
Object to manage the locking loop.
- class [CtrlChannel](#)
Abstract object to allow outputting a control signal.
- class [ErrorChannel](#)
Abstract object to allow measuring an error signal.
- class [PIDAlgorithm](#)
Implementation of an [Algorithm](#) to perform a PID lock.
- class [TemporaryLooper](#)
Temporarily divert the lock, before returning to normal.
- class [V4_ADC_ChannelPair](#)
Implementation of an [ErrorChannel](#) for the ADS1262.
- class [V4_OPA_OutputChannel](#)
Implementation of a [CtrlChannel](#) for a single-ended output.
- class [V4_OPA_OutputChannelBipolar](#)
Implementation of a [CtrlChannel](#) for a bipolar output.

Enumerations

10.1.1 Detailed Description

Namespace to hold all the temperature controller classes.

All the classes contained in the [YbCtrl](#) namespace are involved with the input of error signals, output of control signals or calculation of the appropriate ctrl signal from the corresponding error signal.

10.1.2 Enumeration Type Documentation

10.1.2.1 enum YbCtrl::CtrlChannelReturn [strong]

Error codes for [CtrlChannel](#) operation.

Todo Change to enum class

Enumerator

- NO_ERROR** No error
- NOT_IMPLEMENTED** Feature not implemented by derived class
- NO_SUCH_CHANNEL** This channel does not exist
- CHANNEL_NOT_MANAGED** This channel has no managing [Controller](#)
- INVALID_PARAMETER** Parameter passed was invalid
- OUT_OF_MEMORY** Out of memory

10.1.2.2 enum YbCtrl::ErrorChannelReturn [strong]

Error codes for [ErrorChannel](#) operation.

Enumerator

NO_ERROR No error
NOT_IMPLEMENTED Feature not implemented by derived class
WRITING_TO_REG_FAILED ADC comms failed
TIMEOUT Reading timeout
OUT_OF_RANGE Reading out of PGA range
PGA_ERROR Undefined PGA error
NO_SUCH_CHANNEL Channel does not exist

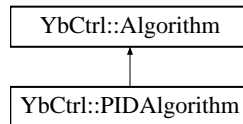
11 Class Documentation

11.1 YbCtrl::Algorithm Class Reference

Abstract class for algorithms.

```
#include <Algorithm.h>
```

Inheritance diagram for YbCtrl::Algorithm:



Public Member Functions

- virtual double [output](#) (double input)=0
Do the locking calculation.
- virtual char * [reportState](#) (char *ptr)=0
Report state.
- virtual void [setOutput](#) (double [output](#))=0
Sets the output.
- virtual void [setSetpoint](#) (double setpoint)=0
Sets the setpoint.
- virtual double [getSetpoint](#) ()=0
Gets the setpoint.
- virtual void [setLimits](#) (double min, double max)=0
Sets output limits.
- virtual bool [lockingAlgo](#) ()
Does this algorithm lock?
- virtual [operator bool](#) ()
Is this algorithm valid?

11.1.1 Detailed Description

Abstract class for algorithms.

This abstract class defines a template for a lock algorithm. The main method is `output()` which is responsible for taking an error signal and calculating a subsequent output from it. `output()` obeys the convention followed everywhere in this code that both input and output signals can range from -1 to +1.

As with all abstract classes, the pure virtual methods in this class **MUST** be overridden by derived classes. The virtual methods **SHOULD** be overridden.

For an example implementation, see [PIDAlgorithm](#).

11.1.2 Member Function Documentation

11.1.2.1 `virtual double YbCtrl::Algorithm::getSetpoint () [pure virtual]`

Gets the setpoint.

Returns

The current setpoint.

Implemented in [YbCtrl::PIDAlgorithm](#).

11.1.2.2 `virtual bool YbCtrl::Algorithm::lockingAlgo () [inline],[virtual]`

Does this algorithm lock?

Returns a boolean describing whether this [Algorithm](#) implementation locks or not. E.g. the [PIDAlgorithm](#), a derived implementation of this class, does lock so it overrides this method with

```
return true;
```

Currently there are no examples of Algorithms that do not lock.

Returns

true / false

Reimplemented in [YbCtrl::PIDAlgorithm](#).

11.1.2.3 `virtual YbCtrl::Algorithm::operator bool () [inline],[explicit],[virtual]`

Is this algorithm valid?

Return true if this [Algorithm](#) is valid for use in calculations. E.g. see [PIDAlgorithm](#) for an example of why this may be useful.

Reimplemented in [YbCtrl::PIDAlgorithm](#).

11.1.2.4 `virtual double YbCtrl::Algorithm::output (double input) [pure virtual]`

Do the locking calculation.

Parameters

in	<i>input</i>	The error signal, ranging from -1 to +1
----	--------------	-----------------------------------------

Returns

The new ctrl level, ranging from -1 to +1

Implemented in [YbCtrl::PIDAlgorithm](#).

11.1.2.5 `virtual char* YbCtrl::Algorithm::reportState (char * ptr) [pure virtual]`

Report state.

Send a string that identifies the currently running algorithm.

Parameters

out	<i>ptr</i>	A buffer to contain the output. Should be at least 128 chars long.
-----	------------	--------------------------------------------------------------------

Returns

Returns *ptr*

Implemented in [YbCtrl::PIDAlgorithm](#).

11.1.2.6 `virtual void YbCtrl::Algorithm::setLimits (double min, double max) [pure virtual]`

Sets output limits.

Set limits on the algorithm's output. Values are from -1 to +1.

Parameters

in	<i>min</i>	The new minimum
in	<i>max</i>	The new maximum

Implemented in [YbCtrl::PIDAlgorithm](#).

11.1.2.7 `virtual void YbCtrl::Algorithm::setOutput (double output) [pure virtual]`

Sets the output.

Set the output to a specified level, smoothly if possible.

Parameters

in	<i>output</i>	The new output, from -1 to +1.
----	---------------	--------------------------------

Implemented in [YbCtrl::PIDAlgorithm](#).

11.1.2.8 `virtual void YbCtrl::Algorithm::setSetpoint (double setpoint) [pure virtual]`

Sets the setpoint.

Change setpoint, smoothly if possible.

Parameters

<code>in</code>	<code><i>setpoint</i></code>	The new setpoint, from -1 to +1
-----------------	------------------------------	---------------------------------

Implemented in [YbCtrl::PIDAlgorithm](#).

The documentation for this class was generated from the following file:

- `Algorithm.h`

11.2 YbCtrl::Controller Class Reference

Object to manage the locking loop.

```
#include <Controller.h>
```

Public Member Functions

- [Controller](#) ()
Construct an empty [Controller](#).
- [Controller](#) ([ErrorChannel](#) *errorInt, [CtrlChannel](#) *ctrlInt, [Algorithm](#) *algorithm)
Construct a [Controller](#).
- `operator bool` () const
Is this [Controller](#) valid?
- `void reset` ()
Wipe this [Controller](#).
- `int doLoop` ()
Do a locking cycle.
- [CtrlChannel](#) * `getCtrlChannel` ()
Gets the control channel.
- [ErrorChannel](#) * `getErrorInterface` ()
Gets the error interface.
- [Algorithm](#) * `getAlgorithm` ()
Gets the algorithm.
- `int replaceCtrlChannel` (bool)
Remove the current [CtrlChannel](#).
- `int replaceCtrlChannel` ([CtrlChannel](#) *newInterface)
Replace the current [CtrlChannel](#).
- `char * reportState` (char *ptr)
Reports the state.

11.2.1 Detailed Description

Object to manage the locking loop.

This object manages pointers to an [ErrorChannel](#), a [CtrlChannel](#) and an [Algorithm](#).

It uses these objects to get the error signal, calculate a new output level and send that output every time [doLoop\(\)](#) is called.

This object can be marked as valid or invalid. It is considered invalid if any of its three contained objects are a) not present or b) themselves marked invalid.

11.2.2 Constructor & Destructor Documentation

11.2.2.1 YbCtrl::Controller::Controller ()

Construct an empty [Controller](#).

This empty controller is marked invalid.

11.2.2.2 YbCtrl::Controller::Controller ([ErrorChannel](#) * *errorInt*, [CtrlChannel](#) * *ctrlInt*, [Algorithm](#) * *algorithm*)

Construct a [Controller](#).

Construct a controller using pointers to its substituents. This is the recommended constructor form.

N.B. a [CtrlChannel](#) may only be assigned to a single [Controller](#) at a time. To ensure this, a pointer is placed within the [CtrlChannel](#) to its controlling [Controller](#), if it exists. This constructor checks for the presence of this pointer and, if it points to another rival [Controller](#), it refuses to use the passed [CtrlChannel](#) and this [Controller](#) is marked invalid.

Parameters

in	<i>errorInt</i>	The error interface
in	<i>ctrlInt</i>	The control interface
in	<i>algorithm</i>	The algorithm

11.2.3 Member Function Documentation

11.2.3.1 int YbCtrl::Controller::doLoop ()

Do a locking cycle.

Does one cycle of the loop. Reads error, performs calculation and outputs result

Returns

Returns 0 (error code not implemented)

11.2.3.2 `Algorithm* YbCtrl::Controller::getAlgorithm () [inline]`

Gets the algorithm.

Returns

The algorithm.

11.2.3.3 `CtrlChannel* YbCtrl::Controller::getCtrlChannel () [inline]`

Gets the control channel.

Returns

The control channel.

11.2.3.4 `ErrorChannel* YbCtrl::Controller::getErrorInterface () [inline]`

Gets the error interface.

Returns

The error interface.

11.2.3.5 `YbCtrl::Controller::operator bool () const [explicit]`

Is this [Controller](#) valid?

Return true if the [ErrorInterface](#), [CtrlChannel](#) and [Algorithm](#) are all present and valid. Otherwise return false.

11.2.3.6 `int YbCtrl::Controller::replaceCtrlChannel (bool) [inline]`

Remove the current [CtrlChannel](#).

Calls [replaceCtrlChannel\(\)](#) with a null pointer.

Parameters

in	<unnamed>	Any bool. Value is ignored
----	-----------	----------------------------

Returns

Error codes

Return values

0	Success
---	---------

11.2.3.7 `int YbCtrl::Controller::replaceCtrlChannel (CtrlChannel * newInterface) [inline]`

Replace the current [CtrlChannel](#).

Replaces the current [CtrlChannel](#) with a new one, releasing control of the current [CtrlChannel](#).

Parameters

in	<i>newInterface</i>	The new CtrlChannel
----	---------------------	-------------------------------------

Returns

Error codes

Return values

0	Success
1	Error: new CtrlChannel is already assigned.

11.2.3.8 `char* YbCtrl::Controller::reportState (char * ptr) [inline]`

Reports the state.

Write into ptr a string that identifies the currently running loop. This is delegated to the [Algorithm](#)

Parameters

out	<i>ptr</i>	Pointer to a buffer. Should hold at least 128 chars.
-----	------------	------------------------------------------------------

Returns

Returns ptr

11.2.3.9 `void YbCtrl::Controller::reset ()`

Wipe this [Controller](#).

Relinquishes control of the current [CtrlChannel](#) and wipes all parameters.

The documentation for this class was generated from the following files:

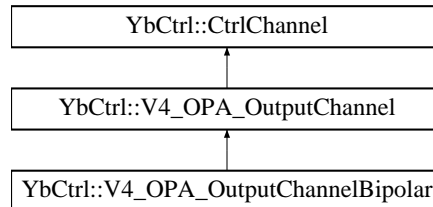
- Controller.h
- Controller.cpp

11.3 YbCtrl::CtrlChannel Class Reference

Abstract object to allow outputting a control signal.

```
#include <CtrlChannel.h>
```

Inheritance diagram for YbCtrl::CtrlChannel:



Public Member Functions

- virtual double [recallCtrl](#) ()=0
Read out the current output.
- virtual [CtrlChannelReturn](#) [setLimits](#) (double minimum, double maximum) final
Set software limits on the max/min ctrl signal.
- virtual [operator bool](#) () const =0
Is this object valid?
- virtual [CtrlChannelReturn](#) [setCurrentLimit](#) (double val)
Limits the current.
- virtual double [getCurrentLimit](#) ()
Gets the current limit.
- virtual [CtrlChannelReturn](#) [isOverheated](#) (bool &state)
Check for overheated.
- virtual [CtrlChannelReturn](#) [getLimits](#) (double &min, double &max)=0
Gets the output limits.
- virtual [CtrlChannelReturn](#) [getCurrentLimit](#) (double &out)
Gets the current limit.
- virtual void [setCtrl](#) (double val) final
Sets the control signal.
- void [setContainingController](#) ([Controller](#) *newController)
Sets the containing Controller.
- [Controller](#) * [getContainingController](#) ()
Gets the containing Controller.
- [CtrlChannelReturn](#) [getContainingController](#) ([Controller](#) *&out)
Gets the containing controller.
- [CtrlChannelReturn](#) [addConflictingChannel](#) ([CtrlChannel](#) *newConflict)
Adds a conflicting channel.
- [CtrlChannelReturn](#) [closeConflictingControllers](#) ()
Closes conflicting controllers.

11.3.1 Detailed Description

Abstract object to allow outputting a control signal.

This object is a template for class that writes an output to the real world. Child classes will implement specifics, e.g. writing a voltage to an OPA.

As with the [Algorithm](#) class, this class is abstract and so derived classes must override many of its methods.

For an example, see [V4_OPA_OutputChannel](#).

11.3.2 Member Function Documentation

11.3.2.1 CtrlChannelReturn YbCtrl::CtrlChannel::addConflictingChannel (CtrlChannel * newConflict) [inline]

Adds a conflicting channel.

Adds a channel that is considered to be conflicting with this one. I.e. this channel and the conflicting one should not be simultaneously controlled.

This is useful e.g. for bipolar vs. single-sided channels which use the same outputs.

Up to 2 channels can be added using this function. Any Controllers associated with these channels can be closed with closeConflictingControllers

Returns

Error code

11.3.2.2 CtrlChannelReturn YbCtrl::CtrlChannel::closeConflictingControllers ()

Closes conflicting controllers.

Close any Controllers that are managing either this channel or any conflicting ones. Conflicting channels are identified by adding pointers to them with addConflictingChannel.

Note

Similarly to [setCtrl\(\)](#) vs [writeCtrl\(\)](#), this is the public interface which is defined by the base class. The derived classes will overwrite the private pure virtual member [writeLimits\(\)](#) which is called by this function and which will actually set the limits.

Returns

Error code

11.3.2.3 Controller* YbCtrl::CtrlChannel::getContainingController () [inline]

Gets the containing [Controller](#).

Returns

The containing controller. NULL if none present.

11.3.2.4 CtrlChannelReturn YbCtrl::CtrlChannel::getContainingController (Controller *& out) [inline]

Gets the containing controller.

Parameters

out	out	The containing Controller
-----	-----	-------------------------------------------

Returns

Error code

11.3.2.5 `virtual double YbCtrl::CtrlChannel::getCurrentLimit() [inline],[virtual]`

Gets the current limit.

Gets the current limit. Value in amps

Returns

The current limit in amps. -999 if not implemented

Reimplemented in [YbCtrl::V4_OPA_OutputChannel](#).

11.3.2.6 `virtual CtrlChannelReturn YbCtrl::CtrlChannel::getCurrentLimit(double & out) [inline],[virtual]`

Gets the current limit.

Gets the current limit and stores it in out. Value in amps

Parameters

out	out	The current limit in amps
-----	-----	---------------------------

Returns

Error code

Reimplemented in [YbCtrl::V4_OPA_OutputChannel](#).

11.3.2.7 `virtual CtrlChannelReturn YbCtrl::CtrlChannel::getLimits(double & min, double & max) [pure virtual]`

Gets the output limits.

Store the output limits in the passed references.

Parameters

out	min	The minimum
out	max	The maximum

Returns

Error code

Implemented in [YbCtrl::V4_OPA_OutputChannel](#).

11.3.2.8 `virtual CtrlChannelReturn YbCtrl::CtrlChannel::isOverheated (bool & state) [inline],[virtual]`

Check for overheated.

Check if this channel is overheated

Parameters

out	<i>state</i>	True if overheated, false otherwise
-----	--------------	-------------------------------------

Returns

Error return

Reimplemented in [YbCtrl::V4_OPA_OutputChannel](#), and [YbCtrl::V4_OPA_OutputChannelBipolar](#).

11.3.2.9 `virtual YbCtrl::CtrlChannel::operator bool () const [explicit],[pure virtual]`

Is this object valid?

Return true if this object has been constructed with valid parameters

Implemented in [YbCtrl::V4_OPA_OutputChannel](#).

11.3.2.10 `virtual double YbCtrl::CtrlChannel::recallCtrl () [pure virtual]`

Read out the current output.

Read out the current output. Values from -1 to 1.

Returns

Current ctrl level

Implemented in [YbCtrl::V4_OPA_OutputChannel](#).

11.3.2.11 `void YbCtrl::CtrlChannel::setContainingController (Controller * newController) [inline]`

Sets the containing [Controller](#).

Set this object's managing [Controller](#) to the given target

Parameters

<i>newController</i>	The new managing Controller
----------------------	---------------------------------------------

11.3.2.12 `void YbCtrl::CtrlChannel::setCtrl (double val) [final],[virtual]`

Sets the control signal.

Set the control signal to the given value. Also, if this channel is managed by a [Controller](#), inform that [Controller's Algorithm](#) of the updated output level

Note

This method is declared in an external object file. This is necessary because, at the point of including this header file, the [Controller](#) object has not yet been fully defined. Since `setCtrl()` uses methods in [Controller](#), it cannot be compiled until [Controller](#) is fully defined. Therefore the code must be in a .cpp file, to be compiled later after all the headers are resolved.

See the documentation for [setLimits](#) for an explanation as to why this structure is needed

Parameters

<i>val</i>	The value, from -1 to +1
------------	--------------------------

11.3.2.13 `virtual CtrlChannelReturn YbCtrl::CtrlChannel::setCurrentLimit (double val) [inline],[virtual]`

Limits the current.

If implemented, limit the current to the given value.

N.B. this method DOES NOT use -1 -> +1 notation; the input is in amps.

Parameters

<i>in</i>	<i>val</i>	The current limit in Amps
-----------	------------	---------------------------

Returns

Error code

Reimplemented in [YbCtrl::V4_OPA_OutputChannel](#), and [YbCtrl::V4_OPA_OutputChannelBipolar](#).

11.3.2.14 `CtrlChannelReturn YbCtrl::CtrlChannel::setLimits (double minimum, double maximum) [final],[virtual]`

Set software limits on the max/min ctrl signal.

Also, if this channel is managed by a [Controller](#), inform that [Controller's Algorithm](#) of the new limits

Note

This method is defined in an external object file. This is necessary because, at the point of including this header file, the [Controller](#) object has not yet been fully defined. Since `setCtrl()` uses methods in [Controller](#), it cannot be compiled until [Controller](#) is fully defined. Therefore the code must be in a .cpp file, to be compiled later after all the headers are resolved.

Similarly to `setCtrl()` vs `writeCtrl()`, this is the public interface which is defined by the base class. The derived classes will overwrite the private pure virtual member `writeLimits()` which is called by this function and which will actually set the limits.

Parameters

in	<i>minimum</i>	The new minimum
in	<i>maximum</i>	The new maximum

Returns

Error status

The documentation for this class was generated from the following files:

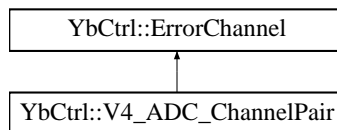
- CtrlChannel.h
- CtrlChannel.cpp

11.4 YbCtrl::ErrorChannel Class Reference

Abstract object to allow measuring an error signal.

```
#include <ErrorChannel.h>
```

Inheritance diagram for YbCtrl::ErrorChannel:



Public Member Functions

- virtual double `recallError` ()=0
Recall error signal.
- virtual `ErrorChannelReturn` `recallError` (double &output)=0
Recall error signal.
- virtual bool `readInProgress` ()
Check if there's a read in progress by this channel.
- virtual bool `globalReadInProgress` ()
Check if there's a read in progress globally.
- virtual bool `readingReady` ()
Check if the current reading is ready.
- virtual bool `readingTimeout` ()
Check if the current reading is has timed out.
- virtual void `abortReading` ()
Aborts a reading currently in progress.
- virtual `ErrorChannelReturn` `startReading` ()
Start reading.
- virtual `ErrorChannelReturn` `getReading` (double &readingOutput)=0
Get the latest reading.
- virtual `operator bool` () const =0
Is this object valid?

11.4.1 Detailed Description

Abstract object to allow measuring an error signal.

This object is a template for class that reads an input from the real world. Child classes will implement specifics, e.g. reading a voltage from an ADC.

As with the [Algorithm](#) class, this class is abstract and so derived classes must override many of its methods.

For an example, see [V4_ADC_ChannelPair](#).

11.4.2 Member Function Documentation

11.4.2.1 `virtual void YbCtrl::ErrorChannel::abortReading () [inline],[virtual]`

Aborts a reading currently in progress.

If a reading is in progress, abort it, If not, do nothing

Child objects should either override all of `readInProgress`, `readingReady`, `readingTimeout` and `startReading` or none of them.

Reimplemented in [YbCtrl::V4_ADC_ChannelPair](#).

11.4.2.2 `virtual ErrorChannelReturn YbCtrl::ErrorChannel::getReading (double & readingOutput) [pure virtual]`

Get the latest reading.

Gets the completed reading and stores it in `readingOutput`. The reading uses the format -1 -> +1, where -1 corresponds to the minimum value the [ErrorChannel](#) can read and vice versa.

If no conversion is in progress / the conversion failed, return an error message.

Otherwise return [ErrorChannelReturn::NO_ERROR](#).

This method must be overridden by child objects.

Parameters

out	<i>readingOutput</i>	Output variable for the reading.
-----	----------------------	----------------------------------

Returns

`ErrorChannelReturn` error code.

Implemented in [YbCtrl::V4_ADC_ChannelPair](#).

11.4.2.3 `virtual bool YbCtrl::ErrorChannel::globalReadInProgress () [inline],[virtual]`

Check if there's a read in progress globally.

Returns true if a reading has been started with startReading, has not been aborted with abortReading and has not been returned by getReading BY ANY [ErrorChannel](#).

Otherwise returns false

Before it is overridden, this function always returns true. This is so that simple ErrorChannels that do not require this start / check / read method can instead just return a value from getReading each time.

Child objects should either override all of readInProgress, readingReady, readingTimeout and startReading or none of them.

Returns

true / false

Reimplemented in [YbCtrl::V4_ADC_ChannelPair](#).

11.4.2.4 `virtual YbCtrl::ErrorChannel::operator bool () const [explicit],[pure virtual]`

Is this object valid?

Return true if this object has been constructed with valid parameters

Implemented in [YbCtrl::V4_ADC_ChannelPair](#).

11.4.2.5 `virtual bool YbCtrl::ErrorChannel::readingReady () [inline],[virtual]`

Check if the current reading is ready.

Returns true if a reading has been started with startReading, has not been aborted with abortReading, has not been returned by getReading and has successfully completed

Otherwise returns false

Before it is overridden, this function always returns true. This is so that simple ErrorChannels that do not require this start / check / read method can instead just return a value from getReading each time.

Child objects should either override all of readInProgress, readingReady, readingTimeout and startReading or none of them.

Returns

true / false

Reimplemented in [YbCtrl::V4_ADC_ChannelPair](#).

11.4.2.6 `virtual bool YbCtrl::ErrorChannel::readingTimeout () [inline],[virtual]`

Check if the current reading is has timed out.

Returns true if a reading has been started with startReading, has not been aborted with abortReading, has not been returned by getReading, has not successfully completed and has exceeded a maximum timeout.

Otherwise returns false

Before it is overridden, this function always returns false. This is so that simple ErrorChannels that do not require this start / check / read method can instead just return a value from getReading each time.

Child objects should either override all of readInProgress, readingReady, readingTimeout and startReading or none of them.

Returns

true / false

Reimplemented in [YbCtrl::V4_ADC_ChannelPair](#).

11.4.2.7 `virtual bool YbCtrl::ErrorChannel::readInProgress () [inline],[virtual]`

Check if there's a read in progress by this channel.

Returns true if a reading has been started with `startReading`, has not been aborted with `abortReading` and has not been returned by `getReading` BY THIS [ErrorChannel](#)

Otherwise returns false

Before it is overridden, this function always returns true. This is so that simple `ErrorChannels` that do not require this start / check / read method can instead just return a value from `getReading` each time.

Child objects should either override all of `readInProgress`, `readingReady`, `readingTimeout` and `startReading` or none of them.

Returns

true / false

Reimplemented in [YbCtrl::V4_ADC_ChannelPair](#).

11.4.2.8 `virtual double YbCtrl::ErrorChannel::recallError () [pure virtual]`

Recall error signal.

Read out the last measured error. Values from -1 to 1.

Returns

Last measured error signal from -1 to +1

Implemented in [YbCtrl::V4_ADC_ChannelPair](#).

11.4.2.9 `virtual ErrorChannelReturn YbCtrl::ErrorChannel::recallError (double & output) [pure virtual]`

Recall error signal.

Read out the last measured error into output. Values from -1 to 1.

Parameters

out	<i>output</i>	Target for the error signal
-----	---------------	-----------------------------

Returns

Error code

Implemented in [YbCtrl::V4_ADC_ChannelPair](#).

11.4.2.10 `virtual ErrorChannelReturn YbCtrl::ErrorChannel::startReading () [inline],[virtual]`

Start reading.

Start a new reading if none in progress (by this channel or globally).

If a reading is already in progress, abort it and start another one.

Child objects should either override all of readInProgress, readingReady, readingTimeout and startReading or none of them.

Returns

ErrorChannelReturn error code.

Reimplemented in [YbCtrl::V4_ADC_ChannelPair](#).

The documentation for this class was generated from the following file:

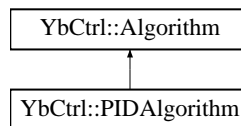
- ErrorChannel.h

11.5 YbCtrl::PIDAlgorithm Class Reference

Implementation of an [Algorithm](#) to perform a PID lock.

```
#include <PIDAlgorithm.h>
```

Inheritance diagram for YbCtrl::PIDAlgorithm:



Public Member Functions

- [PIDAlgorithm](#) (double K, double Ti, double Td, double initialOutput, double target=0, double N=10, bool disableProportional=false)
Constructor.
- [PIDAlgorithm](#) ()
Alternative constructor.
- bool [lockingAlgo](#) () override
Does this algorithm lock?
- [operator bool](#) () override
Returns true if this object has been initialised with valid parameters.
- double [output](#) (double input) override
Do the PID calculation.
- void [setOutput](#) (double [output](#)) override
Sets the output.
- void [setSetpoint](#) (double setpoint) override
Sets the setpoint.
- void [setLimits](#) (double min, double max) override
Sets output limits.
- double [getSetpoint](#) () override
Gets the setpoint.

Protected Member Functions

- `char * reportState (char *ptr)` override
Report state.

11.5.1 Detailed Description

Implementation of an [Algorithm](#) to perform a PID lock.

An implementation of [Algorithm](#) that performs a PID lock based on the parameters passed during its construction.

The lock performed by this object implements the following transfer function:

$$G(s) = -K \left(1 + \frac{1}{sT_I} + \frac{sT_D}{1 + \frac{sT_D}{N}} \right)$$

When created without parameters passed to the constructor, this object will be marked invalid. This allows the user to reserve space on the stack, while still recording which PIDAlgorithms are properly setup.

As usual, all inputs / outputs are to be given in -1 to +1 format.

11.5.2 Constructor & Destructor Documentation

11.5.2.1 YbCtrl::PIDAlgorithm::PIDAlgorithm (double *K*, double *Ti*, double *Td*, double *initialOutput*, double *target* = 0, double *N* = 10, bool *disableProportional* = false)

Constructor.

Parameters

in	<i>K</i>	Proportional gain
in	<i>Ti</i>	Integral time constant
in	<i>Td</i>	Differential time constant
in	<i>initialOutput</i>	The initial output
in	<i>target</i>	The target error signal
in	<i>N</i>	High frequency damping coefficient. ~10 is a good value.
in	<i>disableProportional</i>	Disable proportional gain
in	<i>timingCycles</i>	To speed up execution, the PID parameters are recalculated according to the measured loop speed every timingCycles loops.
in	<i>debug</i>	Enable debug output

11.5.2.2 YbCtrl::PIDAlgorithm::PIDAlgorithm () [inline]

Alternative constructor.

A [PIDAlgorithm](#) constructed without parameters is marked invalid. This can be useful for reserving stack space for these objects, allowing them to be declared in advance.

E.g.

```

PIDAlgorithm algos[3];

bool valid = algos[1]; // returns false

algos[1] = PIDAlgorithm(1,1,0,0,0);

bool valid = algos[1]; // returns true

```

11.5.3 Member Function Documentation

11.5.3.1 double YbCtrl::PIDAlgorithm::getSetpoint () [inline],[override],[virtual]

Gets the setpoint.

Returns

The current setpoint.

Implements [YbCtrl::Algorithm](#).

11.5.3.2 bool YbCtrl::PIDAlgorithm::lockingAlgo () [inline],[override],[virtual]

Does this algorithm lock?

Returns a boolean describing whether this [Algorithm](#) implementation locks or not. E.g. the [PIDAlgorithm](#), a derived implementation of this class, does lock so it overrides this method with

```
return true;
```

Currently there are no examples of Algorithms that do not lock.

Returns

true / false

Reimplemented from [YbCtrl::Algorithm](#).

11.5.3.3 double YbCtrl::PIDAlgorithm::output (double *input*) [override],[virtual]

Do the PID calculation.

See p242 of <http://www.cds.caltech.edu/~murray/courses/cds101/fa02/caltech/astrom-ch6.pdf> for an in depth explanation of how this works

Parameters

in	<i>input</i>	The current error signal input. Values from -1 to +1
----	--------------	------------------------------------------------------

Returns

The calculated new ctrl signal. Values from -1 to +1

Implements [YbCtrl::Algorithm](#).

11.5.3.4 `char* YbCtrl::PIDAlgorithm::reportState (char * ptr)` `[inline], [override], [protected], [virtual]`

Report state.

Send a string that identifies the currently running algorithm.

Parameters

out	<i>ptr</i>	A buffer to contain the output. Should be at least 128 chars long.
-----	------------	--------------------------------------------------------------------

Returns

Returns *ptr*

Implements [YbCtrl::Algorithm](#).

11.5.3.5 `void YbCtrl::PIDAlgorithm::setLimits (double min, double max)` `[override], [virtual]`

Sets output limits.

Set limits on the algorithm's output. Values are from -1 to +1.

Parameters

in	<i>min</i>	The new minimum
in	<i>max</i>	The new maximum

Implements [YbCtrl::Algorithm](#).

11.5.3.6 `void YbCtrl::PIDAlgorithm::setOutput (double output)` `[override], [virtual]`

Sets the output.

Set the output to a specified level, smoothly if possible.

Parameters

in	<i>output</i>	The new output, from -1 to +1.
----	---------------	--------------------------------

Implements [YbCtrl::Algorithm](#).

11.5.3.7 `void YbCtrl::PIDAlgorithm::setSetpoint (double setpoint)` `[override], [virtual]`

Sets the setpoint.

Change setpoint, smoothly if possible.

Parameters

in	setpoint	The new setpoint, from -1 to +1
----	----------	---------------------------------

Implements [YbCtrl::Algorithm](#).

The documentation for this class was generated from the following files:

- PIDAlgorithm.h
- PIDAlgorithm.cpp

11.6 pinToggler< numPins > Class Template Reference

Class for toggling pins.

```
#include <pinToggler.h>
```

Inherits pinTogglerBase.

Static Public Member Functions

- static int [init](#) (const uint8_t *LED_Pins)
Initiate the pins and TIMER1.
- static int [setFlashRate](#) (const size_t LED, const [FLASHRATE](#) rate)
Change the flash rate of an LED managed by this routine.
- static int [getPin](#) (const size_t LED)
Return the pin corresponding to a given LED.

11.6.1 Detailed Description

```
template<int numPins>
class pinToggler< numPins >
```

Class for toggling pins.

This class maintains a list of pins that it controls as outputs. It initiates TIMER1 and uses this to trigger an interrupt routine, allowing arbitrary numbers of pins to be toggled at selectable rates.

This class implements a singleton and all access to it is via static methods. The first usage of [init\(\)](#) will define the value of numPins. Attempts to use [init\(\)](#) with another value of numPins after this will fail with an error code.

For example usage see the Arduino sketches located in the `examples` directory.

Template Parameters

<i>numPins</i>	The number of pins that this object will handle.
----------------	--------------------------------------------------

11.6.2 Member Function Documentation

11.6.2.1 `template<int numPins> static int pinToggler< numPins >::getPin (const size_t LED) [inline], [static]`

Return the pin corresponding to a given LED.

Parameters

<i>LED</i>	The zero reference LED to get the pin of. The LED number should correspond to the index of this pin in the array passed to init() .
------------	-----------------------------------------------------------------------------------------------------------------------------------------------------

Returns

Pin number. N.B. If an error occurs the return value will be negative, according to the error codes in [setFlashRate](#).

11.6.2.2 `template<int numPins> static int pinToggler< numPins >::init (const uint8_t * LED_Pins) [inline], [static]`

Initiate the pins and TIMER1.

Setup and start the ISR triggered by TIMER1, and set TIMER1 running. Also, set all input pins as outputs and set their output to LOW.

Parameters

<i>in</i>	<i>LED_Pins</i>	Pointer to a <numPins> dimension array of pins to be controlled. The values in this array will be copied so it can be deleted from memory after this function call is complete.
-----------	-----------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Return values

0	No error
-1	init() has been called already
-2	Stack assignment failed: out of memory

Returns

Error code

11.6.2.3 `template<int numPins> static int pinToggler< numPins >::setFlashRate (const size_t LED, const FLASHRATE rate) [inline], [static]`

Change the flash rate of an LED managed by this routine.

This method sets the flash rate of one of the pins controlled by the [pinToggler](#) class. Pins are placed under control by passing them to [init\(\)](#), after which they are referred to by their zero-indexed position in the array passed to [init\(\)](#).

Parameters

<i>LED</i>	The zero reference LED to set the flash rate of. The LED number should correspond to the index of this pin in the array passed to init() .
<i>rate</i>	The rate at which to flash this LED. Options are enumerated in FLASHRATE .

Return values

0	No error
-1	init() has not been called.
-2	init() was called with a different value of numPins
-3	The given pin does not exist

Returns

Error code

The documentation for this class was generated from the following file:

- [arduino-pin-toggler/pinToggler.h](#)

11.7 YbCtrl::TemporaryLooper Class Reference

Temporarily divert the lock, before returning to normal.

```
#include <TemporaryLooper.h>
```

11.7.1 Detailed Description

Temporarily divert the lock, before returning to normal.

Class for defining actions that need to take place instead of the loop, for a short time, after which the controller reverts to locking as normal.

Todo This class is currently unused: the code in [Controller](#) exists but is commented out. It could be used to implement e.g. an autotuning lock or a relocking routine

The documentation for this class was generated from the following file:

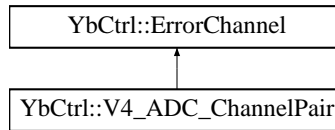
- [TemporaryLooper.h](#)

11.8 YbCtrl::V4_ADC_ChannelPair Class Reference

Implementation of an [ErrorChannel](#) for the ADS1262.

```
#include <V4_ADC_ChannelPair.h>
```

Inheritance diagram for YbCtrl::V4_ADC_ChannelPair:



Public Member Functions

- virtual double [recallError](#) () override
Recall error signal.
- virtual [ErrorChannelReturn](#) [recallError](#) (double &output)
Recall error signal.
- virtual [operator bool](#) () const override
Is this object valid?
- bool [readInProgress](#) () override
Check if there's a read in progress by this channel.
- virtual bool [globalReadInProgress](#) () override
Check if there's a read in progress globally.
- bool [readingReady](#) () override
Check if the current reading is ready.
- bool [readingTimeout](#) () override
Check if the current reading is has timed out.
- void [abortReading](#) () override
Aborts a reading currently in progress.
- [ErrorChannelReturn](#) [startReading](#) () override
Start reading.
- [ErrorChannelReturn](#) [getReading](#) (double &readingOutput) override
Get the latest reading.
- [V4_ADC_ChannelPair](#) ()
Default constructor.
- [V4_ADC_ChannelPair](#) (uint8_t channel1, uint8_t channel2, bool highRes=true)
Constructor.

Static Protected Attributes

- static constexpr uint8_t [__maxLowReadings](#) = 3
- static constexpr double [__lowGainThreshold](#) = 0.4

11.8.1 Detailed Description

Implementation of an [ErrorChannel](#) for the ADS1262.

Reads two channels from the ADS1261 and return the difference. The ADS1262 can read a differential of at most 2.5V in either direction, so the -1 -> +1 scale used in this code corresponds to a -2.5 -> +2.5V scale at the ADC

11.8.2 Constructor & Destructor Documentation

11.8.2.1 YbCtrl::V4_ADC_ChannelPair::V4_ADC_ChannelPair () [inline]

Default constructor.

The object created thus will be marked as invalid

11.8.2.2 YbCtrl::V4_ADC_ChannelPair::V4_ADC_ChannelPair (uint8_t *channel1*, uint8_t *channel2*, bool *highRes* = true) [inline]

Constructor.

The error signal measured by readError(double&) will be channel 1 - channel 2

Parameters

in	<i>channel1</i>	ADC channel 1
in	<i>channel2</i>	ADC channel 2
in	<i>highRes</i>	Should this channel be measured quickly or carefully?

11.8.3 Member Function Documentation

11.8.3.1 void YbCtrl::V4_ADC_ChannelPair::abortReading () [override], [virtual]

Aborts a reading currently in progress.

If a reading is in progress, abort it, If not, do nothing

Child objects should either override all of readInProgress, readingReady, readingTimeout and startReading or none of them.

Reimplemented from [YbCtrl::ErrorChannel](#).

11.8.3.2 ErrorChannelReturn YbCtrl::V4_ADC_ChannelPair::getReading (double & *readingOutput*) [override], [virtual]

Get the latest reading.

Gets the completed reading and stores it in readingOutput. The reading uses the format -1 -> +1, where -1 corresponds to the minimum value the [ErrorChannel](#) can read and vice versa.

If no conversion is in progress / the conversion failed, return an error message.

Otherwise return [ErrorChannelReturn::NO_ERROR](#).

This method must be overridden by child objects.

Parameters

out	<i>readingOutput</i>	Output variable for the reading.
-----	----------------------	----------------------------------

Returns

ErrorChannelReturn error code.

Implements [YbCtrl::ErrorChannel](#).

```
11.8.3.3 virtual bool YbCtrl::V4_ADC_ChannelPair::globalReadInProgress ( ) [inline],[override],
[virtual]
```

Check if there's a read in progress globally.

Returns true if a reading has been started with startReading, has not been aborted with abortReading and has not been returned by getReading BY ANY [ErrorChannel](#).

Otherwise returns false

Before it is overridden, this function always returns true. This is so that simple ErrorChannels that do not require this start / check / read method can instead just return a value from getReading each time.

Child objects should either override all of readInProgress, readingReady, readingTimeout and startReading or none of them.

Returns

true / false

Reimplemented from [YbCtrl::ErrorChannel](#).

```
11.8.3.4 virtual YbCtrl::V4_ADC_ChannelPair::operator bool ( ) const [inline],[explicit],[override],
[virtual]
```

Is this object valid?

Return true if this object has been constructed with valid parameters

Implements [YbCtrl::ErrorChannel](#).

```
11.8.3.5 bool YbCtrl::V4_ADC_ChannelPair::readingReady ( ) [override],[virtual]
```

Check if the current reading is ready.

Returns true if a reading has been started with startReading, has not been aborted with abortReading, has not been returned by getReading and has successfully completed

Otherwise returns false

Before it is overridden, this function always returns true. This is so that simple ErrorChannels that do not require this start / check / read method can instead just return a value from getReading each time.

Child objects should either override all of readInProgress, readingReady, readingTimeout and startReading or none of them.

Returns

true / false

Reimplemented from [YbCtrl::ErrorChannel](#).

11.8.3.6 bool YbCtrl::V4_ADC_ChannelPair::readingTimeout () [override],[virtual]

Check if the current reading is has timed out.

Returns true if a reading has been started with startReading, has not been aborted with abortReading, has not been returned by getReading, has not successfully completed and has exceeded a maximum timeout.

Otherwise returns false

Before it is overridden, this function always returns false. This is so that simple ErrorChannels that do not require this start / check / read method can instead just return a value from getReading each time.

Child objects should either override all of readInProgress, readingReady, readingTimeout and startReading or none of them.

Returns

true / false

Reimplemented from [YbCtrl::ErrorChannel](#).

11.8.3.7 bool YbCtrl::V4_ADC_ChannelPair::readInProgress () [inline],[override],[virtual]

Check if there's a read in progress by this channel.

Returns true if a reading has been started with startReading, has not been aborted with abortReading and has not been returned by getReading BY THIS [ErrorChannel](#)

Otherwise returns false

Before it is overridden, this function always returns true. This is so that simple ErrorChannels that do not require this start / check / read method can instead just return a value from getReading each time.

Child objects should either override all of readInProgress, readingReady, readingTimeout and startReading or none of them.

Returns

true / false

Reimplemented from [YbCtrl::ErrorChannel](#).

11.8.3.8 virtual double YbCtrl::V4_ADC_ChannelPair::recallError () [inline],[override],[virtual]

Recall error signal.

Read out the last measured error. Values from -1 to 1.

Returns

Last measured error signal from -1 to +1

Implements [YbCtrl::ErrorChannel](#).

11.8.3.9 virtual ErrorChannelReturn YbCtrl::V4_ADC_ChannelPair::recallError (double & output) [inline],[virtual]

Recall error signal.

Read out the last measured error into output. Values from -1 to 1.

Parameters

out	output	Target for the error signal
-----	--------	-----------------------------

Returns

Error code

Implements [YbCtrl::ErrorChannel](#).

11.8.3.10 ErrorChannelReturn YbCtrl::V4_ADC_ChannelPair::startReading () [override],[virtual]

Start reading.

Start a new reading if none in progress (by this channel or globally).

If a reading is already in progress, abort it and start another one.

Child objects should either override all of readInProgress, readingReady, readingTimeout and startReading or none of them.

Returns

ErrorChannelReturn error code.

Reimplemented from [YbCtrl::ErrorChannel](#).

11.8.4 Member Data Documentation

11.8.4.1 constexpr double YbCtrl::V4_ADC_ChannelPair::__lowGainThreshold = 0.4 [static],[protected]

The threshold below which a reading is considered to have required a higher gain

11.8.4.2 constexpr uint8_t YbCtrl::V4_ADC_ChannelPair::__maxLowReadings = 3 [static],[protected]

The max number of readings to take getting less than __lowGainThreshold before boosting the gain

The documentation for this class was generated from the following files:

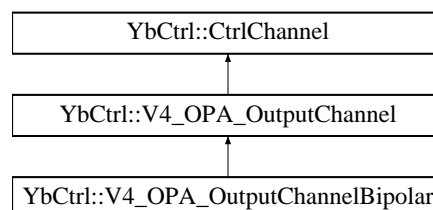
- V4_ADC_ChannelPair.h
- V4_ADC_ChannelPair.cpp

11.9 YbCtrl::V4_OPA_OutputChannel Class Reference

Implementation of a [CtrlChannel](#) for a single-ended output.

```
#include <V4_OPA_OutputChannel.h>
```

Inheritance diagram for YbCtrl::V4_OPA_OutputChannel:



Public Member Functions

- virtual `operator bool` () const override
Is this object valid?
- virtual `CtrlChannelReturn setCurrentLimit` (double val) override
Limits the current.
- virtual double `getCurrentLimit` () override
Gets the current limit.
- virtual `CtrlChannelReturn getCurrentLimit` (double &out) override
Gets the current limit.
- virtual `CtrlChannelReturn isOverheated` (bool &state)
Check for overheated.
- virtual void `enableOutput` ()
Enable OPA output.
- virtual void `disableOutput` ()
Disable OPA output.
- virtual double `recallCtrl` () override
Read out the current output.
- `V4_OPA_OutputChannel` (uint8_t channelVPlus, uint8_t channelVLim, uint8_t OPA_ES, bool smallerOPA, uint8_t DAC_CS_pin)
Constructor.
- `CtrlChannelReturn getLimits` (double &min, double &max) override
Gets the output limits.
- virtual `CtrlChannelReturn setLimits` (double minimum, double maximum) final
Set software limits on the max/min ctrl signal.
- virtual void `setCtrl` (double val) final
Sets the control signal.
- void `setContainingController` (`Controller` *newController)
Sets the containing Controller.
- `Controller` * `getContainingController` ()
Gets the containing Controller.
- `CtrlChannelReturn getContainingController` (`Controller` *&out)
Gets the containing controller.
- `CtrlChannelReturn addConflictingChannel` (`CtrlChannel` *newConflict)
Adds a conflicting channel.
- `CtrlChannelReturn closeConflictingControllers` ()
Closes conflicting controllers.

11.9.1 Detailed Description

Implementation of a `CtrlChannel` for a single-ended output.

This object manages output via a single OPA548 / OPA549. It allows for control of the output voltage between 0 and `MAX_VOLTAGE`.

Its most important method is `setCtrl(double)`

11.9.2 Constructor & Destructor Documentation

- 11.9.2.1 `YbCtrl::V4_OPA_OutputChannel::V4_OPA_OutputChannel` (uint8_t channelVPlus, uint8_t channelVLim, uint8_t OPA_ES, bool smallerOPA, uint8_t DAC_CS_pin) [inline]

Constructor.

Parameters

in	<i>channelVPlus</i>	DAC channel corresponding to V+
in	<i>channelVLim</i>	DAC channel corresponding to Vlim
in	<i>OPA_ES</i>	ATMega pin corresponding to the OPA's E/S pin
in	<i>smallerOPA</i>	Is this OPA an OPA548? If so, true. Else if it's an OPA549, false

11.9.3 Member Function Documentation

11.9.3.1 CtrlChannelReturn YbCtrl::CtrlChannel::addConflictingChannel (CtrlChannel * *newConflict*) [inline], [inherited]

Adds a conflicting channel.

Adds a channel that is considered to be conflicting with this one. I.e. this channel and the conflicting one should not be simultaneously controlled.

This is useful e.g. for bipolar vs. single-sided channels which use the same outputs.

Up to 2 channels can be added using this function. Any Controllers associated with these channels can be closed with `closeConflictingControllers`

Returns

Error code

11.9.3.2 CtrlChannelReturn YbCtrl::CtrlChannel::closeConflictingControllers () [inherited]

Closes conflicting controllers.

Close any Controllers that are managing either this channel or any conflicting ones. Conflicting channels are identified by adding pointers to them with `addConflictingChannel`.

Note

Similarly to `setCtrl()` vs `writeCtrl()`, this is the public interface which is defined by the base class. The derived classes will overwrite the private pure virtual member `writeLimits()` which is called by this function and which will actually set the limits.

Returns

Error code

11.9.3.3 Controller* YbCtrl::CtrlChannel::getContainingController () [inline], [inherited]

Gets the containing [Controller](#).

Returns

The containing controller. NULL if none present.

11.9.3.4 CtrlChannelReturn YbCtrl::CtrlChannel::getContainingController (Controller *& *out*) [inline], [inherited]

Gets the containing controller.

Parameters

out	out	The containing Controller
-----	-----	-------------------------------------------

Returns

Error code

11.9.3.5 `virtual double YbCtrl::V4_OPA_OutputChannel::getCurrentLimit () [inline], [override], [virtual]`

Gets the current limit.

Gets the current limit. Value in amps

Returns

The current limit in amps. -999 if not implemented

Reimplemented from [YbCtrl::CtrlChannel](#).

11.9.3.6 `virtual CtrlChannelReturn YbCtrl::V4_OPA_OutputChannel::getCurrentLimit (double & out) [inline], [override], [virtual]`

Gets the current limit.

Gets the current limit and stores it in out. Value in amps

Parameters

out	out	The current limit in amps
-----	-----	---------------------------

Returns

Error code

Reimplemented from [YbCtrl::CtrlChannel](#).

11.9.3.7 `CtrlChannelReturn YbCtrl::V4_OPA_OutputChannel::getLimits (double & min, double & max) [inline], [override], [virtual]`

Gets the output limits.

Store the output limits in the passed references.

Parameters

out	min	The minimum
out	max	The maximum

Returns

Error code

Implements [YbCtrl::CtrlChannel](#).

11.9.3.8 `CtrlChannelReturn YbCtrl::V4_OPA_OutputChannel::isOverheated (bool & state)` `[virtual]`

Check for overheated.

Check if this channel is overheated

Parameters

out	<i>state</i>	True if overheated, false otherwise
-----	--------------	-------------------------------------

Returns

Error return

Reimplemented from [YbCtrl::CtrlChannel](#).

Reimplemented in [YbCtrl::V4_OPA_OutputChannelBipolar](#).

11.9.3.9 `virtual YbCtrl::V4_OPA_OutputChannel::operator bool () const` `[inline],[explicit],[override],[virtual]`

Is this object valid?

Return true if this object has been constructed with valid parameters

Implements [YbCtrl::CtrlChannel](#).

11.9.3.10 `virtual double YbCtrl::V4_OPA_OutputChannel::recallCtrl ()` `[inline],[override],[virtual]`

Read out the current output.

Read out the current output. Values from -1 to 1.

Returns

Current ctrl level

Implements [YbCtrl::CtrlChannel](#).

11.9.3.11 `void YbCtrl::CtrlChannel::setContainingController (Controller * newController)` `[inline],[inherited]`

Sets the containing [Controller](#).

Set this object's managing [Controller](#) to the given target

Parameters

<i>newController</i>	The new managing Controller
----------------------	---------------------------------------------

11.9.3.12 `void YbCtrl::CtrlChannel::setCtrl (double val)` `[final], [virtual], [inherited]`

Sets the control signal.

Set the control signal to the given value. Also, if this channel is managed by a [Controller](#), inform that [Controller's Algorithm](#) of the updated output level

Note

This method is declared in an external object file. This is necessary because, at the point of including this header file, the [Controller](#) object has not yet been fully defined. Since `setCtrl()` uses methods in [Controller](#), it cannot be compiled until [Controller](#) is fully defined. Therefore the code must be in a .cpp file, to be compiled later after all the headers are resolved.

See the documentation for [setLimits](#) for an explanation as to why this structure is needed

Parameters

<i>val</i>	The value, from -1 to +1
------------	--------------------------

11.9.3.13 `virtual CtrlChannelReturn YbCtrl::V4_OPA_OutputChannel::setCurrentLimit (double val)` `[inline], [override], [virtual]`

Limits the current.

If implemented, limit the current to the given value.

N.B. this method DOES NOT use -1 -> +1 notation; the input is in amps.

Parameters

<i>in</i>	<i>val</i>	The current limit in Amps
-----------	------------	---------------------------

Returns

Error code

Reimplemented from [YbCtrl::CtrlChannel](#).

Reimplemented in [YbCtrl::V4_OPA_OutputChannelBipolar](#).

11.9.3.14 `CtrlChannelReturn YbCtrl::CtrlChannel::setLimits (double minimum, double maximum)` `[final], [virtual], [inherited]`

Set software limits on the max/min ctrl signal.

Also, if this channel is managed by a [Controller](#), inform that [Controller's Algorithm](#) of the new limits

Note

This method is defined in an external object file. This is necessary because, at the point of including this header file, the [Controller](#) object has not yet been fully defined. Since [setCtrl\(\)](#) uses methods in [Controller](#), it cannot be compiled until [Controller](#) is fully defined. Therefore the code must be in a .cpp file, to be compiled later after all the headers are resolved.

Similarly to [setCtrl\(\)](#) vs [writeCtrl\(\)](#), this is the public interface which is defined by the base class. The derived classes will overwrite the private pure virtual member [writeLimits\(\)](#) which is called by this function and which will actually set the limits.

Parameters

in	<i>minimum</i>	The new minimum
in	<i>maximum</i>	The new maximum

Returns

Error status

The documentation for this class was generated from the following files:

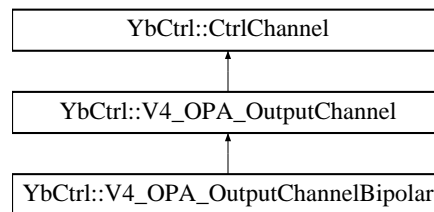
- V4_OPA_OutputChannel.h
- V4_OPA_OutputChannel.cpp

11.10 YbCtrl::V4_OPA_OutputChannelBipolar Class Reference

Implementation of a [CtrlChannel](#) for a bipolar output.

```
#include <V4_OPA_OutputChannelBipolar.h>
```

Inheritance diagram for YbCtrl::V4_OPA_OutputChannelBipolar:

**Public Member Functions**

- virtual [CtrlChannelReturn](#) [setCurrentLimit](#) (double val) override
Limits the current.
- virtual void [enableOutput](#) () override
Enable OPA output.
- virtual void [disableOutput](#) () override
Disable OPA output.
- virtual [CtrlChannelReturn](#) [isOverheated](#) (bool &state) override
Check for overheated.

- virtual `operator bool ()` const override
Is this object valid?
- virtual double `getCurrentLimit ()` override
Gets the current limit.
- virtual `CtrlChannelReturn getCurrentLimit (double &out)` override
Gets the current limit.
- virtual double `recallCtrl ()` override
Read out the current output.
- `CtrlChannelReturn getLimits (double &min, double &max)` override
Gets the output limits.
- virtual `CtrlChannelReturn setLimits (double minimum, double maximum)` final
Set software limits on the max/min ctrl signal.
- virtual void `setCtrl (double val)` final
Sets the control signal.
- void `setContainingController (Controller *newController)`
Sets the containing Controller.
- `Controller * getContainingController ()`
Gets the containing Controller.
- `CtrlChannelReturn getContainingController (Controller *&out)`
Gets the containing controller.
- `CtrlChannelReturn addConflictingChannel (CtrlChannel *newConflict)`
Adds a conflicting channel.
- `CtrlChannelReturn closeConflictingControllers ()`
Closes conflicting controllers.

11.10.1 Detailed Description

Implementation of a `CtrlChannel` for a bipolar output.

Using both output channels simultaneously, allow for positive / negative voltages from `-MAX_VOLTAGE` to `+MAX_VOLTAGE`

This object manages output via both OPA548 / OPA549 s. Its most important method is `setCtrl(double)`

11.10.2 Member Function Documentation

11.10.2.1 `CtrlChannelReturn YbCtrl::CtrlChannel::addConflictingChannel (CtrlChannel * newConflict) [inline], [inherited]`

Adds a conflicting channel.

Adds a channel that is considered to be conflicting with this one. I.e. this channel and the conflicting one should not be simultaneously controlled.

This is useful e.g. for bipolar vs. single-sided channels which use the same outputs.

Up to 2 channels can be added using this function. Any Controllers associated with these channels can be closed with `closeConflictingControllers`

Returns

Error code

11.10.2.2 CtrlChannelReturn YbCtrl::CtrlChannel::closeConflictingControllers () [inherited]

Closes conflicting controllers.

Close any Controllers that are managing either this channel or any conflicting ones. Conflicting channels are identified by adding pointers to them with addConflictingChannel.

Note

Similarly to [setCtrl\(\)](#) vs [writeCtrl\(\)](#), this is the public interface which is defined by the base class. The derived classes will overwrite the private pure virtual member [writeLimits\(\)](#) which is called by this function and which will actually set the limits.

Returns

Error code

11.10.2.3 Controller* YbCtrl::CtrlChannel::getContainingController () [inline],[inherited]

Gets the containing [Controller](#).

Returns

The containing controller. NULL if none present.

11.10.2.4 CtrlChannelReturn YbCtrl::CtrlChannel::getContainingController (Controller *& out) [inline],[inherited]

Gets the containing controller.

Parameters

out	out	The containing Controller
-----	-----	-------------------------------------------

Returns

Error code

11.10.2.5 virtual double YbCtrl::V4_OPA_OutputChannel::getCurrentLimit () [inline],[override],[virtual],[inherited]

Gets the current limit.

Gets the current limit. Value in amps

Returns

The current limit in amps. -999 if not implemented

Reimplemented from [YbCtrl::CtrlChannel](#).

11.10.2.6 `virtual CtrlChannelReturn YbCtrl::V4_OPA_OutputChannel::getCurrentLimit (double & out)` `[inline]`,
`[override]`, `[virtual]`, `[inherited]`

Gets the current limit.

Gets the current limit and stores it in out. Value in amps

Parameters

out	out	The current limit in amps
-----	-----	---------------------------

Returns

Error code

Reimplemented from [YbCtrl::CtrlChannel](#).

11.10.2.7 `CtrlChannelReturn YbCtrl::V4_OPA_OutputChannel::getLimits (double & min, double & max)` `[inline]`,
`[override]`, `[virtual]`, `[inherited]`

Gets the output limits.

Store the output limits in the passed references.

Parameters

out	min	The minimum
out	max	The maximum

Returns

Error code

Implements [YbCtrl::CtrlChannel](#).

11.10.2.8 `CtrlChannelReturn YbCtrl::V4_OPA_OutputChannelBipolar::isOverheated (bool & state)` `[override]`,
`[virtual]`

Check for overheated.

Check if this channel is overheated

Parameters

out	state	True if overheated, false otherwise
-----	-------	-------------------------------------

Returns

Error return

Reimplemented from [YbCtrl::V4_OPA_OutputChannel](#).

11.10.2.9 `virtual YbCtrl::V4_OPA_OutputChannel::operator bool () const [inline],[explicit],[override],[virtual],[inherited]`

Is this object valid?

Return true if this object has been constructed with valid parameters

Implements [YbCtrl::CtrlChannel](#).

11.10.2.10 `virtual double YbCtrl::V4_OPA_OutputChannel::recallCtrl () [inline],[override],[virtual],[inherited]`

Read out the current output.

Read out the current output. Values from -1 to 1.

Returns

Current ctrl level

Implements [YbCtrl::CtrlChannel](#).

11.10.2.11 `void YbCtrl::CtrlChannel::setContainingController (Controller * newController) [inline],[inherited]`

Sets the containing [Controller](#).

Set this object's managing [Controller](#) to the given target

Parameters

<i>newController</i>	The new managing Controller
----------------------	---------------------------------------------

11.10.2.12 `void YbCtrl::CtrlChannel::setCtrl (double val) [final],[virtual],[inherited]`

Sets the control signal.

Set the control signal to the given value. Also, if this channel is managed by a [Controller](#), inform that [Controller's Algorithm](#) of the updated output level

Note

This method is declared in an external object file. This is necessary because, at the point of including this header file, the [Controller](#) object has not yet been fully defined. Since [setCtrl\(\)](#) uses methods in [Controller](#), it cannot be compiled until [Controller](#) is fully defined. Therefore the code must be in a .cpp file, to be compiled later after all the headers are resolved.

See the documentation for [setLimits](#) for an explanation as to why this structure is needed

Parameters

<i>val</i>	The value, from -1 to +1
------------	--------------------------

11.10.2.13 CtrlChannelReturn YbCtrl::V4_OPA_OutputChannelBipolar::setCurrentLimit (double *val*) [override],
[virtual]

Limits the current.

If implemented, limit the current to the given value.

N.B. this method DOES NOT use -1 -> +1 notation; the input is in amps.

Parameters

in	<i>val</i>	The current limit in Amps
----	------------	---------------------------

Returns

Error code

Reimplemented from [YbCtrl::V4_OPA_OutputChannel](#).

11.10.2.14 CtrlChannelReturn YbCtrl::CtrlChannel::setLimits (double *minimum*, double *maximum*) [final],
[virtual],[inherited]

Set software limits on the max/min ctrl signal.

Also, if this channel is managed by a [Controller](#), inform that [Controller's Algorithm](#) of the new limits

Note

This method is defined in an external object file. This is necessary because, at the point of including this header file, the [Controller](#) object has not yet been fully defined. Since [setCtrl\(\)](#) uses methods in [Controller](#), it cannot be compiled until [Controller](#) is fully defined. Therefore the code must be in a .cpp file, to be compiled later after all the headers are resolved.

Similarly to [setCtrl\(\)](#) vs [writeCtrl\(\)](#), this is the public interface which is defined by the base class. The derived classes will overwrite the private pure virtual member [writeLimits\(\)](#) which is called by this function and which will actually set the limits.

Parameters

in	<i>minimum</i>	The new minimum
in	<i>maximum</i>	The new maximum

Returns

Error status

The documentation for this class was generated from the following files:

- V4_OPA_OutputChannelBipolar.h
- V4_OPA_OutputChannelBipolar.cpp

12 File Documentation

12.1 arduino-pin-toggler/pinToggler.h File Reference

Contains all the code for the arduino-pin-toggler library.

Classes

- class `pinToggler< numPins >`
Class for toggling pins.

Macros

- `#define PRESCALER 1024`
The prescaler for TIMER1.
- `#define FLASH_FREQ_HZ 8`
How often the ISR will trigger.

Enumerations

12.1.1 Detailed Description

Contains all the code for the arduino-pin-toggler library.

12.1.2 Macro Definition Documentation

12.1.2.1 `#define FLASH_FREQ_HZ 8`

How often the ISR will trigger.

This defines the max toggle rate. In combination with the `FLASHRATE` chosen with `pinToggler::setFlashRate`, this determines the rate at which the pins will toggle.

12.1.2.2 `#define PRESCALER 1024`

The prescaler for TIMER1.

This value is used to calculate the correct value to start TIMER1 on, given this prescaler factor.

This does not need to be changed unless you need particularly fast flash rates.

Specifically, if your application requires $16000000 / \text{PRESCALER} / \text{FLASH_FREQ_HZ} \geq 65536$ you will need to alter the prescaler value both here and where it is set in the private `setupSingleton()` method.

12.1.3 Enumeration Type Documentation

12.1.3.1 enum FLASHRATE

Flash rate options.

These rates are used as parameters to the setFlashRate method.

In combination with FLASH_FREQ_HZ, these set the flash rate of an LED.

OFF and ON allow you to disable flashing, leaving the LED either on or off.

Enumerator

ON Always on
OFF Always off
SLOW Flash at FLASH_FREQ_HZ / 8
MEDIUM Flash at FLASH_FREQ_HZ / 2
FAST Flash at FLASH_FREQ_HZ / 4
MAX Flash at FLASH_FREQ_HZ

12.2 CommandDefinitions.h File Reference

Defines all the commands that the device listens for.

```
#include "CommandHandler\CommandHandler.h"
```

Functions

- `template<size_t size>`
`CommandHandlerReturn registerCommands (CommandHandler< size > *h)`
Register the commands.

12.2.1 Detailed Description

Defines all the commands that the device listens for.

This file contains definitions of all the commands that can be issued to the device.

Command processing is done via the CommandHandler class which has its own documentation.

All functions to be called by CommandHandler take exactly the same form: they return void and take a const ParameterLookup object as an argument. This object dispenses c strings as the params, e.g. `params[1]` is the first parameter, etc. (`params[0]` is the command itself). See the definition of `#commandFunction` for the required spec.

All commands are parsed and stored using a hash function for memory efficiency, and are case-insensitive.

The following is a summary of the available commands. If query form of a command with different functionality exists is it listed separately. If it's identical, this is shown as e.g. `*TST(?)`. If no query / non-query form exists, it is not shown in this table; calling it will result in an error.

Command	Description	Params	Output
*TST	Test comms	0	"Loud and clear!"
*TST?	Test comms	0	"Query received"
*IDN(?)	Identify	0	"ARDUINO PID"
*VER(?)	Output version string	0	e.g. "v4.1"
*RST	Reset the device (N.B. does not clear EEPROM)	0	-
STOR	Store a command in EEPROM to be executed on startup	"command to be called on startup"	"Done"
		Multiple commands can be stored by separating them with a ','	"#StoreCommand error: command too long"
RETR	Retrieve any stored commands from EEPROM	0	Stored command or "None"
WIPE	Erase any stored commands in EEPROM	0	"Done"
ERRO(?)	Query the error signal on given channel.	[channel = 1t,2t,1v,2v] for signals 1 or 2 from thermistor input or voltage input	Error signal float
			· If thermistor reading, voltage read from -2.5 -] +2.5
	If this channel is being controlled, return the previously measured value. Else, measure it now and then return.		· If voltage reading, return voltage from -5 -] +5 V
			e.g. "-0.00151"
STAT(?)	Return a string describing the status of the controller	0	t.b.c.
CONT	Set the control signal.	[output channel = 1, 2, 3, 4, BPA, BPB] [voltage = 0 - max for single, -max - +max for BP]	e.g. "#SetControl BP 2.500"
	N.B. If this is called during a lock this will set the control signal to the given value, but this may subsequently be changed by the locking algorithm		
CONT?	Query the control signal	[output channel = 1, 2, 3, 4, BPA, BPB]	e.g. "2.500"
THER?	Check the thermal state of the given channel. For the bipolar channel, return "BAD" if either OPA is overheated	[output channel = 1, 2, 3, 4, BPA, BPB]	e.g. "GOOD" / "BAD"
LIMI	Set software limits on the output voltage	[output channel = 1, 2, 3, 4, BPA, BPB]	e.g. "#SetLimits 1 0.000 3.000"
		[min voltage]	
		[max voltage]	
LIMI?	Query limits	[output channel = 1, 2, 3, 4, BPA, BPB]	e.g. "0.000 3.000"
THRE	Set thresholds at which the LEDs will start flashing.	[thresholdHigh] [thresholdLow]	e.g. "#setThresholds 0.1 0.01"

Command	Description	Params	Output
	LEDs will flash fast when the absolute error signal is] thresholdHigh, slow when it's between thresholdHigh and thresholdLow and light solidly when it's [thresholdLow		
THRE?	Query current LED thresholds	0	e.g. "0.1, 0.01"
CLIM	Set hardware limits on the output current	[output channel = 1, 2, 3, 4, BPA, BPB]	e.g. "#SetCurrentLimit 1 1.↔000"
		[max current in amps]	
CLIM?	Get current hardware on output current	[output channel = 1, 2, 3, 4, BPA, BPB]	e.g. "1.000"
SETP	Set setpoint for lock	[output channel = 1, 2, 3, 4, BPA, BPB]	· If no lock running on this channel: "#SetSetpoint error: no lock running on channel xxx"
			· Else, e.g. "#SetSetpoint 1 0.000"
SETP?	Query lock setpoint	[output channel = 1, 2, 3, 4, BPA, BPB]	· If no lock running on this channel: "#SetSetpoint error: no lock running on channel xxx"
			· Else, e.g. "0.000"
LOCK	Start PID lock	[input channel = 1t,2t,1v,2v]	e.g. "#StartLock 1t BP 0.000 1 0.5 0 10"
		[output channel = 1, 2, 3, 4, BPA, BPB]	
		[setpoint]	
		[Kp] [Ki] [Kd] [N=10]	
VOLT	Output a constant voltage	[output channel = 1, 2, 3, 4, BPA, BPB] [voltage = 0 - max for single, -max - +max for BP]	e.g. "#ConstVoltage 1 2.500"
	N.B. Unlike "CONT", this function will output a constant voltage, disabling any PID lock that was previously running on this output channel		

12.2.2 Function Documentation

12.2.2.1 `template<size_t size> CommandHandlerReturn registerCommands (CommandHandler< size > * h)`

Register the commands.

Register the commandFunction functions with the CommandHandler object, defining:

- The command used to call them
- The number of parameters they must be called with

Parameters

<i>h</i>	Pointer to a CommandHandler object
----------	------------------------------------

Template Parameters

<i>size</i>	Number of commands this CommandHandler can contain. This will be inferred by the number specified in its definition.
-------------	----------------------------------------------------------------------------------------------------------------------

Returns

A CommandHandlerReturn enum detailing any errors that occurred during the execution of this method.

12.3 CommandHandler/CommandHandler.h File Reference

Contains all the code for the Arduino CommandHandler library.

```
#include <Arduino.h>
#include "Microprocessor_Debugging\debugging_disable.h"
#include <EEPROM.h>
```

12.3.1 Detailed Description

Contains all the code for the Arduino CommandHandler library.

Todo Write the CommandHandler documentation

12.4 Pins.h File Reference

Hardware limits and pins.

```
#include "Pins_4chan.h"
```

12.4.1 Detailed Description

Hardware limits and pins.

This file allows the user to select between pin setups for the 2 channel or 4 channel board.

12.5 Pins_2chan.h File Reference

Hardware limits and pins.

```
#include <Arduino.h>
```

Variables

- const uint8_t [ADC_CS](#) = A3
- const uint8_t [ADC_START](#) = 2
- const uint8_t [ADC_DRDY](#) = A2
- const uint8_t [ADC_THERM1](#) = 9
- const bool [ADC_THERM1_HIGHRES](#) = false
- const uint8_t [ADC_THERM2](#) = 8
- const bool [ADC_THERM2_HIGHRES](#) = false
- const uint8_t [ADC_REF](#) = 10
- const uint8_t [ADC_VOLT1_P](#) = 6
- const uint8_t [ADC_VOLT1_N](#) = 7
- const bool [ADC_VOLT1_HIGHRES](#) = false
- const uint8_t [ADC_VOLT2_P](#) = 4
- const uint8_t [ADC_VOLT2_N](#) = 5
- const bool [ADC_VOLT2_HIGHRES](#) = false
- const uint8_t [ADC_TMP](#) = 11
- const uint8_t [ADC_POW](#) = 12
- const uint8_t [DAC_CS](#) = 9
- const uint8_t [OPA_ES1](#) = A0
- const uint8_t [OPA_ES2](#) = A1
- const bool [OPA_1_IS_548](#) = true
- const bool [OPA_2_IS_548](#) = true
- const uint8_t [VPLUS_CHAN_1](#) = 1
- const uint8_t [VPLUS_CHAN_2](#) = 0
- const uint8_t [VLIM_CHAN_1](#) = 2
- const uint8_t [VLIM_CHAN_2](#) = 3
- double [MAX_VOLTAGE](#)
- const double [MAX_INITIAL_CURRENT](#) = 2.0
- const double [OPA_R1](#) = 50
- const double [OPA_R2](#) = 200
- const double [OPA_GAIN](#)
- const uint8_t [LED_1](#) = A5
- const uint8_t [LED_2](#) = A4

12.5.1 Detailed Description

Hardware limits and pins.

This file contains pin numbers and hard-coded limits specific to the 2 channel board

[OPA_GAIN](#) is the gain of the OPA compared to the DAC's output voltage of 2.5V. This is set by resistors on the board, calculated according to the values of [OPA_R1](#) and [OPA_R2](#). It is currently assumed that both OPAs have the same gain.

Todo Add option for user to configure different hardware gains on the two OPAs.

12.5.2 Variable Documentation

12.5.2.1 const uint8_t [ADC_CS](#) = A3

Chip select pin of the ADC on the ATmega

12.5.2.2 `const uint8_t ADC_DRDY = A2`

DRDY pin of the ADC on the ATmega

12.5.2.3 `const uint8_t ADC_POW = 12`

ADC channel for chip power level measurement

12.5.2.4 `const uint8_t ADC_REF = 10`

ADC channel for common thermistor reference input

12.5.2.5 `const uint8_t ADC_START = 2`

~START pin of the ADC on the ATmega

12.5.2.6 `const uint8_t ADC_THERM1 = 9`

ADC channel for thermistor 1 input

12.5.2.7 `const bool ADC_THERM1_HIGHRES = false`

Should thermistor 1 be a high-res measurement?

12.5.2.8 `const uint8_t ADC_THERM2 = 8`

ADC channel for thermistor 2 input

12.5.2.9 `const bool ADC_THERM2_HIGHRES = false`

Should thermistor 2 be a high-res measurement?

12.5.2.10 `const uint8_t ADC_TMP = 11`

ADC channel for chip temperature measurement

12.5.2.11 `const bool ADC_VOLT1_HIGHRES = false`

Should voltage 1 be a high-res measurement?

12.5.2.12 `const uint8_t ADC_VOLT1_N = 7`

ADC channel for arbitrary voltage negative input 1

12.5.2.13 `const uint8_t ADC_VOLT1_P = 6`

ADC channel for arbitrary voltage positive input 1

12.5.2.14 `const bool ADC_VOLT2_HIGHRES = false`

Should voltage 2 be a high-res measurement?

12.5.2.15 `const uint8_t ADC_VOLT2_N = 5`

ADC channel for arbitrary voltage negative input 2

12.5.2.16 `const uint8_t ADC_VOLT2_P = 4`

ADC channel for arbitrary voltage positive input 2

12.5.2.17 `const uint8_t DAC_CS = 9`

Chip select pin of the DAC on the ATmega

12.5.2.18 `const uint8_t LED_1 = A5`

ATmega pin for LED 1, Arduino labelling

12.5.2.19 `const uint8_t LED_2 = A4`

ATmega pin for LED 2, Arduino labelling

12.5.2.20 `const double MAX_INITIAL_CURRENT = 2.0`

Maximum current that the OPAs will be limited to on startup

12.5.2.21 `double MAX_VOLTAGE`

Maximum voltage for the OPAs to output. Calculated in setup()

12.5.2.22 `const bool OPA_1_IS_548 = true`

Is OPA 1 an OPA548 (instead of an OPA549)?

12.5.2.23 `const bool OPA_2_IS_548 = true`

Is OPA 2 an OPA548 (instead of an OPA549)?

12.5.2.24 `const uint8_t OPA_ES1 = A0`

E/S pin for first OPA

12.5.2.25 `const uint8_t OPA_ES2 = A1`

E/S pin for second OPA

12.5.2.26 const double OPA_GAIN

Initial value:

=
$$(OPA_R1 + OPA_R2) / OPA_R1$$

DAC -> OPA gain, set by on-board resistors.

12.5.2.27 const double OPA_R1 = 50

OPA gain resistor 1

12.5.2.28 const double OPA_R2 = 200

OPA gain resistor 2

12.5.2.29 const uint8_t VLIM_CHAN_1 = 2

DAC channel for first OPA's Vlim input

12.5.2.30 const uint8_t VLIM_CHAN_2 = 3

DAC channel for second OPA's Vlim input

12.5.2.31 const uint8_t VPLUS_CHAN_1 = 1

DAC channel for first OPA's V+ input

12.5.2.32 const uint8_t VPLUS_CHAN_2 = 0

DAC channel for second OPA's V+ input

12.6 Pins_4chan.h File Reference

Hardware limits and pins.

```
#include <Arduino.h>
```

Variables

- const uint8_t [ADC_CS](#) = A3
- const uint8_t [ADC_START](#) = 2
- const uint8_t [ADC_DRDY](#) = A2
- const uint8_t [ADC_THERM1](#) = 9
- const bool [ADC_THERM1_HIGHRES](#) = false
- const uint8_t [ADC_THERM2](#) = 8
- const bool [ADC_THERM2_HIGHRES](#) = false
- const uint8_t [ADC_REF](#) = 10
- const uint8_t [ADC_TMP](#) = 11
- const uint8_t [ADC_POW](#) = 12
- const uint8_t [DAC_CS](#) = 10
- const uint8_t [OPA_ES1](#) = A0
- const uint8_t [OPA_ES2](#) = A1
- const bool [OPA_1_IS_548](#) = true
- const bool [OPA_2_IS_548](#) = true
- const uint8_t [DAC_CS_Alt](#) = 9
- const uint8_t [OPA_ES3](#) = 7
- const uint8_t [OPA_ES4](#) = 8
- const bool [OPA_3_IS_548](#) = true
- const bool [OPA_4_IS_548](#) = true
- const uint8_t [VPLUS_CHAN_1](#) = 0
- const uint8_t [VPLUS_CHAN_2](#) = 1
- const uint8_t [VLIM_CHAN_1](#) = 3
- const uint8_t [VLIM_CHAN_2](#) = 2
- const uint8_t [VPLUS_CHAN_3](#) = 0
- const uint8_t [VPLUS_CHAN_4](#) = 1
- const uint8_t [VLIM_CHAN_3](#) = 2
- const uint8_t [VLIM_CHAN_4](#) = 3
- double [MAX_VOLTAGE](#)
- const double [MAX_INITIAL_CURRENT](#) = 2.0
- const double [OPA_R1](#) = 50
- const double [OPA_R2](#) = 200
- const double [OPA_GAIN](#)
- const uint8_t [LED_1](#) = A5
- const uint8_t [LED_2](#) = A4
- const uint8_t [DISABLE_SERIAL_CTRL](#) = A6
- const uint8_t [DIVIDED_SUPPLY_VOLTAGE](#) = A7
- const double [DIVIDED_SUPPLY_FACTOR](#) = 11.0

12.6.1 Detailed Description

Hardware limits and pins.

This file contains pin numbers and hard-coded limits specific to the 4 channel board

[OPA_GAIN](#) is the gain of the OPA compared to the DAC's output voltage of 2.5V. This is set by resistors on the board, calculated according to the values of [OPA_R1](#) and [OPA_R2](#). It is currently assumed that both OPAs have the same gain.

Todo Add option for user to configure different hardware gains on the two OPAs.

12.6.2 Variable Documentation

12.6.2.1 `const uint8_t ADC_CS = A3`

Chip select pin of the ADC on the ATmega

12.6.2.2 `const uint8_t ADC_DRDY = A2`

DRDY pin of the ADC on the ATmega

12.6.2.3 `const uint8_t ADC_POW = 12`

ADC channel for chip power level measurement

12.6.2.4 `const uint8_t ADC_REF = 10`

ADC channel for common thermistor reference input

12.6.2.5 `const uint8_t ADC_START = 2`

~START pin of the ADC on the ATmega

12.6.2.6 `const uint8_t ADC_THERM1 = 9`

ADC channel for thermistor 1 input

12.6.2.7 `const bool ADC_THERM1_HIGHRES = false`

Should thermistor 1 be a high-res measurement?

12.6.2.8 `const uint8_t ADC_THERM2 = 8`

ADC channel for thermistor 2 input

12.6.2.9 `const bool ADC_THERM2_HIGHRES = false`

Should thermistor 2 be a high-res measurement?

12.6.2.10 `const uint8_t ADC_TMP = 11`

ADC channel for chip temperature measurement

12.6.2.11 `const uint8_t DAC_CS = 10`

Chip select pin of the DAC on the ATmega

12.6.2.12 `const uint8_t DAC_CS_Alt = 9`

Chip select pin of the DAC on the ATmega

12.6.2.13 `const uint8_t DISABLE_SERIAL_CTRL = A6`

ATMega pin for disabling serial control: 0 or 5V

12.6.2.14 `const double DIVIDED_SUPPLY_FACTOR = 11.0`

Factor by which the supply voltage has been divided

12.6.2.15 `const uint8_t DIVIDED_SUPPLY_VOLTAGE = A7`

ATMega pin for reading supply voltage: analog voltage

12.6.2.16 `const uint8_t LED_1 = A5`

ATMega pin for LED 1, Arduino labelling

12.6.2.17 `const uint8_t LED_2 = A4`

ATMega pin for LED 2, Arduino labelling

12.6.2.18 `const double MAX_INITIAL_CURRENT = 2.0`

Maximum current that the OPAs will be limited to on startup

12.6.2.19 `double MAX_VOLTAGE`

Maximum voltage for the OPAs to output. Calculated in setup()

12.6.2.20 `const bool OPA_1_IS_548 = true`

Is OPA 1 an OPA548 (instead of an OPA549)?

12.6.2.21 `const bool OPA_2_IS_548 = true`

Is OPA 2 an OPA548 (instead of an OPA549)?

12.6.2.22 `const bool OPA_3_IS_548 = true`

Is OPA 1 an OPA548 (instead of an OPA549)?

12.6.2.23 `const bool OPA_4_IS_548 = true`

Is OPA 2 an OPA548 (instead of an OPA549)?

12.6.2.24 `const uint8_t OPA_ES1 = A0`

E/S pin for first OPA

12.6.2.25 `const uint8_t OPA_ES2 = A1`

E/S pin for second OPA

12.6.2.26 `const uint8_t OPA_ES3 = 7`

E/S pin for first OPA

12.6.2.27 `const uint8_t OPA_ES4 = 8`

E/S pin for second OPA

12.6.2.28 `const double OPA_GAIN`

Initial value:

=

$$(OPA_R1 + OPA_R2) / OPA_R1$$

DAC -> OPA gain, set by on-board resistors.

12.6.2.29 `const double OPA_R1 = 50`

OPA gain resistor 1

12.6.2.30 `const double OPA_R2 = 200`

OPA gain resistor 2

12.6.2.31 `const uint8_t VLIM_CHAN_1 = 3`

DAC channel for second OPA's Vlim input

12.6.2.32 `const uint8_t VLIM_CHAN_2 = 2`

DAC channel for first OPA's Vlim input

12.6.2.33 `const uint8_t VLIM_CHAN_3 = 2`

DAC channel for second OPA's Vlim input

12.6.2.34 `const uint8_t VLIM_CHAN_4 = 3`

DAC channel for first OPA's Vlim input

12.6.2.35 `const uint8_t VPLUS_CHAN_1 = 0`

DAC channel for second OPA's V+ input

12.6.2.36 `const uint8_t VPLUS_CHAN_2 = 1`

DAC channel for first OPA's V+ input

12.6.2.37 `const uint8_t VPLUS_CHAN_3 = 0`

DAC channel for second OPA's V+ input

12.6.2.38 `const uint8_t VPLUS_CHAN_4 = 1`

DAC channel for first OPA's V+ input

Index

- [__lowGainThreshold](#)
 - [YbCtrl::V4_ADC_ChannelPair, 38](#)
 - [__maxLowReadings](#)
 - [YbCtrl::V4_ADC_ChannelPair, 38](#)
- [ADC_CS](#)
 - [Pins_2chan.h, 55](#)
 - [Pins_4chan.h, 60](#)
- [ADC_DRDY](#)
 - [Pins_2chan.h, 55](#)
 - [Pins_4chan.h, 60](#)
- [ADC_POW](#)
 - [Pins_2chan.h, 56](#)
 - [Pins_4chan.h, 60](#)
- [ADC_REF](#)
 - [Pins_2chan.h, 56](#)
 - [Pins_4chan.h, 60](#)
- [ADC_START](#)
 - [Pins_2chan.h, 56](#)
 - [Pins_4chan.h, 60](#)
- [ADC_THERM1](#)
 - [Pins_2chan.h, 56](#)
 - [Pins_4chan.h, 60](#)
- [ADC_THERM1_HIGHRES](#)
 - [Pins_2chan.h, 56](#)
 - [Pins_4chan.h, 60](#)
- [ADC_THERM2](#)
 - [Pins_2chan.h, 56](#)
 - [Pins_4chan.h, 60](#)
- [ADC_THERM2_HIGHRES](#)
 - [Pins_2chan.h, 56](#)
 - [Pins_4chan.h, 60](#)
- [ADC_TMP](#)
 - [Pins_2chan.h, 56](#)
 - [Pins_4chan.h, 60](#)
- [ADC_VOLT1_HIGHRES](#)
 - [Pins_2chan.h, 56](#)
- [ADC_VOLT1_N](#)
 - [Pins_2chan.h, 56](#)
- [ADC_VOLT1_P](#)
 - [Pins_2chan.h, 56](#)
- [ADC_VOLT2_HIGHRES](#)
 - [Pins_2chan.h, 56](#)
- [ADC_VOLT2_N](#)
 - [Pins_2chan.h, 57](#)
- [ADC_VOLT2_P](#)
 - [Pins_2chan.h, 57](#)
- [abortReading](#)
 - [YbCtrl::ErrorChannel, 24](#)
 - [YbCtrl::V4_ADC_ChannelPair, 35](#)
- [addConflictingChannel](#)
 - [YbCtrl::CtrlChannel, 19](#)
 - [YbCtrl::V4_OPA_OutputChannel, 40](#)
 - [YbCtrl::V4_OPA_OutputChannelBipolar, 45](#)
- [Algorithms, 7](#)
- [arduino-pin-toggler/pinToggler.h, 50](#)
- [CHANNEL_NOT_MANAGED](#)
 - [YbCtrl, 10](#)
- [closeConflictingControllers](#)
 - [YbCtrl::CtrlChannel, 19](#)
 - [YbCtrl::V4_OPA_OutputChannel, 40](#)
 - [YbCtrl::V4_OPA_OutputChannelBipolar, 45](#)
- [CommandDefinitions.h, 51](#)
 - [registerCommands, 53](#)
- [CommandHandler/CommandHandler.h, 54](#)
- [Control signal output, 8](#)
- [Controller](#)
 - [YbCtrl::Controller, 15](#)
- [CtrlChannelReturn](#)
 - [YbCtrl, 10](#)
- [DAC_CS_Alt](#)
 - [Pins_4chan.h, 60](#)
- [DAC_CS](#)
 - [Pins_2chan.h, 57](#)
 - [Pins_4chan.h, 60](#)
- [DISABLE_SERIAL_CTRL](#)
 - [Pins_4chan.h, 60](#)
- [DIVIDED_SUPPLY_FACTOR](#)
 - [Pins_4chan.h, 61](#)
- [DIVIDED_SUPPLY_VOLTAGE](#)
 - [Pins_4chan.h, 61](#)
- [doLoop](#)
 - [YbCtrl::Controller, 15](#)
- [Error signal input, 9](#)
- [ErrorChannelReturn](#)
 - [YbCtrl, 10](#)
- [FAST](#)
 - [pinToggler.h, 51](#)
- [FLASH_FREQ_HZ](#)
 - [pinToggler.h, 50](#)
- [FLASHRATE](#)
 - [pinToggler.h, 51](#)
- [getAlgorithm](#)
 - [YbCtrl::Controller, 15](#)
- [getContainingController](#)
 - [YbCtrl::CtrlChannel, 19](#)
 - [YbCtrl::V4_OPA_OutputChannel, 40](#)
 - [YbCtrl::V4_OPA_OutputChannelBipolar, 46](#)
- [getCtrlChannel](#)
 - [YbCtrl::Controller, 16](#)
- [getCurrentLimit](#)
 - [YbCtrl::CtrlChannel, 20](#)
 - [YbCtrl::V4_OPA_OutputChannel, 41](#)
 - [YbCtrl::V4_OPA_OutputChannelBipolar, 46](#)
- [getErrorInterface](#)
 - [YbCtrl::Controller, 16](#)

- getLimits
 - YbCtrl::CtrlChannel, 20
 - YbCtrl::V4_OPA_OutputChannel, 41
 - YbCtrl::V4_OPA_OutputChannelBipolar, 47
- getPin
 - pinToggler, 32
- getReading
 - YbCtrl::ErrorChannel, 24
 - YbCtrl::V4_ADC_ChannelPair, 35
- getSetpoint
 - YbCtrl::Algorithm, 12
 - YbCtrl::PIDAlgorithm, 29
- globalReadInProgress
 - YbCtrl::ErrorChannel, 24
 - YbCtrl::V4_ADC_ChannelPair, 36
- INVALID_PARAMETER
 - YbCtrl, 10
- init
 - pinToggler, 32
- isOverheated
 - YbCtrl::CtrlChannel, 21
 - YbCtrl::V4_OPA_OutputChannel, 42
 - YbCtrl::V4_OPA_OutputChannelBipolar, 47
- LED_1
 - Pins_2chan.h, 57
 - Pins_4chan.h, 61
- LED_2
 - Pins_2chan.h, 57
 - Pins_4chan.h, 61
- lockingAlgo
 - YbCtrl::Algorithm, 12
 - YbCtrl::PIDAlgorithm, 29
- MAX_INITIAL_CURRENT
 - Pins_2chan.h, 57
 - Pins_4chan.h, 61
- MAX_VOLTAGE
 - Pins_2chan.h, 57
 - Pins_4chan.h, 61
- MAX
 - pinToggler.h, 51
- MEDIUM
 - pinToggler.h, 51
- NO_ERROR
 - YbCtrl, 10, 11
- NO_SUCH_CHANNEL
 - YbCtrl, 10, 11
- NOT_IMPLEMENTED
 - YbCtrl, 10, 11
- OFF
 - pinToggler.h, 51
- OPA_1_IS_548
 - Pins_2chan.h, 57
 - Pins_4chan.h, 61
- OPA_2_IS_548
 - Pins_2chan.h, 57
 - Pins_4chan.h, 61
- OPA_3_IS_548
 - Pins_4chan.h, 61
- OPA_4_IS_548
 - Pins_4chan.h, 61
- OPA_ES1
 - Pins_2chan.h, 57
 - Pins_4chan.h, 61
- OPA_ES2
 - Pins_2chan.h, 57
 - Pins_4chan.h, 61
- OPA_ES3
 - Pins_4chan.h, 62
- OPA_ES4
 - Pins_4chan.h, 62
- OPA_GAIN
 - Pins_2chan.h, 57
 - Pins_4chan.h, 62
- OPA_R1
 - Pins_2chan.h, 58
 - Pins_4chan.h, 62
- OPA_R2
 - Pins_2chan.h, 58
 - Pins_4chan.h, 62
- OUT_OF_MEMORY
 - YbCtrl, 10
- OUT_OF_RANGE
 - YbCtrl, 11
- ON
 - pinToggler.h, 51
- operator bool
 - YbCtrl::Algorithm, 12
 - YbCtrl::Controller, 16
 - YbCtrl::CtrlChannel, 21
 - YbCtrl::ErrorChannel, 25
 - YbCtrl::V4_ADC_ChannelPair, 36
 - YbCtrl::V4_OPA_OutputChannel, 42
 - YbCtrl::V4_OPA_OutputChannelBipolar, 47
- output
 - YbCtrl::Algorithm, 12
 - YbCtrl::PIDAlgorithm, 29
- PGA_ERROR
 - YbCtrl, 11
- PIDAlgorithm
 - YbCtrl::PIDAlgorithm, 28
- PRESCALER
 - pinToggler.h, 50
- pinToggler
 - getPin, 32
 - init, 32
 - setFlashRate, 32
- pinToggler< numPins >, 31
- pinToggler.h
 - FAST, 51
 - FLASH_FREQ_HZ, 50
 - FLASHRATE, 51
 - MAX, 51

- MEDIUM, [51](#)
- OFF, [51](#)
- ON, [51](#)
- PRESCALER, [50](#)
- SLOW, [51](#)
- Pins.h, [54](#)
- Pins_2chan.h, [54](#)
 - ADC_CS, [55](#)
 - ADC_DRDY, [55](#)
 - ADC_POW, [56](#)
 - ADC_REF, [56](#)
 - ADC_START, [56](#)
 - ADC_THERM1, [56](#)
 - ADC_THERM1_HIGHRES, [56](#)
 - ADC_THERM2, [56](#)
 - ADC_THERM2_HIGHRES, [56](#)
 - ADC_TMP, [56](#)
 - ADC_VOLT1_HIGHRES, [56](#)
 - ADC_VOLT1_N, [56](#)
 - ADC_VOLT1_P, [56](#)
 - ADC_VOLT2_HIGHRES, [56](#)
 - ADC_VOLT2_N, [57](#)
 - ADC_VOLT2_P, [57](#)
 - DAC_CS, [57](#)
 - LED_1, [57](#)
 - LED_2, [57](#)
 - MAX_INITIAL_CURRENT, [57](#)
 - MAX_VOLTAGE, [57](#)
 - OPA_1_IS_548, [57](#)
 - OPA_2_IS_548, [57](#)
 - OPA_ES1, [57](#)
 - OPA_ES2, [57](#)
 - OPA_GAIN, [57](#)
 - OPA_R1, [58](#)
 - OPA_R2, [58](#)
 - VLIM_CHAN_1, [58](#)
 - VLIM_CHAN_2, [58](#)
 - VPLUS_CHAN_1, [58](#)
 - VPLUS_CHAN_2, [58](#)
- Pins_4chan.h, [58](#)
 - ADC_CS, [60](#)
 - ADC_DRDY, [60](#)
 - ADC_POW, [60](#)
 - ADC_REF, [60](#)
 - ADC_START, [60](#)
 - ADC_THERM1, [60](#)
 - ADC_THERM1_HIGHRES, [60](#)
 - ADC_THERM2, [60](#)
 - ADC_THERM2_HIGHRES, [60](#)
 - ADC_TMP, [60](#)
 - DAC_CS_Alt, [60](#)
 - DAC_CS, [60](#)
 - DISABLE_SERIAL_CTRL, [60](#)
 - DIVIDED_SUPPLY_FACTOR, [61](#)
 - DIVIDED_SUPPLY_VOLTAGE, [61](#)
 - LED_1, [61](#)
 - LED_2, [61](#)
 - MAX_INITIAL_CURRENT, [61](#)
 - MAX_VOLTAGE, [61](#)
 - OPA_1_IS_548, [61](#)
 - OPA_2_IS_548, [61](#)
 - OPA_3_IS_548, [61](#)
 - OPA_4_IS_548, [61](#)
 - OPA_ES1, [61](#)
 - OPA_ES2, [61](#)
 - OPA_ES3, [62](#)
 - OPA_ES4, [62](#)
 - OPA_GAIN, [62](#)
 - OPA_R1, [62](#)
 - OPA_R2, [62](#)
 - VLIM_CHAN_1, [62](#)
 - VLIM_CHAN_2, [62](#)
 - VLIM_CHAN_3, [62](#)
 - VLIM_CHAN_4, [62](#)
 - VPLUS_CHAN_1, [62](#)
 - VPLUS_CHAN_2, [62](#)
 - VPLUS_CHAN_3, [62](#)
 - VPLUS_CHAN_4, [62](#)
- readInProgress
 - YbCtrl::ErrorChannel, [25](#)
 - YbCtrl::V4_ADC_ChannelPair, [37](#)
- readingReady
 - YbCtrl::ErrorChannel, [25](#)
 - YbCtrl::V4_ADC_ChannelPair, [36](#)
- readingTimeout
 - YbCtrl::ErrorChannel, [25](#)
 - YbCtrl::V4_ADC_ChannelPair, [36](#)
- recallCtrl
 - YbCtrl::CtrlChannel, [21](#)
 - YbCtrl::V4_OPA_OutputChannel, [42](#)
 - YbCtrl::V4_OPA_OutputChannelBipolar, [48](#)
- recallError
 - YbCtrl::ErrorChannel, [26](#)
 - YbCtrl::V4_ADC_ChannelPair, [37](#)
- registerCommands
 - CommandDefinitions.h, [53](#)
- replaceCtrlChannel
 - YbCtrl::Controller, [16](#)
- reportState
 - YbCtrl::Algorithm, [13](#)
 - YbCtrl::Controller, [17](#)
 - YbCtrl::PIDAlgorithm, [30](#)
- reset
 - YbCtrl::Controller, [17](#)
- SLOW
 - pinToggler.h, [51](#)
- setContainingController
 - YbCtrl::CtrlChannel, [21](#)
 - YbCtrl::V4_OPA_OutputChannel, [42](#)
 - YbCtrl::V4_OPA_OutputChannelBipolar, [48](#)
- setCtrl
 - YbCtrl::CtrlChannel, [22](#)
 - YbCtrl::V4_OPA_OutputChannel, [43](#)
 - YbCtrl::V4_OPA_OutputChannelBipolar, [48](#)
- setCurrentLimit

- YbCtrl::CtrlChannel, 22
- YbCtrl::V4_OPA_OutputChannel, 43
- YbCtrl::V4_OPA_OutputChannelBipolar, 49
- setFlashRate
 - pinToggler, 32
- setLimits
 - YbCtrl::Algorithm, 13
 - YbCtrl::CtrlChannel, 22
 - YbCtrl::PIDAlgorithm, 30
 - YbCtrl::V4_OPA_OutputChannel, 43
 - YbCtrl::V4_OPA_OutputChannelBipolar, 49
- setOutput
 - YbCtrl::Algorithm, 13
 - YbCtrl::PIDAlgorithm, 30
- setSetpoint
 - YbCtrl::Algorithm, 14
 - YbCtrl::PIDAlgorithm, 30
- startReading
 - YbCtrl::ErrorChannel, 26
 - YbCtrl::V4_ADC_ChannelPair, 38
- TIMEOUT
 - YbCtrl, 11
- V4_ADC_ChannelPair
 - YbCtrl::V4_ADC_ChannelPair, 35
- V4_OPA_OutputChannel
 - YbCtrl::V4_OPA_OutputChannel, 39
- VLIM_CHAN_1
 - Pins_2chan.h, 58
 - Pins_4chan.h, 62
- VLIM_CHAN_2
 - Pins_2chan.h, 58
 - Pins_4chan.h, 62
- VLIM_CHAN_3
 - Pins_4chan.h, 62
- VLIM_CHAN_4
 - Pins_4chan.h, 62
- VPLUS_CHAN_1
 - Pins_2chan.h, 58
 - Pins_4chan.h, 62
- VPLUS_CHAN_2
 - Pins_2chan.h, 58
 - Pins_4chan.h, 62
- VPLUS_CHAN_3
 - Pins_4chan.h, 62
- VPLUS_CHAN_4
 - Pins_4chan.h, 62
- WRITING_TO_REG_FAILED
 - YbCtrl, 11
- YbCtrl, 10
 - CHANNEL_NOT_MANAGED, 10
 - CtrlChannelReturn, 10
 - ErrorChannelReturn, 10
 - INVALID_PARAMETER, 10
 - NO_ERROR, 10, 11
 - NO_SUCH_CHANNEL, 10, 11
 - NOT_IMPLEMENTED, 10, 11
 - OUT_OF_MEMORY, 10
 - OUT_OF_RANGE, 11
 - PGA_ERROR, 11
 - TIMEOUT, 11
 - WRITING_TO_REG_FAILED, 11
- YbCtrl::Algorithm, 11
 - getSetpoint, 12
 - lockingAlgo, 12
 - operator bool, 12
 - output, 12
 - reportState, 13
 - setLimits, 13
 - setOutput, 13
 - setSetpoint, 14
- YbCtrl::Controller, 14
 - Controller, 15
 - doLoop, 15
 - getAlgorithm, 15
 - getCtrlChannel, 16
 - getErrorInterface, 16
 - operator bool, 16
 - replaceCtrlChannel, 16
 - reportState, 17
 - reset, 17
- YbCtrl::CtrlChannel, 18
 - addConflictingChannel, 19
 - closeConflictingControllers, 19
 - getContainingController, 19
 - getCurrentLimit, 20
 - getLimits, 20
 - isOverheated, 21
 - operator bool, 21
 - recallCtrl, 21
 - setContainingController, 21
 - setCtrl, 22
 - setCurrentLimit, 22
 - setLimits, 22
- YbCtrl::ErrorChannel, 23
 - abortReading, 24
 - getReading, 24
 - globalReadInProgress, 24
 - operator bool, 25
 - readInProgress, 25
 - readingReady, 25
 - readingTimeout, 25
 - recallError, 26
 - startReading, 26
- YbCtrl::PIDAlgorithm, 27
 - getSetpoint, 29
 - lockingAlgo, 29
 - output, 29
 - PIDAlgorithm, 28
 - reportState, 30
 - setLimits, 30
 - setOutput, 30
 - setSetpoint, 30
- YbCtrl::TemporaryLooper, 33

- YbCtrl::V4_ADC_ChannelPair, 34
 - __lowGainThreshold, 38
 - __maxLowReadings, 38
 - abortReading, 35
 - getReading, 35
 - globalReadInProgress, 36
 - operator bool, 36
 - readInProgress, 37
 - readingReady, 36
 - readingTimeout, 36
 - recallError, 37
 - startReading, 38
 - V4_ADC_ChannelPair, 35
- YbCtrl::V4_OPA_OutputChannel, 38
 - addConflictingChannel, 40
 - closeConflictingControllers, 40
 - getContainingController, 40
 - getCurrentLimit, 41
 - getLimits, 41
 - isOverheated, 42
 - operator bool, 42
 - recallCtrl, 42
 - setContainingController, 42
 - setCtrl, 43
 - setCurrentLimit, 43
 - setLimits, 43
 - V4_OPA_OutputChannel, 39
- YbCtrl::V4_OPA_OutputChannelBipolar, 44
 - addConflictingChannel, 45
 - closeConflictingControllers, 45
 - getContainingController, 46
 - getCurrentLimit, 46
 - getLimits, 47
 - isOverheated, 47
 - operator bool, 47
 - recallCtrl, 48
 - setContainingController, 48
 - setCtrl, 48
 - setCurrentLimit, 49
 - setLimits, 49