Yb Temperature Controller

Generated by Doxygen 1.8.14

Contents

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.

Alternativly, 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 arbitary 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 framentation, 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:

5 Namespace Index

5.1 Namespace List

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

6 Hierarchical Index

6.1 Class Hierarchy

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

7 Class Index

7.1 Class List

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

8 File Index

8.1 File List

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

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 collectivly 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 collectivly 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

- enum CtrlChannelReturn {
  CtrlChannelReturn::NO_ERROR = 0, CtrlChannelReturn::NOT_IMPLEMENTED, CtrlChannelReturn::NO_SUCH_CHANNEL,
  CtrlChannelReturn::CHANNEL_NOT_MANAGED,
  CtrlChannelReturn::INVALID_PARAMETER, CtrlChannelReturn::OUT_OF_MEMORY }

    Error codes for CtrlChannel operation.
- enum ErrorChannelReturn {
  ErrorChannelReturn::NO_ERROR = 0, ErrorChannelReturn::NOT_IMPLEMENTED, ErrorChannelReturn::WRITING_TO_REG
  ErrorChannelReturn::TIMEOUT,
  ErrorChannelReturn::OUT_OF_RANGE, ErrorChannelReturn::PGA_ERROR, ErrorChannelReturn::NO_SUCH_CHANNEL
  }

    Error codes for ErrorChannel operation.

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 CtrlChannelReturn

```
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 ErrorChannelReturn

```
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

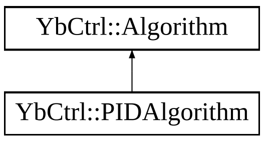
11 Class Documentation

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 getSetpoint()

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

Gets the setpoint.

Returns

The current setpoint.

Implemented in YbCtrl::PIDAlgorithm.

11.1.2.2 lockingAlgo()

```
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 operator bool()

```
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 output()

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

Do the locking calculation.

Parameters

in	*input*	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 reportState()

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

Report state.

Send a string that identifies the currently running algorithm.

Parameters

out	*ptr*	A buffer to contain the output. Should be at least 128 chars long.

Returns

Returns ptr

Implemented in YbCtrl::PIDAlgorithm.

11.1.2.6 setLimits()

```
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	*min*	The new minimum
in	*max*	The new maximum

Implemented in YbCtrl::PIDAlgorithm.

11.1.2.7 setOutput()

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

Sets the output.

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

Parameters

in	*output*	The new output, from -1 to +1.

Implemented in YbCtrl::PIDAlgorithm.

11.1.2.8 setSetpoint()

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

Sets the setpoint.

Change setpoint, smoothly if possible.

Parameters

in	*setpoint*	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 ()
  Constuct an empty Controller.
- Controller (ErrorChannel ∗errorInt, CtrlChannel ∗ctrlInt, Algorithm ∗algorithm)
  Constuct 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 Controller() [1/2]

```
YbCtrl::Controller::Controller ( )
```

Constuct an empty Controller.

This empty controller is marked invalid.

11.2.2.2 Controller() [2/2]

```
YbCtrl::Controller::Controller (
             ErrorChannel ∗ errorInt,
             CtrlChannel ∗ ctrlInt,
             Algorithm ∗ algorithm )
```

Constuct a Controller.

Constuct 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 constuctor 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	*errorInt*	The error interface
in	*ctrlInt*	The control interface
in	*algorithm*	The algorithm

11.2.3 Member Function Documentation

11.2.3.1 doLoop()

```
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 getAlgorithm()

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

Gets the algorithm.

Returns

The algorithm.

11.2.3.3 getCtrlChannel()

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

Gets the control channel.

Returns

The control channel.

11.2.3.4 getErrorInterface()

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

Gets the error interface.

Returns

The error interface.

11.2.3.5 operator bool()

```
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 replaceCtrlChannel() [1/2]

```
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 replaceCtrlChannel() [2/2]

```
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`	*newInterface*	The new CtrlChannel

Returns

Error codes

Return values

0	Success
1	Error: new CtrlChannel is already assigned.

11.2.3.8 reportState()

```
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`	*ptr*	Pointer to a buffer. Should hold at least 128 chars.

Returns

Returns ptr

11.2.3.9 reset()

```
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:

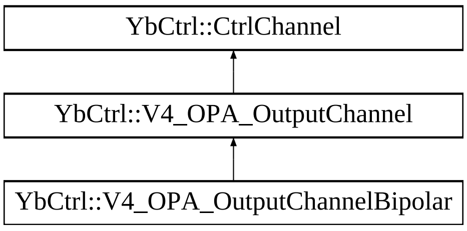
- Controller.h
- Controller.cpp

11.3 YbCtrl::CtrlChannel Class Reference

Abstract object to allow outputting a control signal.

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

Inheritance diagram for YbCtrl::CtrlChannel:


YbCtrl::CtrlChannel
YbCtrl::V4_OPA_OutputChannel
YbCtrl::V4_OPA_OutputChannelBipolar

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 tempate 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 addConflictingChannel()

```
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 closeConflictingControllers()

```
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

Similarily 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 getContainingController() [1/2]

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

Gets the containing Controller.

Returns

The containing controller. NULL if none present.

11.3.2.4 getContainingController() [2/2]

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

Gets the containing controller.

Parameters

out	*out*	The containing Controller

Returns

Error code

11.3.2.5 getCurrentLimit() [1/2]

```
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 getCurrentLimit() [2/2]

```
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 getLimits()

```
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 isOverheated()

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

Check for overheated.

Check if this channel is overheated

Parameters

out	*state*	True if overheated, false otherwise

Returns

Error return

Reimplemented in YbCtrl::V4_OPA_OutputChannel, and YbCtrl::V4_OPA_OutputChannelBipolar.

11.3.2.9 operator bool()

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

Is this object valid?

Return true if this object has been constucted with valid parameters

Implemented in YbCtrl::V4_OPA_OutputChannel.

11.3.2.10 recallCtrl()

```
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 setContainingController()

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

Sets the containing Controller.

Set this object's managing Controller to the given target

Parameters

newController	The new managing Controller

11.3.2.12 setCtrl()

```
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 explaination as to why this structure is needed

Parameters

val	The value, from -1 to +1

11.3.2.13 setCurrentLimit()

```
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

`in`	*val*	The current limit in Amps

Returns

> Error code

Reimplemented in YbCtrl::V4_OPA_OutputChannel, and YbCtrl::V4_OPA_OutputChannelBipolar.

11.3.2.14 setLimits()

```
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.
Similarily 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	*minimum*	The new minimum
in	*maximum*	The new maximum

Returns

Error status

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

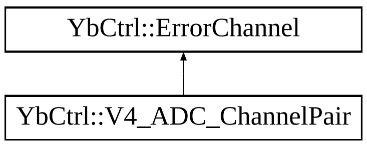
- CtrlChannel.h
- CtrlChannel.cpp

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 tempate 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 abortReading()

```
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 getReading()

```
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	*readingOutput*	Output variable for the reading.

Returns

ErrorChannelReturn error code.

Implemented in YbCtrl::V4_ADC_ChannelPair.

11.4.2.3 globalReadInProgress()

```
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 operator bool()

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

Is this object valid?

Return true if this object has been constucted with valid parameters

Implemented in YbCtrl::V4_ADC_ChannelPair.

11.4.2.5 readingReady()

```
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 readingTimeout()

```
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 readInProgress()

```
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 recallError() [1/2]

```
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 recallError() [2/2]

```
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	*output*	Target for the error signal

Returns

Error code

Implemented in YbCtrl::V4_ADC_ChannelPair.

11.4.2.10 startReading()

```
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:

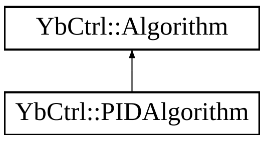
- ErrorChannel.h

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)

  Constuctor.
- PIDAlgorithm ()

  Alternative constuctor.
- 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 constuction.

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 consructor, 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 PIDAlgorithm() [1/2]

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

Constuctor.

Parameters

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

11.5.2.2 PIDAlgorithm() `[2/2]`

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

Alternative constuctor.

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 getSetpoint()

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

Gets the setpoint.

Returns

The current setpoint.

Implements YbCtrl::Algorithm.

11.5.3.2 lockingAlgo()

```
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 output()

```
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 explaination of how this works

Parameters

in	*input*	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 reportState()

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

Report state.

Send a string that identifies the currently running algorithm.

Parameters

out	*ptr*	A buffer to contain the output. Should be at least 128 chars long.

Returns

Returns ptr

Implements YbCtrl::Algorithm.

11.5.3.5 setLimits()

```
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	*min*	The new minimum
in	*max*	The new maximum

Implements YbCtrl::Algorithm.

11.5.3.6 setOutput()

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

Sets the output.

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

Parameters

in	*output*	The new output, from -1 to +1.

Implements YbCtrl::Algorithm.

11.5.3.7 setSetpoint()

```
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 arbitary 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

numPins	The number of pins that this object will handle.

11.6.2 Member Function Documentation

11.6.2.1 getPin()

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

Return the pin corresponding to a given LED

Parameters

LED	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 init()

```
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

in	*LED_Pins*	Pointer to a <numPins> dimention 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

Return values

-2	Stack assignment failed: out of memory

Returns

Error code

11.6.2.3 setFlashRate()

```
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 refered to by their zero-indexed position in the array passed to init().

Parameters

LED	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().
rate	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:

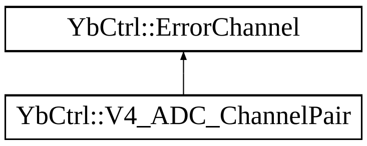
- TemporaryLooper.h

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 constuctor.
- V4_ADC_ChannelPair (uint8_t channel1, uint8_t channel2, bool highRes=true)

  Constuctor.

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 V4_ADC_ChannelPair() [1/2]

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

Default constuctor.

The object created thus will be marked as invalid

11.8.2.2 V4_ADC_ChannelPair() [2/2]

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

Constuctor.

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

Parameters

in	*channel1*	ADC channel 1
in	*channel2*	ADC channel 2
in	*highRes*	Should this channel be measured quickly or carefully?

11.8.3 Member Function Documentation

11.8.3.1 abortReading()

```
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 getReading()

```
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	*readingOutput*	Output variable for the reading.

Returns

ErrorChannelReturn error code.

Implements YbCtrl::ErrorChannel.

11.8.3.3 globalReadInProgress()

```
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 operator bool()

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

Is this object valid?

Return true if this object has been constucted with valid parameters

Implements YbCtrl::ErrorChannel.

11.8.3.5 readingReady()

```
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 readingTimeout()

```
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 readInProgress()

```
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 recallError() [1/2]

```
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 recallError() [2/2]

```
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 startReading()

```
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 __lowGainThreshold

```
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 __maxLowReadings

```
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:

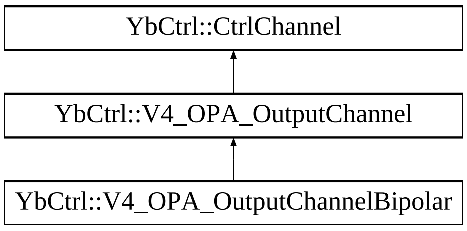
- V4_ADC_ChannelPair.h
- V4_ADC_ChannelPair.cpp

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:


YbCtrl::CtrlChannel
YbCtrl::V4_OPA_OutputChannel
YbCtrl::V4_OPA_OutputChannelBipolar

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 V4_OPA_OutputChannel()

```
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	*channelVPlus*	DAC channel corresponding to V+
in	*channelVLim*	DAC channel corresponsing to Vlim
in	*OPA_ES*	ATMega pin corresponding to the OPA's E/S pin
in	*smallerOPA*	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 addConflictingChannel()

```
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 closeConflictingControllers()

```
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

Similarily 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 getContainingController() [1/2]

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

Gets the containing Controller.

Returns

The containing controller. NULL if none present.

11.9.3.4 getContainingController() [2/2]

```
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 getCurrentLimit() [1/2]

```
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 getCurrentLimit() [2/2]

```
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 getLimits()

```
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 isOverheated()

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

Check for overheated.

Check if this channel is overheated

Parameters

out	*state*	True if overheated, false otherwise

Returns

Error return

Reimplemented from YbCtrl::CtrlChannel.

Reimplemented in YbCtrl::V4_OPA_OutputChannelBipolar.

11.9.3.9 operator bool()

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

Is this object valid?

Return true if this object has been constucted with valid parameters

Implements YbCtrl::CtrlChannel.

11.9.3.10 recallCtrl()

```
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 setContainingController()

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

Sets the containing Controller.

Set this object's managing Controller to the given target

Parameters

newController	The new managing Controller

11.9.3.12 setCtrl()

```
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 explaination as to why this structure is needed

Parameters

val	The value, from -1 to +1

11.9.3.13 setCurrentLimit()

```
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

in	*val*	The current limit in Amps

Returns

Error code

Reimplemented from YbCtrl::CtrlChannel.

Reimplemented in YbCtrl::V4_OPA_OutputChannelBipolar.

11.9.3.14 setLimits()

```
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	*minimum*	The new minimum
in	*maximum*	The new maximum

Returns

Error status

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

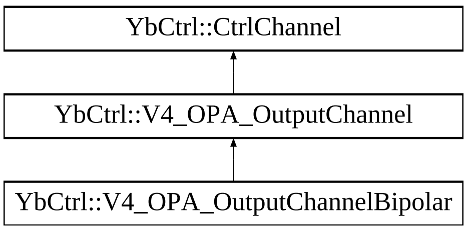
- V4_OPA_OutputChannel.h
- V4_OPA_OutputChannel.cpp

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:


YbCtrl::CtrlChannel
YbCtrl::V4_OPA_OutputChannel
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 addConflictingChannel()

```
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 closeConflictingControllers()

```
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

Similarily 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 getContainingController() [1/2]

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

Gets the containing Controller.

Returns

The containing controller. NULL if none present.

11.10.2.4 getContainingController() [2/2]

```
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 getCurrentLimit() [1/2]

```
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 getCurrentLimit() [2/2]

```
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 getLimits()

```
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 isOverheated()

```
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 operator bool()

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

Is this object valid?

Return true if this object has been constucted with valid parameters

Implements YbCtrl::CtrlChannel.

11.10.2.10 recallCtrl()

```
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 setContainingController()

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

Sets the containing Controller.

Set this object's managing Controller to the given target

Parameters

newController	The new managing Controller

11.10.2.12 setCtrl()

```
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 explaination as to why this structure is needed

Parameters

val	The value, from -1 to +1

11.10.2.13 setCurrentLimit()

```
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	*val*	The current limit in Amps

Returns

Error code

Reimplemented from YbCtrl::V4_OPA_OutputChannel.

11.10.2.14 setLimits()

```
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.
Similarily 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`	*minimum*	The new minimum
`in`	*maximum*	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

- enum FLASHRATE {
  ON = -1, OFF = 0, SLOW = 1, MEDIUM = 2,
  FAST = 4, MAX = 8 }

  Flash rate options.

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 FLASH_FREQ_HZ

```
#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 PRESCALER

```
#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 / PRESCALER / FLASH_FREQ_HZ >= 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 FLASHRATE

```
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"
*GIT(?)	Output git version string	0	e.g. "5a21cb0"
*RST	Reset the device (N.B. does not clear EEPROM)	0	-
*DFU	Puts the device into DFU mode (does 10x resets, expecting code upload)	0	-
STOR	Store a command in EEPR↩OM 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"
APPE	Append a command to E↩EPROM to be executed on startup	"command to be called on startup"	"Done"

Command	Description	Params	Output
		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] [threshold↩Low]	e.g. "#setThresholds 0.1 0.↩01"
	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"
CURR	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]	
CURR?	Get current hardware on output current	[output channel = 1, 2, 3, 4, BPA, BPB]	e.g. "1.000"

Command	Description	Params	Output
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 registerCommands()

```
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

h	Pointer to a CommandHandler object

Template Parameters

size	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 occured 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 "compileTimeCRC32.h"
#include "Microprocessor_Debugging\debugging_disable.h"
#include <EEPROM.h>
```

12.3.1 Detailed Description

Contains all the code for the Arduino CommandHandler library.

Space requirements: 6 bytes + 8 per command + buffer size

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 = false
- const bool OPA_2_IS_548 = false
- const uint8_t VPLUS_CHAN_1 = 1
- const uint8_t VPLUS_CHAN_2 = 0
- const uint8_t VLIM_CHAN_1 = 3
- const uint8_t VLIM_CHAN_2 = 2
- double MAX_VOLTAGE
- const double MAX_INITIAL_CURRENT = 2.0
- const double OPA_R1 = 40.2
- const double OPA_R2 = 200.2
- const double OPA_GAIN
- const uint8_t LED_1 = A5
- const uint8_t LED_2 = A4
- const uint8_t DISABLE_SERIAL_CTRL = 0xFF
- const uint8_t DIVIDED_SUPPLY_VOLTAGE = 0xFF
- const double FALLBACK_SUPPLY_VOLTAGE = 15
- const double DIVIDED_SUPPLY_FACTOR = 0

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 ADC_CS

```
const uint8_t ADC_CS = A3
```

Chip select pin of the ADC on the ATMega

12.5.2.2 ADC_DRDY

```
const uint8_t ADC_DRDY = A2
```

DRDY pin of the ADC on the ATMega

12.5.2.3 ADC_POW

```
const uint8_t ADC_POW = 12
```

ADC channel for chip power level measurement

12.5.2.4 ADC_REF

```
const uint8_t ADC_REF = 10
```

ADC channel for common thermistor reference input

12.5.2.5 ADC_START

```
const uint8_t ADC_START = 2
```

∼START pin of the ADC on the ATMega

12.5.2.6 ADC_THERM1

```
const uint8_t ADC_THERM1 = 9
```

ADC channel for thermistor 1 input

12.5.2.7 ADC_THERM1_HIGHRES

```
const bool ADC_THERM1_HIGHRES = false
```

Should thermistor 1 be a high-res measurement?

12.5.2.8 ADC_THERM2

```
const uint8_t ADC_THERM2 = 8
```

ADC channel for thermistor 2 input

12.5.2.9 ADC_THERM2_HIGHRES

```
const bool ADC_THERM2_HIGHRES = false
```

Should thermistor 2 be a high-res measurement?

12.5.2.10 ADC_TMP

```
const uint8_t ADC_TMP = 11
```

ADC channel for chip temperature measurement

12.5.2.11 ADC_VOLT1_HIGHRES

```
const bool ADC_VOLT1_HIGHRES = false
```

Should voltage 1 be a high-res measurement?

12.5.2.12 ADC_VOLT1_N

```
const uint8_t ADC_VOLT1_N = 7
```

ADC channel for arbitary voltage negative input 1

12.5.2.13 ADC_VOLT1_P

```
const uint8_t ADC_VOLT1_P = 6
```

ADC channel for arbitary voltage positive input 1

12.5.2.14 ADC_VOLT2_HIGHRES

```
const bool ADC_VOLT2_HIGHRES = false
```

Should voltage 2 be a high-res measurement?

12.5.2.15 ADC_VOLT2_N

```
const uint8_t ADC_VOLT2_N = 5
```

ADC channel for arbitary voltage negative input 2

12.5.2.16 ADC_VOLT2_P

```
const uint8_t ADC_VOLT2_P = 4
```

ADC channel for arbitary voltage positive input 2

12.5.2.17 DAC_CS

```
const uint8_t DAC_CS = 9
```

Chip select pin of the DAC on the ATMega

12.5.2.18 DISABLE_SERIAL_CTRL

```
const uint8_t DISABLE_SERIAL_CTRL = 0xFF
```

ATMega pin for disabling serial control: 0 or 5V (set to 0xFF if not present)

12.5.2.19 DIVIDED_SUPPLY_FACTOR

```
const double DIVIDED_SUPPLY_FACTOR = 0
```

Factor by which the supply voltage has been divided

12.5.2.20 DIVIDED_SUPPLY_VOLTAGE

```
const uint8_t DIVIDED_SUPPLY_VOLTAGE = 0xFF
```

ATMega pin for reading supply voltage: analog voltage (set to 0xFF if not present)

12.5.2.21 FALLBACK_SUPPLY_VOLTAGE

```
const double FALLBACK_SUPPLY_VOLTAGE = 15
```

Assumed supply voltage if DIVIDED_SUPPLY_VOLTAGE pin is not present

12.5.2.22 LED_1

```
const uint8_t LED_1 = A5
```

ATMega pin for LED 1, Arduino labelling

12.5.2.23 LED_2

```
const uint8_t LED_2 = A4
```

ATMega pin for LED 2, Arduino labelling

12.5.2.24 MAX_INITIAL_CURRENT

```
const double MAX_INITIAL_CURRENT = 2.0
```

Maximum current that the OPAs will be limited to on startup

12.5.2.25 MAX_VOLTAGE

```
double MAX_VOLTAGE
```

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

12.5.2.26 OPA_1_IS_548

```
const bool OPA_1_IS_548 = false
```

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

12.5.2.27 OPA_2_IS_548

```
const bool OPA_2_IS_548 = false
```

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

12.5.2.28 OPA_ES1

```
const uint8_t OPA_ES1 = A0
```

E/S pin for first OPA

12.5.2.29 OPA_ES2

```
const uint8_t OPA_ES2 = A1
```

E/S pin for second OPA

12.5.2.30 OPA_GAIN

```
const double OPA_GAIN
```

Initial value:

```
=
           (OPA_R1 + OPA_R2) / OPA_R1
```

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

12.5.2.31 OPA_R1

```
const double OPA_R1 = 40.2
```

OPA gain resistor 1

12.5.2.32 OPA_R2

```
const double OPA_R2 = 200.2
```

OPA gain resistor 2

12.5.2.33 VLIM_CHAN_1

```
const uint8_t VLIM_CHAN_1 = 3
```

DAC channel for first OPA's Vlim input

12.5.2.34 VLIM_CHAN_2

```
const uint8_t VLIM_CHAN_2 = 2
```

DAC channel for second OPA's Vlim input

12.5.2.35 VPLUS_CHAN_1

```
const uint8_t VPLUS_CHAN_1 = 1
```

DAC channel for first OPA's V+ input

12.5.2.36 VPLUS_CHAN_2

```
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 = A1
- const uint8_t OPA_ES2 = A0
- 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 = 8
- const uint8_t OPA_ES4 = 7
- 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
- const double FALLBACK_SUPPLY_VOLTAGE = 15

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 ADC_CS

```
const uint8_t ADC_CS = A3
```

Chip select pin of the ADC on the ATMega

12.6.2.2 ADC_DRDY

```
const uint8_t ADC_DRDY = A2
```

DRDY pin of the ADC on the ATMega

12.6.2.3 ADC_POW

```
const uint8_t ADC_POW = 12
```

ADC channel for chip power level measurement

12.6.2.4 ADC_REF

```
const uint8_t ADC_REF = 10
```

ADC channel for common thermistor reference input

12.6.2.5 ADC_START

```
const uint8_t ADC_START = 2
```

∼START pin of the ADC on the ATMega

12.6.2.6 ADC_THERM1

```
const uint8_t ADC_THERM1 = 9
```

ADC channel for thermistor 1 input

12.6.2.7 ADC_THERM1_HIGHRES

```
const bool ADC_THERM1_HIGHRES = false
```

Should thermistor 1 be a high-res measurement?

12.6.2.8 ADC_THERM2

```
const uint8_t ADC_THERM2 = 8
```

ADC channel for thermistor 2 input

12.6.2.9 ADC_THERM2_HIGHRES

```
const bool ADC_THERM2_HIGHRES = false
```

Should thermistor 2 be a high-res measurement?

12.6.2.10 ADC_TMP

```
const uint8_t ADC_TMP = 11
```

ADC channel for chip temperature measurement

12.6.2.11 DAC_CS

```
const uint8_t DAC_CS = 10
```

Chip select pin of the DAC on the ATMega

12.6.2.12 DAC_CS_Alt

```
const uint8_t DAC_CS_Alt = 9
```

Chip select pin of the DAC on the ATMega

12.6.2.13 DISABLE_SERIAL_CTRL

```
const uint8_t DISABLE_SERIAL_CTRL = A6
```

ATMega pin for disabling serial control: 0 or 5V

12.6.2.14 DIVIDED_SUPPLY_FACTOR

```
const double DIVIDED_SUPPLY_FACTOR = 11.0
```

Factor by which the supply voltage has been divided

12.6.2.15 DIVIDED_SUPPLY_VOLTAGE

```
const uint8_t DIVIDED_SUPPLY_VOLTAGE = A7
```

ATMega pin for reading supply voltage: analog voltage

12.6.2.16 FALLBACK_SUPPLY_VOLTAGE

```
const double FALLBACK_SUPPLY_VOLTAGE = 15
```

Assumed supply voltage if DIVIDED_SUPPLY_VOLTAGE pin is not present

12.6.2.17 LED_1

```
const uint8_t LED_1 = A5
```

ATMega pin for LED 1, Arduino labelling

12.6.2.18 LED_2

```
const uint8_t LED_2 = A4
```

ATMega pin for LED 2, Arduino labelling

12.6.2.19 MAX_INITIAL_CURRENT

```
const double MAX_INITIAL_CURRENT = 2.0
```

Maximum current that the OPAs will be limited to on startup

12.6.2.20 MAX_VOLTAGE

```
double MAX_VOLTAGE
```

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

12.6.2.21 OPA_1_IS_548

```
const bool OPA_1_IS_548 = true
```

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

12.6.2.22 OPA_2_IS_548

```
const bool OPA_2_IS_548 = true
```

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

12.6.2.23 OPA_3_IS_548

```
const bool OPA_3_IS_548 = true
```

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

12.6.2.24 OPA_4_IS_548

```
const bool OPA_4_IS_548 = true
```

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

12.6.2.25 OPA_ES1

```
const uint8_t OPA_ES1 = A1
```

E/S pin for first OPA

12.6.2.26 OPA_ES2

```
const uint8_t OPA_ES2 = A0
```

E/S pin for second OPA

12.6.2.27 OPA_ES3

```
const uint8_t OPA_ES3 = 8
```

E/S pin for first OPA

12.6.2.28 OPA_ES4

```
const uint8_t OPA_ES4 = 7
```

E/S pin for second OPA

12.6.2.29 OPA_GAIN

```
const double OPA_GAIN
```

Initial value:

```
=
           (OPA_R1 + OPA_R2) / OPA_R1
```

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

12.6.2.30 OPA_R1

```
const double OPA_R1 = 50
```

OPA gain resistor 1

12.6.2.31 OPA_R2

```
const double OPA_R2 = 200
```

OPA gain resistor 2

12.6.2.32 VLIM_CHAN_1

```
const uint8_t VLIM_CHAN_1 = 3
```

DAC channel for second OPA's Vlim input

12.6.2.33 VLIM_CHAN_2

```
const uint8_t VLIM_CHAN_2 = 2
```

DAC channel for first OPA's Vlim input

12.6.2.34 VLIM_CHAN_3

```
const uint8_t VLIM_CHAN_3 = 2
```

DAC channel for second OPA's Vlim input

12.6.2.35 VLIM_CHAN_4

```
const uint8_t VLIM_CHAN_4 = 3
```

DAC channel for first OPA's Vlim input

12.6.2.36 VPLUS_CHAN_1

```
const uint8_t VPLUS_CHAN_1 = 0
```

DAC channel for second OPA's V+ input

12.6.2.37 VPLUS_CHAN_2

```
const uint8_t VPLUS_CHAN_2 = 1
```

DAC channel for first OPA's V+ input

12.6.2.38 VPLUS_CHAN_3

```
const uint8_t VPLUS_CHAN_3 = 0
```

DAC channel for second OPA's V+ input

12.6.2.39 VPLUS_CHAN_4

```
const uint8_t VPLUS_CHAN_4 = 1
```

DAC channel for first OPA's V+ input

Index