labscript

Release 3.3.0rc2.dev33+g24f2d6a

labscript suite contributors

Mar 19, 2024

DOCUMENTATION

labscript, a component of the labscript suite, is an API used to define the experiment logic of a buffered experiment shot. This documentation will outline the general device hierarchy used when defining a connection table, and the **labscript** classes used to command input and output. For device specific documentation, and documentation for adding support for new devices, please refer to the labscript-devices documentation.

CHAPTER

ONE

INTRODUCTION

The labscript API is used to define the logic of an experiment that you wish to run. It is recommended that you read our paper before this documentation, so you are familiar with terms like pseudoclock. It would also be a good idea to familiarise yourself with the Python programming language and object oriented (OO) programming if you are not already.

To give you an idea of what a sample experiment looks like, the simplest experiment script (that does something) using the labscript API is below:

```
from labscript import *
from labscript_devices.PulseBlaster import PulseBlaster

# Connection Table
PulseBlaster(name='pulseblaster_0', board_number=0)
DigitalOut(name='my_digital_out', parent_device=pulseblaster_0.direct_outputs,␣
↪connection='flag 2')

#Experiment Logic
start()
my_digital_out.go_low(t=0)  # start low at the start
my_digital_out.go_high(t=1) # go high at 1s
stop(2)                     # stop at 2s
```

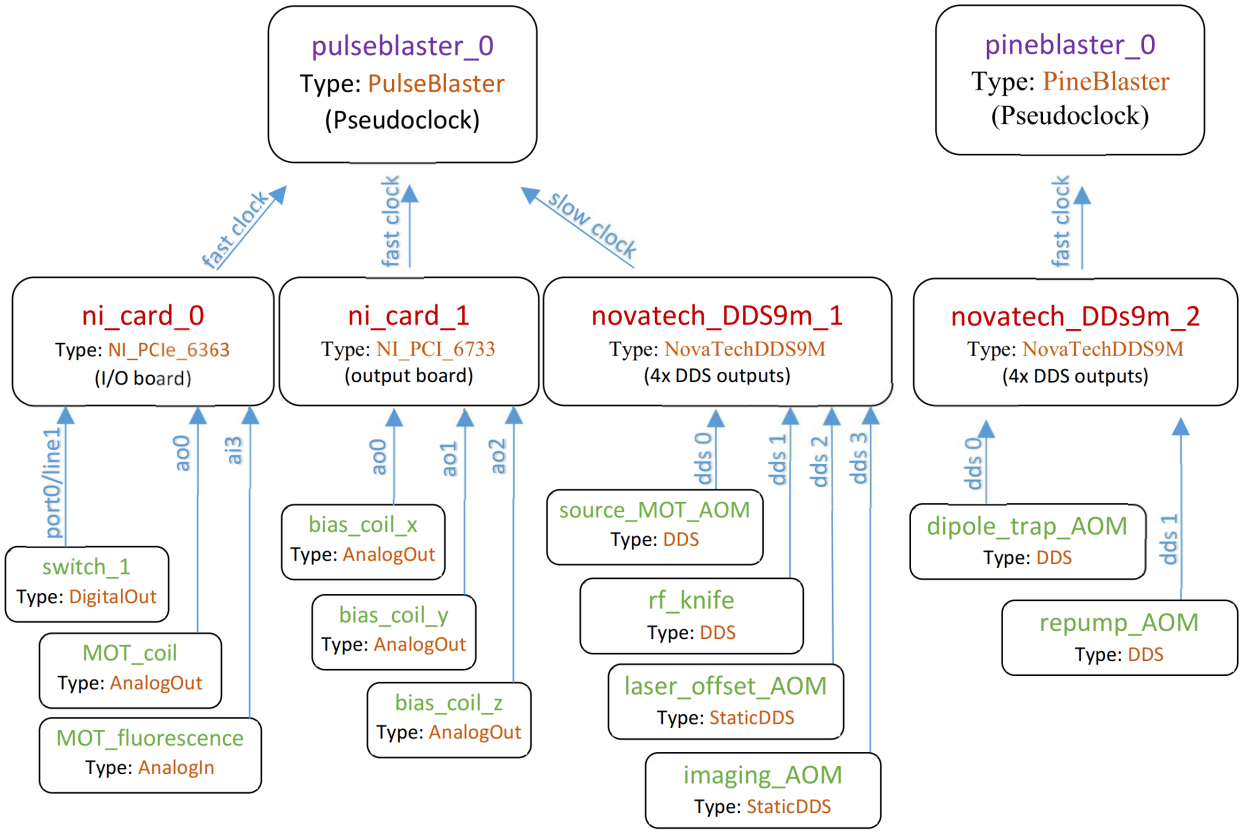
The script consists of two parts, the connection table and the experiment logic which will be discussed in the following sections.

CHAPTER

TWO

CONNECTION TABLE

The connection table maps out the way input/output devices are connected to each other in your lab, and the channels (individual inputs/outputs) they have. The devices in your lab should be connected in a similar way to that shown in the figure below.



Here we see two *PseudoclockDevice* instances in the top tier of the diagram. They do not have a parent device that tells them when to update their output (this is true for all *PseudoclockDevice* instances). However, all but one (the master pseudoclock device) must be triggered by an output clocked by the master pseudoclock device.

Each *PseudoclockDevice* instance should have one or more *Pseudoclock* children. Some *PseudoclockDevice* instances may automatically create these children for you (check the device specific documentation). In turn, each *Pseudoclock* will have one of more *ClockLine* instances connected to it. These *ClockLine* instances generally refer to physical outputs of a device which will be used to clock another device. However, in some cases, one or more *ClockLine* instances may be internally created for you (check the device specific documentation).

If a device is not a *PseudoclockDevice*, it must be connected to one via a clockline. such devices inherit

from *IntermediateDevice*. Inputs and outputs are then connected to these devices. For example, *DigitalOut*, *AnalogOut*, and *DDS*. See *labscript.outputs* and *labscript.inputs* for a complete list. Note that devices determine what types of inputs and outputs can be connected, see labscript-devices for device information.

If a *PseudoclockDevice* also has outputs that are not used for a *ClockLine*, then an *IntermediateDevice* is internally instantiated, and should be made available through the `PseudoclockDevice.direct_outputs` attribute (for example see the `PulseBlaster` implementation).

Note: Most user's will not need to use *PseudoclockDevice*, *Pseudoclock*, and *IntermediateDevice* directly. These are generic classes that are subclassed by device implementations in labscript-devices. It is these device implementations that you are most likely to use.

CHAPTER

THREE

API REFERENCE

labscript.core	Core classes containing common device functionality - these are used in labscript-devices when adding support for a hardware device.
labscript.outputs	Classes for devices channels that are outputs
labscript.inputs	Classes for device channels that are inputs
labscript.remote	Classes for configuring remote/secondary BLACS and/or device workers
labscript.constants	Common constant factors for time and frequency
labscript.labscript	Everything else including the `start()`, `stop()`, and `wait()` functions - all other classes are also imported here for backwards compatibility
labscript.functions	Contains the functional forms of analog output ramps - these are not used directly, instead see the interfaces in `AnalogQuantity`/`AnalogOut`.
labscript.base	The labscript base class for all I/O/Device classes
labscript.utils	Utility functions

3.1 labscript.core

Core classes containing common device functionality - these are used in labscript-devices when adding support for a hardware device.

Classes

ClockLine(name, pseudoclock, connection[, ...])	
IntermediateDevice(name, parent_device, **kwargs)	Base class for all devices that are to be clocked by a pseudoclock.
Pseudoclock(name, pseudoclock_device, ...)	Parent class of all pseudoclocks.
PseudoclockDevice(name[, trigger_device, ...])	Device that implements a pseudoclock.
TriggerableDevice(name, parent_device, ...)	A triggerable version of `Device`.

3.1.1 labscript.core.ClockLine

class ClockLine(*name*, *pseudoclock*, *connection*, *ramping_allowed=True*, ***kwargs*)

Bases: *Device*

__init__(*name*, *pseudoclock*, *connection*, *ramping_allowed=True*, ***kwargs*)

Creates a Device.

Parameters

- **name** (*str*) – python variable name to assign this device to.
- **parent_device** (Device) – Parent of this device.
- **connection** (*str*) – Connection on this device that links to parent.
- **call_parents_add_device** (*bool, optional*) – Flag to command device to call its parent device's add_device when adding a device.
- **added_properties** (*dict, optional*) –
- **gui** –
- **worker** –
- **start_order** (*int, optional*) – Priority when starting, sorted with all devices.
- **stop_order** (*int, optional*) – Priority when stopping, sorted with all devices.
- ****kwargs** – Other options to pass to parent.

Methods

__init__(name, pseudoclock, connection[, ...])	Creates a Device.
add_device(device)	Adds a child device to this device.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using Device.set_property().
init_device_group(hdf5_file)	Creates the device group in the shot file.
quantise_to_pseudoclock(times)	Quantises times to the resolution of the controlling pseudoclock.
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
clock_limit	Clock limit for this line, typically set by speed of child Intermediate Devices.
description	Brief description of the device.
minimum_clock_high_time	The minimum time a clock tick must be in the logical high state
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
t0	The earliest time output can be commanded from this device at the start of the experiment.

add_device(*device*)

Adds a child device to this device.

Parameters
device (Device) – Device to add.

Raises
LabscriptError – If device is not an allowed child of this device.

allowed_children = [<class 'labscript.core.IntermediateDevice'>]

Defines types of devices that are allowed to be children of this device.

Type
list

property clock_limit

Clock limit for this line, typically set by speed of child Intermediate Devices.

Type
float

description = 'Generic ClockLine'

Brief description of the device.

property minimum_clock_high_time

The minimum time a clock tick must be in the logical high state

Type
float

3.1.2 labscript.core.IntermediateDevice

class IntermediateDevice(*name*, *parent_device*, ***kwargs*)

Bases: *Device*

Base class for all devices that are to be clocked by a pseudoclock.

__init__(*name*, *parent_device*, ***kwargs*)

Provides some error checking to ensure parent_device is a *ClockLine*.

Calls Device.__init__().

Parameters

- **name** (*str*) – python variable name to assign to device
- **parent_device** (*ClockLine*) – Parent ClockLine device.

Methods

__init__(name, parent_device, **kwargs)	Provides some error checking to ensure parent_device is a *ClockLine*.
add_device(device)	Adds a child device to this device.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using `Device.set_property()`.
init_device_group(hdf5_file)	Creates the device group in the shot file.
quantise_to_pseudoclock(times)	Quantises `times` to the resolution of the controlling pseudoclock.
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
description	Brief description of the device.
minimum_clock_high_time	
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
t0	The earliest time output can be commanded from this device at the start of the experiment.

property minimum_clock_high_time

3.1.3 labscript.core.Pseudoclock

class Pseudoclock(*name*, *pseudoclock_device*, *connection*, ***kwargs*)

Bases: *Device*

Parent class of all pseudoclocks.

You won't usually interact with this class directly, unless you are implementing a new PsedoclockDevice in labscript-devices. It provides common functionality for generating pseudoclock instructions..

__init__(*name*, *pseudoclock_device*, *connection*, ***kwargs*)

Creates a Pseudoclock.

Parameters

- **name** (*str*) – python variable name to assign the device instance to.
- **pseudoclock_device** (*PseudoclockDevice*) – Parent pseudoclock device
- **connection** (*str*) – Connection on this device that links to parent
- ****kwargs** – Passed to `Device()`.

Methods

__init__(name, pseudoclock_device, ...)	Creates a Pseudoclock.
add_device(device)	Adds a child device to this device.
collect_change_times(all_outputs, ...)	Asks all connected outputs for a list of times that they change state.
expand_change_times(all_change_times, ...)	For each time interval delimited by change_times, constructs an array of times at which the clock for this device needs to tick.
generate_clock()	Generate the pseudoclock and configure outputs for each tick of the clock.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_outputs_by_clockline()	Obtain all outputs by clockline.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using `Device.set_property()`.
init_device_group(hdf5_file)	Creates the device group in the shot file.
quantise_to_pseudoclock(times)	Quantises `times` to the resolution of the controlling pseudoclock.
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
t0	The earliest time output can be commanded from this device at the start of the experiment.

add_device(*device*)

Adds a child device to this device.

Parameters
device (Device) – Device to add.

Raises
LabscriptError – If device is not an allowed child of this device.

allowed_children = [<class 'labscript.core.ClockLine'>]

Defines types of devices that are allowed to be children of this device.

Type
list

collect_change_times(*all_outputs*, *outputs_by_clockline*)

Asks all connected outputs for a list of times that they change state.

Takes the union of all of these times. Note that at this point, a change from holding-a-constant-value to ramping-through-values is considered a single state change. The clocking times will be filled in later in the expand_change_times function, and the ramp values filled in with expand_timeseries.

Parameters

- **all_outputs** (*list*) – List of all outputs connected to this pseudoclock.
- **outputs_by_clockline** (*dict*) – List of all outputs connected to this pseudoclock, organized by clockline.

Returns

Tuple containing:

- **all_change_times** (list): List of all change times.
- **change_times** (dict): Dictionary of all change times organised by which clock they are attached to.

Return type
tuple

description = 'Generic Pseudoclock'

Brief description of the device.

expand_change_times(*all_change_times*, *change_times*, *outputs_by_clockline*)

For each time interval delimited by change_times, constructs an array of times at which the clock for this device needs to tick. If the interval has all outputs having constant values, then only the start time is stored. If one or more outputs are ramping, then the clock ticks at the maximum clock rate requested by any of the outputs. Also produces a higher level description of the clocking; self.clock. This list contains the information that facilitates programming a pseudo clock using loops.

generate_clock()

Generate the pseudoclock and configure outputs for each tick of the clock.

generate_code(*hdf5_file*)

Generate hardware instructions for device and children, then save to h5 file.

Will recursively call generate_code for all children devices.

Parameters
hdf5_file (h5py.File) – Handle to shot file.

get_outputs_by_clockline()

Obtain all outputs by clockline.

Returns

Tuple containing:

- **all_outputs** (list): List of all outputs, obtained from `get_all_outputs()`.
- **outputs_by_clockline** (dict): Dictionary of outputs, organised by clockline.

Return type
tuple

3.1.4 labscript.core.PseudoclockDevice

class PseudoclockDevice(*name*, *trigger_device=None*, *trigger_connection=None*, ***kwargs*)

Bases: *TriggerableDevice*

Device that implements a pseudoclock.

__init__(*name*, *trigger_device=None*, *trigger_connection=None*, ***kwargs*)

Instantiates a pseudoclock device.

Parameters

- **name** (*str*) – python variable to assign to this device.
- **trigger_device** (DigitalOut) – Sets the parent triggering output. If None, this is considered the master pseudoclock.
- **trigger_connection** (*str, optional*) – Must be provided if trigger_device is provided. Specifies the channel of the parent device.
- ****kwargs** – Passed to *TriggerableDevice.__init__()*.

Methods

__init__(name[, trigger_device, ...])	Instantiates a pseudoclock device.
add_device(device)	Adds a child device to this device.
do_checks(outputs)	Basic error checking to ensure the user's instructions make sense.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using `Device.set_property()`.
init_device_group(hdf5_file)	Creates the device group in the shot file.
offset_instructions_from_trigger(outputs)	Offset instructions for child devices by the appropriate trigger times.
quantise_to_pseudoclock(times)	Quantises `times` to the resolution of the controlling pseudoclock.
set_initial_trigger_time(t)	Sets the initial trigger time of the pseudoclock.
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.
trigger(t, duration[, wait_delay])	Ask the trigger device to produce a digital pulse of a given duration to trigger this pseudoclock.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
description	Brief description of the device.
is_master_pseudoclock	Whether this device is the master pseudoclock.
minimum_recovery_time	Minimum time required before another trigger can occur.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
t0	The earliest time output can be commanded from this device at the start of the experiment.
trigger_delay	
trigger_edge_type	Type of trigger.
trigger_minimum_duration	
wait_delay	

allowed_children = [<class 'labscript.core.Pseudoclock'>]

Defines types of devices that are allowed to be children of this device.

Type
list

description = 'Generic Pseudoclock Device'

Brief description of the device.

do_checks(*outputs*)

Basic error checking to ensure the user's instructions make sense.

Parameters
outputs (*list*) – List of outputs to check.

generate_code(*hdf5_file*)

Generate hardware instructions for device and children, then save to h5 file.

Will recursively call `generate_code` for all children devices.

Parameters
hdf5_file (h5py.File) – Handle to shot file.

property is_master_pseudoclock

Whether this device is the master pseudoclock.

Type
bool

offset_instructions_from_trigger(*outputs*)

Offset instructions for child devices by the appropriate trigger times.

Parameters
outputs (*list*) – List of outputs to offset.

set_initial_trigger_time(*t*)

Sets the initial trigger time of the pseudoclock.

If this is the master pseudoclock, time must be 0.

Parameters
t (*float*) – Time, in seconds, to trigger this device.

trigger(*t*, *duration*, *wait_delay=0*)

Ask the trigger device to produce a digital pulse of a given duration to trigger this pseudoclock.

Parameters

- **t** (*float*) – Time, in seconds, to trigger this device.
- **duration** (*float*) – Duration, in seconds, of the trigger pulse.
- **wait_delay** (*float, optional*) – Time, in seconds, to delay the trigger.

trigger_delay = 0

trigger_edge_type = 'rising'

Type of trigger. Must be `'rising'` or `'falling'`.

Type
str

trigger_minimum_duration = 0

wait_delay = 0

3.1.5 labscript.core.TriggerableDevice

class TriggerableDevice(*name*, *parent_device*, *connection*, *parentless=False*, ***kwargs*)

Bases: *Device*

A triggerable version of `Device`.

This class is for devices that do not require a pseudoclock, but do require a trigger. This enables them to have a Trigger device as a parent.

__init__(*name*, *parent_device*, *connection*, *parentless=False*, ***kwargs*)

Instantiate a Triggerable Device.

Parameters

- **name** (*str*) –
- **()** (*parent_device*) –
- **connection** (*str*) –
- **parentless** (*bool, optional*) –
- ****kwargs** – Passed to `Device.__init__()`.

Raises

LabscriptError – If trigger type of this device does not match the trigger type of the parent Trigger.

Methods

__init__(name, parent_device, connection[, ...])	Instantiate a Triggerable Device.
add_device(device)	Adds a child device to this device.
do_checks()	Check that all devices sharing a trigger device have triggers when this device has a trigger.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using `Device.set_property()`.
init_device_group(hdf5_file)	Creates the device group in the shot file.
quantise_to_pseudoclock(times)	Quantises `times` to the resolution of the controlling pseudoclock.
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.
trigger(t, duration)	Request parent trigger device to produce a trigger.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
description	Brief description of the device.
minimum_recovery_time	Minimum time required before another trigger can occur.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
t0	The earliest time output can be commanded from this device at the start of the experiment.
trigger_edge_type	Type of trigger.

do_checks()

Check that all devices sharing a trigger device have triggers when this device has a trigger.

Raises
LabscriptError – If correct triggers do not exist for all devices.

generate_code(*hdf5_file*)

Generate hardware instructions for device and children, then save to h5 file.

Will recursively call generate_code for all children devices.

Parameters
hdf5_file (h5py.File) – Handle to shot file.

minimum_recovery_time = 0

Minimum time required before another trigger can occur.

Type
float

trigger(*t*, *duration*)

Request parent trigger device to produce a trigger.

Parameters

- **t** (*float*) – Time, in seconds, to produce a trigger.
- **duration** (*float*) – Duration, in seconds, of the trigger pulse.

trigger_edge_type = 'rising'

Type of trigger. Must be 'rising' or 'falling'.

Type
str

3.2 labscript.outputs

Classes for devices channels that are outputs

Classes

AnalogOut(name, parent_device, connection[, ...])	Analog Output class for use with all devices that support timed analog outputs.
AnalogQuantity(name, parent_device, connection)	Base class for *AnalogOut*.
DDS(name, parent_device, connection[, ...])	DDS class for use with all devices that have DDS-like outputs.
DDSQuantity(name, parent_device, connection)	Used to define a DDS output.
DigitalOut(name, parent_device, connection)	Digital output class for use with all devices.
DigitalQuantity(name, parent_device, connection)	Base class for *DigitalOut*.
Output(name, parent_device, connection[, ...])	Base class for all output classes.
Shutter(name, parent_device, connection[, ...])	Customized version of *DigitalOut* that accounts for the open/close delay of a shutter automatically.
StaticAnalogOut(*args, **kwargs)	Static Analog Output class for use with all devices that have constant outputs.
StaticAnalogQuantity(*args, **kwargs)	Base class for *StaticAnalogOut*.
StaticDDS(name, parent_device, connection[, ...])	Static DDS class for use with all devices that have static DDS-like outputs.
StaticDigitalOut(*args, **kwargs)	Static Digital Output class for use with all devices that have constant outputs.
StaticDigitalQuantity(*args, **kwargs)	Base class for *StaticDigitalOut*.
Trigger(name, parent_device, connection[, ...])	Customized version of *DigitalOut* that tracks edge type.

3.2.1 labscript.outputs.AnalogOut

class AnalogOut(*name*, *parent_device*, *connection*, *limits=None*, *unit_conversion_class=None*, *unit_conversion_parameters=None*, *default_value=None*, ***kwargs*)

Bases: *AnalogQuantity*

Analog Output class for use with all devices that support timed analog outputs.

__init__(*name*, *parent_device*, *connection*, *limits=None*, *unit_conversion_class=None*, *unit_conversion_parameters=None*, *default_value=None*, ***kwargs*)

Instantiate an Output.

Parameters

- **name** (*str*) – python variable name to assign the Output to.
- **parent_device** (IntermediateDevice) – Parent device the output is connected to.
- **connection** (*str*) – Channel of parent device output is connected to.
- **limits** (*tuple, optional*) – (min,max) allowed for the output.
- **unit_conversion_class** (labscript_utils:labscript_utils.unitconversions, optional) – Unit concersion class to use for the output.

- **unit_conversion_parameters** (*dict, optional*) – Dictonary or kwargs to pass to the unit conversion class.
- **default_value** (*float, optional*) – Default value of the output if no output is commanded.
- ****kwargs** – Passed to `Device.__init__()`.

Raises

LabscriptError – Limits tuple is invalid or unit conversion class units don't line up.

Methods

__init__(name, parent_device, connection[, ...])	Instantiate an Output.
add_device(device)	Adds a child device to this device.
add_instruction(time, instruction[, units])	Adds a hardware instruction to the device instruction list.
apply_calibration(value, units)	Apply the calibration defined by the unit conversion class, if present.
constant(t, value[, units])	Sets the output to a constant value at time `t`.
customramp(t, duration, function, *args, ...)	Define a custom function for the output.
do_checks(trigger_times)	Basic error checking to ensure the user's instructions make sense.
exp_ramp(t, duration, initial, final, samplerate)	Exponential ramp whose rate of change is set by an asymptotic value (zero argument).
exp_ramp_t(t, duration, initial, final, ...)	Exponential ramp whose rate of change is set by the time_constant.
expand_timeseries(all_times, flat_all_times_len)	This function evaluates the ramp functions in self.timeseries at the time points in all_times, and creates an array of output values at those times.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_change_times()	If this function is being called, it means that the parent Pseudoclock has requested a list of times that this output changes state.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using `Device.set_property()`.
get_ramp_times()	If this is being called, then it means the parent Pseudoclock has asked for a list of the output ramp start and stop times.
init_device_group(hdf5_file)	Creates the device group in the shot file.
instruction_to_string(instruction)	Gets a human readable description of an instruction.
make_timeseries(change_times)	If this is being called, then it means the parent Pseudoclock has asked for a list of this output's states at each time in change_times.
offset_instructions_from_trigger(trigger_tim	Subtracts self.trigger_delay from all instructions at or after each trigger_time.
piecewise_accel_ramp(t, duration, initial, ...)	Changes the output so that the second derivative follows one period of a triangle wave.

continues on next page

Table 1 – continued from previous page

quantise_to_pseudoclock(times)	Quantises times to the resolution of the controlling pseudoclock.
ramp(t, duration, initial, final, samplerate)	Command the output to perform a linear ramp.
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.
sine(t, duration, amplitude, angfreq, phase, ...)	Command the output to perform a sinusoidal modulation.
sine4_ramp(t, duration, initial, final, ...)	Command the output to perform an increasing ramp defined by one half period of a quartic sine wave.
sine4_reverse_ramp(t, duration, initial, ...)	Command the output to perform a decreasing ramp defined by one half period of a quartic sine wave.
sine_ramp(t, duration, initial, final, ...)	Command the output to perform a ramp defined by one half period of a squared sine wave.
square_wave(t, duration, amplitude, ...[, ...])	A standard square wave.
square_wave_levels(t, duration, level_0, ...)	A standard square wave.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
allowed_states	
clock_limit	Returns the parent clock line's clock limit.
default_value	
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
scale_factor	
t0	The earliest time output can be commanded from this device at the start of the experiment.
trigger_delay	The earliest time output can be commanded from this device after a trigger.
wait_delay	The earliest time output can be commanded from this device after a wait.

description = 'analog output'

Brief description of the device.

3.2.2 labscript.outputs.AnalogQuantity

class AnalogQuantity(*name*, *parent_device*, *connection*, *limits=None*, *unit_conversion_class=None*, *unit_conversion_parameters=None*, *default_value=None*, ***kwargs*)

Bases: *Output*

Base class for *AnalogOut*.

It is also used internally by *DDS*. You should never instantiate this class directly.

__init__(*name*, *parent_device*, *connection*, *limits=None*, *unit_conversion_class=None*, *unit_conversion_parameters=None*, *default_value=None*, ***kwargs*)

Instantiate an Output.

Parameters

- **name** (*str*) – python variable name to assign the Output to.
- **parent_device** (IntermediateDevice) – Parent device the output is connected to.
- **connection** (*str*) – Channel of parent device output is connected to.
- **limits** (*tuple, optional*) – (min,max) allowed for the output.
- **unit_conversion_class** (labscript_utils:labscript_utils.unitconversions, optional) – Unit concersion class to use for the output.
- **unit_conversion_parameters** (*dict, optional*) – Dictonary or kwargs to pass to the unit conversion class.
- **default_value** (*float, optional*) – Default value of the output if no output is commanded.
- ****kwargs** – Passed to Device.__init__().

Raises

LabscriptError – Limits tuple is invalid or unit conversion class units don't line up.

Methods

__init__(name, parent_device, connection[, ...])	Instantiate an Output.
add_device(device)	Adds a child device to this device.
add_instruction(time, instruction[, units])	Adds a hardware instruction to the device instruction list.
apply_calibration(value, units)	Apply the calibration defined by the unit conversion class, if present.
constant(t, value[, units])	Sets the output to a constant value at time t.
customramp(t, duration, function, *args, ...)	Define a custom function for the output.
do_checks(trigger_times)	Basic error checking to ensure the user's instructions make sense.
exp_ramp(t, duration, initial, final, samplerate)	Exponential ramp whose rate of change is set by an asymptotic value (zero argument).
exp_ramp_t(t, duration, initial, final, ...)	Exponential ramp whose rate of change is set by the time_constant.
expand_timeseries(all_times, flat_all_times_len)	This function evaluates the ramp functions in self.timeseries at the time points in all_times, and creates an array of output values at those times.

continues on next page

Table 2 – continued from previous page

generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_change_times()	If this function is being called, it means that the parent Pseudoclock has requested a list of times that this output changes state.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using `Device.set_property()`.
get_ramp_times()	If this is being called, then it means the parent Pseuedoclock has asked for a list of the output ramp start and stop times.
init_device_group(hdf5_file)	Creates the device group in the shot file.
instruction_to_string(instruction)	Gets a human readable description of an instruction.
make_timeseries(change_times)	If this is being called, then it means the parent Pseudoclock has asked for a list of this output's states at each time in change_times.
offset_instructions_from_trigger(trigger_tim	Subtracts self.trigger_delay from all instructions at or after each trigger_time.
piecewise_accel_ramp(t, duration, initial, ...)	Changes the output so that the second derivative follows one period of a triangle wave.
quantise_to_pseudoclock(times)	Quantises `times` to the resolution of the controlling pseudoclock.
ramp(t, duration, initial, final, samplerate)	Command the output to perform a linear ramp.
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.
sine(t, duration, amplitude, angfreq, phase, ...)	Command the output to perform a sinusoidal modulation.
sine4_ramp(t, duration, initial, final, ...)	Command the output to perform an increasing ramp defined by one half period of a quartic sine wave.
sine4_reverse_ramp(t, duration, initial, ...)	Command the output to perform a decreasing ramp defined by one half period of a quartic sine wave.
sine_ramp(t, duration, initial, final, ...)	Command the output to perform a ramp defined by one half period of a squared sine wave.
square_wave(t, duration, amplitude, ...[, ...])	A standard square wave.
square_wave_levels(t, duration, level_0, ...)	A standard square wave.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
allowed_states	
clock_limit	Returns the parent clock line's clock limit.
default_value	
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
scale_factor	
t0	The earliest time output can be commanded from this device at the start of the experiment.
trigger_delay	The earliest time output can be commanded from this device after a trigger.
wait_delay	The earliest time output can be commanded from this device after a wait.

constant(*t*, *value*, *units=None*)

Sets the output to a constant value at time `t`.

Parameters

- **t** (*float*) – Time, in seconds, to set the constant output.
- **value** (*float*) – Value to set.
- **units** – Units, defined by the unit conversion class, the value is in.

customramp(*t*, *duration*, *function*, **args*, ***kwargs*)

Define a custom function for the output.

Parameters

- **t** (*float*) – Time, in seconds, to start the function.
- **duration** (*float*) – Length in time, in seconds, to perform the function.
- **function** (*func*) – Function handle that defines the output waveform. First argument is the relative time from function start, in seconds.
- ***args** – Arguments passed to `function`.
- ****kwargs** – Keyword arguments pass to `function`. Standard kwargs common to other output functions are: `units`, `samplerate`, and `truncation`. These kwargs are optional, but will not be passed to `function` if present.

Returns

Duration the function is to be evaluate for. Equivalent to `truncation*duration`.

Return type

float

default_value = 0

description = 'analog quantity'

Brief description of the device.

exp_ramp(*t*, *duration*, *initial*, *final*, *samplerate*, *zero=0*, *units=None*, *truncation=None*, *truncation_type='linear'*, ***kwargs*)

Exponential ramp whose rate of change is set by an asymptotic value (zero argument).

Parameters

- **t** (*float*) – time to start the ramp
- **duration** (*float*) – duration of the ramp
- **initial** (*float*) – initial value of the ramp (sans truncation)
- **final** (*float*) – final value of the ramp (sans truncation)
- **zero** (*float*) – asymptotic value of the exponential decay/rise, i.e. limit as t –> inf
- **samplerate** (*float*) – rate to sample the function
- **units** – unit conversion to apply to specified values before generating raw output
- **truncation_type** (*str*) –
  - 'linear' truncation stops the ramp when it reaches the value given by the truncation parameter, which must be between initial and final
  - 'exponential' truncation stops the ramp after a period of truncation*duration In this instance, the truncation parameter should be between 0 (full truncation) and 1 (no truncation).

exp_ramp_t(*t*, *duration*, *initial*, *final*, *time_constant*, *samplerate*, *units=None*, *truncation=None*, *truncation_type='linear'*, ***kwargs*)

Exponential ramp whose rate of change is set by the time_constant.

Parameters

- **t** (*float*) – time to start the ramp
- **duration** (*float*) – duration of the ramp
- **initial** (*float*) – initial value of the ramp (sans truncation)
- **final** (*float*) – final value of the ramp (sans truncation)
- **time_constant** (*float*) – 1/e time of the exponential decay/rise
- **samplerate** (*float*) – rate to sample the function
- **units** – unit conversion to apply to specified values before generating raw output
- **truncation_type** (*str*) –
  - 'linear' truncation stops the ramp when it reaches the value given by the truncation parameter, which must be between initial and final
  - 'exponential' truncation stops the ramp after a period of truncation*duration In this instance, the truncation parameter should be between 0 (full truncation) and 1 (no truncation).

piecewise_accel_ramp(*t*, *duration*, *initial*, *final*, *samplerate*, *units=None*, *truncation=1.0*)

Changes the output so that the second derivative follows one period of a triangle wave.

Parameters

- **t** (*float*) – Time, in seconds, at which to begin the ramp.
- **duration** (*float*) – Duration of the ramp, in seconds.
- **initial** (*float*) – Initial output value at time `t`.
- **final** (*float*) – Final output value at time `t+duration`.
- **samplerate** (*float*) – Update rate of the output, in Hz.
- **units** – Units, defined by the unit conversion class, the value is in.
- **truncation** (*float, optional*) – Fraction of ramp to perform. Default 1.0.

Returns
Time the ramp will take to complete.

Return type
float

ramp(*t*, *duration*, *initial*, *final*, *samplerate*, *units=None*, *truncation=1.0*)

Command the output to perform a linear ramp.

Defined by `f(t) = ((final - initial)/duration)*t + initial`

Parameters

- **t** (*float*) – Time, in seconds, to begin the ramp.
- **duration** (*float*) – Length, in seconds, of the ramp.
- **initial** (*float*) – Initial output value, at time `t`.
- **final** (*float*) – Final output value, at time `t+duration`.
- **samplerate** (*float*) – Rate, in Hz, to update the output.
- **units** – Units the output values are given in, as specified by the unit conversion class.
- **truncation** (*float, optional*) – Fraction of ramp to perform. Must be between 0 and 1.

Returns
Length of time ramp will take to complete.

Return type
float

sine(*t*, *duration*, *amplitude*, *angfreq*, *phase*, *dc_offset*, *samplerate*, *units=None*, *truncation=1.0*)

Command the output to perform a sinusoidal modulation.

Defined by `f(t) = amplitude*sin(angfreq*t + phase) + dc_offset`

Parameters

- **t** (*float*) – Time, in seconds, to begin the ramp.
- **duration** (*float*) – Length, in seconds, of the ramp.
- **amplitude** (*float*) – Amplitude of the modulation.
- **angfreq** (*float*) – Angular frequency, in radians per second.
- **phase** (*float*) – Phase offset of the sine wave, in radians.
- **dc_offset** (*float*) – DC offset of output away from 0.
- **samplerate** (*float*) – Rate, in Hz, to update the output.

- **units** – Units the output values are given in, as specified by the unit conversion class.
- **truncation** (*float, optional*) – Fraction of duration to perform. Must be between 0 and 1.

Returns
Length of time modulation will take to complete. Equivalent to `truncation*duration`.

Return type
float

sine4_ramp(*t*, *duration*, *initial*, *final*, *samplerate*, *units=None*, *truncation=1.0*)

Command the output to perform an increasing ramp defined by one half period of a quartic sine wave.

Defined by `f(t) = (final-initial)*(sin(pi*t/(2*duration)))^4 + initial`

Parameters

- **t** (*float*) – Time, in seconds, to begin the ramp.
- **duration** (*float*) – Length, in seconds, of the ramp.
- **initial** (*float*) – Initial output value, at time `t`.
- **final** (*float*) – Final output value, at time `t+duration`.
- **samplerate** (*float*) – Rate, in Hz, to update the output.
- **units** – Units the output values are given in, as specified by the unit conversion class.
- **truncation** (*float, optional*) – Fraction of ramp to perform. Must be between 0 and 1.

Returns
Length of time ramp will take to complete.

Return type
float

sine4_reverse_ramp(*t*, *duration*, *initial*, *final*, *samplerate*, *units=None*, *truncation=1.0*)

Command the output to perform a decreasing ramp defined by one half period of a quartic sine wave.

Defined by `f(t) = (final-initial)*(sin(pi*t/(2*duration)))^4 + initial`

Parameters

- **t** (*float*) – Time, in seconds, to begin the ramp.
- **duration** (*float*) – Length, in seconds, of the ramp.
- **initial** (*float*) – Initial output value, at time `t`.
- **final** (*float*) – Final output value, at time `t+duration`.
- **samplerate** (*float*) – Rate, in Hz, to update the output.
- **units** – Units the output values are given in, as specified by the unit conversion class.
- **truncation** (*float, optional*) – Fraction of ramp to perform. Must be between 0 and 1.

Returns
Length of time ramp will take to complete.

Return type
float

sine_ramp(*t*, *duration*, *initial*, *final*, *samplerate*, *units=None*, *truncation=1.0*)

Command the output to perform a ramp defined by one half period of a squared sine wave.

Defined by `f(t) = (final-initial)*(sin(pi*t/(2*duration)))^2 + initial`

Parameters

- **t** (*float*) – Time, in seconds, to begin the ramp.
- **duration** (*float*) – Length, in seconds, of the ramp.
- **initial** (*float*) – Initial output value, at time `t`.
- **final** (*float*) – Final output value, at time `t+duration`.
- **samplerate** (*float*) – Rate, in Hz, to update the output.
- **units** – Units the output values are given in, as specified by the unit conversion class.
- **truncation** (*float, optional*) – Fraction of ramp to perform. Must be between 0 and 1.

Returns
Length of time ramp will take to complete.

Return type
float

square_wave(*t*, *duration*, *amplitude*, *frequency*, *phase*, *offset*, *duty_cycle*, *samplerate*, *units=None*, *truncation=1.0*)

A standard square wave.

This method generates a square wave which starts HIGH (when its phase is zero) then transitions to/from LOW at the specified `frequency` in Hz. The `amplitude` parameter specifies the peak-to-peak amplitude of the square wave which is centered around `offset`. For example, setting `amplitude=1` and `offset=0` would give a square wave which transitions between `0.5` and `-0.5`. Similarly, setting `amplitude=2` and `offset=3` would give a square wave which transitions between `4` and `2`. To instead specify the HIGH/LOW levels directly, use `square_wave_levels()`.

Note that because the transitions of a square wave are sudden and discontinuous, small changes in timings (e.g. due to numerical rounding errors) can affect the output value. This is particularly relevant at the end of the waveform, as the final output value may be different than expected if the end of the waveform is close to an edge of the square wave. Care is taken in the implementation of this method to avoid such effects, but it still may be desirable to call `constant()` after `square_wave()` to ensure a particular final value. The output value may also be different than expected at certain moments in the middle of the waveform due to the finite samplerate (which may be different than the requested `samplerate`), particularly if the actual samplerate is not a multiple of `frequency`.

Parameters

- **t** (*float*) – The time at which to start the square wave.
- **duration** (*float*) – The duration for which to output a square wave when `truncation` is set to 1. When `truncation` is set to a value less than 1, the actual duration will be shorter than `duration` by that factor.
- **amplitude** (*float*) – The peak-to-peak amplitude of the square wave. See above for an example of how to calculate the HIGH/LOW output values given the `amplitude` and `offset` values.
- **frequency** (*float*) – The frequency of the square wave, in Hz.

- **phase** (*float*) – The initial phase of the square wave. Note that the square wave is defined such that the phase goes from 0 to 1 (NOT 2 pi) over one cycle, so setting `phase=0.5` will start the square wave advanced by 1/2 of a cycle. Setting `phase` equal to `duty_cycle` will cause the waveform to start LOW rather than HIGH.
- **offset** (*float*) – The offset of the square wave, which is the value halfway between the LOW and HIGH output values. Note that this is NOT the LOW output value; setting `offset` to `0` will cause the HIGH/LOW values to be symmetrically split around `0`. See above for an example of how to calculate the HIGH/LOW output values given the `amplitude` and `offset` values.
- **duty_cycle** (*float*) – The fraction of the cycle for which the output should be HIGH. This should be a number between zero and one inclusively. For example, setting `duty_cycle=0.1` will create a square wave which outputs HIGH over 10% of the cycle and outputs LOW over 90% of the cycle.
- **samplerate** (*float*) – The requested rate at which to update the output value. Note that the actual samplerate used may be different if, for example, another output of the same device has a simultaneous ramp with a different requested `samplerate`, or if `1 / samplerate` isn't an integer multiple of the pseudoclock's timing resolution.
- **units** (*str, optional*) – The units of the output values. If set to `None` then the output's base units will be used. Defaults to `None`.
- **truncation** (*float, optional*) – The actual duration of the square wave will be `duration * truncation` and `truncation` must be set to a value in the range [0, 1] (inclusively). Set to `1` to output the full duration of the square wave. Setting it to `0` will skip the square wave entirely. Defaults to `1.`.

Returns

The actual duration of the square wave, accounting for `truncation`.

Return type
duration (float)

square_wave_levels(*t*, *duration*, *level_0*, *level_1*, *frequency*, *phase*, *duty_cycle*, *samplerate*, *units=None*, *truncation=1.0*)

A standard square wave.

This method generates a square wave which starts at `level_0` (when its phase is zero) then transitions to/from `level_1` at the specified `frequency`. This is the same waveform output by `square_wave()`, but parameterized differently. See that method's docstring for more information.

Parameters

- **t** (*float*) – The time at which to start the square wave.
- **duration** (*float*) – The duration for which to output a square wave when `truncation` is set to 1. When `truncation` is set to a value less than `1`, the actual duration will be shorter than `duration` by that factor.
- **level_0** (*float*) – The initial level of the square wave, when the phase is zero.
- **level_1** (*float*) – The other level of the square wave.
- **frequency** (*float*) – The frequency of the square wave, in Hz.
- **phase** (*float*) – The initial phase of the square wave. Note that the square wave is defined such that the phase goes from 0 to 1 (NOT 2 pi) over one cycle, so setting `phase=0.5` will

start the square wave advanced by 1/2 of a cycle. Setting `phase` equal to `duty_cycle` will cause the waveform to start at `level_1` rather than `level_0`.

- **duty_cycle** (*float*) – The fraction of the cycle for which the output should be set to `level_0`. This should be a number between zero and one inclusively. For example, setting `duty_cycle=0.1` will create a square wave which outputs `level_0` over 10% of the cycle and outputs `level_1` over 90% of the cycle.
- **samplerate** (*float*) – The requested rate at which to update the output value. Note that the actual samplerate used may be different if, for example, another output of the same device has a simultaneous ramp with a different requested `samplerate`, or if `1 / samplerate` isn't an integer multiple of the pseudoclock's timing resolution.
- **units** (*str, optional*) – The units of the output values. If set to `None` then the output's base units will be used. Defaults to `None`.
- **truncation** (*float, optional*) – The actual duration of the square wave will be `duration * truncation` and `truncation` must be set to a value in the range [0, 1] (inclusively). Set to `1` to output the full duration of the square wave. Setting it to `0` will skip the square wave entirely. Defaults to `1.`.

Returns

The actual duration of the square wave, accounting for `truncation`.

Return type
duration (float)

3.2.3 labscript.outputs.DDS

class DDS(*name*, *parent_device*, *connection*, *digital_gate=None*, *freq_limits=None*, *freq_conv_class=None*, *freq_conv_params=None*, *amp_limits=None*, *amp_conv_class=None*, *amp_conv_params=None*, *phase_limits=None*, *phase_conv_class=None*, *phase_conv_params=None*, *call_parents_add_device=True*, ***kwargs*)

Bases: *DDSQuantity*

DDS class for use with all devices that have DDS-like outputs.

__init__(*name*, *parent_device*, *connection*, *digital_gate=None*, *freq_limits=None*, *freq_conv_class=None*, *freq_conv_params=None*, *amp_limits=None*, *amp_conv_class=None*, *amp_conv_params=None*, *phase_limits=None*, *phase_conv_class=None*, *phase_conv_params=None*, *call_parents_add_device=True*, ***kwargs*)

Instantiates a DDS quantity.

Parameters

- **name** (*str*) – python variable for the object created.
- **parent_device** (`IntermediateDevice`) – Device this output is connected to.
- **connection** (*str*) – Output of parent device this DDS is connected to.
- **digital_gate** (*dict, optional*) – Configures a digital output to use as an enable/disable gate for the output. Should contain keys `'device'` and `'connection'` with arguments for the `parent_device` and `connection` for instantiating the *DigitalOut*. All other (optional) keys are passed as kwargs.
- **freq_limits** (*tuple, optional*) – `(lower, upper)` limits for the frequency of the output

- **freq_conv_class** (labscript_utils:labscript_utils.unitconversions, optional) – Unit conversion class for the frequency of the output.
- **freq_conv_params** (*dict, optional*) – Keyword arguments passed to the unit conversion class for the frequency of the output.
- **amp_limits** (*tuple, optional*) – (lower, upper) limits for the amplitude of the output
- **amp_conv_class** (labscript_utils:labscript_utils.unitconversions, optional) – Unit conversion class for the amplitude of the output.
- **amp_conv_params** (*dict, optional*) – Keyword arguments passed to the unit conversion class for the amplitude of the output.
- **phase_limits** (*tuple, optional*) – (lower, upper) limits for the phase of the output
- **phase_conv_class** (labscript_utils:labscript_utils.unitconversions, optional) – Unit conversion class for the phase of the output.
- **phase_conv_params** (*dict, optional*) – Keyword arguments passed to the unit conversion class for the phase of the output.
- **call_parents_add_device** (*bool, optional*) – Have the parent device run its add_device method.
- ****kwargs** – Keyword arguments passed to Device.__init__().

Methods

__init__(name, parent_device, connection[, ...])	Instantiates a DDS quantity.
add_device(device)	Adds a child device to this device.
disable(t)	Disable the Output.
enable(t)	Enable the Output.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using Device.set_property().
init_device_group(hdf5_file)	Creates the device group in the shot file.
pulse(t, duration, amplitude, frequency[, ...])	Pulse the output.
quantise_to_pseudoclock(times)	Quantises times to the resolution of the controlling pseudoclock.
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.
setamp(t, value[, units])	Set the amplitude of the output.
setfreq(t, value[, units])	Set the frequency of the output.
setphase(t, value[, units])	Set the phase of the output.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
t0	The earliest time output can be commanded from this device at the start of the experiment.

3.2.4 labscript.outputs.DDSQuantity

class DDSQuantity(*name*, *parent_device*, *connection*, *digital_gate=None*, *freq_limits=None*, *freq_conv_class=None*, *freq_conv_params=None*, *amp_limits=None*, *amp_conv_class=None*, *amp_conv_params=None*, *phase_limits=None*, *phase_conv_class=None*, *phase_conv_params=None*, *call_parents_add_device=True*, ***kwargs*)

Bases: *Device*

Used to define a DDS output.

It is a container class, with properties that allow access to a frequency, amplitude, and phase of the output as *AnalogQuantity*. It can also have a gate, which provides enable/disable control of the output as *DigitalOut*.

This class instantiates channels for frequency/amplitude/phase (and optionally the gate) itself.

__init__(*name*, *parent_device*, *connection*, *digital_gate=None*, *freq_limits=None*, *freq_conv_class=None*, *freq_conv_params=None*, *amp_limits=None*, *amp_conv_class=None*, *amp_conv_params=None*, *phase_limits=None*, *phase_conv_class=None*, *phase_conv_params=None*, *call_parents_add_device=True*, ***kwargs*)

Instantiates a DDS quantity.

Parameters

- **name** (*str*) – python variable for the object created.
- **parent_device** (IntermediateDevice) – Device this output is connected to.
- **connection** (*str*) – Output of parent device this DDS is connected to.
- **digital_gate** (*dict, optional*) – Configures a digital output to use as an enable/disable gate for the output. Should contain keys `'device'` and `'connection'` with arguments for the `parent_device` and `connection` for instantiating the *DigitalOut*. All other (optional) keys are passed as kwargs.
- **freq_limits** (*tuple, optional*) – `(lower, upper)` limits for the frequency of the output
- **freq_conv_class** (labscript_utils:labscript_utils.unitconversions, optional) – Unit conversion class for the frequency of the output.
- **freq_conv_params** (*dict, optional*) – Keyword arguments passed to the unit conversion class for the frequency of the output.
- **amp_limits** (*tuple, optional*) – `(lower, upper)` limits for the amplitude of the output

- **amp_conv_class** (labscript_utils:labscript_utils.unitconversions, optional) – Unit conversion class for the amplitude of the output.
- **amp_conv_params** (*dict, optional*) – Keyword arguments passed to the unit conversion class for the amplitude of the output.
- **phase_limits** (*tuple, optional*) – (lower, upper) limits for the phase of the output
- **phase_conv_class** (labscript_utils:labscript_utils.unitconversions, optional) – Unit conversion class for the phase of the output.
- **phase_conv_params** (*dict, optional*) – Keyword arguments passed to the unit conversion class for the phase of the output.
- **call_parents_add_device** (*bool, optional*) – Have the parent device run its add_device method.
- ****kwargs** – Keyword arguments passed to Device.__init__().

Methods

__init__(name, parent_device, connection[, ...])	Instantiates a DDS quantity.
add_device(device)	Adds a child device to this device.
disable(t)	Disable the Output.
enable(t)	Enable the Output.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using Device.set_property().
init_device_group(hdf5_file)	Creates the device group in the shot file.
pulse(t, duration, amplitude, frequency[, ...])	Pulse the output.
quantise_to_pseudoclock(times)	Quantises times to the resolution of the controlling pseudoclock.
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.
setamp(t, value[, units])	Set the amplitude of the output.
setfreq(t, value[, units])	Set the frequency of the output.
setphase(t, value[, units])	Set the phase of the output.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
t0	The earliest time output can be commanded from this device at the start of the experiment.

allowed_children = [<class 'labscript.outputs.AnalogQuantity'>, <class 'labscript.outputs.DigitalOut'>, <class 'labscript.outputs.DigitalQuantity'>]

Defines types of devices that are allowed to be children of this device.

Type
list

description = 'DDS'

Brief description of the device.

disable(*t*)

Disable the Output.

Parameters
t (*float*) – Time, in seconds, to disable the output at.

Raises
LabscriptError – If the DDS is not instantiated with a digital gate.

enable(*t*)

Enable the Output.

Parameters
t (*float*) – Time, in seconds, to enable the output at.

Raises
LabscriptError – If the DDS is not instantiated with a digital gate.

pulse(*t*, *duration*, *amplitude*, *frequency*, *phase=None*, *amplitude_units=None*, *frequency_units=None*, *phase_units=None*, *print_summary=False*)

Pulse the output.

Parameters

- **t** (*float*) – Time, in seconds, to start the pulse at.
- **duration** (*float*) – Length of the pulse, in seconds.
- **amplitude** (*float*) – Amplitude to set the output to during the pulse.
- **frequency** (*float*) – Frequency to set the output to during the pulse.
- **phase** (*float, optional*) – Phase to set the output to during the pulse.
- **amplitude_units** – Units of `amplitude`.
- **frequency_units** – Units of `frequency`.
- **phase_units** – Units of `phase`.
- **print_summary** (*bool, optional*) – Print a summary of the pulse during compilation time.

Returns
Duration of the pulse, in seconds.

Return type
float

setamp(*t*, *value*, *units=None*)

Set the amplitude of the output.

Parameters

- **t** (*float*) – Time, in seconds, when the amplitude is set.
- **value** (*float*) – Amplitude to set to.
- **units** – Units that the value is defined in.

setfreq(*t*, *value*, *units=None*)

Set the frequency of the output.

Parameters

- **t** (*float*) – Time, in seconds, when the frequency is set.
- **value** (*float*) – Frequency to set to.
- **units** – Units that the value is defined in.

setphase(*t*, *value*, *units=None*)

Set the phase of the output.

Parameters

- **t** (*float*) – Time, in seconds, when the phase is set.
- **value** (*float*) – Phase to set to.
- **units** – Units that the value is defined in.

3.2.5 labscript.outputs.DigitalOut

class DigitalOut(*name*, *parent_device*, *connection*, *inverted=False*, ***kwargs*)

Bases: *DigitalQuantity*

Digital output class for use with all devices.

__init__(*name*, *parent_device*, *connection*, *inverted=False*, ***kwargs*)

Instantiate a digital quantity.

Parameters

- **name** (*str*) – python variable name to assign the quantity to.
- **parent_device** (IntermediateDevice) – Device this quantity is attached to.
- **connection** (*str*) – Connection on parent device we are connected to.
- **inverted** (*bool, optional*) – If True, output is logic inverted.
- ****kwargs** – Passed to *Output.__init__()*.

Methods

`__init__`(name, parent_device, connection[, ...])	Instantiate a digital quantity.
`add_device`(device)	Adds a child device to this device.
`add_instruction`(time, instruction[, units])	Adds a hardware instruction to the device instruction list.
`apply_calibration`(value, units)	Apply the calibration defined by the unit conversion class, if present.
`disable`(t)	Commands the output to disable.
`do_checks`(trigger_times)	Basic error checking to ensure the user's instructions make sense.
`enable`(t)	Commands the output to enable.
`expand_timeseries`(all_times, flat_all_times_len)	This function evaluates the ramp functions in self.timeseries at the time points in all_times, and creates an array of output values at those times.
`generate_code`(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
`get_all_children`()	Get all children devices for this device.
`get_all_outputs`()	Get all children devices that are outputs.
`get_change_times`()	If this function is being called, it means that the parent Pseudoclock has requested a list of times that this output changes state.
`get_properties`([location])	Get all properties in location.
`get_property`(name[, location])	Method to get a property of this device already set using `Device.set_property()`.
`get_ramp_times`()	If this is being called, then it means the parent Pseuedoclock has asked for a list of the output ramp start and stop times.
`go_high`(t)	Commands the output to go high.
`go_low`(t)	Commands the output to go low.
`init_device_group`(hdf5_file)	Creates the device group in the shot file.
`instruction_to_string`(instruction)	Gets a human readable description of an instruction.
`make_timeseries`(change_times)	If this is being called, then it means the parent Pseudoclock has asked for a list of this output's states at each time in change_times.
`offset_instructions_from_trigger`(trigger_tim	Subtracts self.trigger_delay from all instructions at or after each trigger_time.
`quantise_to_pseudoclock`(times)	Quantises `times` to the resolution of the controlling pseudoclock.
`repeat_pulse_sequence`(t, duration, ...)	This function only works if the DigitalQuantity is on a fast clock
`set_properties`(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
`set_property`(name, value[, location, overwrite])	Method to set a property for this device.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
allowed_states	
clock_limit	Returns the parent clock line's clock limit.
default_value	
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
scale_factor	
t0	The earliest time output can be commanded from this device at the start of the experiment.
trigger_delay	The earliest time output can be commanded from this device after a trigger.
wait_delay	The earliest time output can be commanded from this device after a wait.

description = 'digital output'

Brief description of the device.

3.2.6 labscript.outputs.DigitalQuantity

class DigitalQuantity(*name*, *parent_device*, *connection*, *inverted=False*, ***kwargs*)

Bases: *Output*

Base class for *DigitalOut*.

It is also used internally by other, more complex, output types.

__init__(*name*, *parent_device*, *connection*, *inverted=False*, ***kwargs*)

Instantiate a digital quantity.

Parameters

- **name** (*str*) – python variable name to assign the quantity to.
- **parent_device** (IntermediateDevice) – Device this quantity is attached to.
- **connection** (*str*) – Connection on parent device we are connected to.
- **inverted** (*bool, optional*) – If True, output is logic inverted.
- ****kwargs** – Passed to *Output.__init__()*.

Methods

__init__(name, parent_device, connection[, ...])	Instantiate a digital quantity.
add_device(device)	Adds a child device to this device.
add_instruction(time, instruction[, units])	Adds a hardware instruction to the device instruction list.
apply_calibration(value, units)	Apply the calibration defined by the unit conversion class, if present.
disable(t)	Commands the output to disable.
do_checks(trigger_times)	Basic error checking to ensure the user's instructions make sense.
enable(t)	Commands the output to enable.
expand_timeseries(all_times, flat_all_times_len)	This function evaluates the ramp functions in self.timeseries at the time points in all_times, and creates an array of output values at those times.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_change_times()	If this function is being called, it means that the parent Pseudoclock has requested a list of times that this output changes state.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using `Device.set_property()`.
get_ramp_times()	If this is being called, then it means the parent Pseuedoclock has asked for a list of the output ramp start and stop times.
go_high(t)	Commands the output to go high.
go_low(t)	Commands the output to go low.
init_device_group(hdf5_file)	Creates the device group in the shot file.
instruction_to_string(instruction)	Gets a human readable description of an instruction.
make_timeseries(change_times)	If this is being called, then it means the parent Pseudoclock has asked for a list of this output's states at each time in change_times.
offset_instructions_from_trigger(trigger_tim	Subtracts self.trigger_delay from all instructions at or after each trigger_time.
quantise_to_pseudoclock(times)	Quantises `times` to the resolution of the controlling pseudoclock.
repeat_pulse_sequence(t, duration, ...)	This function only works if the DigitalQuantity is on a fast clock
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
allowed_states	
clock_limit	Returns the parent clock line's clock limit.
default_value	
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
scale_factor	
t0	The earliest time output can be commanded from this device at the start of the experiment.
trigger_delay	The earliest time output can be commanded from this device after a trigger.
wait_delay	The earliest time output can be commanded from this device after a wait.

allowed_states = {0: 'low', 1: 'high'}

default_value = 0

description = 'digital quantity'

Brief description of the device.

disable(*t*)

Commands the output to disable.

If `inverted=True`, this will set the output high.

Parameters
t (*float*) – Time, in seconds, when the output disables.

dtype

alias of `uint32`

enable(*t*)

Commands the output to enable.

If `inverted=True`, this will set the output low.

Parameters
t (*float*) – Time, in seconds, when the output enables.

go_high(*t*)

Commands the output to go high.

Parameters
t (*float*) – Time, in seconds, when the output goes high.

go_low(*t*)

Commands the output to go low.

Parameters
t (*float*) – Time, in seconds, when the output goes low.

repeat_pulse_sequence(*t*, *duration*, *pulse_sequence*, *period*, *samplerate*)

This function only works if the DigitalQuantity is on a fast clock

The pulse sequence specified will be repeated from time t until t+duration.

Note 1: The samplerate should be significantly faster than the smallest time difference between two states in the pulse sequence, or else points in your pulse sequence may never be evaluated.

Note 2: The time points your pulse sequence is evaluated at may be different than you expect, if another output changes state between t and t+duration. As such, you should set the samplerate high enough that even if this rounding of tie points occurs (to fit in the update required to change the other output) your pulse sequence will not be significantly altered)

Parameters

- **t** (*float*) – Time, in seconds, to start the pulse sequence.
- **duration** (*float*) – How long, in seconds, to repeat the sequence.
- **pulse_sequence** (*list*) – List of tuples, with each tuple of the form `(time, state)`.
- **period** (*float*) – Defines how long the final tuple will be held for before repeating the pulse sequence. In general, should be longer than the entire pulse sequence.
- **samplerate** (*float*) – How often to update the output, in Hz.

3.2.7 labscript.outputs.Output

class Output(*name*, *parent_device*, *connection*, *limits=None*, *unit_conversion_class=None*, *unit_conversion_parameters=None*, *default_value=None*, ***kwargs*)

Bases: *Device*

Base class for all output classes.

__init__(*name*, *parent_device*, *connection*, *limits=None*, *unit_conversion_class=None*, *unit_conversion_parameters=None*, *default_value=None*, ***kwargs*)

Instantiate an Output.

Parameters

- **name** (*str*) – python variable name to assign the Output to.
- **parent_device** (`IntermediateDevice`) – Parent device the output is connected to.
- **connection** (*str*) – Channel of parent device output is connected to.
- **limits** (*tuple, optional*) – `(min,max)` allowed for the output.
- **unit_conversion_class** (`labscript_utils:labscript_utils.unitconversions`, optional) – Unit concersion class to use for the output.
- **unit_conversion_parameters** (*dict, optional*) – Dictonary or kwargs to pass to the unit conversion class.
- **default_value** (*float, optional*) – Default value of the output if no output is commanded.
- ****kwargs** – Passed to `Device.__init__()`.

Raises

LabscriptError – Limits tuple is invalid or unit conversion class units don't line up.

Methods

__init__(name, parent_device, connection[, ...])	Instantiate an Output.
add_device(device)	Adds a child device to this device.
add_instruction(time, instruction[, units])	Adds a hardware instruction to the device instruction list.
apply_calibration(value, units)	Apply the calibration defined by the unit conversion class, if present.
do_checks(trigger_times)	Basic error checking to ensure the user's instructions make sense.
expand_timeseries(all_times, flat_all_times_len)	This function evaluates the ramp functions in self.timeseries at the time points in all_times, and creates an array of output values at those times.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_change_times()	If this function is being called, it means that the parent Pseudoclock has requested a list of times that this output changes state.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using `Device.set_property()`.
get_ramp_times()	If this is being called, then it means the parent Pseudoclock has asked for a list of the output ramp start and stop times.
init_device_group(hdf5_file)	Creates the device group in the shot file.
instruction_to_string(instruction)	Gets a human readable description of an instruction.
make_timeseries(change_times)	If this is being called, then it means the parent Pseudoclock has asked for a list of this output's states at each time in change_times.
offset_instructions_from_trigger(trigger_tim	Subtracts self.trigger_delay from all instructions at or after each trigger_time.
quantise_to_pseudoclock(times)	Quantises `times` to the resolution of the controlling pseudoclock.
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
allowed_states	
clock_limit	Returns the parent clock line's clock limit.
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
scale_factor	
t0	The earliest time output can be commanded from this device at the start of the experiment.
trigger_delay	The earliest time output can be commanded from this device after a trigger.
wait_delay	The earliest time output can be commanded from this device after a wait.

add_instruction(*time*, *instruction*, *units=None*)

Adds a hardware instruction to the device instruction list.

Parameters

- **time** (*float*) – Time, in seconds, that the instruction begins.
- **instruction** (*dict or float*) – Instruction to add.
- **units** (*str, optional*) – Units instruction is in, if `instruction` is a `float`.

Raises

LabscriptError – If time requested is not allowed or samplerate is too fast.

allowed_states = None

apply_calibration(*value*, *units*)

Apply the calibration defined by the unit conversion class, if present.

Parameters

- **value** (*float*) – Value to apply calibration to.
- **units** (*str*) – Units to convert to. Must be defined by the unit conversion class.

Returns

Converted value.

Return type

float

Raises

LabscriptError – If no unit conversion class is defined or `units` not in that class.

property clock_limit

Returns the parent clock line's clock limit.

Type

float

description = 'generic output'

Brief description of the device.

do_checks(*trigger_times*)

Basic error checking to ensure the user's instructions make sense.

Parameters
trigger_times (*iterable*) – Times to confirm don't conflict with instructions.

Raises
LabscriptError – If a trigger time conflicts with an instruction.

dtype

alias of `float64`

expand_timeseries(*all_times*, *flat_all_times_len*)

This function evaluates the ramp functions in self.timeseries at the time points in all_times, and creates an array of output values at those times. These are the values that this output should update to on each clock tick, and are the raw values that should be used to program the output device. They are stored in self.raw_output.

get_all_outputs()

Get all children devices that are outputs.

For `Output`, this is `self`.

Returns
List of children *Output*.

Return type
list

get_change_times()

If this function is being called, it means that the parent Pseudoclock has requested a list of times that this output changes state.

Returns
List of times output changes values.

Return type
list

get_ramp_times()

If this is being called, then it means the parent Pseuedoclock has asked for a list of the output ramp start and stop times.

Returns
List of (start, stop) times of ramps for this Output.

Return type
list

instruction_to_string(*instruction*)

Gets a human readable description of an instruction.

Parameters
instruction (*dict or str*) – Instruction to get description of, or a fixed instruction defined in *allowed_states*.

Returns
Instruction description.

Return type
str

`make_timeseries`(*change_times*)

If this is being called, then it means the parent Pseudoclock has asked for a list of this output's states at each time in change_times. (Which are the times that one or more connected outputs in the same pseudoclock change state). By state, I don't mean the value of the output at that moment, rather I mean what instruction it has. This might be a single value, or it might be a reference to a function for a ramp etc. This list of states is stored in self.timeseries rather than being returned.

`offset_instructions_from_trigger`(*trigger_times*)

Subtracts self.trigger_delay from all instructions at or after each trigger_time.

Parameters
trigger_times (*iterable*) – Times of all trigger events.

`scale_factor = 1`

`property trigger_delay`

The earliest time output can be commanded from this device after a trigger. This is nonzeo on secondary pseudoclocks due to triggering delays.

Type
float

`property wait_delay`

The earliest time output can be commanded from this device after a wait. This is nonzeo on secondary pseudoclocks due to triggering delays and the fact that the master clock doesn't provide a resume trigger to secondary clocks until a minimum time has elapsed: compiler.wait_delay. This is so that if a wait is extremely short, the child clock is actually ready for the trigger.

Type
float

3.2.8 labscript.outputs.Shutter

`class Shutter`(*name*, *parent_device*, *connection*, *delay=(0, 0)*, *open_state=1*, ***kwargs*)

Bases: *DigitalOut*

Customized version of *DigitalOut* that accounts for the open/close delay of a shutter automatically.

When using the methods *open()* and *close()*, the shutter open and close times are precise without haveing to track the delays. Note: delays can be set using runmanager globals and periodically updated via a calibration.

Warning: If the shutter is asked to do something at `t=0`, it cannot start moving earlier than that. This means the initial shutter states will have imprecise timing.

`__init__`(*name*, *parent_device*, *connection*, *delay=(0, 0)*, *open_state=1*, ***kwargs*)

Instantiates a Shutter.

Parameters

- **name** (*str*) – python variable to assign the object to.
- **parent_device** (`IntermediateDevice`) – Parent device the digital output is connected to.

- **connection** (*str*) – Physical output port of the device the digital output is connected to.
- **delay** (*tuple, optional*) – Tuple of the (open, close) delays, specified in seconds.
- **open_state** (*int, optional*) – Allowed values are `0` or `1`. Defines which state of the digital output opens the shutter.

Raises

LabscriptError – If the `open_state` is not `0` or `1`.

Methods

__init__(name, parent_device, connection[, ...])	Instantiates a Shutter.
add_device(device)	Adds a child device to this device.
add_instruction(time, instruction[, units])	Adds a hardware instruction to the device instruction list.
apply_calibration(value, units)	Apply the calibration defined by the unit conversion class, if present.
close(t)	Command the shutter to close at time t.
disable(t)	Commands the output to disable.
do_checks(trigger_times)	Basic error checking to ensure the user's instructions make sense.
enable(t)	Commands the output to enable.
expand_timeseries(all_times, flat_all_times_len)	This function evaluates the ramp functions in self.timeseries at the time points in all_times, and creates an array of output values at those times.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_change_times(*args, **kwargs)	If this function is being called, it means that the parent Pseudoclock has requested a list of times that this output changes state.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using Device.set_property().
get_ramp_times()	If this is being called, then it means the parent Pseuedoclock has asked for a list of the output ramp start and stop times.
go_high(t)	Commands the output to go high.
go_low(t)	Commands the output to go low.
init_device_group(hdf5_file)	Creates the device group in the shot file.
instruction_to_string(instruction)	Gets a human readable description of an instruction.
make_timeseries(change_times)	If this is being called, then it means the parent Pseudoclock has asked for a list of this output's states at each time in change_times.
offset_instructions_from_trigger(trigger_tim	Subtracts self.trigger_delay from all instructions at or after each trigger_time.
open(t)	Command the shutter to open at time t.
quantise_to_pseudoclock(times)	Quantises times to the resolution of the controlling pseudoclock.
repeat_pulse_sequence(t, duration, ...)	This function only works if the DigitalQuantity is on a fast clock
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
allowed_states	
clock_limit	Returns the parent clock line's clock limit.
default_value	
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
scale_factor	
t0	The earliest time output can be commanded from this device at the start of the experiment.
trigger_delay	The earliest time output can be commanded from this device after a trigger.
wait_delay	The earliest time output can be commanded from this device after a wait.

close(*t*)

Command the shutter to close at time `t`.

Takes the close delay time into account.

Note that the delay time will not be take into account the close delay if the command is made at t=0 (or other times less than the close delay). No warning will be issued for this loss of precision during compilation.

Parameters
t (*float*) – Time, in seconds, when shutter should be closed.

description = 'shutter'

Brief description of the device.

generate_code(*hdf5_file*)

Generate hardware instructions for device and children, then save to h5 file.

Will recursively call `generate_code` for all children devices.

Parameters
hdf5_file (h5py.File) – Handle to shot file.

get_change_times(**args*, ***kwargs*)

If this function is being called, it means that the parent Pseudoclock has requested a list of times that this output changes state.

Returns
List of times output changes values.

Return type
list

open(*t*)

Command the shutter to open at time `t`.

Takes the open delay time into account.

Note that the delay time will not be take into account the open delay if the command is made at t=0 (or other times less than the open delay). No warning will be issued for this loss of precision during compilation.

Parameters

t (*float*) – Time, in seconds, when shutter should be open.

3.2.9 labscript.outputs.StaticAnalogOut

class StaticAnalogOut(**args*, ***kwargs*)

Bases: *StaticAnalogQuantity*

Static Analog Output class for use with all devices that have constant outputs.

__init__(**args*, ***kwargs*)

Instatiantes the static analog quantity.

Defines an internal tracking variable of the static output value and calls *Output.__init__()*.

Parameters

- ***args** – Passed to *Output.__init__()*.
- ****kwargs** – Passed to *Output.__init__()*.

Methods

`__init__`(*args, **kwargs)	Instatiantes the static analog quantity.
`add_device`(device)	Adds a child device to this device.
`add_instruction`(time, instruction[, units])	Adds a hardware instruction to the device instruction list.
`apply_calibration`(value, units)	Apply the calibration defined by the unit conversion class, if present.
`constant`(value[, units])	Set the static output value of the output.
`do_checks`(trigger_times)	Basic error checking to ensure the user's instructions make sense.
`expand_timeseries`(*args, **kwargs)	Defines the `raw_output` attribute.
`generate_code`(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
`get_all_children`()	Get all children devices for this device.
`get_all_outputs`()	Get all children devices that are outputs.
`get_change_times`()	Enforces no change times.
`get_properties`([location])	Get all properties in location.
`get_property`(name[, location])	Method to get a property of this device already set using `Device.set_property()`.
`get_ramp_times`()	If this is being called, then it means the parent Pseuedoclock has asked for a list of the output ramp start and stop times.
`init_device_group`(hdf5_file)	Creates the device group in the shot file.
`instruction_to_string`(instruction)	Gets a human readable description of an instruction.
`make_timeseries`(change_times)	Since output is static, does nothing.
`offset_instructions_from_trigger`(trigger_tim	Subtracts self.trigger_delay from all instructions at or after each trigger_time.
`quantise_to_pseudoclock`(times)	Quantises `times` to the resolution of the controlling pseudoclock.
`set_properties`(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
`set_property`(name, value[, location, overwrite])	Method to set a property for this device.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
allowed_states	
clock_limit	Returns the parent clock line's clock limit.
default_value	Value of output if no constant value is commanded.
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
scale_factor	
static_value	The value of the static output.
t0	The earliest time output can be commanded from this device at the start of the experiment.
trigger_delay	The earliest time output can be commanded from this device after a trigger.
wait_delay	The earliest time output can be commanded from this device after a wait.

description = 'static analog output'

Brief description of the device.

3.2.10 labscript.outputs.StaticAnalogQuantity

class StaticAnalogQuantity(**args*, ***kwargs*)

Bases: *Output*

Base class for *StaticAnalogOut*.

It can also be used internally by other more complex output types.

__init__(**args*, ***kwargs*)

Instatiantes the static analog quantity.

Defines an internal tracking variable of the static output value and calls *Output.__init__()*.

Parameters

- ***args** – Passed to *Output.__init__()*.
- ****kwargs** – Passed to *Output.__init__()*.

Methods

__init__(*args, **kwargs)	Instatiantes the static analog quantity.
add_device(device)	Adds a child device to this device.
add_instruction(time, instruction[, units])	Adds a hardware instruction to the device instruction list.
apply_calibration(value, units)	Apply the calibration defined by the unit conversion class, if present.
constant(value[, units])	Set the static output value of the output.
do_checks(trigger_times)	Basic error checking to ensure the user's instructions make sense.
expand_timeseries(*args, **kwargs)	Defines the `raw_output` attribute.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_change_times()	Enforces no change times.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using `Device.set_property()`.
get_ramp_times()	If this is being called, then it means the parent Pseuedoclock has asked for a list of the output ramp start and stop times.
init_device_group(hdf5_file)	Creates the device group in the shot file.
instruction_to_string(instruction)	Gets a human readable description of an instruction.
make_timeseries(change_times)	Since output is static, does nothing.
offset_instructions_from_trigger(trigger_tim	Subtracts self.trigger_delay from all instructions at or after each trigger_time.
quantise_to_pseudoclock(times)	Quantises `times` to the resolution of the controlling pseudoclock.
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
allowed_states	
clock_limit	Returns the parent clock line's clock limit.
default_value	Value of output if no constant value is commanded.
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
scale_factor	
static_value	The value of the static output.
t0	The earliest time output can be commanded from this device at the start of the experiment.
trigger_delay	The earliest time output can be commanded from this device after a trigger.
wait_delay	The earliest time output can be commanded from this device after a wait.

constant(*value*, *units=None*)

Set the static output value of the output.

Parameters

- **value** (*float*) – Value to set the output to.
- **units** – Units, defined by the unit conversion class, the value is in.

Raises

LabscriptError – If static output has already been set to another value or the value lies outside the output limits.

default_value = 0.0

Value of output if no constant value is commanded.

Type

float

description = 'static analog quantity'

Brief description of the device.

expand_timeseries(**args*, ***kwargs*)

Defines the raw_output attribute.

get_change_times()

Enforces no change times.

Returns

An empty list, as expected by the parent pseudoclock.

Return type

list

make_timeseries(*change_times*)

Since output is static, does nothing.

property static_value

The value of the static output.

Type
float

3.2.11 labscript.outputs.StaticDDS

class StaticDDS(*name*, *parent_device*, *connection*, *digital_gate=None*, *freq_limits=None*, *freq_conv_class=None*, *freq_conv_params=None*, *amp_limits=None*, *amp_conv_class=None*, *amp_conv_params=None*, *phase_limits=None*, *phase_conv_class=None*, *phase_conv_params=None*, ***kwargs*)

Bases: *Device*

Static DDS class for use with all devices that have static DDS-like outputs.

__init__(*name*, *parent_device*, *connection*, *digital_gate=None*, *freq_limits=None*, *freq_conv_class=None*, *freq_conv_params=None*, *amp_limits=None*, *amp_conv_class=None*, *amp_conv_params=None*, *phase_limits=None*, *phase_conv_class=None*, *phase_conv_params=None*, ***kwargs*)

Instantiates a Static DDS quantity.

Parameters

- **name** (*str*) – python variable for the object created.
- **parent_device** (IntermediateDevice) – Device this output is connected to.
- **connection** (*str*) – Output of parent device this DDS is connected to.
- **digital_gate** (*dict, optional*) – Configures a digital output to use as an enable/disable gate for the output. Should contain keys 'device' and 'connection' with arguments for the parent_device and connection for instantiating the *DigitalOut*. All other (optional) keys are passed as kwargs.
- **freq_limits** (*tuple, optional*) – (lower, upper) limits for the frequency of the output
- **freq_conv_class** (labscript_utils:labscript_utils.unitconversions, optional) – Unit conversion class for the frequency of the output.
- **freq_conv_params** (*dict, optional*) – Keyword arguments passed to the unit conversion class for the frequency of the output.
- **amp_limits** (*tuple, optional*) – (lower, upper) limits for the amplitude of the output
- **amp_conv_class** (labscript_utils:labscript_utils.unitconversions, optional) – Unit conversion class for the amplitude of the output.
- **amp_conv_params** (*dict, optional*) – Keyword arguments passed to the unit conversion class for the amplitude of the output.
- **phase_limits** (*tuple, optional*) – (lower, upper) limits for the phase of the output
- **phase_conv_class** (labscript_utils:labscript_utils.unitconversions, optional) – Unit conversion class for the phase of the output.
- **phase_conv_params** (*dict, optional*) – Keyword arguments passed to the unit conversion class for the phase of the output.

- **call_parents_add_device** (*bool, optional*) – Have the parent device run its add_device method.
- ****kwargs** – Keyword arguments passed to `Device.__init__()`.

Methods

__init__(name, parent_device, connection[, ...])	Instantiates a Static DDS quantity.
add_device(device)	Adds a child device to this device.
disable([t])	Disable the Output.
enable([t])	Enable the Output.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using `Device.set_property()`.
init_device_group(hdf5_file)	Creates the device group in the shot file.
quantise_to_pseudoclock(times)	Quantises `times` to the resolution of the controlling pseudoclock.
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.
setamp(value[, units])	Set the static amplitude of the output.
setfreq(value[, units])	Set the static frequency of the output.
setphase(value[, units])	Set the static phase of the output.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
t0	The earliest time output can be commanded from this device at the start of the experiment.

allowed_children = [<class 'labscript.outputs.StaticAnalogQuantity'>, <class 'labscript.outputs.DigitalOut'>, <class 'labscript.outputs.StaticDigitalOut'>]

Defines types of devices that are allowed to be children of this device.

Type
list

description = 'Static RF'

Brief description of the device.

disable(*t=None*)

Disable the Output.

Parameters
t (*float, optional*) – Time, in seconds, to disable the output at.

Raises
LabscriptError – If the DDS is not instantiated with a digital gate.

enable(*t=None*)

Enable the Output.

Parameters
t (*float, optional*) – Time, in seconds, to enable the output at.

Raises
LabscriptError – If the DDS is not instantiated with a digital gate.

setamp(*value*, *units=None*)

Set the static amplitude of the output.

Parameters

- **value** (*float*) – Amplitude to set to.
- **units** – Units that the value is defined in.

setfreq(*value*, *units=None*)

Set the static frequency of the output.

Parameters

- **value** (*float*) – Frequency to set to.
- **units** – Units that the value is defined in.

setphase(*value*, *units=None*)

Set the static phase of the output.

Parameters

- **value** (*float*) – Phase to set to.
- **units** – Units that the value is defined in.

3.2.12 labscript.outputs.StaticDigitalOut

class StaticDigitalOut(**args*, ***kwargs*)

Bases: *StaticDigitalQuantity*

Static Digital Output class for use with all devices that have constant outputs.

__init__(**args*, ***kwargs*)

Instatiantes the static digital quantity.

Defines an internal tracking variable of the static output value and calls *Output.__init__()*.

Parameters

- ***args** – Passed to *Output.__init__()*.
- ****kwargs** – Passed to *Output.__init__()*.

Methods

`__init__`(*args, **kwargs)	Instatiantes the static digital quantity.
`add_device`(device)	Adds a child device to this device.
`add_instruction`(time, instruction[, units])	Adds a hardware instruction to the device instruction list.
`apply_calibration`(value, units)	Apply the calibration defined by the unit conversion class, if present.
`disable`(t)	Commands the output to disable.
`do_checks`(trigger_times)	Basic error checking to ensure the user's instructions make sense.
`enable`(t)	Commands the output to enable.
`expand_timeseries`(*args, **kwargs)	Defines the `raw_output` attribute.
`generate_code`(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
`get_all_children`()	Get all children devices for this device.
`get_all_outputs`()	Get all children devices that are outputs.
`get_change_times`()	Enforces no change times.
`get_properties`([location])	Get all properties in location.
`get_property`(name[, location])	Method to get a property of this device already set using `Device.set_property()`.
`get_ramp_times`()	If this is being called, then it means the parent Pseuedoclock has asked for a list of the output ramp start and stop times.
`go_high`()	Command a static high output.
`go_low`()	Command a static low output.
`init_device_group`(hdf5_file)	Creates the device group in the shot file.
`instruction_to_string`(instruction)	Gets a human readable description of an instruction.
`make_timeseries`(change_times)	Since output is static, does nothing.
`offset_instructions_from_trigger`(trigger_tim	Subtracts self.trigger_delay from all instructions at or after each trigger_time.
`quantise_to_pseudoclock`(times)	Quantises `times` to the resolution of the controlling pseudoclock.
`repeat_pulse_sequence`(t, duration, ...)	This function only works if the DigitalQuantity is on a fast clock
`set_properties`(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
`set_property`(name, value[, location, overwrite])	Method to set a property for this device.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
allowed_states	
clock_limit	Returns the parent clock line's clock limit.
default_value	Value of output if no constant value is commanded.
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
scale_factor	
static_value	The value of the static output.
t0	The earliest time output can be commanded from this device at the start of the experiment.
trigger_delay	The earliest time output can be commanded from this device after a trigger.
wait_delay	The earliest time output can be commanded from this device after a wait.

description = 'static digital output'

Brief description of the device.

3.2.13 labscript.outputs.StaticDigitalQuantity

class StaticDigitalQuantity(**args*, ***kwargs*)

Bases: *DigitalQuantity*

Base class for *StaticDigitalOut*.

It can also be used internally by other, more complex, output types.

__init__(**args*, ***kwargs*)

Instatiantes the static digital quantity.

Defines an internal tracking variable of the static output value and calls *Output.__init__()*.

Parameters

- ***args** – Passed to *Output.__init__()*.
- ****kwargs** – Passed to *Output.__init__()*.

Methods

__init__(*args, **kwargs)	Instatiantes the static digital quantity.
add_device(device)	Adds a child device to this device.
add_instruction(time, instruction[, units])	Adds a hardware instruction to the device instruction list.
apply_calibration(value, units)	Apply the calibration defined by the unit conversion class, if present.
disable(t)	Commands the output to disable.
do_checks(trigger_times)	Basic error checking to ensure the user's instructions make sense.
enable(t)	Commands the output to enable.
expand_timeseries(*args, **kwargs)	Defines the `raw_output` attribute.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_change_times()	Enforces no change times.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using `Device.set_property()`.
get_ramp_times()	If this is being called, then it means the parent Pseudoclock has asked for a list of the output ramp start and stop times.
go_high()	Command a static high output.
go_low()	Command a static low output.
init_device_group(hdf5_file)	Creates the device group in the shot file.
instruction_to_string(instruction)	Gets a human readable description of an instruction.
make_timeseries(change_times)	Since output is static, does nothing.
offset_instructions_from_trigger(trigger_tim	Subtracts self.trigger_delay from all instructions at or after each trigger_time.
quantise_to_pseudoclock(times)	Quantises `times` to the resolution of the controlling pseudoclock.
repeat_pulse_sequence(t, duration, ...)	This function only works if the DigitalQuantity is on a fast clock
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
allowed_states	
clock_limit	Returns the parent clock line's clock limit.
default_value	Value of output if no constant value is commanded.
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
scale_factor	
static_value	The value of the static output.
t0	The earliest time output can be commanded from this device at the start of the experiment.
trigger_delay	The earliest time output can be commanded from this device after a trigger.
wait_delay	The earliest time output can be commanded from this device after a wait.

default_value = 0

Value of output if no constant value is commanded.

Type
float

description = 'static digital quantity'

Brief description of the device.

expand_timeseries(**args*, ***kwargs*)

Defines the raw_output attribute.

get_change_times()

Enforces no change times.

Returns
An empty list, as expected by the parent pseudoclock.

Return type
list

go_high()

Command a static high output.

Raises
LabscriptError – If output has already been set low.

go_low()

Command a static low output.

Raises
LabscriptError – If output has already been set high.

make_timeseries(*change_times*)

Since output is static, does nothing.

property static_value

The value of the static output.

Type
float

3.2.14 labscript.outputs.Trigger

class Trigger(*name*, *parent_device*, *connection*, *trigger_edge_type='rising'*, ***kwargs*)

Bases: *DigitalOut*

Customized version of *DigitalOut* that tracks edge type.

__init__(*name*, *parent_device*, *connection*, *trigger_edge_type='rising'*, ***kwargs*)

Instantiates a DigitalOut object that tracks the trigger edge type.

Parameters

- **name** (*str*) – python variable name to assign the quantity to.
- **parent_device** (IntermediateDevice) – Device this quantity is attached to.
- **trigger_edge_type** (*str, optional*) – Allowed values are `'rising'` and `'falling'`.
- ****kwargs** – Passed to *Output.__init__()*.

Methods

__init__(name, parent_device, connection[, ...])	Instantiates a DigitalOut object that tracks the trigger edge type.
add_device(device)	Adds a child device to this device.
add_instruction(time, instruction[, units])	Adds a hardware instruction to the device instruction list.
apply_calibration(value, units)	Apply the calibration defined by the unit conversion class, if present.
disable(t)	Commands the output to disable.
do_checks(trigger_times)	Basic error checking to ensure the user's instructions make sense.
enable(t)	Commands the output to enable.
expand_timeseries(all_times, flat_all_times_len)	This function evaluates the ramp functions in self.timeseries at the time points in all_times, and creates an array of output values at those times.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_change_times()	If this function is being called, it means that the parent Pseudoclock has requested a list of times that this output changes state.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using `Device.set_property()`.
get_ramp_times()	If this is being called, then it means the parent Pseuedoclock has asked for a list of the output ramp start and stop times.
go_high(t)	Commands the output to go high.
go_low(t)	Commands the output to go low.
init_device_group(hdf5_file)	Creates the device group in the shot file.
instruction_to_string(instruction)	Gets a human readable description of an instruction.
make_timeseries(change_times)	If this is being called, then it means the parent Pseudoclock has asked for a list of this output's states at each time in change_times.
offset_instructions_from_trigger(trigger_tim	Subtracts self.trigger_delay from all instructions at or after each trigger_time.
quantise_to_pseudoclock(times)	Quantises `times` to the resolution of the controlling pseudoclock.
repeat_pulse_sequence(t, duration, ...)	This function only works if the DigitalQuantity is on a fast clock
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.
trigger(t, duration)	Command a trigger pulse.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
allowed_states	
clock_limit	Returns the parent clock line's clock limit.
default_value	
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
scale_factor	
t0	The earliest time output can be commanded from this device at the start of the experiment.
trigger_delay	The earliest time output can be commanded from this device after a trigger.
wait_delay	The earliest time output can be commanded from this device after a wait.

add_device(*device*)

Adds a child device to this device.

Parameters
device (Device) – Device to add.

Raises
LabscriptError – If device is not an allowed child of this device.

allowed_children = [<class 'labscript.core.TriggerableDevice'>]

Defines types of devices that are allowed to be children of this device.

Type
list

description = 'trigger device'

Brief description of the device.

trigger(*t*, *duration*)

Command a trigger pulse.

Parameters

- **t** (*float*) – Time, in seconds, for the trigger edge to occur.
- **duration** (*float*) – Duration of the trigger, in seconds.

3.3 labscript.inputs

Classes for device channels that are inputs

Classes

AnalogIn(name, parent_device, connection[, ...])	Analog Input for use with all devices that have an analog input.

3.3.1 labscript.inputs.AnalogIn

class AnalogIn(*name*, *parent_device*, *connection*, *scale_factor=1.0*, *units='Volts'*, ***kwargs*)

Bases: *Device*

Analog Input for use with all devices that have an analog input.

__init__(*name*, *parent_device*, *connection*, *scale_factor=1.0*, *units='Volts'*, ***kwargs*)

Instantiates an Analog Input.

Parameters

- **name** (*str*) – python variable to assign this input to.
- **parent_device** (IntermediateDevice) – Device input is connected to.
- **scale_factor** (*float, optional*) – Factor to scale the recorded values by.
- **units** (*str, optional*) – Units of the input.
- ****kwargs** – Keyword arguments passed to `Device.__init__()`.

Methods

__init__(name, parent_device, connection[, ...])	Instantiates an Analog Input.
acquire(label, start_time, end_time[, ...])	Command an acquisition for this input.
add_device(device)	Adds a child device to this device.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using `Device.set_property()`.
init_device_group(hdf5_file)	Creates the device group in the shot file.
quantise_to_pseudoclock(times)	Quantises `times` to the resolution of the controlling pseudoclock.
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
t0	The earliest time output can be commanded from this device at the start of the experiment.

acquire(*label*, *start_time*, *end_time*, *wait_label=''*, *scale_factor=None*, *units=None*)

Command an acquisition for this input.

Parameters

- **label** (*str*) – Unique label for the acquisition. Used to identify the saved trace.
- **start_time** (*float*) – Time, in seconds, when the acquisition should start.
- **end_time** (*float*) – Time, in seconds, when the acquisition should end.
- **wait_label** (*str, optional*) –
- **scale_factor** (*float*) – Factor to scale the saved values by.
- **units** – Units of the input, consistent with the unit conversion class.

Returns

Duration of the acquistion, equivalent to `end_time - start_time`.

Return type

float

description = 'Analog Input'

Brief description of the device.

3.4 labscript.remote

Classes for configuring remote/secondary BLACS and/or device workers

Classes

RemoteBLACS(name, host[, port, parent])	
SecondaryControlSystem(name, host, port[, ...])	

3.4.1 labscript.remote.RemoteBLACS

class RemoteBLACS(*name*, *host*, *port=7341*, *parent=None*)

Bases: _RemoteConnection

__init__(*name*, *host*, *port=7341*, *parent=None*)

Creates a Device.

Parameters

- **name** (*str*) – python variable name to assign this device to.
- **parent_device** (Device) – Parent of this device.
- **connection** (*str*) – Connection on this device that links to parent.
- **call_parents_add_device** (*bool, optional*) – Flag to command device to call its parent device's add_device when adding a device.
- **added_properties** (*dict, optional*) –
- **gui** –
- **worker** –
- **start_order** (*int, optional*) – Priority when starting, sorted with all devices.
- **stop_order** (*int, optional*) – Priority when stopping, sorted with all devices.
- ****kwargs** – Other options to pass to parent.

Methods

__init__(name, host[, port, parent])	Creates a Device.
add_device(device)	Adds a child device to this device.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using Device.set_property().
init_device_group(hdf5_file)	Creates the device group in the shot file.
quantise_to_pseudoclock(times)	Quantises times to the resolution of the controlling pseudoclock.
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
t0	The earliest time output can be commanded from this device at the start of the experiment.

3.4.2 labscript.remote.SecondaryControlSystem

class SecondaryControlSystem(*name*, *host*, *port*, *parent=None*)

Bases: _RemoteConnection

__init__(*name*, *host*, *port*, *parent=None*)

Creates a Device.

Parameters

- **name** (*str*) – python variable name to assign this device to.
- **parent_device** (Device) – Parent of this device.
- **connection** (*str*) – Connection on this device that links to parent.
- **call_parents_add_device** (*bool, optional*) – Flag to command device to call its parent device's add_device when adding a device.
- **added_properties** (*dict, optional*) –
- **gui** –
- **worker** –
- **start_order** (*int, optional*) – Priority when starting, sorted with all devices.
- **stop_order** (*int, optional*) – Priority when stopping, sorted with all devices.
- ****kwargs** – Other options to pass to parent.

Methods

__init__(name, host, port[, parent])	Creates a Device.
add_device(device)	Adds a child device to this device.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using `Device.set_property()`.
init_device_group(hdf5_file)	Creates the device group in the shot file.
quantise_to_pseudoclock(times)	Quantises `times` to the resolution of the controlling pseudoclock.
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
t0	The earliest time output can be commanded from this device at the start of the experiment.

3.5 labscript.constants

Common constant factors for time and frequency

Module Attributes

ns	Conversion factor between nanoseconds and seconds
us	Conversion factor between microseconds and seconds
ms	Conversion factor between milliseconds and seconds
s	Conversion factor between seconds and seconds
Hz	Conversion factor between hertz and hertz
kHz	Conversion factor between kilohertz and hertz
MHz	Conversion factor between megahertz and hertz
GHz	Conversion factor between gigahertz and hertz

3.5.1 labscript.constants.ns

ns = 1e-09

Conversion factor between nanoseconds and seconds

3.5.2 labscript.constants.us

us = 1e-06

Conversion factor between microseconds and seconds

3.5.3 labscript.constants.ms

ms = 0.001

Conversion factor between milliseconds and seconds

3.5.4 labscript.constants.s

s = 1

Conversion factor between seconds and seconds

3.5.5 labscript.constants.Hz

Hz = 1

Conversion factor between hertz and hertz

3.5.6 labscript.constants.kHz

kHz = 1000.0

Conversion factor between kilohertz and hertz

3.5.7 labscript.constants.MHz

MHz = 1000000.0

Conversion factor between megahertz and hertz

3.5.8 labscript.constants.GHz

GHz = 1000000000.0

Conversion factor between gigahertz and hertz

3.6 labscript.labscript

Everything else including the `start()`, `stop()`, and `wait()` functions - all other classes are also imported here for backwards compatibility

Functions

add_time_marker(t, label[, color, verbose])	Add a marker for the specified time.
generate_code()	Compiles a shot and saves it to the shot file.
generate_connection_table(hdf5_file)	Generates the connection table for the compiled shot.
generate_wait_table(hdf5_file)	Generates the wait table for the shot and saves it to the shot file.
labscript_cleanup()	restores builtins and the labscript module to its state before labscript_init() was called
labscript_init(hdf5_filename[, ...])	Initialises labscript and prepares for compilation.
load_globals(hdf5_filename)	
save_labscripts(hdf5_file)	Writes the script files for the compiled shot to the shot file.
save_time_markers(hdf5_file)	Save shot time markers to the shot file.
start()	Indicates the end of the connection table and the start of the experiment logic.
stop(t[, target_cycle_time, ...])	Indicate the end of an experiment at the given time, and initiate compilation of instructions, saving them to the HDF5 file.
trigger_all_pseudoclocks([t])	
wait(label, t[, timeout])	Commands pseudoclocks to pause until resumed by an external trigger, or a timeout is reached.
write_device_properties(hdf5_file)	Writes device_properties for each device in compiled shot to shto file.

3.6.1 labscript.labscript.add_time_marker

add_time_marker(*t*, *label*, *color=None*, *verbose=False*)

Add a marker for the specified time. These markers are saved in the HDF5 file. This allows one to label that time with a string label, and a color that applications may use to represent this part of the experiment. The color may be specified as an RGB tuple, or a string of the color name such as 'red', or a string of its hex value such as '#ff88g0'. If verbose=True, this funtion also prints the label and time, which can be useful to view during shot compilation.

Runviewer displays these markers and allows one to manipulate the time axis based on them, and BLACS' progress bar plugin displays the name and colour of the most recent marker as the shot is running

3.6.2 labscript.labscript.generate_code

generate_code()

Compiles a shot and saves it to the shot file.

3.6.3 labscript.labscript.generate_connection_table

generate_connection_table(*hdf5_file*)

Generates the connection table for the compiled shot.

Parameters

hdf5_file (h5py.File) – Handle to file to save to.

3.6.4 labscript.labscript.generate_wait_table

generate_wait_table(*hdf5_file*)

Generates the wait table for the shot and saves it to the shot file.

Parameters

hdf5_file (h5py.File) – Handle to file to save to.

3.6.5 labscript.labscript.labscript_cleanup

labscript_cleanup()

restores builtins and the labscript module to its state before labscript_init() was called

3.6.6 labscript.labscript.labscript_init

labscript_init(*hdf5_filename*, *labscript_file=None*, *new=False*, *overwrite=False*, *load_globals_values=True*)

Initialises labscript and prepares for compilation.

Parameters

- **hdf5_filename** (*str*) – Path to shot file to compile.
- **labscript_file** – Handle to the labscript file.
- **new** (*bool, optional*) – If True, ensure a new shot file is created.
- **overwrite** (*bool, optional*) – If True, overwrite existing shot file, if it exists.
- **load_globals_values** (*bool, optional*) – If True, load global values from the existing shot file.

3.6.7 labscript.labscript.load_globals

load_globals(*hdf5_filename*)

3.6.8 labscript.labscript.save_labscripts

save_labscripts(*hdf5_file*)

Writes the script files for the compiled shot to the shot file.

If `save_hg_info` labconfig parameter is `True`, will attempt to save hg version info as an attribute.

Parameters

hdf5_file (h5py.File) – Handle to file to save to.

3.6.9 labscript.labscript.save_time_markers

save_time_markers(*hdf5_file*)

Save shot time markers to the shot file.

Parameters

hdf5_file (h5py.File) – Handle to file to save to.

3.6.10 labscript.labscript.start

start()

Indicates the end of the connection table and the start of the experiment logic.

Returns

Time required for all pseudoclocks to start execution.

Return type

float

3.6.11 labscript.labscript.stop

stop(*t*, *target_cycle_time=None*, *cycle_time_delay_after_programming=False*)

Indicate the end of an experiment at the given time, and initiate compilation of instructions, saving them to the HDF5 file. Configures some shot options.

Parameters

- **t** (*float*) – The end time of the experiment.
- **target_cycle_time** (*float, optional*) – default: `None` How long, in seconds, after the previous shot was started, should this shot be started by BLACS. This allows one to run shots at a constant rate even if they are of different durations. If `None`, BLACS will run the next shot immediately after the previous shot completes. Otherwise, BLACS will delay starting this shot until the cycle time has elapsed. This is a request only, and may not be met if running/programming/saving data from a shot takes long enough that it cannot be met. This functionality requires the BLACS `cycle_time` plugin to be enabled in labconfig. Its accuracy is also limited by software timing, requirements of exact cycle times beyond software timing should be instead done using hardware triggers to Pseudoclocks.

- **cycle_time_delay_after_programming** (*bool, optional*) – default: False Whether the BLACS cycle_time plugin should insert the required delay for the target cycle time *after* programming devices, as opposed to before programming them. This is more precise, but may cause some devices to output their first instruction for longer than desired, since some devices begin outputting their first instruction as soon as they are programmed rather than when they receive their first clock tick. If not set, the *average* cycle time will still be just as close to as requested (so long as there is adequate time available), however the time interval between the same part of the experiment from one shot to the next will not be as precise due to variations in programming time.

3.6.12 labscript.labscript.trigger_all_pseudoclocks

trigger_all_pseudoclocks(*t='initial'*)

3.6.13 labscript.labscript.wait

wait(*label*, *t*, *timeout=5*)

Commands pseudoclocks to pause until resumed by an external trigger, or a timeout is reached.

Parameters

- **label** (*str*) – Unique name for wait.
- **t** (*float*) – Time, in seconds, at which experiment should begin the wait.
- **timeout** (*float, optional*) – Maximum length of the wait, in seconds. After this time, the pseudoclocks are automatically re-triggered.

Returns

Time required for all pseudoclocks to resume execution once wait has completed.

Return type

float

3.6.14 labscript.labscript.write_device_properties

write_device_properties(*hdf5_file*)

Writes device_properties for each device in compiled shot to shto file.

Parameters

hdf5_file (h5py.File) – Handle to file to save to.

Classes

WaitMonitor(name, parent_device, connection, ...)

3.6.15 labscript.labscript.WaitMonitor

class WaitMonitor(*name*, *parent_device*, *connection*, *acquisition_device*, *acquisition_connection*, *timeout_device=None*, *timeout_connection=None*, *timeout_trigger_type='rising'*, ***kwargs*)

Bases: *Trigger*

__init__(*name*, *parent_device*, *connection*, *acquisition_device*, *acquisition_connection*, *timeout_device=None*, *timeout_connection=None*, *timeout_trigger_type='rising'*, ***kwargs*)

Create a wait monitor.

This is a device or devices, one of which:

a) outputs pulses every time the master pseudoclock begins running (either at the start of the shot or after a wait

b) measures the time in between those pulses in order to determine how long the experiment was paused for during waits

c) optionally, produces pulses in software time that can be used to retrigger the master pseudoclock if a wait lasts longer than its specified timeout

Parameters

- **parent_device** (Device) – The device with buffered digital outputs that should be used to produce the wait monitor pulses. This device must be one which is clocked by the master pseudoclock.
- **connection** (*str*) – The name of the output connection of parent_device that should be used to produce the pulses.
- **acquisition_device** (Device) – The device which is to receive those pulses as input, and that will measure how long between them. This does not need to be the same device as the wait monitor output device (corresponding to `parent_device` and `connection`). At time of writing, the only devices in labscript that can be a wait monitor acquisition device are NI DAQmx devices that have counter inputs.
- **acquisition_connection** (*str*) – The name of the input connection on `acquisition_device` that is to read the wait monitor pulses. The user must manually connect the output device (`parent_device/connection`) to this input with a cable, in order that the pulses can be read by the device. On NI DAQmx devices, the acquisition_connection should be the name of the counter to be used such as 'Ctr0'. The physical connection should be made to the input terminal corresponding to the gate of that counter.
- **timeout_device** (Device, *optional*) – The device that should be used to produce pulses in software time if a wait lasts longer than its prescribed timeout. These pulses can connected to the trigger input of the master pseudoclock, via a digital logic to 'AND' it with other trigger sources, in order to resume the master pseudoclock upon a wait timing out. To produce these pulses during a shot requires cooperation between the acquisition device and the timeout device code, and at present this means only NI DAQmx devices can be used as the timeout device (though it need not be the same device as the acquisition device). If not set, timeout pulses will not be produced and the user must manually resume the master pseudoclock via other means, or abort a shot if the indended resumption mechanism fails.
- **timeout_connection** (*str*, *optional*) – Which connection on the timeout device should be used to produce timeout pulses. Since only NI DAQmx devices are supported at the moment, this must be a digital output on a port on the NI DAQmx device that is not

being used. Most NI DAQmx devices have both buffered and unbuffered ports, so typically one would use one line of one of the unbuffered ports for the timeout output.

- **timeout_trigger_type** (*str*) – The edge type to be used for the timeout signal, either `'rising'` or `'falling'`

Methods

__init__(name, parent_device, connection, ...)	Create a wait monitor.
add_device(device)	Adds a child device to this device.
add_instruction(time, instruction[, units])	Adds a hardware instruction to the device instruction list.
apply_calibration(value, units)	Apply the calibration defined by the unit conversion class, if present.
disable(t)	Commands the output to disable.
do_checks(trigger_times)	Basic error checking to ensure the user's instructions make sense.
enable(t)	Commands the output to enable.
expand_timeseries(all_times, flat_all_times_len)	This function evaluates the ramp functions in self.timeseries at the time points in all_times, and creates an array of output values at those times.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_change_times()	If this function is being called, it means that the parent Pseudoclock has requested a list of times that this output changes state.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using `Device.set_property()`.
get_ramp_times()	If this is being called, then it means the parent Pseudoclock has asked for a list of the output ramp start and stop times.
go_high(t)	Commands the output to go high.
go_low(t)	Commands the output to go low.
init_device_group(hdf5_file)	Creates the device group in the shot file.
instruction_to_string(instruction)	Gets a human readable description of an instruction.
make_timeseries(change_times)	If this is being called, then it means the parent Pseudoclock has asked for a list of this output's states at each time in change_times.
offset_instructions_from_trigger(trigger_tim	Subtracts self.trigger_delay from all instructions at or after each trigger_time.
quantise_to_pseudoclock(times)	Quantises `times` to the resolution of the controlling pseudoclock.
repeat_pulse_sequence(t, duration, ...)	This function only works if the DigitalQuantity is on a fast clock
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.
trigger(t, duration)	Command a trigger pulse.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
allowed_states	
clock_limit	Returns the parent clock line's clock limit.
default_value	
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
scale_factor	
t0	The earliest time output can be commanded from this device at the start of the experiment.
trigger_delay	The earliest time output can be commanded from this device after a trigger.
wait_delay	The earliest time output can be commanded from this device after a wait.

3.7 labscript.functions

Contains the functional forms of analog output ramps - these are not used directly, instead see the interfaces in AnalogQuantity/AnalogOut.

Functions

exp_ramp(duration, initial, final, zero)	Defines an exponential ramp via offset value.
exp_ramp_t(duration, initial, final, ...)	Defines an exponential ramp via time constant.
piecewise_accel(duration, initial, final)	Defines a piecewise acceleration.
pulse_sequence(pulse_sequence, period)	Returns a function that interpolates a pulse sequence.
ramp(duration, initial, final)	Defines a linear ramp.
sine(duration, amplitude, angfreq, phase, ...)	Defines a sine wave.
sine4_ramp(duration, initial, final)	Defines a quartic sinusoidally increasing ramp.
sine4_reverse_ramp(duration, initial, final)	Defines a quartic sinusoidally decreasing ramp.
sine_ramp(duration, initial, final)	Defines a square sinusoidally increasing ramp.
square_wave(duration, level_0, level_1, ...)	

3.7.1 labscript.functions.exp_ramp

exp_ramp(*duration*, *initial*, *final*, *zero*)

Defines an exponential ramp via offset value.

f(t) = (initial-zero)*e^(-rate*t) + zero rate = log((initial-zero)/(final-zero))/duration

Parameters

- **duration** (*float*) – Length of time for the ramp to complete
- **initial** (*float*) – Initial value of ramp.
- **final** (*float*) – Final value of ramp.
- **zero** (*float*) – Zero offset of ramp.

Returns
Function that takes a single parameter `t`.

Return type
func

3.7.2 labscript.functions.exp_ramp_t

exp_ramp_t(*duration*, *initial*, *final*, *time_constant*)

Defines an exponential ramp via time constant.

f(t) = (initial-zero)*e^(-t/time_constant) + zero zero = (final-initial*e^(-duration/time_constant))/(1-e^(-duration/time_constant))

Parameters

- **duration** (*float*) – Length of time for the ramp to complete
- **initial** (*float*) – Initial value of ramp.
- **final** (*float*) – Final value of ramp.
- **zero** (*float*) – Zero offset of ramp.

Returns
Function that takes a single parameter `t`.

Return type
func

3.7.3 labscript.functions.piecewise_accel

piecewise_accel(*duration*, *initial*, *final*)

Defines a piecewise acceleration.

Parameters

- **duration** (*float*) – Length of time for the acceleration to complete.
- **initial** (*float*) – Initial value.
- **final** (*float*) – Final value.

3.7.4 labscript.functions.pulse_sequence

pulse_sequence(*pulse_sequence*, *period*)

Returns a function that interpolates a pulse sequence.

Relies on `numpy.digitize` to perform the interpolation.

Parameters

- **pulse_sequence** (`numpy.ndarray`) – 2-D timeseries of change times and associated states.
- **period** (*float*) – How long, in seconds, to hold the final state before repeating the sequence.

Returns

Interpolating function that takes a single parameter `t`. Only well defined if `t` falls within the `pulse_sequence` change times.

Return type

func

3.7.5 labscript.functions.ramp

ramp(*duration*, *initial*, *final*)

Defines a linear ramp.

f(t) = (final - initial)*t/duration + initial

Parameters

- **duration** (*float*) – Duration of ramp
- **initial** (*float*) – Starting value of ramp
- **final** (*float*) – Ending value of ramp

Returns

Function that takes a single parameter `t`.

Return type

func

3.7.6 labscript.functions.sine

sine(*duration*, *amplitude*, *angfreq*, *phase*, *dc_offset*)

Defines a sine wave.

f(t) = amplitude*sin(angfreq*t + phase) + dc_offset

Parameters

- **duration** (*float*) – Not used.
- **amplitude** (*float*) – Amplitude of sine wave.
- **angfreq** (*float*) – Angular frequency of sine wave.
- **phase** (*float*) – Phase of sine wave.
- **dc_offset** (*float*) – Verticle offset of sine wave.

Returns

Function that takes a single parameter `t`.

Return type
func

3.7.7 labscript.functions.sine4_ramp

sine4_ramp(*duration*, *initial*, *final*)

Defines a quartic sinusoidally increasing ramp.

f(t) = (final-initial)*(sin(pi*t/(2*duration)))^4 + initial

Parameters

- **duration** (*float*) – Length of time for the ramp to complete.
- **initial** (*float*) – Initial value of ramp.
- **final** (*float*) – Final value of ramp.

Returns
Function that takes a single parameter `t`.

Return type
func

3.7.8 labscript.functions.sine4_reverse_ramp

sine4_reverse_ramp(*duration*, *initial*, *final*)

Defines a quartic sinusoidally decreasing ramp.

f(t) = (final-initial)*(sin(pi/2+pi*t/(2*duration)))^4 + initial

Parameters

- **duration** (*float*) – Length of time for the ramp to complete.
- **initial** (*float*) – Initial value of ramp.
- **final** (*float*) – Final value of ramp.

Returns
Function that takes a single parameter `t`.

Return type
func

3.7.9 labscript.functions.sine_ramp

sine_ramp(*duration*, *initial*, *final*)

Defines a square sinusoidally increasing ramp.

f(t) = (final-initial)*(sin(pi*t/(2*duration)))^2 + initial

Parameters

- **duration** (*float*) – Length of time for the ramp to complete.
- **initial** (*float*) – Initial value of ramp.
- **final** (*float*) – Final value of ramp.

Returns
Function that takes a single parameter `t`.

Return type
func

3.7.10 labscript.functions.square_wave

square_wave(*duration*, *level_0*, *level_1*, *frequency*, *phase*, *duty_cycle*)

3.8 labscript.base

The labscript base class for all I/O/Device classes

Classes

Device(name, parent_device, connection[, ...])	Parent class of all device and input/output channels.

3.8.1 labscript.base.Device

class Device(*name*, *parent_device*, *connection*, *call_parents_add_device=True*, *added_properties={}*, *gui=None*, *worker=None*, *start_order=None*, *stop_order=None*, ***kwargs*)

Bases: object

Parent class of all device and input/output channels.

You usually won't interact directly with this class directly (i.e. you never instantiate this class directly) but it provides some useful functionality that is then available to all subclasses.

__init__(*name*, *parent_device*, *connection*, *call_parents_add_device=True*, *added_properties={}*, *gui=None*, *worker=None*, *start_order=None*, *stop_order=None*, ***kwargs*)

Creates a Device.

Parameters

- **name** (*str*) – python variable name to assign this device to.
- **parent_device** (*Device*) – Parent of this device.
- **connection** (*str*) – Connection on this device that links to parent.
- **call_parents_add_device** (*bool, optional*) – Flag to command device to call its parent device's add_device when adding a device.
- **added_properties** (*dict, optional*) –
- **gui** –
- **worker** –
- **start_order** (*int, optional*) – Priority when starting, sorted with all devices.
- **stop_order** (*int, optional*) – Priority when stopping, sorted with all devices.
- ****kwargs** – Other options to pass to parent.

Methods

__init__(name, parent_device, connection[, ...])	Creates a Device.
add_device(device)	Adds a child device to this device.
generate_code(hdf5_file)	Generate hardware instructions for device and children, then save to h5 file.
get_all_children()	Get all children devices for this device.
get_all_outputs()	Get all children devices that are outputs.
get_properties([location])	Get all properties in location.
get_property(name[, location])	Method to get a property of this device already set using *Device.set_property()*.
init_device_group(hdf5_file)	Creates the device group in the shot file.
quantise_to_pseudoclock(times)	Quantises `times` to the resolution of the controlling pseudoclock.
set_properties(properties_dict, property_names)	Add one or a bunch of properties packed into properties_dict
set_property(name, value[, location, overwrite])	Method to set a property for this device.

Attributes

allowed_children	Defines types of devices that are allowed to be children of this device.
description	Brief description of the device.
parent_clock_line	Stores the clocking clockline, which may be itself.
pseudoclock_device	Stores the clocking pseudoclock, which may be itself.
t0	The earliest time output can be commanded from this device at the start of the experiment.

add_device(*device*)

Adds a child device to this device.

Parameters
device (*Device*) – Device to add.

Raises
LabscriptError – If `device` is not an allowed child of this device.

allowed_children = None

Defines types of devices that are allowed to be children of this device.

Type
list

description = 'Generic Device'

Brief description of the device.

generate_code(*hdf5_file*)

Generate hardware instructions for device and children, then save to h5 file.

Will recursively call `generate_code` for all children devices.

Parameters
hdf5_file (`h5py.File`) – Handle to shot file.

`get_all_children`()

Get all children devices for this device.

Returns
List of children *Device*.

Return type
list

`get_all_outputs`()

Get all children devices that are outputs.

Recursively calls `get_all_outputs()` on each child device. `Output`'s will return a list containing just themselves.

Returns
List of children `Output`.

Return type
list

`get_properties`(*location=None*)

Get all properties in location.

Parameters
location (*str, optional*) – Location to get properties from. If `None`, return all properties.

Returns
Dictionary of properties.

Return type
dict

`get_property`(*name*, *location=None*, **args*, ***kwargs*)

Method to get a property of this device already set using *Device.set_property()*.

If the property is not already set, a default value will be returned if specified as the argument after `'name'`, if there is only one argument after `'name'` and the argument is either not a keyword argurment or is a keyword argument with the name `'default'`.

Parameters

- **name** (*str*) – Name of property to get.
- **location** (*str, optional*) – If not `None`, only search for `name` in `location`.
- **default** – The default value. If not provided, an exception is raised if the value is not set.

Returns
Property value.

Raises
LabscriptError – If property not set and default not provided, or default conventions not followed.

Examples

Examples of acceptable signatures:

```
>>> get_property('example')             # 'example' will be returned if set, or␣
↪an exception raised
>>> get_property('example', 7)          # 7 returned if 'example' is not set
>>> get_property('example', default=7)  # 7 returnd if 'example' is not set
```

Example signatures that WILL ALWAYS RAISE AN EXCEPTION:

```
>>> get_property('example', 7, 8)
>>> get_property('example', 7, default=9)
>>> get_property('example', default=7, x=9)
```

init_device_group(*hdf5_file*)

Creates the device group in the shot file.

Parameters
hdf5_file (h5py.File) – File handle to create the group in.

Returns
Created group handle.

Return type
h5py.Group

property parent_clock_line

Stores the clocking clockline, which may be itself.

Type
ClockLine

property pseudoclock_device

Stores the clocking pseudoclock, which may be itself.

Type
PseudoclockDevice

quantise_to_pseudoclock(*times*)

Quantises `times` to the resolution of the controlling pseudoclock.

Parameters
times (numpy.ndarray or list or set or float) – Time, in seconds, to quantise.

Returns
Quantised times.

Return type
same type as `times`

set_properties(*properties_dict*, *property_names*, *overwrite=False*)

Add one or a bunch of properties packed into properties_dict

Parameters

- **properties_dict** (*dict*) – Dictionary of properties and their values.
- **property_names** (*dict*) – Is a dictionary {key:val, ...} where each val is a list [var1, var2, ...] of variables to be pulled from properties_dict and added to the property localtion with name `key`

- **overwrite** (*bool, optional*) – Toggles overwriting of existing properties.

set_property(*name*, *value*, *location=None*, *overwrite=False*)

Method to set a property for this device.

Property will be stored in the connection table and used during connection table comparisons.

Value must satisfy `eval(repr(value)) == value`.

Parameters

- **name** (*str*) – Name to save property value to.
- **value** – Value to set property to.
- **location** (*str, optional*) – Specify a location to save property to, such as `'device_properties'` or `'connection_table_properties'`.
- **overwrite** (*bool, optional*) – If `True`, allow overwriting a property already set.

Raises

LabscriptError – If `'location'` is not valid or trying to overwrite an existing property with `'overwrite'=False`.

property t0

The earliest time output can be commanded from this device at the start of the experiment. This is nonzero on secondary pseudoclock devices due to triggering delays.

Type

float

3.9 labscript.utils

Utility functions

Functions

bitfield(arrays, dtype)	Converts a list of arrays of ones and zeros into a single array of unsigned ints of the given datatype.
fastflatten(inarray, dtype)	A faster way of flattening our arrays than pylab.flatten.
is_clock_line(device)	Returns whether the connection is an instance of `ClockLine`
is_pseudoclock_device(device)	Returns whether the connection is an instance of `PseudoclockDevice`
is_remote_connection(connection)	Returns whether the connection is an instance of `_RemoteConnection`
max_or_zero(*args, **kwargs)	Returns max of the arguments or zero if sequence is empty.
print_time(t, description)	Print time with a descriptive string.
set_passed_properties([property_names])	Decorator for device __init__ methods that saves the listed arguments/keyword arguments as properties.
suppress_all_warnings([state])	A context manager which modifies compiler.suppress_all_warnings
suppress_mild_warnings([state])	A context manager which modifies compiler.suppress_mild_warnings

3.9.1 labscript.utils.bitfield

bitfield(*arrays*, *dtype*)

Converts a list of arrays of ones and zeros into a single array of unsigned ints of the given datatype.

Parameters

- **arrays** (*list*) – List of numpy arrays consisting of ones and zeros.
- **dtype** (*data-type*) – Type to convert to.

Returns
Numpy array with data type `dtype`.

Return type
numpy.ndarray

3.9.2 labscript.utils.fastflatten

fastflatten(*inarray*, *dtype*)

A faster way of flattening our arrays than pylab.flatten.

pylab.flatten returns a generator which takes a lot of time and memory to convert into a numpy array via array(list(generator)). The problem is that generators don't know how many values they'll return until they're done. This algorithm produces a numpy array directly by first calculating what the length will be. It is several orders of magnitude faster. Note that we can't use numpy.ndarray.flatten here since our inarray is really a list of 1D arrays of varying length and/or single values, not a N-dimenional block of homogeneous data like a numpy array.

Parameters

- **inarray** (*list*) – List of 1-D arrays to flatten.
- **dtype** (*data-type*) – Type of the data in the arrays.

Returns
Flattened array.

Return type
numpy.ndarray

3.9.3 labscript.utils.is_clock_line

is_clock_line(*device*)

Returns whether the connection is an instance of `ClockLine`

This function defers and caches the import of `ClockLine`. This both breaks the circular import between `Device` and `ClockLine`, while maintaining reasonable performance (this performs better than importing each time as the lookup in the modules hash table is slower).

3.9.4 labscript.utils.is_pseudoclock_device

is_pseudoclock_device(*device*)

Returns whether the connection is an instance of `PseudoclockDevice`

This function defers and caches the import of `PseudoclockDevice`. This both breaks the circular import between `Device` and `PseudoclockDevice`, while maintaining reasonable performance (this performs better than importing each time as the lookup in the modules hash table is slower).

3.9.5 labscript.utils.is_remote_connection

is_remote_connection(*connection*)

Returns whether the connection is an instance of `_RemoteConnection`

This function defers and caches the import of `_RemoteConnection`. This both breaks the circular import between `Device` and `_RemoteConnection`, while maintaining reasonable performance (this performs better than importing each time as the lookup in the modules hash table is slower).

3.9.6 labscript.utils.max_or_zero

max_or_zero(**args*, ***kwargs*)

Returns max of the arguments or zero if sequence is empty.

This protects the call to `max()` which would normally throw an error on an empty sequence.

Parameters

- ***args** – Items to compare.
- ****kwargs** – Passed to `max()`.

Returns

Max of *args.

3.9.7 labscript.utils.print_time

print_time(*t*, *description*)

Print time with a descriptive string.

Useful debug tool to print time at a specific point in the shot, during shot compilation. Helpful when the time is calculated.

Parameters

- **t** (*float*) – Time to print
- **description** (*str*) – Descriptive label to print with it

3.9.8 labscript.utils.set_passed_properties

set_passed_properties(*property_names=None*)

Decorator for device __init__ methods that saves the listed arguments/keyword arguments as properties.

Argument values as passed to __init__ will be saved, with the exception that if an instance attribute exists after __init__ has run that has the same name as an argument, the instance attribute will be saved instead of the argument value. This allows code within __init__ to process default arguments before they are saved.

Internally, all properties are accessed by calling `self.get_property()`.

Parameters

property_names (*dict*) – A dictionary {key:val}, where each `val` is a list [var1, var2, ...] of instance attribute names and/or method call arguments (of the decorated method). Values of the instance attributes/method call arguments will be saved to the location specified by `key`.

3.9.9 labscript.utils.suppress_all_warnings

suppress_all_warnings(*state=True*)

A context manager which modifies compiler.suppress_all_warnings

Allows the user to suppress (or show) all warnings for specific lines. Useful when you want to hide/show all warnings from specific lines.

Parameters

- **state** (*bool*) – The new state for `compiler.suppress_all_warnings`. Defaults to
- **provided.** (*True if not explicitly*) –

3.9.10 labscript.utils.suppress_mild_warnings

suppress_mild_warnings(*state=True*)

A context manager which modifies compiler.suppress_mild_warnings

Allows the user to suppress (or show) mild warnings for specific lines. Useful when you want to hide/show all warnings from specific lines.

Parameters

- **state** (*bool*) – The new state for `compiler.suppress_mild_warnings`. Defaults to
- **provided.** (*True if not explicitly*) –

Exceptions

LabscriptError	A *labscript* error.

3.9.11 labscript.utils.LabscriptError

exception LabscriptError

A *labscript* error.

This is used to denote an error within the labscript suite itself. Is a thin wrapper of `Exception`.

CHAPTER

FOUR

LABSCRIPT SUITE COMPONENTS

The *labscript suite* is modular by design, and is comprised of:

Table 1: Python libraries

labscript — Expressive composition of hardware-timed experiments
labscript-devices — Plugin architecture for controlling experiment hardware
labscript-utils — Shared modules used by the *labscript suite*

Table 2: Graphical applications

runmanager — Graphical and remote interface to parameterized experiments
blacs — Graphical interface to scientific instruments and experiment supervision
lyse — Online analysis of live experiment data
runviewer — Visualize hardware-timed experiment instructions

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

W