Labscript FPGA Documentation

Matt Earnshaw
matt@earnshaw.org.uk

August 2014

Contents

1 Installation

The labscript install guide is not recommended (as of August 2014). It is preferable to follow the method below which installs the various labscript suite modules automatically using the pip utility. This should work on Windows or Linux, however for ease of installing libftdi, Linux is the recommended platform.

1.1 Prerequisites

Install the following using your OS's package manager or otherwise. You may also need to install a FORTRAN compiler (eg. gcc-fortran) if you have issues installing NumPy/SciPy.

- Python 2.7
- python-pip
- libftdi (version 1.0+, aka libftdi1 - including python bindings)
- PyQt4
- PyGTK
- SciPy
- NumPy
- matplotlib

1.2 Installing labscript suite

Use pip to install the tarballs from my forks of the labscript suite repositories. pip should automatically resolve and install the extra dependencies.

```
$ sudo pip install https://bitbucket.org/mearnshaw/labscript_utils/get/1.1.0-dev.tar.gz
$ sudo pip install https://bitbucket.org/mearnshaw/labscript/get/gated-clocks.tar.gz
$ sudo pip install https://bitbucket.org/mearnshaw/labscript_devices/get/fpga-device.tar.gz
$ sudo pip install https://bitbucket.org/mearnshaw/blacs/get/1.1.0-dev.tar.gz
$ sudo pip install https://bitbucket.org/mearnshaw/mise/get/default.tar.gz
$ sudo pip install https://bitbucket.org/mearnshaw/runmanager/get/default.tar.gz
$ sudo pip install https://bitbucket.org/mearnshaw/runviewer/get/gated-clocks.tar.gz
```

If any problems are encountered in the above, try manually installing the extra dependencies (see below), then retry the above.

```
$ pip install h5py pyzmq==13.1.0 pyside pandas PyDAQmx spinapi qtutils zprocess
```

1.3 Testing installation

In a Python shell run the following (for example - requires FPGA to be connected):

```
>>> import labscript
>>> from labscript_devices.FPGADevice import FPGADevice
>>> from labscript_devices.FPGADevice.fpga_api import FPGAInterface
>>> i = FPGAInterface()
>>> i.start()
```

no errors = success!

2 Setting up BLACS/runmanager

A configuration file is required before runmanager will talk to BLACS. On Windows this file should be in `C:\labconfig\`, and on Linux either in `/home/[user]/labconfig/`, or `/etc/labconfig/`. The name of the file should be <hostname>.ini where <hostname> is the hostname of the machine (under Linux this is the string returned by the hostname command, or see this for Windows). See below for an example configuration file.

```
[DEFAULT]
experiment_name = test

[paths]
shared_drive = /tmp/
experiment_shot_storage = /home/matt
labscriptlib = /
connection_table_h5 = /home/matt/labconfig/connection_table.h5
connection_table_py = /home/matt/labconfig/connection_table.py

[programs]
text_editor = vim
text_editor_arguments =

[ports]
BLACS = 42517
lyse = 42519
mise = 42520
zlock = 7339

[servers]
zlock = localhost
```

The connection_table fields are paths to files containing the experiment's connection table, which is everything in the labscript experiment before start(), ie. where the devices involved and their connections are specified. Example:

```
# connection_table.py

from labscript import *
from labscript_devices.FPGADevice import FPGADevice, FPGAAnalogOut, FPGADigitalOut

FPGADevice(name='my_fpga')

FPGAAnalogOut('analog0', my_fpga.outputs, board_number=1, channel_number=0)
FPGAAnalogOut('analog1', my_fpga.outputs, board_number=1, channel_number=1)
FPGADigitalOut('digital1', my_fpga.outputs, board_number=1, channel_number=8)

start()
stop(1)
```

Note that you must include a "start(); stop(1)" at the end so that runmanager will compile it.

Compile your connection table with runmanager and move the resultant .h5 file to the path specified in the config file. This should only need to be done once as any subsequent time the connection table is change, BLACS will automatically detects the changes and will offer to recompile for you.

3 Using BLACS/runmanager

To start the programs simply run blacs/main.pyw and runmanager/main.pyw which can be found in your site-packages directory (typically /usr/lib/python2.7/site-packages). Refer to Using Runmanger and Using BLACS for more details.

4 General overview of using the FPGADevice in labscript

The FPGADevice class (in labscript_devices/FPGADevice/FPGADevice.py) represents one or more boards connected to the computer through a single USB connection. It can be used in the connection table as follows:

```
from labscript_devices.FPGADevice import FPGADevice
FPGADevice(name, n_analog=8, n_digital=25)
```

If the parameters n_analog, n_digital are provided, then clocks/data will be generated for that many channels regardless of how many are specified in the connection table by generating default (zero) clocks/data on channels that

are left unspecified.

Labscript binds an instance of the device to the name provided in the first parameter as a string. Thus as with all labscript objects, the name must be a valid Python identifier (no spaces for example, see Using the Labscript API and the following for more info).

FPGADevice instances have an `outputs` attribute which returns an `OutputIntermediateDevice` which should be specified as the parent device of any `Outputs`. The supported output types are `FPGAAnalogOut` and `FPGADigitalOut` (labscript_devices/FPGADevice/fpga_outputs.py). As these subclass labscript's `AnalogOut` and `DigitalOut`, all the same functions can be used on them (eg. constant, ramp, sine etc.).

```
FPGAAnalogOut(name, parent_device, board_number, channel_number, group_name=None, limits=None)
FPGADigitalOut(name, parent_device, board_number, channel_number, group_name=None)
```

`limits` is is an optional parameter which if specified should be a tuple of the min,max values allowed on that output (see example). The other optional parameter `group_name` is currently only used internally but could be used in the future to group outputs in the GUI, for example.

```
from labscript_devices.FPGADevice import FPGADevice, FPGAAnalogOut, FPGADigitalOut

# instantiate an FPGADevice bound to my_fpga, with 8 digital channels and 25 analog channels (default)
FPGADevice("my_fpga")

# attach an analog out to board 0 (default), channel 1, constrained to -5 to 5 V
# and bound to name: red_MOT
FPGAAnalogOut("red_MOT", my_fpga.outputs, channel_number=1, limits=(-5,5))

# attach a digital output to board 1, channel 2, bound to name: shutter
FPGADigitalOut("shutter", my_fpga.outputs, board_number=1, channel_number=2)

shot_reps = 2  # run the shot twice

# do an experiment
start()
shutter.go_high(t=0)

# see labscript API documentation for the full format of these commands
red_MOT.ramp(t=0, ...)
stop(5)
```

5 Waits

FPGADevice provides a wait method for producing global waits on the FPGA.

```
wait(at_time, board_number=None, channel_number=None, value=None, comparison=None)
```

There are 3 categories of wait classified according to the type of input controlling the wait: "PC Wait", "Digital Wait", "Analog Wait". The type of wait required is inferred from the parameters provided to wait. A "PC Wait" only requires the time at which it is to occur, a "Digital Wait" requires the time, board number, channel number and value to wait on (True/False, 1/0), and an "Analog Wait" requires these and an additional comparison operator character (less than: '<', greater than: '>').

```
FPGADevice("my_fpga")
shot_reps = 4  # repeat the shot 4 times

start()
# analog wait
my_fpga.wait(at_time=0, board_number=1, channel_number=2, value=5.2, comparison='<')
# digital wait
my_fpga.wait(at_time=1, board_number=1, channel_number=2, value=True)
# PC wait
my_fpga.wait(at_time=2)
stop(10)
```

6 Shot Reps

The number of repetitions of the shot can be specified by including a shot_reps variable in the script, as in the above examples. If unspecified the default shot repetitions is 1.

7 Implementation details

7.1 FPGADevice.py

The FPGADevice class subclasses labscript's generic PseudoclockDevice class, which represents some Device with multiple pseudoclocks.

7.1.1 Board configuration

The boards_configuration attribute of FPGADevice is a dictionary mapping board numbers to dictionaries with the format

```
{'analog': [analog_channel_numbers], 'digital': [digital_channel_numbers]}
```

so when multiple boards are introduced this should be updated to reflect the new board/channel numbering structure.

7.1.2 H5 file generation

The generate_code method of FPGADevice first creates placeholder outputs on any unconnected channels and then calls the generate_code method of PseudoclockDevice which iterates through all the Pseudoclocks on the device and produces their clocking signals and data (pseudoclock.clock, output.raw_output). FPGADevice takes these raw clocking signals (period in seconds which labscript calls 'step', and repetitions or 'reps') and processes them into a format understood by the FPGA (see section 7.2), writing them to the H5 file.

7.1.3 Output connection name/output name

Each output has a name, which is provided as the first parameter - and a connection name which is automatically generated. util.py provides various functions to retrieve these names because BLACS needs to know how they correspond.

7.1.4 Changing chunksize/delays

At the start of FPGADevice.py there are four variables that can be used to control the chunksize/delay between chunks (in seconds) for clocks/data. To remove the delay or to maximize the chunks, simply set the appropriate variables to None, which will let everything run as fast as possible (default chunksize=512, no delays)

7.2 clock_processing.py

All of the raw signals produced by labscript (for both analog and digital channels) are first reduced by reduce_clock_instructions which simply combines compounds the reps of consecutive instructions with the same period (although due to

rounding/floating point errors, some periods that should be the same actually differ by a small amount and so these can't be combined).

For the analog channels the clock is then processed by the process_analog_clock function. This first discards the last clock instruction, which in all cases tested appears to be unnecessary junk. The formula for transforming the labscript period (seconds)/reps is then

$$\text{period}_{\text{FPGA}} = \frac{1}{2}\text{round}(\text{period}_{\text{labscript}} \times \text{FPGA\_clock}) - 1 \tag{1}$$

$$\text{reps}_{\text{FPGA}} = \text{reps}_{\text{labscript}} - 1 \tag{2}$$

where FPGA_clock is the fundamental frequency, 30 MHz. Any instructions for which labscript_period * FPGA_clock is less than 1000 is discarded as the DACs cannot update this quickly. This behaviour should probably be improved.

Digital channel clocks are processed by convert_to_clocks_and_toggles function. Again this discards the last instruction which is unneeded (in all tests to date). Clocks/toggles are calculated according to

$$\text{clocks}_{\text{FPGA}} = \text{round}(\text{period}_{\text{labscript}} \times \text{FPGA\_clock}) - 1 \tag{3}$$

$$\text{toggles}_{\text{FPGA}} = \text{reps}_{\text{labscript}} - 1 \tag{4}$$

except for the first instruction where toggles is the initial state of the channel. If the first instruction in the labscript clock has more than 1 rep, we simply decrement the number of reps by 1 having created this special first instruction, and then proceed to process what remains according to the above formulae. Finally we subtract 1 from the toggles of the final instruction to compensate for the final autotoggle.

8 FPGADeviceWorker

FPGADeviceWorker is responsible for ultimately coordinating dispatching shots to the FPGA via BLACS. This is achieved via the FPGAInterface class (in fpga_api.py) which has the methods for encoding the various parts of the shot (clocks, data, etc.) into the FPGA protocol.

9 Troubleshooting

9.1 "Cannot connect more than n outputs to the device 'fpga...'"

Ensure the correct number of outputs has been specified for the FPGA in the connection table and that it's the same in the shot file.
eg. FPGADevice(name='fpga', n_analog=8, n_digital=25)

9.2 "The device name … already exists in the Python namespace..."

Ensure all devices in the connection table have unique names.

10 Known Issues