Contents

Chapter 1

Grendel Quickstart

The `Grendel` runtime system (`grendel.py`) enables developers to dispatch commands to the HCDCv2 analog device on the fly. The `Grendel` runtime is used to execute scripts generated by the `Legno` compiler as well as calibrate and profile blocks on the device.

1.0.1 Installation

The `Grendel` runtime is a pure python program. To install the python dependencies, execute the following command:

```
  pip install -r packages.list
```

Both the `Legno` compiler and the `Grendel` runtime work with a `config.py` file that specifies the relevant output directories and database files. To use the default configuration, execute the following copy command:

```
cp util/config_local.py util/config.py
```

1.0.2 Anatomy of a Grendel Script

The following section of the quickstart guide breaks down the `test/cosfun.grendel` script. The `cosfun` script configures the analog device to emit a cosine function at output 0 of the analog device (the `A0` and `A1` differential pair). This differential pair is connected to a Sigilent 1202X-E oscilloscope, which we use to record waveforms **??**.

We describe the anatomy of the `Grendel` script below. Generally, commands prefixed `micro_` are sent to the microcontroller and commands prefixed with `osc_` are sent to the oscilloscope. Any commands lacking these prefixes are dispatched to the analog device through the microcontroller.

Each `Grendel` script begins with a preamble that sets the parameters for the experiment. The `cosfun` script configures the microcontroller to use the analog chip to execute the simulation (`micro_use_chip`) and produce a trigger signal to tell the oscilloscope to start recording (`micro_use_osc`). The `micro_reset` command resets the parameters of the experiment that are stored on the microcontroller.

```
micro_reset
```

```
micro_use_chip
micro_use_osc
micro_set_sim_time 1.587e-03
```

The Grendel script then configures the time and voltage scales of the oscilloscope so that the waveform fits in the viewport. The set_volt_range commands set the minimum and maximum voltages for channels 0 and 1 of the oscilloscope. The osc_sim_time command sets the amount of time the oscilloscope should record for.

```
osc_set_volt_range 0 0.102000 1.310000
osc_set_sim_time 2.063e-03
osc_set_volt_range 1 0.102000 1.310000
osc_set_sim_time 2.063e-03
```

The Grendel script then programs the analog device to implement the cosine function. The cosine function is implemented as a second-order differential equation and consists of two integrators and one current copier (fanout block). The integrators correspond to the position and velocity of the cosine function, and the fanout is used to produce a copy of the position for measurement. Figure **??** presents a schematic of the configured circuit.

```
use_fanout 0 3 0 0  sgn + - + rng m two
use_integ 0 3 0 sgn + val 0.8359375 rng h m debug
use_integ 0 3 1 sgn + val 0.0 rng h m debug
mkconn fanout 0 3 0 0 port 0 tile_output 0 3 0 0
mkconn tile_output 0 3 0 0 chip_output 0 3 2
mkconn integ 0 3 0 fanout 0 3 0 0
mkconn integ 0 3 1 integ 0 3 0
mkconn fanout 0 3 0 0 port 1 integ 0 3 1
```

With the analog device configured, the Grendel script configures the oscilloscope to listen for a trigger (osc_setup_trigger), then executes the simulation (micro_run). The waveform is retrieved from the oscilloscope using the osc_get_values command. This command returns the differential signal between channel 0 and channel 1 of the oscilloscope (CH0-CH1). The measured signal corresponds to the position (Pos) of the cosine function, and is written to the waveform.json file when it's done. The Grendel script also reports the exception status for the integrators that were used by the simulation.

```
osc_setup_trigger
micro_run
osc_get_values differential 0 1 Pos waveform.json
get_integ_status 0 3 0
get_integ_status 0 3 1
micro_get_status
```

After the simulation has finished executing, the Grendel script tears down the analog device configuration. It disables any enabled blocks and unsets and programmed connections.

```
disable fanout 0 3 0 0
disable integ 0 3 0
disable integ 0 3 1
```

```
rmconn fanout 0 3 0 0 port 0 tile_output 0 3 0 0
rmconn tile_output 0 3 0 0 chip_output 0 3 2
rmconn integ 0 3 0 fanout 0 3 0 0
rmconn integ 0 3 1 integ 0 3 0
rmconn fanout 0 3 0 0 port 1 integ 0 3 1
```

1.1 Executing the Grendel Script

Before the `cosfun.grendel` script can be executed, any uncalibrated blocks must be *calibrated*. When a block is configured by the grendel script through a `use_` statement, only some of the parameters are set. We call the parameters that are set through a `Grendel` command *visible* codes. The parameters that are not set through a `Grendel` command are *hidden* or *calibration* codes. These codes are computed by the calibration procedure and cached in the state database (`state.db`). For each set of visible code values, the block must be calibrated. We calibrate all the partially configured blocks that appear in the `cosfun` script with the following command:

```
python3 grendel.py --calib-obj min_error calibrate test/cosfun.grendel
```

This command tells the `Grendel` runtime to calibrate the integrator and current copier blocks in the script. The `--calib-obj` tells the `Grendel` runtime the metric the calibration routine should minimize when looking for the best set of calibration codes. The `min_error` objective function returns the error between the expected and observed behavior for a set of test points.

After the calibration procedure is done, we can then use the grendel runtime to execute the script. This command runs the simulation on the analog device, and writes the measured waveform to `waveform.json`:

```
python3 grendel.py --calib-obj min_error run test/cosfun.grendel
```

The `--calib-obj` flag specifies which set of calibration codes to use to finish configuring the block. We tell the `Grendel` runtime to use the calibration codes that were computed with the `min_error` objective function.

1.2 Analyzing the Results

We can view the waveform using the following python code snippet:

```
import scripts.analysis.quality as qualitylib
import matplotlib.pyplot as plt
times,values = qualitylib.read_meas_data('waveform.json')
plt.plot(times,values)
plt.savefig('waveform.png')
```

The `read_meas_data` script decompresses and parses the `json` file. Once this file is parsed, the data can then be plotted. Figure **??** presents the simulation produced by the analog device. Note that there is not information in the `Grendel` script to recover the original simulation. Refer to the section on ExpDriver for instructions on how to do that.

Chapter 2

Delta Model Inference

Chapter 3

Command Reference

Chapter 4

Batch Execution with Experiment Driver

ExpDriver is a batch processing tool that automatically finds an executes all `Grendel` scripts in the `output` directory. ExpDriver is designed to work with `Grendel` scripts generated by the `Legno` compiler, and uses the `Grendel` runtime as a subroutine for many of its operations. It therefore relies on the naming conventions and directory structure used by the `Legno` compiler.

In addition to batch executing scripts, ExpDriver is able to analyze and produce visualizations for the waveforms generated by the executions. ExpDriver post-processes the collected signals using the scaling factors produced by the `Legno` compiler and compares the recovered signal to a ground-truth reference signal. The ExpDriver is also able to compute the runtime and energy requirements for each execution.

The ExpDriver tracks the state of each experiment in a database located in `outputs/experiment.db`.

4.0.1 Scanning for `Grendel` Scripts

We use ExpDriver to scan for `Grendel` scripts using the following command:

```
python3 exp_driver.py scan
```

This command automatically searches for any `Grendel` scripts in the `outputs` directory and adds them to the experiment database [?]. The experiment database tracks the execution and analysis results of each experiment. All scripts start out as `pending` or `ran`. A script is marked as `ran` if all of the waveforms it is supposed to generate are present.

4.0.2 Executing Pending Grendel Scripts

Next we tell ExpDriver to execute any pending experiments.

```
python3 exp_driver run --calibrate
```

The `--calibrate` flag tells ExpDriver to calibrate any blocks that require calibration before running the experiments. Note that the `--calib-obj` flag is automatically inferred from the model used to compile the script. If the program was scaled by `LScale` using a `naive` model, the `min_error` calibration objective is used for calibration and execution. If the program was scaled by `LScale` using a `partial` or `physical` model, the `max_delta` calibration objective is used for calibration and execution.

4.0.3 Analyzing Grendel Scripts

Assume we have a grendel script for some program `bmark` that was generated with the `Legno` compiler that resides at the following path:

```
outputs/legno/{subset}/{bmark}/grendel/{bmark}_{legno_id}_{tag}_{obj}_{menv}_{
```

ExpDriver supports analyzing the waveforms produced by `Grendel` scripts provided the following criteria are met:

- The`bmark` dynamical system program the `Grendel` script implements exists in the `bmarks` directory and can be found by the `legno` compiler.
- The experiment has been executed. More concretely, the waveform files (`.json`) in the `Grendel` script exist in the `outputs/legno/{subset}/{bmark}/out_waveforms` directory.
- There is an analog device program for the `Grendel` script. This program must exist at the following location:

  ```
  output/legno/{subset}/{bmark}/conc-circ/{bmark}_{lgraph_id}_s{lscale_
  ```

  ExpDriver uses the corresponding analog device program for the `Grendel` script to scale the waveform. Note that the scaling transform computed from the analog device program is produced by the compiler and not derived by using a reference signal in any way.

Provided the above criteria are met, the analysis routine analyzes the waveforms for each experiment. All of these analysis results are stored in the experiment database (`experiments.db`). Once all the waveforms in an experiment have been analyzed, the experiment status is changed to `analyzed`. We present a summary of the analysis results below:

- *Runtime*: Using the time scaling factor computed by the `Legno` compiler, ExpDriver reports the runtime of the simulation in wall clock time [**?**]. This quantity is the same across all waveforms.
- *Energy*: Using the energy model for the hardware and the runtime of the simulation (in hardware time), ExpDriver estimates how many μJ are required to execute the simulation [**?**, **?**]. This quantity is the same across all waveforms.
- *Quality Measurements*: Using the time and amplitude scaling factors computed by the `Legno` compiler, ExpDriver converts the times and values of the measured waveform to simulation times and values. The signal is then aligned in the time domain with a reference signal, which is computed using a high-precision digital solver. This alignment procedure only allows time shifting to account for synchronization issues between unleashing the capacitors and triggering the oscilloscope. After aligning the signals ExpDriver computes a the sum squared errors between the reference and observed signals [**?**, **?**]. This is the quality metric for the execution (lower is better).

ExpDriver produces several helpful plots while computing the quality of the waveform. These plots are written to the `output/legno/{subset}/{bmark}/plots` directory. Each visualization is tagged with a suffix that indicates what kind of plot it is. We present a summary of the suffixes below:

- `_ref.png`: The reference simulation. The time is in simulation units and the value is in state variable units.
- `_meas.png`: The waveform measured from the oscilloscope. The time is in wall clock time units, and the amplitude is in volts.
- `_pred.png`: The waveform that is expected to be generated by the chip. This is computed by applying the time and amplitude scaling factors from the analog device program to the reference waveform. The time is in wall block time units and the amplitude is in volts.
- `_rec.png`: The measured waveform, after the extraneous parts of the signal have been trimmed away and the scaling transform has been un-applied. The time is in simulation time units, and the amplitude is in state variable units. We call this the recovered waveform.
- `_comp.png`: A comparison between the recovered waveform and the reference waveform. This discrepency between these signals informs the quality of the simulation.

4.0.4 The Scaling Transform

In this section, we go into a bit more detail on how the scaling transform is used to recover the desired simulation from the measured signal. The scaling transform is computed from the analog device program that produced the grendel script:

```
output/legno/{subset}/{bmark}/conc-circ/{bmark}_{legno_id}_{tag}_{obj}.circ
```

This program was generated by the `LScale` pass of the `Legno` compiler. This means that the parameters in the program have already been scaled so the signals fit within the operating constraints of the device. ExpDriver reads the following quantities from the program [?]:

- **Time Scaling Factor**: ExpDriver reads the time scaling factor τ from the program.
- **Waveform Scaling Factor**: For each waveform, ExpDriver finds the port in the analog device program that emitted the waveform. The scaling factor at this port, α, is the scaling factor for the waveform.

ExpDriver uses the τ and α parameters to predict the measured waveform from the reference waveform and recover the original dynamics of the signal from the measured waveform. We describe these procedures below:

Recovery of Original Signal

Given a waveform collected from a measurement device, such as an oscilloscope, ExpDriver recovers the original dynamics of the signal by applying the following transforms to the time (t_{meas}) and voltage v_{meas} measurements:

$$t_{rec} = (t_{meas} - \delta) \cdot \tau^{-1} \qquad v_{rec} = v_{meas} \cdot \alpha^{-1} \tag{4.1}$$

(4.2)

In this equation, δ is the time shift found by the alignment procedure. Refer to the `scale_obs_data` function in the `quality.py` file for the implementation. Note that the implementation also trims away any measurements that occur before or after the simulation.

Predicting Measurements from Reference Signal

Given a reference signal computed by digitally executing the dynamical system, the time and waveform scaling factors can be used to predict the measured signal. This is done by applying the following transform to the simulation time (t_{ref}) and amplitude v_{ref}measurements:

$$t_{pred} = t_{ref} \cdot \tau \qquad v_{pred} = v_{ref} \cdot \alpha \tag{4.3}$$

$$\tag{4.4}$$

Refer to the `scale_ref_data` function in the `quality.py` file for the implementation.

Aligning Signals

ExpDriver aligns signals by taking the cross correlation of the resampled signals, and finding the offset that has the highest correlation value. This offset is then converted into a time delay δ. The time delay is the lag of the measured signal with respect to the reference signal.

Bibliography