Contents

Chapter 1

Legno Quickstart

The `Legno` compiler (`legno.py`) enables developers to compile dynamical systems down to configurations for the analog hardware. The `Legno` compiler requires that dynamical systems be specified in the `dynamical system language`, a high level language that supports writing first-order differential equations. The `Legno` compiler also accepts a specification of the target analog device, described using the Analog Device API. The `Legno` compiler generates an `analog device program` which implements the target dynamical system on the specified analog device. In the following quickstart guide, we walk through an example where we compile a dynamical system that models the cosine function for the HCDCv2 analog 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
```

The `Legno` compiler works 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 Dynamical System Program

The following dynamical system program implements the cosine function:

```
def make_prog():
  params = {
    'v(0)':0.0,
    'p(0)':1.0
  }
  prog = DSProg(cos)
  prog.decl_stvar('v','-p',ic='{v(0)}', params=params)
  prog.decl_stvar('p','v',ic='{p(0)}', params=params)
```

```
prog.decl_var('pos','emit p')
prog.range('v',-1,1)
prog.range('p',-1,1)
prog.max_time = 100
prog.check()
menv = DSSim(20)
return prog,menv
```

In this dynamical system, the position `pos` corresponds to the amplitude of the cosine function. The v and p variables are internal variables that are used to model the dynamics of the system. We describe each line of the program below:

- `prog = DSProg('cos')`: This statement creates a new dynamical system program with the name `cos`.
- `prog.decl_stvar('v','-p',ic='{v(0)}',params=params)`: This statement declares a variable v that has the dynamics $v' = -p$, and an initial value of $p(0)$, where $p(0)$ is defined in the parameter dictionary `params`. Note that any variables in curly braces are replaced with values from the parameter dictionary.
- `prog.decl_stvar('p','v',ic='{p(0)}',params=params)`: This statement declares the variable p that has the dynamics $p = v$ and an initial value of 1.0;
- `prog.decl_var('pos', 'emit p')`: this statement communicates to the compiler that we want to measure the position of the system. We name the emitted signal `pos`.
- `prog.range('v',-1,1) and prog.range('p',-1,1)`: These statements promise to the compiler that the position and velocity variables will be between -1 and 1. The range of the `pos` variable is inferred from the ranges of the `p` and `v` variables.
- `prog.max_time=100`: The maximum time this simulation will ever be run for, in simulation units. This is useful for situations where the measurement hardware does not support recording signals for long periods of time.
- `prog.check()`: This command checks that the program is well formed. A program is well formed if all the variables are bounded.
- `menv = DSSim(20)`: This statement defines a simulation that executes for 20 simulation units. Unlike the dynamical system program, which is used throughout the the compilation process, the dynamical system simulation object is only used at the very end.

The above dynamical system program is written to `cos.py` in the `bmark/bmarks/quickstart/` directory. In order for `legno.py` to find the benchmark, the `bmark/bmarks/quickstart/__init__.py` file must be modified to include the cosine benchmark in the array returned by `get_benchmarks`:

```
import bmark.bmarks.quickstart.cos as cos

def get_benchmarks():
   return [cos.model()]
```

This modification to `__init__.py` enables `legno.py` to find the cosine benchmark.

Executing the cos Dynamical System with a ODE Solver

TODO

1.0.3 Compiling the cos Program

The Legno compiler first generates an unscaled analog device program that implements the cos benchmark. An analog device program (.circ) consists of a set of block configurations and digitally programmable connections to write to the device. This program is unscaled, meaning that the parameters have not been changed so that dynamical system operates within the constraints of the device. We generate the analog device program for the cos benchmark with the following command:

```
python3 legno.py --subset standard cos arco --abs-circuits 1
    --conc-circuits 1
```

This step of the compilation process is called the LGraph compilation pass. The --subset flag indicates what subset of device features to use to generate the graph. The standard subset limits the accepted modes for each block, which restricts the dynamic range of the device. The --abs-circuits and --conc-circuits parameters control the search space explored by LGraph compilation pass. Specifying higher numbers to these flags produces more analog device programs.

For the cos dynamical system with the standard set of features, all unscaled circuits are stored in the following directory:

```
outputs/legno/standard/cos/abs-circ
```

The LGraph command presented above produces exactly one unscaled analog device program:

```
cos_0_0_0_0.circ
```

Since the analog device program is not human readable, the compiler also produces a graph that describes the analog device program.

```
outputs/legno/standard/cos/abs-graph/cos_0_0_0_0.png
```

Parameter Scaling with LScale

Next, we direct the Legno compiler to scale each unscaled analog device program in the abs-circ directory:

```
python3 legno.py --subset standard cos jaunt --search
   --model naive --scale-circuits 1
```

This step of the compilation process is called the LScale compilation pass. The --subset flag indicates what subset of device features to use. The --model argument indicates whether delta models should be considered when scaling the system (A delta model is a model that describes the physical hardware behavior). The naive model assumes each block delivers its expected behavior with no deviations. The --scale-circuits parameter determines how many scaled programs to produce from each unscaled program. Finally the --search parameter tells Legno to search for the scaling transform that produces the best signal-to-noise ratio.

For the cos dynamical system with the standard set of features, the resulting scaled programs are written to the following directory:

```
outputs/legno/standard/cos/conc-circ
```

The `LScale` execution presented above produces exactly one scaled analog device program:

```
cos_0_0_0_0_s0_nq1.53d1.10b_obsslow.circ
```

Since the analog device program is not human readable, the compiler also produces graphs that visually depict the circuit each program implements. These graphs are stored in the following directory:

```
outputs/legno/standard/cos/conc-graph
```

Source Generation

For each scaled analog device program, legno generates a low-level `Grendel` script that executes the experiment on the analog hardware.

```
python3 legno.py --subset standard cos srcgen default --trials 1
```

This command generates a `Grendel` script for each scaled analog device program. For the `cos` dynamical system with the `standard` set of features, all the `Grendel` scripts are stored in the following directory:

```
outputs/legno/standard/cos/grendel
```

Since the `cos` benchmark only has one scaled circuit, only one `Grendel` script is produced:

```
cos_0_0_0_0_s0_nq1.53d1.10b_obsslow_t20_default.grendel
```

This grendel script can be dispatched to the HCDCv2 analog device with the following commands:

```
python3 exp_driver.py scan
python3 exp_driver.py run
```

These commands execute the cosine dynamical system described in Section **??**. For the `cos` benchmark with the `standard` set of features, the waveforms are stored in the following directory:

```
outputs/legno/standard/cos/out-waveform
```

The collected waveforms can then be analyzed (compared with the expected cosine function) with the following command:

```
python3 exp_driver.py analyze
```

The relevant visualizations are stored in the following directory:

```
outputs/legno/standard/cos/plots
```

Chapter 2

Legno: An Overview

Chapter 3

Dynamical System Language

coming soon....

Chapter 4

Hardware Specification API

Chapter 5

Analog Device Program Language

Chapter 6

`LScale`: The Legno Scaling Engine

Chapter 7

`LGraph`: The Legno Graph Synthesis Engine

Bibliography