

Introduction

The EAC toolkit is a collection of utilities intended to make it easier to use the extended analog computers designed here at Indiana University. The project grew out of my own research this summer. Originally, the interface was developed to tie together the collection of Perl scripts that are currently used to manipulate the EACs.

I'll spare rest of the details of the history for now. Suffice it to say there are three major components of the toolkit, some of which are relevant to the course, some of which are not. These components include:

1. A simple user interface for direct manipulation of the analog computers.
2. A hardware abstraction layer and pseudo-API to provided a unified programming model for both the silicon and foam substrates of the original EAC, as well as the forthcoming uEAC.
3. A simple collection of tools for my own research; namely, data logging utilities and a genetic algorithms package.

Prerequisites

Development was done using Perl on Linux. The code is tested and known to run on the Burrow machines. Plotting relies on gnuplot, while the default viewer is “Eye of Gnome” (eog). If you want to run try things on your own machine, it is most likely to work under Gnome on Linux.

Mac OSX support is broken right now. If you want to buy me a Mac, I will fix it for you. ;-)

I haven't tried this on Windows.

uEAC support requires more prerequisites, and is therefore disabled for the time being.

Toolkit Layout

Here is a rough breakdown of the modules:

- `config.pm` Default values and global variables.
- `interface.pm` A simple user interface to the analog computer.
- `hardware.pm` Hardware abstraction layer for the analog computer.
- `logging.pm` File I/O and data logging routines.
- `utilities.pm` Various utility functions.
- `ga.pm` Genetic algorithm package for analog.
 - `fitness.pm` Fitness functions for the GA; you'll want to code here.

The first three modules are the most interesting. `config.pm` is where you will find all the settings and global variables. If you would like to get a sense of the breadth of the toolkit, this is a good place to start. I have tried to document my code and maintain coding standards that are conducive to understanding how things work. With that said, it's still Perl, and Perl is an ugly, ugly language.

`interface.pm` provides the user interface, and is otherwise not very interesting. `hardware.pm` is the

hardware abstraction layer. It has developed into a five-layer OSI-like model. There are lots of comments in there, but I will not document it here.

The remaining modules are related to my own work. I am still working on them, so don't be surprised if they are broken!

User Interface

From the command line, type `./interface.pm`. This will get you going. A list of commands is available by type “h” or “help.”

Sources and Sinks

Sources and sinks are manipulated by the `source` and `sink` commands. Each of these accepts two arguments: the channel to use (between 1-8), and the amount of current to source or sink (between 0-200 uA). Remember, all the commands do are set the current values on the wires. You have to manually set up the wires on the machines.

Both commands support ranges. If, for example, you would like to send 110 uA to source 1 through 5, you would enter `source 1 5 110`.

A note about current and electricity. First, I do not really understand the relationships between current and voltage and resistance and all that. I don't want to. My assumption is that many people who will use the analog computer do not want to learn it either. Thus the goal of my project is to create tools that sufficiently abstract away from the details of the electrical properties of the sheet.

With that in mind, I have capped the maximum current at 200 uA. Values much higher than 200 uA will lead to saturation of the sheet. This means that it will be difficult to do anything useful because the voltage gradient will always be near the maximum.

LLAs

LLAs can be configured in much the same way. `lla` takes a channel (between 1-6) and one of the 27 piecewise linear functions to use (between 1-27). These functions are listed on Dr. Mills' homepage.

An LLA differs from a source in that it draws the maximum amount of current off the sheet. It is like a sink set to “full blast.” This means that, once attached, the LLA will always be drawing current off the sheet. The current that goes into a the LLA is run through the piecewise linear transfer function. The output current value can either be:

1. Added back onto the sheet by use of the `LLA_SRC` output.
2. Further subtracted off the sheet by use of the `LLA_SNK` output.
3. Added at one point and subtracted at another by using both outputs.
4. Reported (see note below).

NOTE: due to the design of the EAC, the value that is reported by the LLA **cannot be taken to be accurate**. While the value that is outputted back onto the sheet *is* accurate, the reported value *is not*. To work around this, you should take the output of the LLA and run it into a second LLA, using the second

LLA's input value as the correct output value of the first LLA. Basically, you are daisy-chaining LLAs to get an accurate reading.

Resetting

Resetting is a loaded term. I want to be clear: resetting, in this case, simply means that source and sink values are all set to 0. Due to details of electrical theory that I do not care to understand right now, you cannot assume that the voltage gradient will necessarily respond uniformly to zeroing the inputs onto the sheet. Conceptually, I would like reset to return the sheet to as near 0.0v as possible; however, that simply is not feasible right now. My experience is that resetting the sheet will push the voltage gradient down near -5.0v.

Also remember that if you have an LLA attached to the sheet it will continue to act on the sheet. There is nothing that can be done in software to negate the effects of an LLA, so this must be taken into consideration when working with the original EACs.

Reporting

Three commands are useful for reporting the voltage gradient on the sheet: `report`, `plot`, and `view`. These commands separate the task of reading a voltage gradient, plotting it with `gnuplot`, and viewing it with the local image viewer. Reporting commands may be stacked; that is, you may specify `report plot view` to do all three, or `report plot`, or simply `report`. The output of `report` and `plot` is saved, the interface will tell you where.

While each of these commands have utility when programming against the EAC, you will probably want to use the `rpv` command. It is simply a shortcut for `report plot view`. Optionally, you may specify a number with `rpv`, the number of successive reports to run.

Logging

As I mentioned above, all the files generated by reporting and plotting (and other functions) are saved. Collectively, these files are stored in a *dataset*, stored in sequentially numbered dataset directories.

Sources and sinks can be reset to 0 uA using the `reset` command.

Logging is set up to support genetic algorithms (which I believe is the native programming paradigm of the EAC, but I digress). Thus each plot is given an identifier. I only mention this so you know how things work a little better. The identifier is of the format `00-000-000`. The first number is the generation identifier, the second is the population identifier, and the third is the reading identifier. Since you (probably) won't be using the genetic algorithm, only the third number will update.

The more you know.

Ending Now...

There is a lot more to say about the toolkit, including using the GA, programming against the hardware module, and using debug modes. I just wanted to give a quick introduction to the interface module.