

anabrid

lucidac

handbook



Serial code sticker

Device identification
and authentication

Here you can attach copies of the labels which are permanently put on the back of the LUCIDAC corresponding to this manual. This helps finding relevant data quicker.



Never connect this device directly to the main power line. Do not apply voltages greater than ± 2 V to this device's front connectors. The application of voltages greater than ± 2 V to this device's front panel connectors may cause damage to property, personal injury, or death. Only connect the supplied power supply to the power socket on the back of the device.

This device is designed for users aged 12 years and older. Users below the age of 12 require adult supervision.

LUCIDAC User Manual

Copyright ©2024 anabrid GmbH

Jan 2025, V. 3.0 (third edition)



This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA. “This Work” means this booklet, the figures, code snippets shown, and the text.

anabrid™, LUCIDAC™, REDAC™ and the product logos are registered trademarks of anabrid GmbH. For further legal information please see page 54.

anabrid GmbH,

Am Stadtpark 3,

12167 Berlin, Germany.

Phone: +49 30 6293047 20

Email: hello@anabrid.com

Web: www.anabrid.com



You can read the latest version of this document online at
<https://anabrid.com/lucidac-user-manual.pdf>



Contents

Preface	1
1 Introduction and Getting started	3
1.1 Requirements	3
1.2 What is in the box	5
1.3 First time hardware setup	6
1.4 Device connectivity	7
1.5 The LUCIDAC co-processor design	9
1.6 Installing the <i>lucipy</i> reference code	10
2 LUCIDAC architecture	11
2.1 Interconnection network	11
2.2 LUCIDAC Blocks	15
2.3 Math blocks	16
2.4 Circuit Configuration	17
2.5 Coupling multiple LUCIDACs together	19
2.6 Coupling THAT with LUCIDAC	21
3 Example applications	24
3.1 Lorenz attractor	25
3.2 Rössler attractor	27
3.3 Hindmarsh Rose Neuronal Bursting	29
3.4 Van der Pol oscillator	31

4	Device administration and maintenance	33
4.1	The Lucigo administrative code	33
4.2	Permanent settings and network configuration	34
4.3	Access control	35
4.4	Device identification	35
4.5	System logs and usage metrics	36
4.6	Firmware Update	36
4.7	Physical maintenance	37
4.8	Testing and calibration	37
5	Embedded programming	39
5.1	Front panel digital connector	40
5.2	Software architecture and communication interface	41
5.3	System states	42
5.4	Writing a compile time plugin	43
5.5	Extending the existing network protocol	43
6	Troubleshooting	45
6.1	Basic connectivity and startup	45
6.2	Analog programming problems	47
6.3	Frequently asked questions (FAQ)	49
7	Resources and further reading	52
8	Legal documents	53
8.1	European Union CE/RoHS Conformity Declaration	53
8.2	Safety Instructions, Maintenance, Warranty and Liability	54

Preface

Welcome to Analog Computing

Analog computers differ substantially from current digital computers in that they do not work by executing an algorithm in a step-by-step fashion. Instead they consist of a number of *computing elements*, each capable of performing a certain mathematical operation such as summation, multiplication or time-integration. A *program* for an analog computer describes how these computing elements are to be connected in order to create a *model* (an *analogue*) of the problem to be solved.

A (very) simple problem like computing $a(b + c)$ could be implemented on a classic digital computer as shown in figure 1. This straightforward algorithm requires six individual steps to compute the desired result. Contrast this with the analog computer program shown in figure 2. This setup requires just two computing elements, one summer and one multiplier. The summer is fed with the values b and c while the multiplier is connected to the output of this summer and to the value a .

```
LOAD A, R0
LOAD B, R1
LOAD C, R2
ADD R1, R2, R1
MULT R0, R1, R0
STORE R0, ...
```

Figure 1: Computing $x = a(b+c)$ on a digital computer

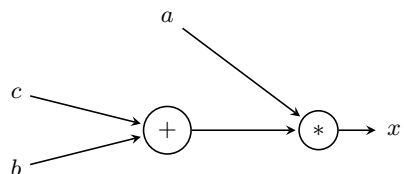


Figure 2: Analog computer setup for solving $x = a(b+c)$

Typically, values are represented by voltages or currents in an analog computer so that only a single connection is necessary between computing elements.

The advantages of this analog computing approach are manifold. Most notable are the extremely high degree of parallelism (there are no central memory, no data dependencies, no synchronisation points, etc., so that the computing elements work in full parallelism), the resulting high speed of computation and the inherent very high energy efficiency of analog computing.

While a digital computer can basically solve every problem given enough time and memory, an analog computer setup needs as many computing elements as there are operations in the governing equations of the problem to be solved. A little bit more mathematically, one can say that the size of a digital computer is constant while its time to solution typically grows much faster than just linearly with the size of the problem (making many problems basically intractable on classic digital computers). The size of the analog computer on the other hand grows linearly with the problem size but, and this cannot be overestimated, the time to solution remains constant.

Classic analog computers were impressive systems, typically featuring a large patchpanel with thousands of jacks, all connected to a vast number of computing elements, such as integrators, summers, coefficient potentiometers, multipliers, etc. Programming these machines was as much of an art as a science and was quite time consuming due to the hundreds or even thousands of connections that had to be made manually. ([ULMANN, 2023/2] describes the history of analog computing in great detail.)

Nowadays, the patchpanel is a museum piece and the actual connection of the computing elements is done electronically, under the control of an attached digital computer; this greatly simplifies the programming. Using appropriate libraries as shown below, the analog computer can be used as a mathematical machine without having to understand the underlying electronic implementation in detail.

Since the basics of analog computer programming are outside the scope of this user guide, please refer to [ULMANN, 2023] for detailed information on the subject.

Section 1

Introduction and Getting started

The LUCIDAC is a fully software-reconfigurable analog-digital hybrid computer intended to be used as a form of co-processor in conjunction with a digital computer. Its main area of application is the exploration of analog and hybrid computing approaches in a variety of fields including high-performance computing, life sciences, mathematics, hardware-in-the-loop setups for (industrial) control purposes, education and training, and many more.

LUCIDAC is the first commercially available modern hybrid computer, and has eight integrators with two software selectable time scale factors of $k_0 \in \{10^2, 10^4\}$, four four-quadrant multipliers, 32 coefficient elements with 11 bits of resolution plus a sign bit each, and a variable number of (implicit) summers. Using these computing elements it is possible to solve eight coupled differential equations (DEQs) of 1st order or four DEQs of 2nd order, etc. Thanks to the multipliers, non-linear equations can also be easily implemented.

The LUCIDAC system contains a local microcontroller unit (MCU) responsible for the communication with the attached digital computer, configuring the computing elements and their interconnection, reading out values by means of analog-digital-converters (ADCs), etc.

1.1 Requirements

The LUCIDAC system is designed for ease of use. However, the user should have basic programming, ideally some experience with (scientific) Python, as this is the main software environment used here. Furthermore, some math knowledge is required (calculus, differential equations). The minimum hardware requirements are:

- 100 V – 240 V AC mains power connection for the supplied power supply.
- A notebook, desktop or server grade computer with ordinary IPv4 networking and/or a USB2 interface. If the USB interface is to be used, root/administrator rights are very likely to be required (see section 4.2 on page 34).
- The operating systems Apple Mac OS X™ (macOS 10.9 Mavericks or later), Microsoft Windows™ (Windows 7 or later) or GNU/Linux are supported. In addition to this at least Python 3.8 (released in 2019) is required for the reference client software *lucipy*.

It is recommended to use the device in a conventional “local” network with a DHCP server and networking switch. Furthermore, a digital storage oscilloscope (*DSO*) is recommended for convenience. However, note that the following points are not mandatory in order to use LUCIDAC:

- Operating the LUCIDAC system does not require an Internet connection. The LUCIDAC firmware will never intentionally connect to another host if not instructed to do so. The LUCIDAC will never “call home” without being told so.
- Connecting LUCIDAC over USB does not require USB-C or USB3 on the host computer. Classical USB 2.0 is sufficient.
- A conventional local network is not needed but is strongly recommended if connecting via TCP/IP. Technically, connecting to the LUCIDAC over TCP/IP neither requires a DHCP server or networking switch. The LUCIDAC can be configured with a static IPv4 address. Furthermore a cross-over direct connection can be made between any computer and the LUCIDAC without special cables. However, the static IPv4 configuration requires basic knowledge about IP networking (see section 4.2 on page 34).
- LUCIDAC provides internal methods for data acquisition (ADCs) which come in handy with hybrid programming. However, using a DSO provides better user interaction, especially initially.

1.2 What is in the box

The LUCIDAC system is shipped with the following complement of items:

1. The LUCIDAC itself, housed in a durable and easily stackable metal case (due to the low power consumption no active ventilation/cooling is required).
2. A universal power supply with a 24 V DC output. Please note that the user will need to provide a country-specific mains lead with an IEC C13 connector.
3. A collection of MCX-MCX and MCX-BNC cables for interfacing the LUCIDAC system to external signal sources, control equipment, oscilloscopes, etc.
4. A master/minion digital port header as well as a suitable long flat ribbon cable for connecting multiple LUCIDACs together.
5. An RJ45 Ethernet cable suitable for connecting the LUCIDAC system to your inhouse network or to your computer (cross-over).
6. An USB-C to USB-A cable suitable for connecting the LUCIDAC system directly to your computer.
7. This user guide.

1.3 First time hardware setup

We recommend connecting the LUCIDAC to an existing ethernet network. Ensure your end device (notebook, desktop or server) is part of the same IP network. If this is the case, the LUCIDAC device can be easily *autodiscovered* by the LUCIDAC client software within the same multicast (broadcast) domain. This way, you do not have to find out its assigned IP address. If you do not have an existing network available, you can make a direct USB connection instead. The recommended steps are as follows:

1. Connect either Ethernet *or* USB *or* both.
2. If using a DSO, connect the first few LUCIDAC outputs channels to your DSO inputs using the enclosed MCX-BNC cables. If desired you can use the LUCIDAC OP output (this line represents the mode of operation of the LUCIDAC and is a the main trigger source for external devices) as the trigger input for the DSO.
3. Power the LUCIDAC device on by flipping the power switch on the front.

The LUCIDAC system boots within about 10 seconds. The 8 LED array will light up and the system is fully booted when the LEDs turn off again. Note that the system is passively cooled and thus fan less and completely silent. In order to turn off the system again, just flip the power switch. There is no need to perform any shut down procedure.

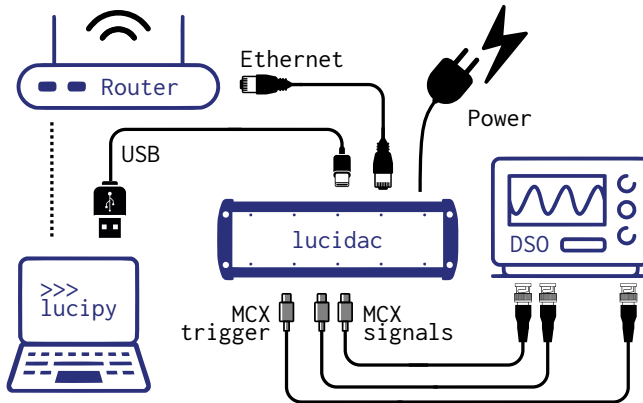


Figure 1.1: Recommended first time setup

1.4 Device connectivity

The LUCIDAC front (Figure 1.2) has a variety of analog and digital inputs and outputs, while the back panel (Figure 1.3) has power and communication ports.

The digital part of the front panel has a 3.3V logic level, regardless of the MCX or pin connector shape. The four binary status lines, available at their respective MCX connectors, are shown in table 1.1.

The analog I/O signals have a $\pm 2V$ voltage range which translates to the ± 1 machine unit domain of the system. The eight output channels are connected to particular system matrix *lanes* and are typically used for connecting external measurement equipment, such as a DSO. The eight input channels can be used as inputs to the system matrix, each replacing one lane. (This will become clearer in section 2 when the LUCIDAC architecture has been described.)

The right side of front panel contains four MCX *outputs* connected to the built-in signal generator. These signals can be either connected to MCX inputs on the front panel or used for other purposes. *Do not* short circuit or overload these outputs, otherwise the signal generator will be damaged.



Never connect sources exceeding $\pm 2V$ to the MCX front panel jacks. This will permanently damage the computing elements!

The inputs and outputs can be damaged by excessive external signal amplitudes. The same is true for the pin header. Using the input/output ports requires a thorough understanding of the signal levels involved.

IC	OP	OL	HALT
Initial Conditions	Operating	Overload	External halt
The system is in preparation mode and the integrators load their initial conditions. This is normally an <i>output</i> signal (see section 2.5 about master/minion mode for alternative modes).	The system is in computing mode and the integrators are operating. This, too, is normally an <i>output</i> signal.	An overload is detected in the computing circuit. This is always an <i>output</i> . Depending on the current configuration of the system, this condition can be used to automatically halt the current computation.	This is an <i>input</i> signal and allows halting the machine controlled by an external, incoming trigger signal.

Table 1.1: Front panel system mode LEDs and MCX sockets and their descriptions

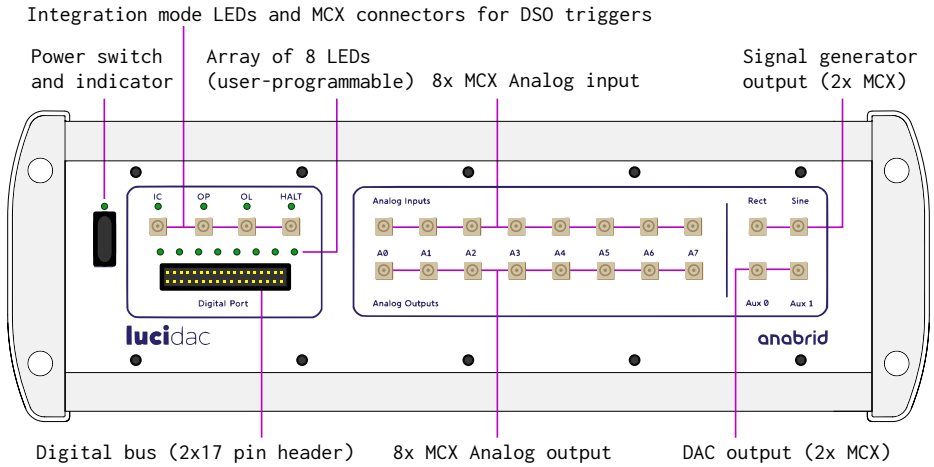


Figure 1.2: Front connectivity

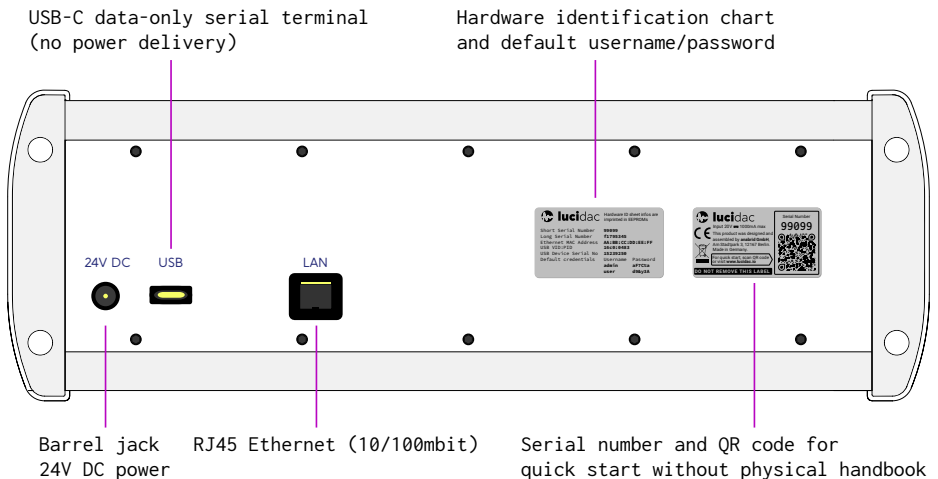


Figure 1.3: Back connectivity. For type plates (labels) see also front matter

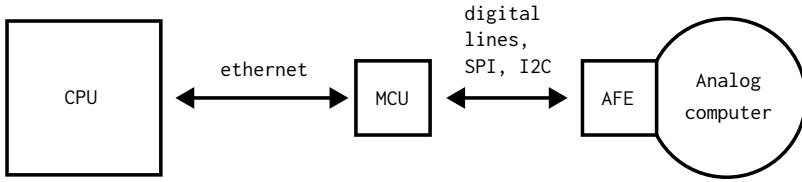


Figure 1.4: CPU (central processing unit)/MCU (microcontroller unit) concept. AFE denotes the analog front-end. SPI and I2C are serial protocols mainly used for communication between integrated circuits.

1.5 The LUCIDAC co-processor design

The LUCIDAC achieves the goal of an analog co-processor with the help of an embedded micro-processor which is connected via ethernet to an upstream computer (Figure 1.4). This creates a heterogenous computing environment with a “big” CPU which is relatively far away from the actual LUCIDAC system (also typically not real-time capable, both in terms of the connectivity as well as with respect to the operating system used) and a “small” MCU which is real-time capable but has relatively low computing power.

Exploiting the performance advantages of analog-digital hybrid algorithms requires clever distribution of an algorithm between the CPU, the MCU, and the analog computing elements of the LUCIDAC. This topic is discussed in more detail in section 5. A simpler way of programming such a hybrid computer, referred to as *client based programming*, is described first. Depending on the connection between CPU and MCU, different names for the roles of MCU and CPU codes are commonly used:

1. When using ethernet, a classical *client-server model* is used. In this setting, the embedded microcontroller within LUCIDAC acts as a network-enabled *server*. A variety of different *client* codes exist which run on the CPU. Multiple clients can connect to one server at the same time. LUCIDAC allows locking the device to a single client.
2. When using USB, the USB *host computer* runs the “client code” with the MCU acting as USB *peripheral*. There can always be only one “client” be connected via USB. Ethernet and USB can be used at the same time.

In any situation, one CPU/client can be connected to multiple LUCIDACs at the time, regardless of the transport method (Ethernet or USB).

1.6 Installing the *lucipy* reference code

For the LUCIDAC, client code implementations are available mainly for Python although other languages will be supported in the near future. Here we focus on *lucipy*, which is currently the reference protocol implementation in Python. It requires at least Python 3.8 and thus should run on any (popular) computer platform built in recent years. The open source code is available at <https://github.com/anabrid/lucipy>. It can be installed with the following command:

```
shell@client $> pip install lucipy
```

lucipy itself comes with no (mandatory) dependencies. Extensive documentation is available at <https://anabrid.dev/docs/lucipy>. The code features:

- a simple syntax to setup and edit LUCIDAC circuit configurations on a netlist level.
- Methods to import and export the LUCIDAC circuit to various other (file) formats.
- A *SciPy*-backed simulator/emulator of LUCIDAC based on an idealized mathematical model, suitable for rapid development, debugging and verification of idealized circuits.
- A client interface to steer the LUCIDAC operations, including data acquisition.

Typically, *lucipy* will find the network- or USB-connected LUCIDAC system automatically. If this fails, you have to tell *lucipy* how to reach the system. To do so, *lucipy* used a notation called *endpoints*. These are similar to a VISA lab device connection string. LUCIDAC endpoints are written similar to URLs used in the web. See the *lucipy* manual for further instructions. Section 6.1 contains a list of tips and tricks if you should encounter problems connecting to your LUCIDAC system.

lucipy abstracts only slightly from the underlying hardware. Computing elements are connected manually, guided by the underlying system of differential equations to be solved. This is quite similar to programming a digital computer in assembler. The example circuits shown below demonstrate how it is done. An in-depth explanation of how to map a mathematical problem to an analog computer circuit is outside of the scope of this handbook. For more details on this refer to the references given in section 7.

Section 2

LUCIDAC architecture

Classic analog computers typically featured a large central patch panel consisting of thousands of sockets by means of which computing elements were connected with each other using patch cables. This was very cumbersome, took a long time to manually program, was error prone, and did not allow for rapid program changes. Fortunately with the LUCIDAC system these patch panels are finally relegated to museums where they belong.

Do not confuse the LUCIDAC front panel connectors with the patch panels of classical analog computers. The LUCIDAC front panel serves only for analog and digital input and output (I/O), while the computing elements are interconnected within the enclosure.

2.1 Interconnection network

LUCIDAC provides all-to-all connectivity between up to 16 computing elements. Coupling is implemented by switching matrices under control of the attached digital computer, basically resembling a *crossbar switch*.

Schematically, Figure 2.1 provides a way to visualize the interconnection matrix with real numbers representing coupling strengths. The LUCIDAC system matrix is the adjacency matrix for the interconnection graph between the computing elements. The entries are referred to as *weights*. As a special property, the system matrix does *implicit summing* which can be thought of as matrix multiplication linear algebra operations: $C_i^{\text{in}} = \sum_{j=0}^{16} M_{ij} C_j^{\text{out}}$, $i, j \in [0, 16]$ where C_i^{in} is the input to the i -th computing element, C_i^{out} is its output, and the M_{ij} are the

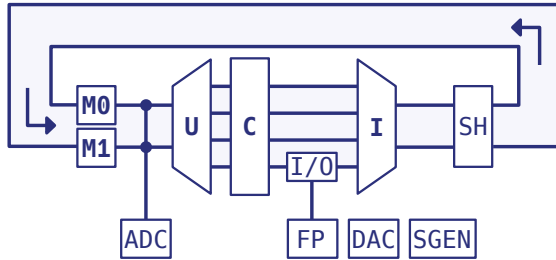


Figure 2.2: The LUCIDAC system as a closed-loop feedback circuit (“turbine diagram”)

Figure 2.2 shows a more detailed block diagram of the LUCIDAC. It shows the closed loop *analog compute path*. The labelled shapes in the diagram each represent a particular *blocks* (in bold letters, referred to as “entities” in the LUCIDAC terminology) and auxiliary *elements* (in normal letters). The figure does not show any digital control/configuration signal paths. Most notably, the **U**-, **C**- and **I**-blocks together form the **UCI** matrix, which corresponds to the simplified *interconnection matrix* shown before in Figure 2.1. In contrast, Figure 2.2 emphasizes the internal *fanout* and *fanin* properties of the U- and I-blocks, resulting in a turbine-like appearance of this circuit representation.

The UCI matrix has 16 inputs, 16 outputs and 32 internal signal paths, each of which contains a coefficient element. These 32 paths are also called *lanes* and correspond to up to 32 nonzero entries in the system matrix, yielding a maximum matrix density of $32/16^2 = 12.5$. This corresponds to a minimum sparsity of 87.5%.

Figure 2.3 shows the most detailed block diagram within this document. Here, the internals of the different blocks are sketched and individual analog data lines are drawn, communicating one voltage or current, respectively. A textual description about the individual blocks is given in the following pages.

LUCIDAC Motherboard (Carrier) Reconfigurable analog computer

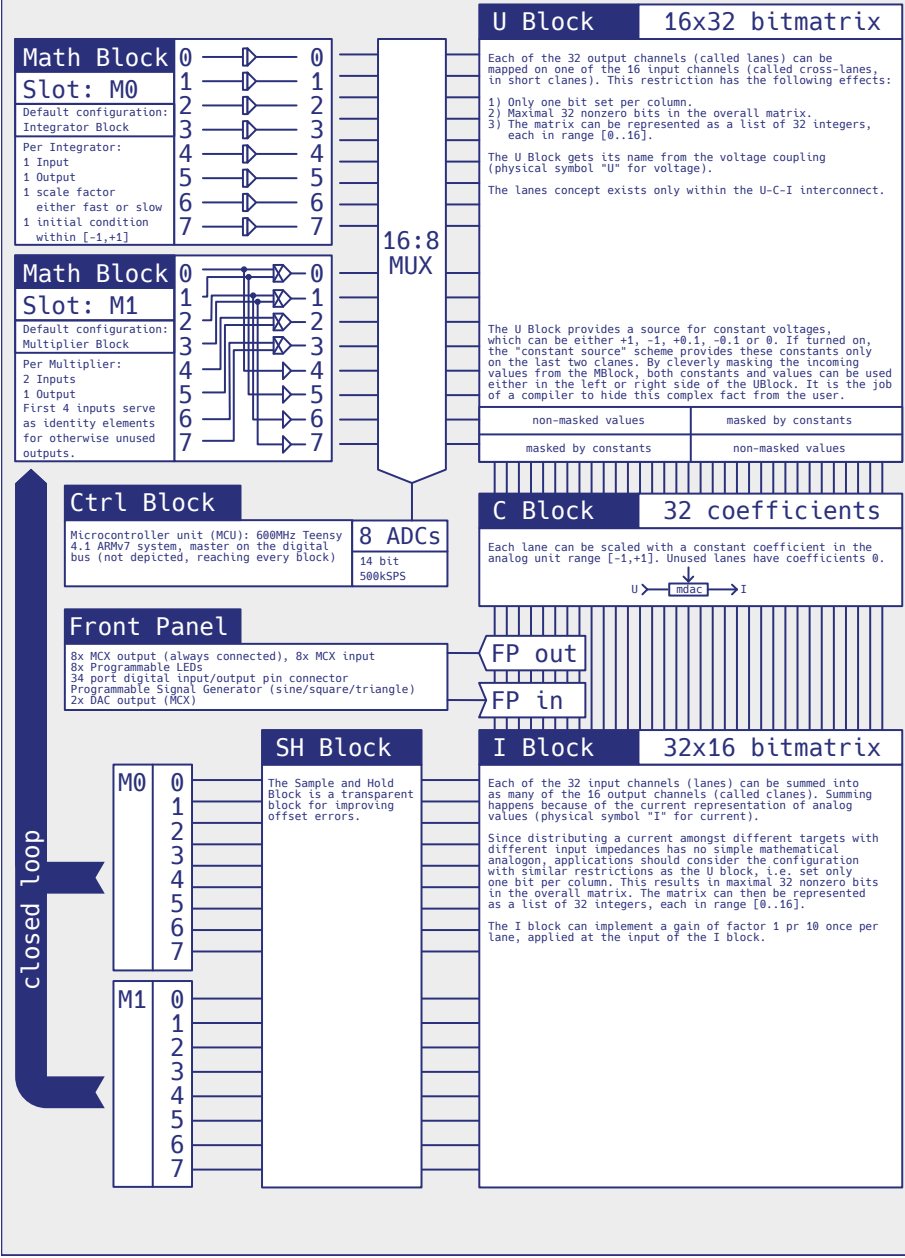


Figure 2.3: LUCIDAC detailed architecture

2.2 LUCIDAC Blocks

M0 and M1 blocks: M-blocks are *math blocks*. Each can have up to 8 analog input and 8 analog output signals. Math blocks contain various elements and allow digital configuration of these elements. In contrast to classic analog computers, there are no summers available as explicit computing elements, as summation is done implicitly in the I-block. This enhances the overall flexibility considerably. For details about the Math blocks, see Section 2.3.

U block: The output signals of these M-blocks are connected to the U-block which contains a 16×32 crossbar switch. Since the output signals of the M-blocks are voltages, this crossbar switch can distribute one signal to several outputs at once. It therefore serves as a 16:32 fan-out, allowing to distribute the input signals arbitrarily on 32 internal lanes within the UCI interconnection matrix.

C block: These 32 output signals (still represented by voltages) are then fed into the *coefficient block*. This contains 32 digitally controlled coefficient potentiometers with 11 bits resolution and an additional sign bit. Thus, the C-Block implements one coefficient per lane by means of a multiplying DAC (a certain type of digital analog converter). These coefficients allow scaling of values with factors in the interval $[-1, 1]$. The output of these coefficients are now currents instead of voltages and feed the I-block.

I block: Following the C-block is the I-block. It, too, is a crossbar switch, this time with 32 inputs and 16 outputs. Working with currents, it is now possible to implicitly sum several inputs, thus eliminating the need for explicit summers as computing elements. The 16 output lines of the I-block are connected to the inputs of the computing elements contained on the M-blocks. The I-block also features a programmable gain of either factor 1 or 10, allowing for a broader dynamical range of the machine.

SH block: This block contains *sample and hold* elements to correct offset errors of the computing elements. It is transparent for the analog signal path and only serves for signal conditioning. It is part of the sophisticated error correction techniques of the LUCIDAC.

CTRL block: This block contains the *hybrid controller* based on the MCU. It is responsible for setting up the computing elements, coefficient potentiometers, crossbar switches. It

also takes care of the communication with the digital upstream computer (the *client* in a networking setting or *host* in a USB setting).

The Control block is not part of the compute path and therefore is not shown in figure 2.2. The following additional *elements* are shown:

ADC: For device-internal data acquisition (DAQ), there are eight 16 bit analog-to-digital converters (ADC). They have access to all 16 M-block outputs by means of a separate 16:8 analog multiplexer.

I/O and FP: The Front panel allows to connect eight analog input and eight output signals. The last eight of the overall 32 lanes are connected to these front panel output jacks. The input signals can be fed into a LUCIDAC setup by means of eight analog switches. In some parts of the documentation these signals are called ACL_IN/OUT (short for Analog Cluster In/Out).

DAC and SGEN: The LUCIDAC also features a signal generator (SGEN) for square/triangle/sine signals with adjustable frequency as well as two digital-to-analog converters (12 to 16bit DAC). Outputs from the SGEN/DACs can be connected to the LUCIDAC computing elements by means of MCX-MCX cables.

2.3 Math blocks

Despite the implicit summing in the networking topology, all linear and nonlinear analog computation is performed on the Math blocks (M-blocks). LUCIDAC math blocks are modular and can be exchanged (although this requires opening the device, see Section 4.7). LUCIDAC has slots for two math blocks. As of now there are two types of M-blocks available and the machine is by default equipped with one of each type:

Math block M0 contains eight integrators (each with one input with implicit summing, one output). Integrators have an internal analog state (the current integration value), a digital state (IC / OP / HALT state machine), and a hybrid state (initial conditions and time scaling factor k_0).

Math block M1 contains four multipliers (each with two implicit summing inputs, one output). The four remaining outputs of this block are used as identity elements. They can be used for more flexibility in the connection topology.

All in all, one LUCIDAC contains eight integrators and four multipliers.

2.4 Circuit Configuration

In LUCIDAC Terminology *circuit configuration* means the set of values required to fully determine the configuration (degrees of freedom) of the blocks mentioned in the previous subsection. Users are not expected to deal with this file format: It is produced automatically by a higher level netlist programmed, for instance, within the lucipy code. Nevertheless, an example circuit configuration is provided for the Lorentz system (section 3.1). The data structure is represented in the popular JSON serialization (see <https://www.json.org>) which is also used in the LUCIDAC networking protocol.

The outer level data structure resembles the hierarchical block structure of LUCIDAC. All configuration options are scoped to the relevant block. LUCIDAC only has a few “global” properties which apply system-wide. An example are the mode lines for steering the computation (IC/OP/HALT), they are not part of the configuration.

```
1  {
2  "/0": {
3  "/U": {
4  "outputs": [
5  0, 2, 0, 1, 1, 0, 0, 8, 1, 9, 2, null,
6  null, null, null, null, null, null, null, null, null, null, null, null,
7  null, null, null, null, null, null, null, null ]
8  },
9  "/C": {
10 "elements": [
11 1, 1, 1,
12 1, -1.0, 1.0,
13 -0.28, -0.5, 0.1,
14 0.5, 0.2666, 0,
15 0, 0, 0,
16 0, 0, 0,
17 0, 0, 0,
18 0, 0, 0,
19 0, 0, 0,
20 0, 0, 0,
21 0, 0 ]
22 },
23 "/I": {
24 "outputs": [
25 [4, 5 ],
26 [6, 7, 8],
27 [9, 10 ],
28 [ ],
29 [ ],
30 [ ],
31 [ ],
32 [ ] ]
}
```

```

33     [0     ],
34     [1     ],
35     [2     ],
36     [3     ],
37     [     ],
38     [     ],
39     [     ],
40     [     ]
41 ],
42 "upscaling": [
43     false, false, false, false, false, false, true , true , false, true ,
44     false, false, false, false, false, false, false, false, false, false,
45     false, false, false, false, false, false, false, false, false, false,
46     false, false
47 ]
48 },
49 "/M0": {
50     "elements": [
51         {"k": 10000, "ic": 0.1},
52         {"k": 10000, "ic": 0  },
53         {"k": 10000, "ic": 0  },
54         {"k": 10000, "ic": 0.0},
55         {"k": 10000, "ic": 0.0},
56         {"k": 10000, "ic": 0.0},
57         {"k": 10000, "ic": 0.0},
58         {"k": 10000, "ic": 0.0}
59     ]
60 },
61 "/M1": {}
62 }
63 }

```

Note that the above listing is intentionally not complete: It does not include all circuit configuration options available but only a minimum relevant for a typical application. For a full list of configuration options, please refer to the firmware documentation (section 5).

2.5 Coupling multiple LUCIDACs together

Generally, digital computers can solve arbitrarily large problems given there is enough time and memory available. In contrast to this, analog computer have to grow linearly with the problem size. The advantage of this is that an analog computer exhibits constant times to solution (see for instance [KÖPPEL et al, 2021] and references therein). Computational circuits in LUCIDACs can grow beyond a single LUCIDAC by making use of the front panel inputs and outputs. This way, a number of LUCIDACs can be connected together, thus forming a larger machine. Such a system is still reconfigurable with respect to its LUCIDAC nodes but not with respect to the connections between these systems which are done using the MCX connectors on the front panel. Connecting several LUCIDACs to form a larger system poses a *synchronization* problem since control of the IC/OP modes as well as ADC acquisition (if applicable) must happen synchronously.

2.5.1 Global mode line and Master/minion mode

LUCIDAC supports several methods for synchronization which meet the timing requirements of its 1 MHz bandwidth computing elements. This section describes one of them; the digital line synchronization in a *master/minion* setup. In this mode of synchronization, one device (master) controls the network's mode while all other systems (minions) are listening by effectively turning their IC/OP lines into inputs (cf. section 5.1 for more information regarding the front panel).

The OP/IC lines can be connected either using the MCX front panel connectors or the digital pin headers of the systems. For more than two LUCIDACs, MCX/BNC T-pieces or something similar are required. Custom headers and flat ribbon cables to simplify this will be provided in the future. The digital ports should be connected when the participating devices are powered off. From the software side, all minion LUCIDACs should receive their proper minion configuration right after startup. With *lucipy*, minion mode is activated with these two lines of python code:

```
minion = LUCIDAC("tcp://hostname-of-minion-lucidac")
minion.manual_mode("minion") # activates minion mode
```

Such a system should always be controlled and triggered via the master LUCIDAC. The minion systems should not be controlled by any other means in such a setup. *Lucipy* provides (at

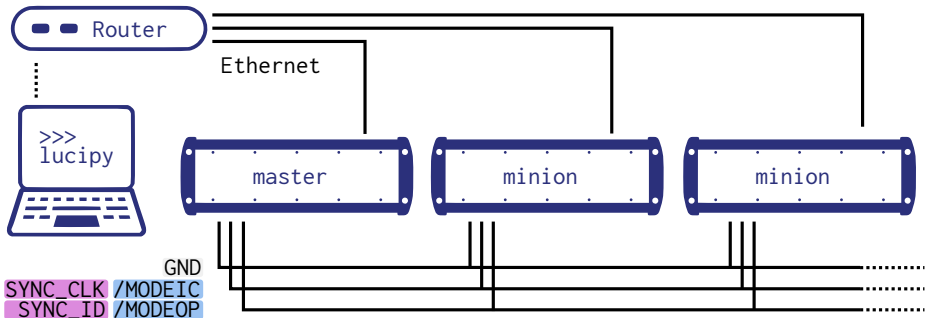


Figure 2.4: Combining multiple LUCIDACs into a larger system requires three digital wires available from the front panel as well as ethernet connections. The analog interconnections (MCX-MCX) are not shown.

time of writing) rudimentary support for such modes with the LUCIGroup class. Note that with master/minion mode, the current version of the firmware does not support data acquisition on the minion nodes.

2.5.2 LUCIDAC group coupling

Another way to synchronize LUCIDAC runs is an custom tailored serial synchronization protocol which is physically exposed on the front panel pin header by means of two lines separate from the OP/IC lines discussed previously. This coupling is incompatible with the master/minion mode described so far. Technically, it allows a set of LUCIDACs to be partitioned dynamically at run time into several groups which are synchronized independently. This is the first step towards programmable connections between LUCIDACs, planned for future members of this product family.

This method, too, requires a “group leader” which triggers the computing run after all followers were set to “armed” mode. This interconnection method is currently favoured in the ongoing development of synchronization mechanisms. It also supports distributed data acquisition with minimal jitter (time spread between sampling points). LUCIDACs ship with tiny headers exposing these two lines in order to make it simple to chain LUCIACs together. However, at the time of writing there is currently no software support for this method.

Note that both coupling variants require three digital lines connected to all systems (IC / OP / GND or SYNC ID / CLK / GND), c.f. Figure 2.4.

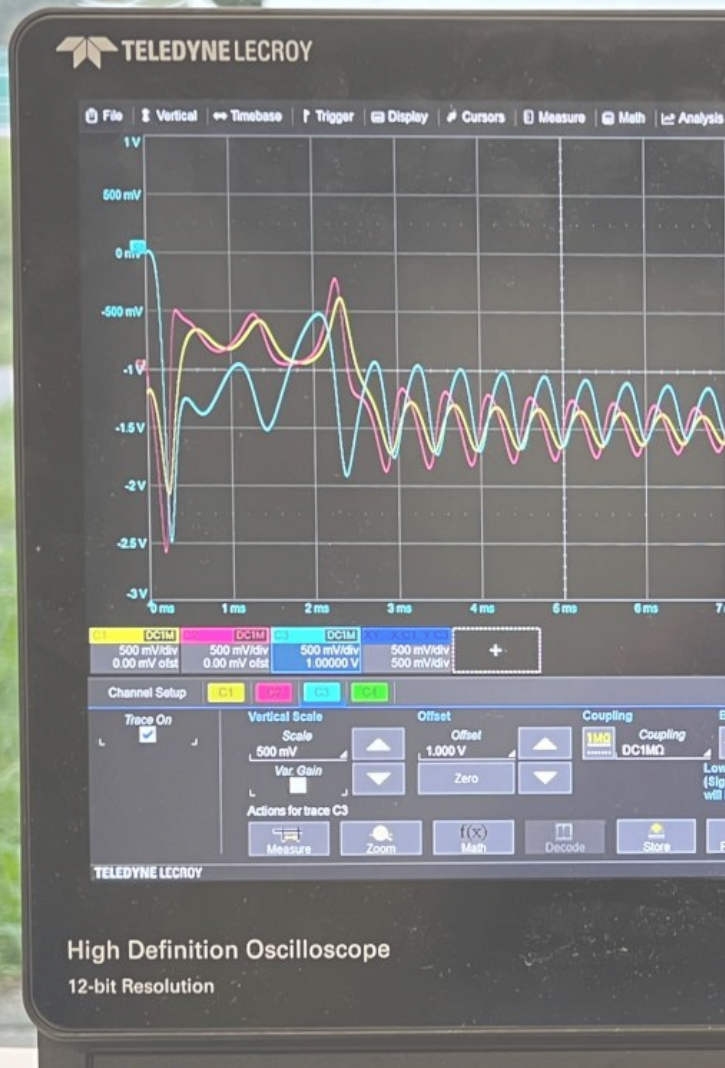
2.6 Coupling THAT with LUCIDAC

The Analog Thing (THAT) is an entry-level classical analog computer with a patch panel (open source/open hardware by anabrid, see <https://the-analog-thing.org> for details). Connecting THAT and LUCIDAC can be useful for some applications which profit from the patch panel and the interactivity provided by THAT. However, keep in mind that THAT computing elements have lower bandwidth and resolution, compared to LUCIDAC.

The master/minion mode is similar to coupling two or more *The Analog Things* (THATs). Connecting a THAT to a LUCIDAC is simple from a digital point of view since the logic levels are compatible. In order to couple a LUCIDAC (digital port) to a THAT (hybrid port), three connections are required (GND, IC and OP). See section 5.1 for the LUCIDAC digital front pinout and for instance <https://anabrid.com/that-hybrid-port-desc.pdf> for a description of the THAT hybrid port.

For connecting analog signals, the four THAT RCA jacks can be used. THAT has up to four analog I/O signals, labelled X, Y, Z and U. The physical connection is easily realized, for instance, with an RCA-BNC adapter and the supplementary BNC-MCX cables of LUCIDAC. Note that LUCIDAC front panel analog signals are directional. Therefore, when interconnecting two LUCIDACs it is obvious that two inputs or two outputs should not be short-circuited. In contrast, on THAT, the direction of the RCA jacks is dictated by the connections made on its patch panel.

Note that THAT RCA jack levels are $\pm 1V$ while LUCIDAC MCX levels are $\pm 2V$. Thus using analog values outside of the interval $[-0.5, 0.5]$ on the LUCIDAC side feeding the THAT will result in erroneous values on the THAT. However, misuse in terms of out of range values will not damage either LUCIDAC or THAT.



There is no need to copy code from this booklet – all example codes shown are also available online in the lucipy repository at the examples folder, i.e. at <https://github.com/anabrid/lucipy/tree/master/examples>. All python filenames given are relative to this directory.



Section 3

Example applications

This section shows example applications for *lucipy* client side programming.

How to run an example

For the sake of brevity, all examples are written without explicit LUCIDAC endpoints. That is, we write `LUCIDAC()` instead of, e.g. `LUCIDAC("tcp://192.168.1.234")`. Conventionally, without a given endpoint *lucipy* will test for the `LUCIDAC_ENDPOINT` environment variable. If not given, it will carry out an auto-detection in order to find LUCIDAC hardware to run the commands on. If there are multiple LUCIDAC systems in a subnet, explicit endpoints will be required to address individual systems. Consult section 6.1 in case of questions how to build up a connection to the system. The simplest way to reproduce the examples on your system is a setup as proposed in figure 1.1 on page 6. Put the oscilloscope in automatic or trigger mode and just execute an example file. The shortest way to do so, without any prior installation, is:

```
shell@client $> git clone https://github.com/anabrid/lucipy.git
shell@client $> cd lucipy/examples/booklet
shell@client $> export PYTHONPATH=../../ # only if you haven't done pip install
shell@client $> export LUCIDAC_ENDPOINT="tcp://192.168.123.123" # if applicable
shell@client $> python lorenz.py
```

3.1 Lorenz attractor

This example shows the *Lorenz system*. This is a set of three coupled first order differential equations for the variables $x(t), y(t), z(t)$. The code presented implements a rescaled version as in https://analogparadigm.com/downloads/alpaca_2.pdf. A circuit is shown in figure 3.1. The system has chaotic behaviour and the (x, y) evolution and phase space computed by lucidac is shown in figure 3.2.

Example listing: **lorenz.py**

```
1  from lucipy import LUCIDAC, Circuit
2
3  a = 1.0
4  b = 2.8
5  c = 2.666 / 10
6
7  l = Circuit()          # Create a circuit
8
9  mx = l.int(ic = 0.1) # Allocate an integrator with initial condition
10 my = l.int()         # two more integrators
11 mz = l.int()
12 xz = l.mul()        # Allocate two multipliers
13 xy = l.mul()
14
15 l.connect(mx, xz.a)  # Product -x * -z = xz
16 l.connect(mz, xz.b)
17
18 l.connect(mx, xy.a)  # Product -x * -y = xy
19 l.connect(my, xy.b)
20
21 l.connect(my, mx, weight = -a)
22 l.connect(mx, mx, weight = a)
23
24 l.connect(mx, my, weight = -b)
25 l.connect(xz, my, weight = -5)
26 l.connect(my, my, weight = 0.1)
27
28 l.connect(xy, mz, weight = 5)
29 l.connect(mz, mz, weight = c)
30
31 l.probe(mx, front_port=0) # Connect a DSO to
32 l.probe(my, front_port=1) # ports 0, 1 or 2
33 l.probe(mz, front_port=2) # to see the output!
34
35 hc = LUCIDAC()
36 hc.set_circuit(l)
37
38 hc.set_op_time(unlimited=True)
39 hc.start_run()
```

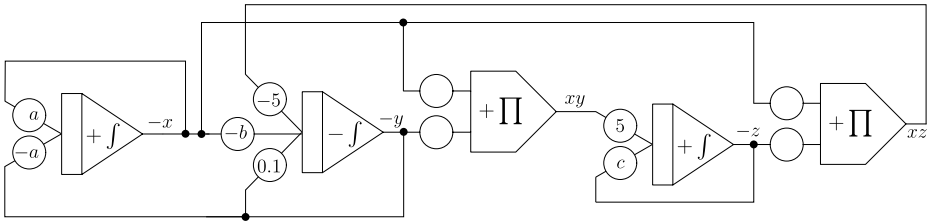


Figure 3.1: Lorenz circuit, empty potentiometers are weight= 1



Figure 3.2: Example display on an attached oscilloscope of the Lorenz attractor

3.2 Rössler attractor

The Rössler attractor is another nonlinear and chaotic ODE example. The code presented here is equivalent to the scaled version presented in https://analogparadigm.com/downloads/alpaca_1.pdf. See figure 3.3 for the output of this program on a digital oscilloscope.

Example listing: **roessler.py**

```
1  from pylab import *
2  from lucipy import Circuit, LUCIDAC
3
4  r = Circuit()                                # Create a circuit
5
6  x    = r.int(ic = .066)
7  my   = r.int()
8  mz   = r.int()
9  prod = r.mul()
10 const = r.const(1)
11
12 r.connect(my,    x, weight = -0.8)
13 r.connect(mz,   x, weight = -2.3)
14
15 r.connect(x,    my, weight = 1.25)
16 r.connect(my,  my, weight = -0.2)
17
18 r.connect(const, mz, weight = 0.005)
19 r.connect(prod, mz, weight = 10)    # We need a weight of 15, split up
20 r.connect(prod, mz, weight = 5)    # into two lanes due to max. coeff = 10
21
22 r.connect(mz,   prod.a, weight = -1)
23 r.connect(x,    prod.b)
24 r.connect(const, prod.b, weight = -0.3796)
25
26 r.probe(x, front_port=5)
27 r.probe(my, front_port=6)
28
29 r.measure(x)
30 r.measure(my)
31
32 hc = LUCIDAC()
33 hc.set_circuit(r)
34
35 hc.run(op_time_unlimited=True)
```

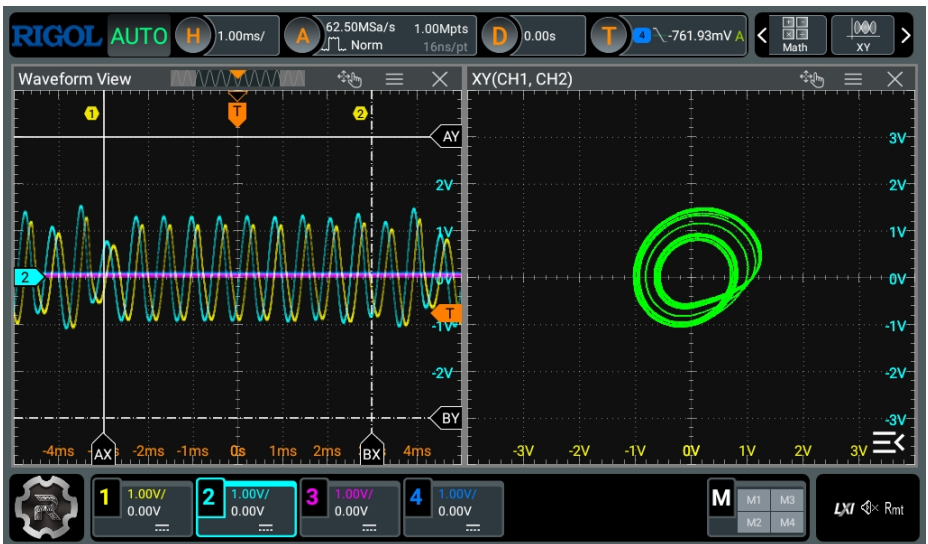



Figure 3.3: Example display of the Roessler attractor (output of previous listing)

3.3 Hindmarsh Rose Neuronal Bursting

This example implements a rather intricate mathematical model of neural bursting and spiking due to Hindmarsh and Rose, see https://analogparadigm.com/downloads/alpaca_28.pdf. The output is shown in figure 3.4.

Example listing: **hindmarsh_rose.py**

```
1 from lucipy import Circuit, LUCIDAC
2
3 hr = Circuit() # Create a circuit
4
5 mx = hr.int(ic = 1)
6 y = hr.int(ic = 1)
7 mz = hr.int(slow = True, ic = -1) # set k0 time scale 100 times slower then others
8 x2 = hr.mul()
9 mx3 = hr.mul()
10 c = hr.const()
11
12 hr.connect(c, mx)
13 hr.connect(mx3, mx, weight = 4)
14 hr.connect(x2, mx, weight = 6)
15 hr.connect(y, mx, weight = 7.5)
16 hr.connect(mz, mx)
17
18 hr.connect(mx, mx3.a)
19 hr.connect(x2, mx3.b)
20
21 hr.connect(mx, x2.a)
22 hr.connect(mx, x2.b)
23
24 hr.connect(x2, y, weight = 1.333)
25 hr.connect(c, y, weight = -0.066)
26 hr.connect(y, y)
27
28 hr.connect(mx, mz, weight = -0.4)
29 hr.connect(c, mz, weight = 0.32)
30 hr.connect(mz, mz, weight = 0.1)
31
32 hr.probe(mx, front_port=5, weight=-1) # switch signs of output channel
33 hr.probe(y, front_port=6)
34 hr.probe(mz, front_port=7, weight=-1)
35
36 hc = LUCIDAC()
37 hc.set_circuit(hr)
38
39 hc.run(op_time_unlimited=True)
```



Figure 3.4: Example display of the Hindmarsh Rose model (output of previous listing)

3.4 Van der Pol oscillator

This example implements a classic amplitude stabilized oscillator first described by VAN DER POL. We also demonstrate how to carry out LUCIDAC-internal data acquisition instead of using an external DSO. See figure 3.5 for the qualitative result.

Example listing: **vdp.py**

```
1  from lucipy import Circuit, LUCIDAC
2  import numpy as np, matplotlib.pyplot as plt
3
4  eta = 4    # tunes the nonlinearity, value 0 results in an harmonic oscillator
5
6  vdp = Circuit()
7
8  mdy = vdp.int()
9  y  = vdp.int(ic = 0.1)
10 y2 = vdp.mul()
11 fb = vdp.mul(2)
12 c  = vdp.const()
13
14 vdp.connect(fb, mdy, weight = -eta)
15 vdp.connect(y,  mdy, weight = -0.5)
16
17 vdp.connect(mdy, y, weight = 2)
18
19 vdp.connect(y, y2.a)
20 vdp.connect(y, y2.b)
21
22 vdp.connect(y2, fb.a, weight = -1)
23 vdp.connect(c,  fb.a, weight = 0.25)
24 vdp.connect(mdy, fb.b)
25
26 vdp.probe(mdy, front_port=5)
27 vdp.probe(y,  front_port=6)
28
29 vdp.measure(mdy) # register variables for internal DAQ
30 vdp.measure(y)  # up to eight variables/channels can be registered
31
32 hc = LUCIDAC()
33 hc.set_circuit(vdp)
34
35 run = hc.run(ic_time_us=200, op_time_ms=6)
36 data = np.array(run.data())
37
38 time = np.linspace(0, 6, num=data.shape[0])
39 plt.title("LUCIDAC-internal_data_aquisition:_Van-der-Pol_oscillator")
40 plt.plot(time, data[:,0], label="-${dot}_y$")
41 plt.plot(time, data[:,1], label="$y$")
42 plt.axhline(0, color="black")
43 plt.xlabel("Time_[ms]")
44 plt.legend()
45 plt.show()
```

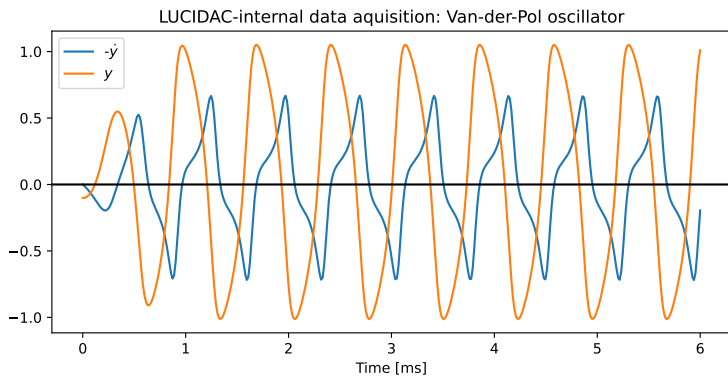


Figure 3.5: Van der Pol oscillator evolution acquired with ADCs, plotted with pyplot

Section 4

Device administration and maintenance

LUCIDAC is primarily meant to be used in a IPv4 network. From an administration point of view, this raises a number of issues, in particular about usage, monitoring, maintenance and access control. These topics are covered in this section.

4.1 The Lucigo administrative code

For IT administrators a dedicated open source administration software called *Lucigo* is provided and can be downloaded from <https://github.com/anabrid/lucigo>. It is meant to be used as a terminal program (standalone CLI tool) and does not require a Python installation. Executables are available for MS Windows, Mac OS X and GNU/Linux at <https://github.com/anabrid/lucigo/releases>. Both Lucigo and Lucipy are clients for LUCIDAC, however lucigo has a focus on device administration while lucipy has a focus on user-level device usage. Both codes respect the LUCIDAC_ENDPOINT environment variable and can autodetect devices via USB or zeroconf multicast calls.

The code snippets shown below assume that the endpoint is suitably set with an environment variable or is autodetected. That means whenever we write `$> lucigo foo bar` this should be substituted by `$> LUCIDAC_ENDPOINT="tcp://120.121.122.123" ./lucigo foo bar` or `$> ./lucigo --endpoint="tcp://120.121.122.123" foo bar`. Note that typically you have to write `$> ./lucigo` instead of `$> lucigo` in case you have the executable in the local directory and not in the system path. Furthermore, depending on what you have downloaded you have to adopt the name of the executable, for instance `$> ./lucigo-amd64-win.exe`.

4.2 Permanent settings and network configuration

LUCIDAC is designed in a way that all circuit configuration is volatile and is intentionally lost at power off/restart. In contrast, there is a small part of device configuration which is permanent. This is referred to as *settings* and primarily covers the device network settings and the user/password database. The permanent device configuration can be easily read with the following command:

```
1 you@host $ ./lucigo get net
2 auth.enable_auth = true
3 auth.enable_users = true
4 auth.users[0].name = "admin"
5 auth.users[0].password = "-redacted-"
6 auth.users[1].name = "user"
7 auth.users[1].password = "-redacted-"
8 net.enable_dhcp = true
9 net.enable_ethernet = true
10 net.enable_jsonl = true
11 net.enable_mdns = true
12 net.hostname = lucidac-16-0A-15
13 net.jsonl_port = 5732
14 net.mac = 04-E9-E5-16-0A-15
15 net.static_dns = 8.8.8.8
16 net.static_gw = 192.168.1.1
17 net.static_ipaddr = 192.168.1.100
18 net.static_netmask = 255.255.255.0
```

Changes can be made with the `lucigo net-set` command. For instance, the following command disables the DHCP client mode and sets a fixed IPv4 address:

```
shell@client $> lucigo net-set ip.local=10.0.0.3 dhcp.active=false
```

Note that changes always apply only after restart, which can be invoked by `lucigo sys-reboot`. Structured data can also be extracted or fed in using the JSON representation from standard input, i.e. `$> lucigo query net-set < input.json`. Thus, the permanent system settings can be dumped to a file and easily rewritten if required.

For further usage patterns of `lucigo`, please run `$> lucigo --help` or refer to the documentation within the code repository at <https://github.com/anabrid/lucigo>.

4.3 Access control

We believe that network-enabled equipment should, by default, be equipped with capable but basic authentication in order to improve IT security and to avoid intentional and unintentional misuse. Authentication also can avoid pitfalls with respect to device identification if multiple devices are available in a network.

LUCIDACs are shipped with a simple application-level authentication/authorization system. By default, there are two users called `user` and `admin` with default passwords provided on the back plate label. It is possible to reset the device to these default passwords. Additional users can be created (the number is only limited by the MCU EEPROM size, which should allow up to 10 user-password combinations).

The authentication scheme in the version 1.0 firmware is very simple and provides a three-step group/authorization structure which allows different protocol method calls for different groups:

1. Anonymous/guest can ask for device identification.
2. Logged in users can use the device (read/write circuit configuration, data acquisition access, run management access, read system logs).
3. Administrator-level users have full access (read/write permanent network configuration, update, device reboot, firmware upgrade, load plugins).

Since anybody with physical access to a device can bypass security barriers only TCP/IP requests require authentication. The USB serial console always provides administrator-level access without a login being required.

The authentication system can be completely disabled by the administrator. This makes device usage more practical in trustworthy networks. With `lucigo`, the relevant command to disable the authentication system is:

```
shell@client $> lucigo net-set auth.enable_auth=false
```

4.4 Device identification

From the vendor point of view, the following information is imprinted onto the microcontroller flash memory: Serial number (a short integer), Serial UUID, Manufacturer name, default user

and admin passwords. This information is assigned by anabrid at fabrication time. Furthermore, the microcontroller holds its one-time-programmed MAC address and USB Serial ID. This information is assigned by the MCU fabrication. All this information is also printed onto the serial plate stickers on the back of the device (cf. figure 1.3 on page 8 as well as the front matter of this booklet). In order to simplify the management of multiple devices in a single network, this information can be queried over network with the `$> lucigo query sys_ident` call.

4.5 System logs and usage metrics

The system holds a volatile (in-memory) log file which is helpful to trace startup problems and usage mistakes. The log is implemented in a ring buffer with limited size. That means if you want to read out the startup log, you should do this as the first operation after startup. The log can be read out with the `$> lucigo query sys_log` call. In some situations, the log also provides useful information as a side-channel where calls have an insufficient error reporting.

The LUCIDAC firmware also locally tracks simple usage counters for evaluation purposes. They can be accessed with the `$> lucigo query sys_stats` call and provide insight of the machine usage since startup.

4.6 Firmware Update

We believe that for the sake of reproducibility of results, predictivity of the device, and availability in critical situations, devices should never autonomously and unannounced run a software update on their own. We also believe devices should never “call home” on their own. The ways of searching for a new firmware image and its installation are presented below.

The embedded microcontroller in the LUCIDAC system runs open sourced firmware available at <http://github.com/anabrid>. Releases with binary builds ready to be flashed on the LUCIDAC MCU are provided there. There are different options for identify the most recent firmware version compatible with the available LUCIDAC hardware. It is sufficient to execute only one of these steps:

- A client code such as lucigo can be used to check and install firmware updates appropriate for the connected LUCIDAC. With lucigo this can be done with the command

```
$> lucigo firmware-upgrade .
```

- You can also visit <http://anabrid.com/product-identification> or scan the QR code on the LUCIDAC back plate in order to visit a website which points you to the most recent firmware version which is compatible with your device along with a guide for the update procedure.

In general, it is recommended to update LUCIDAC regularly to profit from bug fixes, stability improvements and new features. Follow the product identification link provided above in order to register for notifications on new releases.

4.6.1 Manual flashing the MCU

The Teensy MCU cannot be *bricked* since it has an external bootloader. It is also impossible to lock yourself out of the device since the serial connection using the USB port always allows administrator access to the device. If something goes wrong, there is always the option of reflashing the MCU by connecting it via USB and executing the `$> teensy_loader_cli` with appropriate options. For further details and update instructions on the MCU please visit <https://www.pjrc.com/teensy/> or see Section 5 on embedded programming.

4.7 Physical maintenance

Technically, the LUCIDAC device needs *no regular maintenance* (see page 54). The device itself is fully enclosed (it does not need external airflow). Opening the device is *not* part of regular usage and must not be done by untrained personnel. Opening the device without proper ESD protection and gloves can and will damage the hardware!

Although it is easy to open the device by removing the screws holding the front panel in place (the internals can be pulled out on a rail), opening the device will invalidate any warranty claims.

4.8 Testing and calibration

The LUCIDAC device performs self-tests and calibration routines autonomously. This is part of the firmware and happens internally, without user interaction at various times, for instance during startup and before running a computation. Note that it is *not* necessary for the user to calibrate LUCIDAC computing elements. The device comes fully calibrated from the manufacturer. In contrast, certain firmware versions might offer ways to invoke the auto-calibration in

order to achieve accuracy goals. In case of self-test or calibration failures, the device will report these with signalling LEDs and entries in the system log.

In addition to the integrated self-tests, there is a class of test available in various codes. First of all, the firmware repository contains hundreds of *platformio* tests. These are basically stand-alone little firmware images testing particular subsystems of LUCIDAC. At the time of this writing the complete test suite contains over 400 individual tests, c.f. `$> pio test --list-tests`. Running these tests requires an USB connection between host and LUCIDAC. Please refer to the firmware documentation for interpretation of test results and for suggestions of suitable tests in the event of suspected malfunction of the device.

Also, each client code is also equipped with a test suite which typically invokes hardware and integration tests. Client driven tests do not require firmware modifications/flashing and thus can be executed any time in order to verify the functionality of the distributed client/server system. For instance, the *lucipy* code ships with a few dozen tests which can be invoked with `$> make test LUCIDAC_ENDPOINT=...`. Please refer to the *lucipy* documentation for further instructions.

Section 5

Embedded programming

Embedded programming means writing code which runs directly on the MCU of the LUCIDAC. The main reason to do this instead of host based programming is to make use of the real-time capabilities of the MCU, thus allowing the implementation of low-latency analog-digital hybrid algorithms with the following advantages:

1. Saving ethernet or USB latency, which is typically of the order of a few Milliseconds per roundtrip time. This clearly dominates short analog computations which only take a few microseconds to complete.
2. Saving protocol overhead (de-/serialization resources on the MCU). This dominates the digital compute time required for applying circuit configurations.
3. Fine-grained access to the underlying hardware allows advanced techniques inaccessible to the API exposed over the protocol. For instance, custom data acquisition and online postprocessing of data can be done right on the MCU while results can be fed back into the analog circuitry directly.

Additions or changes in the digital communication with respect to the IP networking stack (such as new TCP servers speaking for instance MQTT), protocol extensions to the existing protocol, as well as low-level digital general purpose I/O on the front panel (speaking for instance SPI to another external device) always require modifications in the firmware.

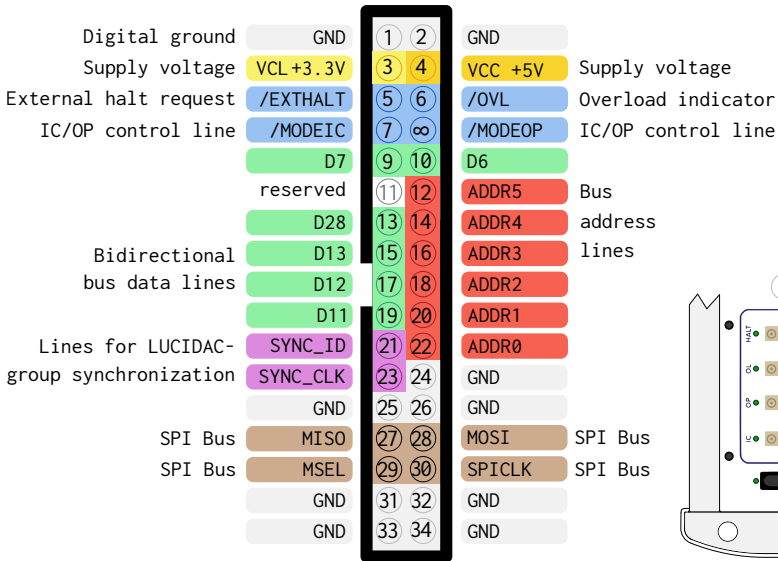


Figure 5.1: Front panel pinout (rotated view from front)

5.1 Front panel digital connector

There is a two pin header on the lower left of the front panel (figure 5.1). In regular use, this port is currently used for inter-LUCIDAC coupling (master/minion mode, cf section 2.5). The front panel digital connector can only be controlled by means of custom firmware code, for instance with a compile time plugin.

Several pins are reserved for future applications. The bidirectional data lines can be used to control external equipment while the six address lines are generated by the built-in control module. They also control the SPI bus. The four IC/OP/OVL/HALT lines are also available on MCX ports. These are primarily used as oscilloscope trigger signals but can in principle also be used for controlling other devices (including LUCIDACs).



Care should be taken when connecting external hardware so as not to damage the system. Digital logic levels are 0 V and 3.3 V.

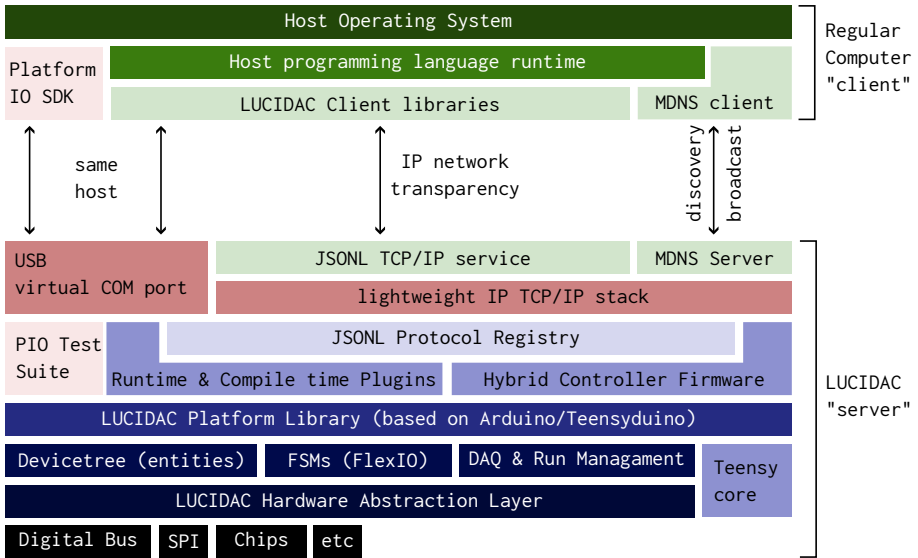


Figure 5.2: LUCIDAC tech stack diagram with focus on communication

5.2 Software architecture and communication interface

The LUCIDAC firmware is open-sourced at <https://github.com/anabrid/lucidac-firmware>. The code is modular/extensible and documented. The documentation is linked in the Github repository. The firmware code is based on Arduino and on the PlatformIO build system (in short *PIO*, see <https://platformio.org>). Please refer to the firmware documentation for getting started with PlatformIO.

Figure 5.2 shows the full digital interface to the LUCIDAC in a layer/tech stack diagram. The individual layers are explained from bottom to top:

1. On the lowest level, the firmware libraries provide a *hardware abstraction layer* to communicate with the various integrated circuits, their data models, and programming functionality.
2. On the next layer, the firmware exposes the *platform library* which collects business logic that implements the blocks as well as administrative infrastructure for hierarchical hardware management, various finite state machines (FSMs), data acquisition, run management, etc.

3. The low level firmware functionality can be accessed as a library “toolbox” where various functions can be used. This allows writing both the main firmware, custom plugins by the user as well as isolated stand-alone tests of particular parts of the software and hardware using the PlatformIO test system.
4. The Remote Procedure Call (RPC) system is based on a JSON protocol registry, which is accessible independently from the transport layer above.
5. Communication takes place either by means of the Arduino USB Serial terminal or the lightweight IP stack (QNEthernet lwIP).
6. The firmware exposes a number of TCP/IP services, such as the JSONL “native” main server or the QNEthernet-integrated MDNS service announcement.
7. At the ethernet client (or USB host) side, various client codes decode the communication, eventually communicating with the host side and application codes.

5.3 System states

The following list provides an overview information about the different states hold by the digital part within the LUCIDAC system. First of all, we differentiate between volatile state (re-set at power cycle) and persistent, non-volatile state (stored in EEPROMs/Flash memories and re-read at startup). The following data are non-volatile: Firmware image, networking configuration/user settings, vendor identification information and finally entity data which can be calibration information. Even a reboot of the machine won't reset the non-volatile information.

The following data are volatile: In general, all analog parts (in particular the reconfigurable interconnection matrix, potentiometer values, initial conditions, overload states), the operating states (IC/OP/HALT state machines), the dynamical runtime ethernet configuration (i.e. DHCP, locking), various software-only subsystem states (such as plugin loaders or ongoing firmware upgrades).

Typically, each subsystem holding volatile information has a reset function on its own. However, a power cycle of the whole system or reboot of the microcontroller will also reset all volatile system states. For further information, please consult the firmware manual.

5.4 Writing a compile time plugin

The firmware is organized as a library “toolbox” exposing several functions. The classical entry point at `src/hybrid_controller.cpp` boils down to the two classical Arduino functions as shown in the following pseudocode:

```
1  #include "lucidac-firmware-libraries.h"
2
3  void setup() {
4      setup_firmware();
5      // This is where your startup code can be placed, which is executed once
6      // at startup.
7  }
8
9  void loop() {
10     loop_firmware_services();
11
12     // This is where any code can be placed which will be executed regularly,
13     // for instance event loops, services, etc.
14 }
```

Typically this or similar patterns are referred to as *compile time plugins*. Rudimentary support for *runtime plugins*, which can load and execute machine code at runtime is provided by the firmware. This is will provide a technological demonstrator basis for tightly-coupled co-processor style hybrid computing. (This is where anabrid is heading towards. For further details, please refer to the firmware documentation.)

5.5 Extending the existing network protocol

The LUCIDAC TCP/IP network model is a JSONL-based command-reply protocol and a simple implementation of a remote procedure call (RPC) scheme. In order to extend the protocol with new calls, some experience with C++ is required. What follows is a demonstration listing which allows to start and stop a front panel LED animation running independently in the main loop:

```
1  #include "lucidac/lucidac.h"
2  #include "protocol/handler.h"
3
4  bool running = false;
5  int i = 0;
6
7  struct MyCustomHandler : public msg::handlers::MessageHandler {
8      int handle(JsonObjectConst msg_in, JsonObject &msg_out) override {
9          LOG_ALWAYS("My_new_request_is_being_called");
10         running = msg_in["running"];
11     }
12 }
```



```

11     }
12 };
13
14 void setup() {
15     // ...
16     LOG_ALWAYS("Registering_my_request_handler...");
17     msg::handlers::Registry::get().set("my_request", new MyCustomHandler());
18 }
19
20 void loop() {
21     // ...
22     auto& lucidac = platform::LUCIDAC::get();
23     lucidac.front_panel->leds.set(i, true);
24     i = (i+1)%8; // advance
25 }

```

From lucipy, invoking this request is as simple as a plain query, for instance in this interactive application:

```

1  from lucipy import LUCIDAC
2  hc = LUCIDAC()
3  while True:
4      # Wait for user input on command line
5      input("Press_enter_to_start_animation_or_CTRL+C_to_stop_program")
6
7      # Turn on the LED animation
8      hc.query("my_request", {"running": True })
9
10     # Wait for user input on command line
11     input("Press_enter_to_stop_animation_or_CTRL+C_to_stop_program")
12
13     # Turn off the LED animation
14     hc.query("my_request", {"running": False })

```

Section 6

Troubleshooting

This section shows some typical problems which could appear and approaches to solve them. If you have fundamental problems with the device which you cannot solve by yourself, please do not hesitate to send a mail to support@anabrid.com.

6.1 Basic connectivity and startup

System remains dead at startup Check if the Power LED at the front panel is on. If it is not, check the power supply and the position of the power switch on the front panel. After power on, some of the IC/OP/OL/... LEDs should also light up, at least briefly. If this does not happen, your MCU or firmware may be corrupted. All eight status LEDs should be lit within three seconds of powering on. If this does not happen, the fundamental LUCIDAC self-check hardware detection has failed. If the eight LEDs do not turn off again, the firmware has got stuck in the networking setup (waiting for an address, etc.)

System cannot be found in the network Make sure the LEDs on the RJ45 network connector at the back of the device are lit, indicating a physical connection. Make sure the system is properly turned on and all LEDs are off after final startup (wait at least 10 to 20 seconds). If possible, check the log of your local DHCP server for an entry containing the hostname "lucidac". Otherwise, make use of a IP network scanner. There are various ones available for all kind of operating systems, even mobile phones. On linux and Mac, you can use the classical <https://nmap.org/> and scan for the typical LUCIDAC TCP port 5024:

```
shell@client $> nmap -p 5024 192.168.1.0/24 -oG - | grep open
```

In this snippet, insert your local IP network instead of 192.168.1.0. If this does not work, connect to the USB terminal and check the output for the allocated IP address, which is shown there. This also works with the client codes itself. For instance `lucigo detect` (see section 4 about `lucigo`) will list USB devices. `Lucipy`, too, can do this, please refer to the `lucipy` documentation for further information.

Cannot connect over USB First of all, make sure the USB connection is working. Use only the enclosed USB-A to C cable, which is certified for USB2. In particular, do not use a USB C-to-C cable, because we experienced issues with such cables. Note that LUCIDAC does not support USB-C Power Delivery. Power to LUCIDAC is provided by the enclosed power supply unit. If you connect LUCIDAC via USB and forget to connect the LUCIDAC to its power supply, the device won't even power up.

Check your USB hub structure. Try connecting the LUCIDAC to other USB hubs or ports on your computer. Make sure the device shows up in your operating system. USB recognizes devices, amongst others, by their vendor ID, product ID and their serial number. These are written on the device back plate. Various operating systems support listing the connected devices (“Device Manager” on Microsoft Windows, “System Information app” in Mac OS X Utilities folder, `lsusb` on Linux). Make sure the device shows up there.

USB Serial Terminal doesn't work On Linux, you need to install the `udev` rules for making sure the virtual serial terminal is registered by the kernel. You find these at <https://www.pjrc.com/teensy/00-teensy.rules> including the installation guide. Also make sure the access rights of the corresponding device file allow your standard user to connect to the USB serial terminal.

In principle, you can use any serial terminal client, such as GNU `screen`, PlatformIO (`pio device manager`) or even the Arduino IDE-integrated terminal program. The terminal speaks the JSONL protocol. That is, you should be able to type in the line

```
1 {"type": "sys_log"}
```

(don't forget to press `enter/newline`) and get some output. Note that all connection details (such as baud rate, stop bit, etc) for the virtual serial terminal (also referred to as

COM port on Windows) are completely ignored in the teensy ecosystem. Typically, the terminal “just works” at full USB2 speed.

Flashing the firmware doesn't work Normally you should just use the self-contained update procedure provided by the firmware itself. However, if you want to flash the firmware explicitly, for instance because you are developing a modified firmware, the following tips might come in handy:

It is a well-known phenomenon that, depending on the operating system and cables used, sometimes the firmware flasher has to be invoked several times. If the

`$> teensy_loader_cli` does not work well for you, you might also want to try out <https://koromix.dev/tytools>. See also https://www.pjrc.com/teensy/check_halfkay.html for the Teensy bootloader.

In rare cases, flashing the firmware only works when pressing the button located on the teensy MCU. Since this button is not accessible from the outside, the system must be opened to get access. Please contact anabrid in such a case.

6.2 Analog programming problems

I can't find a summing element in LUCIDAC programming: The computer performs an implicit sum in the I-block. Accordingly, there are no summers available as explicit computing elements.

System does not compute what was expected: If using lucipy, you might consider to cross-check your circuit configuration using the simulator included in lucipy. This can be as easy as changing the LUCIDAC endpoint to the integrated Emulator, which has the “virtual” endpoint `emu://`. Please refer to the lucipy documentation for further details.

In general note that analog computing is by definition low precision computing. In particular when dealing with highly unstable systems, there can be differences between an idealized simulation and a real analog computation. If you want to dig deeper into these differences, you might want to consider realistic simulations, taking into account transfer functions which model, for instance, the finite bandwidth of the computing elements.

Furthermore, keep in mind that analog computing requires all values to be within a finite domain (the machine unit domain), typically $[-1, +1]$. This system property is called

BIBO (bounded-in, bounded-out) and cannot be violated. Proper scaling of the mathematical equations is required before mapping them onto any analog computer. For more information on how to scale a system of coupled differential equations, see the primers such as [ULMANN, 2023].

Missing signals In order to see what LUCIDAC is computing, you either have to use the internal data acquisition system (DAQ/ADCs) and/or an externally connected oscilloscope. This requires appropriate configuration of the device.

In `lucipy`, this is possible with the `circuit.probe()` and `circuit.measure()` calls to register an external probe or an internal data conversion. Please refer to the `lucipy` documentation for further details and examples. Consider reading the API documentation of the `lucipy.circuits` package or `Circuit` class.

Front panel I/O not working The eight front panel outputs/inputs are mapped to the last eight lanes of the 32 UCI matrix lanes. Counting from 0: Front panel in/out 0 maps to lane 24, front panel in/out 1 maps to lane 25, ... up to front in/out 7 maps to lane 31. Keep in mind that these connections are located between the C and I block. Keep also in mind that for using an input jack, you have to define this as an input, for instance by connecting the corresponding object in `lucipy` to a computing element, which internally triggers an `acl_select` option:

```
1 from lucipy import Circuit
2 circ = Circuit()
3 fp3 = circ.front_panel(3) # front panel I/O 3
4 m = circ.mult()          # multiplier 0
5 circ.connect(fp3, m)     # this connects front panel input 3 to multiplier 0
```

Caution: The front panel connectors are ESD sensitive. Please make sure no excessive voltages are applied. Touch any grounded object before making or breaking connections. Misuse can permanently damage the computing elements, in particular the corresponding lanes (coefficients) and cross lanes (outgoing lanes, SH block, computing elements).

My circuit is too big, I run out of computing elements Keep in mind that analog computing means that every mathematical operation in a set of equations has to be mapped to a physical computing element. LUCIDAC provides eight integrators, four multipliers, 32 coefficients, and a variable number of implicit summers. The interconnect circuits also imposes some limitations onto a configuration (see section 2 for details). If your circuit is

too big (for instance, it requires too many connections), a single LUCIDAC won't be able to solve it. Maybe the circuit can be simplified.

6.3 Frequently asked questions (FAQ)

How to power off the system? Just turn the system off using the power switch at the front panel. This should not be done while writing permanent settings (see Section 4.2) to the flash, which could corrupt the system state.

How to use the system with multiple persons the same time? LUCIDAC allows multiple clients to connect over the network at the same time. This can easily result in problems if two clients want to run different circuits at the same time. Therefore, the device provides a simple exclusive locking mechanism. For further details, please refer to the lucipy client or firmware documentation.

How to distinguish multiple devices in the network? LUCIDACs are primarily meant to be distinguished by their Ethernet MAC address. It is written on the back plate of the device. Furthermore, the firmware exposes an identify call which allows the front panel LEDs to blink and thus identify a particular device. For further details refer to the documentation of the relevant codes.

How to use my favourite programming language? Client libraries for LUCIDAC are mostly ordinary TCP/IP network client applications using JSON for encoding data structures. It is straightforward to write libraries in various programming languages. Clients are already available for C/C++, Rust, Python, Julia, Go, TypeScript, JavaScript and some of them are already released on Github. Contact anabrid if you want to make use of these codes or start a client in a new programming language. It also might be an option to use language bindings and foreign function interfaces instead.

Is there a graphical user interface? We have a web browser-based graphical user interface developed and open-sourced at <https://github.com/anabrid/lucigui>. In the LUCIDAC 1.0.0 firmware, this GUI is not part of the system but will become one in future releases. The GUI simplifies many things such as device configuration/administration, visual circuit programming, mode steering, and many more. Using the GUI also does not require the installation of additional software and enables using the LUCIDAC from mobile devices (in particular tablets) as a *no-code development platform*.

Can I connect peripherals to the LUCIDAC USB port? By default the Teensy MCU within the LUCIDAC does not operate as a USB host but as a USB device. You can modify the firmware to have it operate differently and thereby, for instance, connect a USB keyboard or USB mouse to the LUCIDAC. However, this is beyond the scope of our firmware. Another option connecting digital peripherals to LUCIDAC is the front facing port, which will require suitable firmware code to “drive” the custom additions.

Is there some compiler which automates the translation from a differential equation to the LUCIDAC? A fully-fledged compiler infrastructure is currently being developed which maps mathematical expressions to the LUCIDAC topology. We will provide access to this software as soon as it is ready to use. Users will be able to provide a mathematical system in a domain specific language or in their favourite computer algebra system (CAS). Interoperability with electronics simulators such as *LTSpice* is also planned.

Can I get access to the LUCIDAC schematics? At the time of this writing only the software is open sourced. The schematics are not publicly available. Access can be provided after signing a non-disclosure or license agreement with Anabrid.

LUCIDAC was made possible by the amazing discrete electronics engineering team working in our research lab at the Quantum Computing Initiative (QCI) of the DLR (Deutsches Zentrum für Luft- und Raumfahrt, German Aerospace Agency) in the city of Ulm, Germany.



Section 7

Resources and further reading

This booklet provides only an introduction into the LUCIDAC software and hardware ecosystem. Various resources are available to make your explorations and use of LUCIDAC enjoyable and rewarding. The following web references will help you stay up-to-date with all things LUCIDAC and to get in touch with other members of the analog computing community:

- Anabrid LUCIDAC landing page: <https://anabrid.com/lucidac>
- Anabrid LUCIDAC webshop: <https://shop.anabrid.com>
- Anabrid codes at Github: <https://github.com/anabrid>
- Introductory analog computing circuit examples at <https://anabrid.com/application-notes> and https://the-analog-thing.org/THAT_First_Steps.pdf

For getting started at analog computing in general, you might want to have a look at the following books and papers:

[ULMANN, 2023] BERND ULMANN, *Analog and Hybrid Computer Programming*, 2nd edition, DeGruyter, 2023

[ULMANN, 2023/2] BERND ULMANN, *Analog Computing*, 2nd edition, DeGruyter, 2023

[KÖPPEL et al, 2021] SVEN KÖPPEL, BERND ULMANN, LARS HEIMANN, DIRK KILLAT, *Using analog computers in today's largest computational challenges*, doi:10.5194/ars-19-105-2021 [arxiv:2102.07268]

EU-Konformitätserklärung

gemäß der Richtlinie 2011/65/EU (RoHS) vom 8.07.2011



Nr: AN-4-170000-194873

Hersteller/Bevollmächtigter 1):

anabrid GmbH
Am Stadtpark 3
D-12167 Berlin
E-Mail.: office@anabrid.com

Die alleinige Verantwortung für die Ausstellung dieser Konformitätserklärung trägt der Hersteller (bzw. Installationsbetrieb): anabrid GmbH

Gegenstand der Erklärung: *LUCIDAC - digital programmierbarer Analogcomputer*

Der oben beschriebene Gegenstand der Erklärung erfüllt die Vorschriften der Richtlinie 2011/65/EU des Europäischen Parlaments und des Rates vom 8. Juni 2011 zur Beschränkung der Verwendung bestimmter gefährlicher Stoffe in Elektro- und Elektronikgeräten.

Angewandte harmonisierte Normen insbesondere:

Angewandte sonstige technische Normen und Spezifikationen:

Unterzeichnet für und im Namen von: anabrid GmbH

Ort/Datum der Ausstellung: Berlin / 27.09.2024

Angabe zur Person des Unterzeichners:

Lars Heimann, Geschäftsführer anabrid GmbH
(Name, Position)

Unterschrift:.....



1) „Bevollmächtigter“ ist jede in der Union ansässige natürliche oder juristische Person, die von einem Hersteller schriftlich beauftragt wurde, in seinem Namen bestimmte Aufgaben wahrzunehmen;

LUCIDAC LEGAL DISCLAIMER, SAFETY INSTRUCTIONS, WARRANTY, LIABILITY

1. MANUFACTURER AND CONTACT INFORMATION

Company Name: Anabrid GmbH
Address: Am Stadtpark 3, 12167 Berlin, Germany
Contact Information: Phone: +4930629304720, E-Mail: info@anabrid.com

1.2. PRODUCT IDENTIFICATION

Model Number: LUCIDAC 1.0
Serial Number: Attached to device
Manufacturing Date: Friday Sept 13, 2024

1.3. INTENDED USE

The LUCIDAC is a fully reconfigurable analog computer coprocessor. It is intended for solving complex differential equations, running simulations in physics and engineering, and exploring unconventional computing paradigms. Any use beyond the intended scope (e.g., in non-industrial applications or without appropriate safety precautions) is considered misuse.

2. SAFETY INSTRUCTIONS

2.1. GENERAL SAFETY INFORMATION

This manual complies with ANSI Z535.6 for safety message formatting. Read all instructions thoroughly before installation and operation to ensure compliance with safety regulations.



2.2. SAFETY SIGNAL WORDS:

DANGER: Immediate hazard that will result in serious injury or death.

WARNING: Hazard that could result in serious injury or death.

CAUTION: May result in minor or moderate injury.

2.2 SAFETY SYMBOLS

 Warning: High voltage
 Stop: Emergency power shutoff available

2.3 SAFETY PRECAUTIONS

DANGER: Ensure that all power sources are disconnected before installation.

WARNING: Always use the provided housing to prevent electrical contact.

CAUTION: Only operate the device in an environment with proper ventilation and a stable surface.

2.4 EMERGENCY PROCEDURES

Emergency Stop: Press the power switch located on the front panel. Disconnect all cables.

First Aid: If electrical shock occurs, disconnect power immediately and seek medical assistance.

3. TECHNICAL SPECIFICATIONS

3.1 PERFORMANCE

Power Supply: DC 24V, 1A
Max Operating Pressure: 1030 hPa
Max Operating Temperature: 50° Celsius
Dimensions and Weight: 99 x 357 x 184 mm, weight: 2.3 kg

3.2 MATERIALS AND COATINGS

Body Material: Aluminum chassis (IT 103520 housing by BOPLA GmbH, Germany)
Profiles: Al Mg Si 0.5, die-cast corners: Zinc alloy Z410, seat: TPE.

4. INSTALLATION

4.1 UNPACKING INSTRUCTIONS

* Upon receiving the device, inspect packaging for any damage.

* Ensure the package includes the LUCIDAC main unit, power supply, and accompanying accessories (e.g., RJ45 Ethernet cable, USB-C console cable).

4.2 SITE REQUIREMENTS

Space Requirements: Ensure a minimum clearance of 10 cm on all sides for proper ventilation.

Power Requirements: Standard 240V/120V AC, converted to 24V DC (adapter included).

4.3 INSTALLATION STEPS

Step 1: Place the LUCIDAC unit on a flat, stable surface.

Step 2: Connect the provided power supply to the barrel jack connector.

Step 3: Use the Ethernet cable to connect the LUCIDAC to your network. The device uses a 100Mbit/sec Ethernet interface.

Step 4: If needed, connect the USB-C cable for console access and firmware flashing.

Step 5: Power on the device by connecting it to the electrical outlet.

5. OPERATION

5.1 Control Panel Overview

Power Button: Located at the front.

LED Indicators: 8 user-configurable LEDs, showing system status.

Trigger outputs for I/O/Overload detection.

5.2 OPERATING PROCEDURE

Step 1: Power on the LUCIDAC by connecting the power cable.

Step 2: Access the device through its web-based interface or using the PyAnabrid client on Python.

Step 3: Use the software to configure analog computing elements and run your analog computation.

5.3 SHUTDOWN PROCEDURE

(1) Safely terminate all running computations through the software interface. (2) Disconnect the power supply from the wall socket.

6. MAINTENANCE

6.1 ROUTINE MAINTENANCE

MONTHLY: Check the status LEDs to ensure proper system functionality. Inspect power cables and Ethernet connections for wear.

7. DISPOSAL AND RECYCLING

Dispose of the LUCIDAC in accordance with local electronics recycling regulations. Components such as the aluminum chassis and PCBs should be recycled at appropriate facilities.

APPENDICES

Appendix A: Block Diagram (See main text in document)

Appendix B: Electrical Schematic (Available for licensing or NDA basis upon request).

Appendix C: Warranty Information (Standard 2-year warranty).

APPENDIX C: WARRANTY INFORMATION (STANDARD 2-YEAR WARRANTY) WARRANTY PERIOD

Anabrid GmbH provides a standard 2-year warranty for the LUCIDAC co-processor starting from the date of purchase.

COVERAGE

The warranty covers defects in materials and workmanship under normal use and service conditions. If a defect arises during the warranty period, Anabrid will either repair or replace the defective product at no additional charge.

EXCLUSIONS

This warranty does not cover:

- (1) Damage caused by improper installation, misuse, or unauthorized modifications. (2) Normal wear and tear, cosmetic damage, or accidents. (3) Damage resulting from environmental factors such as excessive heat, humidity, or power surges.

WARRANTY CLAIM PROCESS

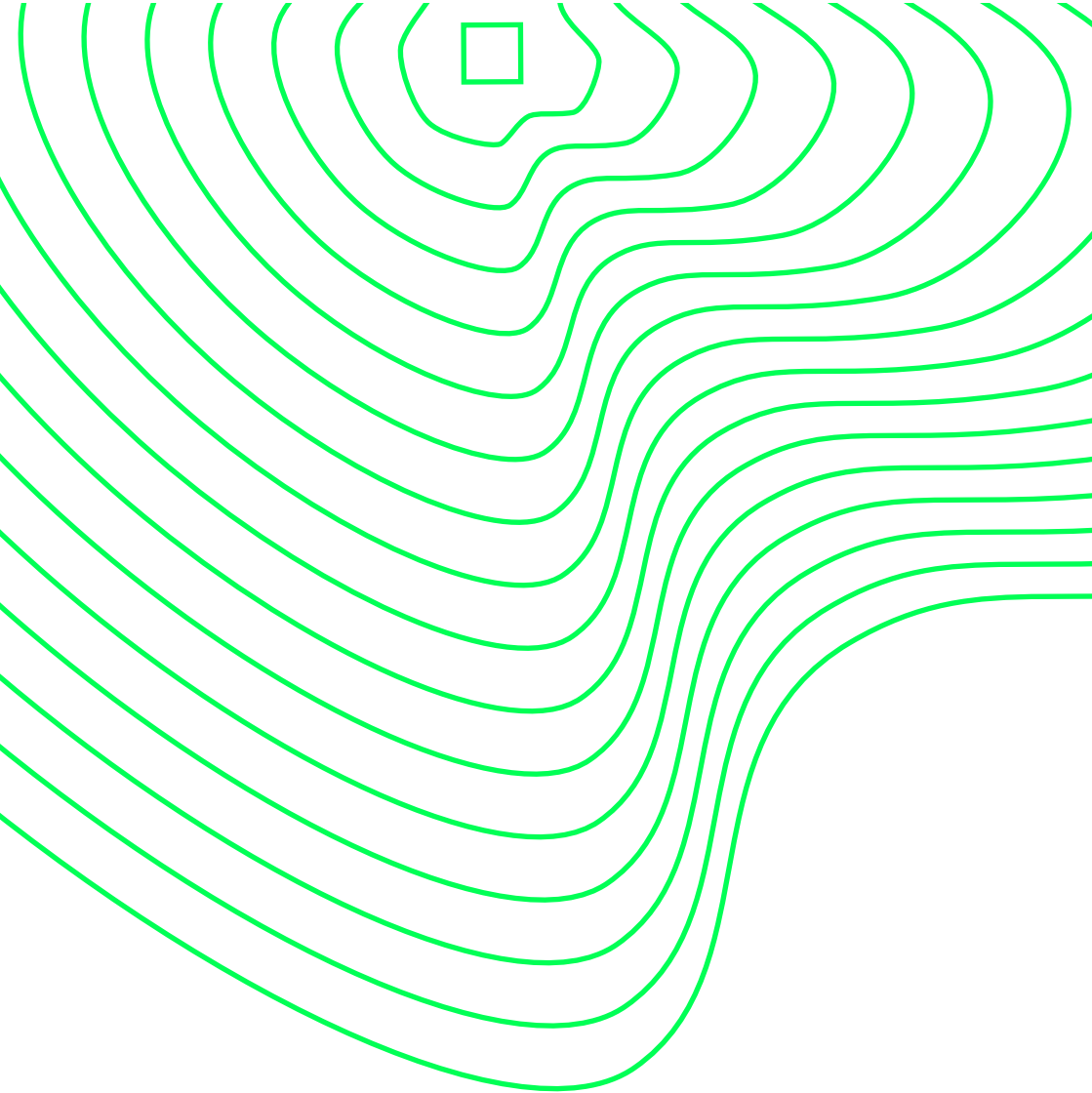
0. Contact Anabrid customer support at hello@anabrid.com with proof of purchase and a detailed description of the issue.

1. Upon approval, you will receive instructions for returning the product for inspection.
2. If the defect is confirmed, a repair or replacement will be provided. Anabrid covers shipping costs for repairs or replacements within the warranty period.

LIMITATION OF LIABILITY

Anabrid's liability is limited to the replacement or repair of the defective product. Under no circumstances shall Anabrid be liable for any indirect, incidental, or consequential damages arising from the use or inability to use the product.

For more details, please refer to the full warranty terms on the Anabrid website or contact customer service.



anabrid