The cic tools

Written by Claude (Anthropic) – ideas and review by Carsten Wulff, carsten@wulff.no

This chapter was written by Claude, Anthropic's AI, from an outline and direction by Carsten Wulff, who reviewed and edited the result. The commit history of the book's repository records precisely who wrote what.

Analog design that cannot be reproduced from a shell command is not done. That is the thesis of this chapter, and the reason the tools in it exist. Every schematic in this course simulates from a script, every testbench sweeps its corners from a one-line command, and one of the ADCs in the converter chapter compiles its own layout. None of that needs a licence server.

The tools share a prefix - cic, for Custom IC Creator - and a philosophy: plain text in, plain text out, and git in between. Each one is small. Together they are a design flow.

CICSIM - SIMULATIONS YOU CAN RERUN

`cicsim` is the workhorse: a script package that controls ngspice. You met it in the tutorial when `cicsim simcell` built the simulation directory for the current mirror. What it buys you over running ngspice by hand:

`cicsim run` takes a testbench and a corner specification - typical, slow, fast, high and low temperature, supply corners - and runs the cross product, in parallel if you ask (`--threads`), with a progress bar and a timeout to kill the simulation that wedged. Add `--count N` and each corner runs N times with mismatch - Monte Carlo from the same command line.

The measurements come back as a results table, one row per corner, which is the artifact that matters: the OTA chapter's advice to verify gain, noise, slew and start-up *over PVT* is one `cicsim run` per testbench, and the table is the evidence.

The yaml files that define a simulation live next to the schematic and go into git. Six months later, `cicsim run` again and you get the same table - or a diff that tells you exactly what the PDK update broke.

- `cicsim run` : corners, in parallel, with a results table
- `--count N` : Monte Carlo from the same command
- yaml + git : the simulation is reproducible, or it is not done

CICCREATOR - THE LAYOUT COMPILER

`ciccreator` is the tool behind the compiled SAR ADC in the converter chapter. The idea dates to 2013: describe the circuit as a SPICE netlist, describe the layout intent as a JSON object definition, add a technology rule file, and let a compiler place the polygons. The prototype was 16 thousand lines of Perl; the current tool is a C++ rewrite of the same input language, fast enough to recompile an ADC while you watch.

The point is portability. The netlist and the object file do not mention a technology; the rule file does. Porting the ADC to a new process - and it has been ported to 22, 28, 55, 65, 130 and 180 nm - means writing a new rule file, not redrawing a layout. That is why the comparison table in the converter chapter has a "compiled" line with a competitive figure of merit, and why the same ADC exists as an open source SKY130 implementation you can read.

- SPICE netlist + JSON object definition + technology rules = layout
- Port to a new process: new rule file, not new polygons
- The compiled SAR ADC of the converter chapter is the proof

CICPY AND CICCONF - THE GLUE

`cicpy` translates. It takes ciccreator output and transpiles it to whatever the rest of your flow speaks: SKILL for Cadence, SPICE, Verilog, Xschem schematics, Magic layout, or SVG when you just want to look at a cell. It also carries two small workhorses - `sch2mag` and `spi2mag` - that netlist a schematic and place-and-route it to Magic, which is how the standard-cell-like blocks in the aicex IPs appear.

`cicconf` manages the projects themselves. An aicex IP is a git repository, and a chip is many of them; `cicconf` keeps the dependency list in one `config.yaml` and clones, updates and templates the collection. When the tutorial told you to run `cicconf clone`, this is what stitched your project together.

- cicpy: transpile the compiled IC to SKILL, SPICE, Verilog, Xschem, Magic or SVG
- cicconf: one config.yaml for a project of many git repositories

CICWAVE - LOOKING AT THE WAVES

`cicwave` is the waveform viewer: it reads ngspice raw files and draws them with a PyQtGraph/Qt6 backend fast enough for long transients. It is the third viewer of its line - the first was written during a summer internship in 2001 - and it exists for the same reason as the rest of the family: the open source flow deserves a viewer that starts instantly, works on every OS, and is scriptable.

CICTIKZ - THE FIGURES OF THIS BOOK

The newest member drew the book you are reading. `cictikz` packages the TikZ symbol library behind every schematic figure in these chapters, gives an AI assistant a render-and-look loop so figures can be drawn and reviewed programmatically, and converts between the book's TikZ dialect and Xschem

schematics - so a figure can start life as a real, simulated schematic and end as a book drawing, or the other way around.

- One symbol library for every schematic in the book
- TikZ to Xschem and back: figures that are also schematics

SUMMARY

The one-page version of this chapter:

- If it does not rerun from a shell command, it is not done
- cicsim: corners and Monte Carlo as one command, results as a table, all of it in git
- ciccreator: netlist + object definition + rules compile to layout; porting is a rule file
- cicpy transpiles to the flow you have; cicconf holds multi-repo projects together
- cicwave views the waves; cictikz draws the book
- None of it needs a licence server

WOULD YOU LIKE TO KNOW MORE?

Every tool lives on GitHub with its own documentation: cicsim, ciccreator (docs), cicpy, cicconf, cicwave (docs) and cictikz (docs). The IPs built with them are collected in aicex.

- github.com/wulffern/aicex - the IPs built with these tools

Carsten Wulff received the M.Sc. and Ph.D. degrees in electrical engineering from the Department of Electronics and Telecommunication, Norwegian University of Science and Technology (NTNU), in 2002 and 2008, respectively. During his Ph.D. work at NTNU, he worked on open-loop sigma-delta modulators and analog-to-digital converters in nanoscale CMOS technologies. In 2006-2007, he was a Visiting Researcher with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada. Since 2008 he's been with Nordic Semiconductor in various roles, from analog designer, to Wireless Group Manager, to currently Principle IC Scientist. From 2014-2017 he did a part time Post.Doc focusing on compiled, ultra low power, SAR ADCs in nanoscale technologies. He's also an Adjunct Associate Professor at NTNU. His present research interests includes analog and mixed-signal CMOS design, design of high-efficiency analog-to-digital converters and low-power wireless transceivers. He is the developer of Custom IC Compiler, a general purpose integrated circuit compiler, and makes the occational video on analog integrated circuits at https://www.youtube.com/@analogicus. For full CV see https://analogicus.com/markdown-cv/.