intro

TODO:

* consuldate
* get consistinat with "we", "you", "I"
* add images and flow diagrams
* fix the markdown headers to work with jupyter book
* figuer out jupyter book

# Welcome to the book “Python & SPICE”[¶](#Welcome-to-the-book-“Python-&-SPICE”)

This work represents an attempt at dragging SPICE from a niche tool of analog circuit designers to a standard part of the workflow of anyone practicing electronic design. And by electronic design that doesn’t just mean for this select few doing custom silicon, it means anyone who is designing any electronics from discrete RLC circuits in class to analog PCBs, and yes those do custom silicon. What is different about this SPICE book from all the other academic SPICE books written for SPICE/ PSPICE/ Spectra/ect.? SPICE netlist are going to be generated and manipulated in Python and SPICE (specifically Ngspice) is going to be controlled via a Python extraction layer. Thus we never have to leave python to do any SPICE simulation and with every part of the process of interacting with SPICE in python, we can take advantage of python’s clean programming language and full-stack to scientific tools to take SPICE into the 21st century. And did I mention that every tool is FREE!

This wouldn’t be possible without the following specific works that allow us to Python & SPICE with ease:

* Dave Vandenbout creator of [SKiDl](https://github.com/xesscorp/skidl)
* Fabrice Salvaire creator of [PySpice](https://github.com/FabriceSalvaire/PySpice)
* The [ngspice development team](http://ngspice.sourceforge.net/devel.html)

In addition to the primary pipeline code, this book also uses the work of

* Michael Hayes creator of [Lcapy](https://github.com/mph-/lcapy) which allows us to create schematics directly in jupyter notebooks.

In addition, this book uses a lot of examples and follows the organization of one of the best YT channels on the analytical analysis of Electronics

* Rushal Shah creator of the YT channel [ALL ABOUT ELECTRONICS](https://www.youtube.com/c/ALLABOUTELECTRONICS/featured)

## What is this book going to do/and not do and why you should keep reading:[¶](#What-is-this-book-going-to-do/and-not-do-and-why-you-should-keep-reading:)

This book is not a guide to circuit and electronics theory and doing all the analytical math that comes with said theory. For that reason, I have included a video from the YT channel ALL ABOUT ELECTRONICS who makes excellent circuit and electronic theory YT videos to consult and where I will be templating a majority of this book from. This is also not a guide to python, so you should already have familiarity with python to the point you can write your own classes and deploy basic Object Oriented Programing.

What this book is going to show you is how to generate or import a SPICE netlist with python via SKiDl and manipulate said netlist. I cant, but I will in more detail shortly, stress how big a deal this is. Since one of the biggest frustrations of using SPICE is dealing with netlist. Even if you are using schematic capture tools to generate your netlist. Dealing with netlist is by far the most exasperating part of SPICE. This book is also going to be a dive into what SPICE can do. Trust me when I say this, there is a 99% chance you have barely put SPICE to work. And at the same time, I am going to point out the lacking of SPICE. And finely this book is going to show you how to maximize what SPICE can do via wrapping it with python to make it easier to use and to analyze the data from SPICE like it, in all respect to EDA tool creators, has never been done before by leveraging the Python Scientific, Data Science, and Machine Learning ecosystems.

### In short:[¶](#In-short:)

* If you’re a student trying to learn circuits and electronics this book is going to show you how to quickly create your circuit understudy and get the most of the values you need lickety-split
* If you’re an educator this books is going to show you how to create interactive simulations
* If you're a practicing designer this book is going to show you how to leverage SPICE with Python to not just simulate things by verify them as much as SPICE can do
* If you’re in test this book is going to show you how to simulate a DUT (to a reasonable degree) like the DUT was on the bench to do give you a starting point as a sanity check

## The most important thing this book is going to do: Revolutionize Analog/mixed single design verification.[¶](#The-most-important-thing-this-book-is-going-to-do:-Revolutionize-Analog/mixed-single-design-verification.)

So I learned electronics in Fort Collins Colorado, a town built on Hewlett-Packard (HP), and even worked for a time at a legacy VLSI group from the HP days. So I am going to use that terminology. And in the VLSI group I worked in I worked with a great Design Verification Group who was technically headed by an author of one of the first books are assertion-based design in HDL. Why am I bringing that up, because I never heard about a problem in the digital logic parts of what we produced, and that was because the Design Verification group was a Digital Design Verification group and I am going to leave the analog and mixed-signal verification in that VLSI group to the readers own imagination.

So I am using HP’s terminology for the design and release process, but no matter where you are there are always two major steps to releasing a product (or at least you hope there are). Where HP called those Phases 1 and 2 and both have numerous sup-phases. Phase 1 is the design stage, this is when HDL is written, put through design verification, RTL’ed, and placed (this is for both custom silicon and FPGA work) on the digital side of things. And in analog/mixed-signal sides this is when circuits are down up, simulations are done, and things are laid out and routed for custom silicon or placed and routed for PCBs while working in tandem with Signal Integrity folks for the important interconnects. After this and after a whole lot of going back and forth with the manufactures till everyone agrees there's Tapeout (Tapeout refers to sending literal stacks of magnetic tape storage to the faps for ASICs creation back in the day) or Gerbered out (Gerber is the standard PCB place and route file format) and thus there's no going back and begins Phase 2. Phase 2 is where you get to verify what was Taped/ Gerbered out. It is when you take what you design and you put it through rigorous testing (hopefully) via stupidly expensive test and measurement equipment and real-world like conditions to find out what you shouldn’t have released from Phase 1. The outcome of Phase 2 is you either release with (again hopefully) realistic scaling of what your product can or can't do, or you don’t release. If you have the resources you do what is called a Phase 1 spin either to get a release or as an updated version. Either way, your back to Phase 1 to fix or improve what should be manufactured and still having to redo Phase 2. And the whole exercise is expensive, especially if you have to spin.

Notice something when I was talking about Phase 1, when talking about digital I said it’s “put through design verification” and when I talked about analog/mixed-signal I said, “simulations are done”. It’s a subtle but big difference. After all, any verification is going to included simulation. This difference is that verification is a systematic and rigorous set of simulations to verify functionality or lack thereof. And since the creation of HDL the digital logic verification folks have had it easy and it’s only been getting easier. On the other hand, pure analog simulation has always been hard and that’s ignoring E&M solvers that are even harder but will not be discussed here. And mixed-signal has seen some improvement with things such abstraction of analog blocks via real number modeling, AMI, etc. But the fact of the matter is that things like real number models must be abstracted from analog simulations.

So what is the big deal everything since everything has gone digital anyway and we just use DSP and ADCs and DACs; WRONG! First off, your ADC/DAC is a mixed-signal system and are predominantly analog. And secondly, digital devices are islands in a sea of analog. Think about it, unless you are using 100% verified IP or fully built FPGA boards that have all the analog already sorted out from the power, clocks, and IO your logic is at the mercy of the analog. Whether that be your power distribution system, your clocks, and or your low and or high speed interconnects; it all analog and if the analog isn’t up to snuff you digital now matter how well it’s been verified it’s useless.

So why do the digital logic folk, expressly Verilog, have it so easy and the analog folks don’t? The obvious reason is that you're hitting code with code. In HDL you write a logical or behavioral expression of what you want your logical system to do in code. From there to simulate you just instantiate that chunk of code in another chunk of code that contains your test stimuli vectors to see what happens. Furthermore, with Verilog and assertions, you can not only throw test vectors at the modules under test but also use powerful abstraction verification tools like formal verification that will also check the system when certain conditions are met. And then when all the verification is done you pass the module under test to an RTL synthesizer which then creates the hardware (FPGAs are a bit different but the same idea) from standard cells and there’s the silicon. In fact, there are quite a few commercial and open source tools that will do the formal varication and automatic test vector testing like SymbiYosys and the now being phased out Cadence’s Specman. And note here this is a high-level overview for contrast to analog/ mixed-signal. It's a lot more involved with its own plights than I am letting on here.

But in analog, we really have not had that kind of ease of writing code and smacking it with code to ensure the code works. And sure there are things like Verilog-AMI but that is very specialty stuff. The reality is that someone draws up a circuit in a schematic capture tool. And then if all the elements are there to do the SPICE simulation they have to export the netlist to the SPICE engine and run the simulation. Let alone then grab the data from the simulation and analyze it. If not all the elements are there to do the simulation, then the schematic has to be modified to add those elements, and then after the simulation, those extra elements need to be removed. Going back to the data from the simulation. Yes most SPICE engines have some basic math like FFTs but compared to Python it’s a joke and getting the data from your SPICE engine into a programming environment even like Matlab is not trivial and often evolves a lot of intermediate data steps like databases, excels, and CSVs. There are industry solutions (yeah right) to this problem. The first is interactive SPICE controls via what is typically called probs. Some good examples of these are ANSYS’s Electronics Desktop and NI’s Multisim, but these again add elements to the schematic that need to be removed before place and route. The second solution is typically a programming route using the EDA vendor's built-in API with such horrors as TCL, LISP, SKILL, OCEAN, and even Matlab some rare times. But those are all mostly horrible out of data languages and every one of them has to be used exactly with its respective tool. And therefore, requires teams of just tool builders. Finally, there is the issue of SPICE is just slow. And thus, things like real-world modeling and AMI modeling were developed to create faster emulation objects that then work within the relatively fast Verilog/VHDL simulations. But the problem is that those models have to be extracted from the actual circuit that there emulating and if there are any changes to the circuit while, back to square one and to the problem of setting up those SPICE simulations. And all and all this is why unless you have the team and resources to do it Analog is simulated and not verified because to do verification like the digital logic folk is an astronomic cost and thus the industry has accepted that tradeoff for a very long time. Though things are changing.

So what is going to do about is and improve the discipline of Analog Verification.

1. Where going to use SKiDl to enclose the fundamental building blocks of our circuits. Wither this is directly coded circuits in SKiDl or netlist capture from a schematic tool. Where even with terminal changes SKiDl gives us a way to manipulate the netlist programmatically with a grace that has not really been shown before with SPICE tools. Further, we can create testbench items with SKiDl and the other tools to very easily slap onto our DUT without altering the DUT’s base.
2. Where going to develop wrappers around SPICE calls, even though pyspice, to make performing a simulation trivial
3. Where going to leverage the whole of the python scientific/data ecosystem to analyze the data to verify what is going on.
4. The testbenchs and analysis code we write will try there best to emulate the measurements that would be done in the lab to a physical DUT and thus when we get data from the lab we can do a much more direct comparison to figure out what happened between phase 1 and phase 2 with unparalleled grace to what has been done
5. And we will show how to extract high-level models using traditional real-world and AMI tools along with machine learning tools that Python is light years ahead of other traditional scientific programming languages of