University of California, Santa Cruz
Electrical Engineering Department

The PCB EDA Tool Chain

EE174, Intro to EDA Tools
S.C. Petersen

1. Introduction

Electronic Design Automation , or “EDA”, refers to a set of powerful, complex, computer programs used to facilitate the design and physical realization of electronic circuits. Typically, this can be on a chip or printed circuit board (PCB), and involves layout, verification and performance simulation of electronic circuits. In the 1950s and 1960s, circuits were drawn by hand on drafting boards. By the 1970’s this was supplanted by computer-aided design (CAD) programs used to create circuit schematics, along with computer-aided engineering (CAE) programs for analysis and simulation. In the mid-1980s, all CAD and CAE tools for electronic circuits eventually coalesced into the current broad term "electronic design automation," where automation suggests the whimsical idea of some fixed-sequence of steps achievable by simply standing back and “turning the crank.” Programs associated with EDA tasks are known as *tools* and the sequence of integrated inter-dependent tools employed to realize a particular design is simply called a *tool chain* or *workflow*. Also, as designs became larger and more complex, especially with digital circuits, hardware description languages (HDL) were invented as scalable alternatives to traditional graphic schematic representations.

It follows that custom workflows exist for a variety of common tasks in the electronics industry today. Most can be broadly categorized cleanly into two general parent workflows. The first and most prevalent, making integrated circuits (IC’s), is usually known as “chip-level” design. The second, “board-level” design, means that the electronics end up implemented on discrete printed circuit boards (PCB) rather than monolithic IC’s, where *discrete* simply means “not an IC”. Hence, board-level circuits are composed of many physically distinct inter-wired, typically soldered, devices, some of which might be and usually are already finished and completely encapsulated IC’s. Within these two major workflows are many tributaries, each devoted to specific variations. For example, digital IC’s are usually designed using hardware description languages, hence chip-level workflows with HDL require special tools capable of translating these machine-readable desriptions into some format suitable for physical IC implementation. On the other hand, small analog or RF designs might be expressed using traditional graphic schematics, and so require a different translation tool. It is customary today to encapsulate large digital designs in HDL form, since the objective of true automation is more closely realized.

Generally, both of these workflows can be roughly divided into two temporally distinct phases, *front-end* and *back-end*. Tools associated with front-end tasks are collectively devoted to documenting, or “capturing”, electronic designs. Hence hardware description languages and graphical schematic editors would both be classified as front-end tools. The back-end phase involves everything else necessary to actually implement the resulting hardware. This is often called the “packaging” phase for chip-level circuits or the “layout” phase for board-level designs. Note the term *layout* is just as often applied to both, so one must take care to understand the context of usage; laying out chips is quite different than laying out PCB’s. This nice sequential distinction is complicated when we bring in the many CAE tools devoted to analyzing, simulating, and therefore predicting circuit behavior. I like to classify these with either the front-end or back-end phases, but in fact, some of the “front-end” tools require knowledge of the back-end geometry and other physical considerations before they can be used effectively. Today, the complexity and differentiation between different vendor’s tool chains is remarkably dense and opaque, with many acronyms referring to aspects of particular design flows (this point is explored in the homework). Don’t be daunted. If you take the time to carefully work through the thickets of imprecise vague verbiage and their acronyms, translating much too abstract terms (for example like SDL, an acronym for “system design level”) by actually reviewing and learning what a tool chain assigned to this descriptive term actually *does*, then you will have elucidated it.

2. Course Perspective and Background

This course introduces the board-level EDA tool chain required to design and produce high-quality electronic printed circuit boards through a guided series of tutorial laboratories presenting the necessary practical skills associated with two popular EDA *tools* written by Cadence Corp.: **Allegro Design Entry CIS** (front-end) and **Allegro PCB Editor** (back-end)**.** The front-end tool was originally created by Orcad Corp. and later acquired by Cadence. When Cadence picked up Orcad around 2001, they acquired two major tools: Orcad Capture CIS, used to enter (and hence "capture") engineering design schematics in graphical form, and Orcad Layout Plus, used to design and generate related PCB files based on schematic designs first captured and compiled in Orcad Capture CIS. Cadence immediately began merging them into their own wider set of more powerful tools. During the last nine years, these two Orcad programs were renamed a number of times as each new version release came out. This foster family approach to code assimilation was confusing and often appeared silly when the ghost of Orcad's presence refused to be exorcised (it still appears in several of the help files within Design Entry CIS). Cadence was able to integrate the front-end tool nicely into their toolchains, since it greatly improved and complemented their clunky, unfriendly, closest equivalent: Allegro Design Entry HDL. In 2007, Cadence announced that Layout Plus would be discontinued, began a 2-year phaseout, and encouraged the large base of Layout users to migrate to their more powerful backend tool: Allegro PCB (in its multitudinous masquerades). Many engineers (me included) demurred. I will explain why shortly. However, before proceeding, throughout these labs we will agree to use "Capture" as synonymous with "Allegro Design Entry CIS", after its popular and still prevalent genetic Orcad name. I will also use "Allegro" as a single word to always refer to Allegro PCB Editor, even though Cadence has several other tools prefaced with "Allegro". In those particular cases I will write out the entire name, as in say, "Allegro Package Designer." And whenever I refer to "Layout" (with a capital "L") I will also mean the original Orcad tool, Layout Plus.

Our objective is to learn to competently document existing electronic designs by first capturing them as complete self-documenting engineering schematics, then proceeding to lay them out on a PCB. Since the inception of this course in Fall, 2005, the EDA flow for board-level PCB design was introduced sequentially by first discussing Capture followed by Layout. Although Allegro PCB Design (albeit under different names) has been around for many years, we considered it inappropriate for an introductory course for a variety of reasons and opted to prudently stay with Layout. This proved to be very successful, but faced with having to abandon it, the question about whether we should move to Allegro or adopt an entirely new tool chain needed to be addressed. From my viewpoint, it appears that Cadence acquired Orcad primarily because they wanted Capture CIS with its excellent hierarchical architecture, powerful database management features, and integrated graphical symbol editor. Moreover, Layout had always been moderately priced as the industry-standard "poor engineer's tool" that was an excellent choice for professional-level manual layout chores, but it had a poor autorouter and no signal-integrity or high-speed digital design and analysis capabilities. Allegro, by contrast, is a high-end tool designed to equally handle manual and automatic layout of large many-layered PCB designs, and it has integrated complex signal-integity features applicable to high-speed digital projects. These features are well beyond the scope of this introductory course. Since Capture remains a versatile, powerful tool capable of providing compiled designs targeted for a variety of back-end tools, including Layout and Allegro, I decided to retain Capture and carefully review the issues involved with migrating only from Layout to one of the many so-called "tiered" versions of Allegro. I finally decided to move to Allegro to replace Layout. This has the primary advantage of continuity for those students here at UCSC that took the earlier classes and may have gone on to graduate school, skillfully know the Orcad Capture-Layout combination, by retaining their present skills with the front-end tool, while moving on to Allegro for the backend tool. During the Summer of 2009, I re-wrote the entire laboratory series.

Now, both Capture and Allegro are large complex programs quite difficult for beginners to learn merely by consulting the various references, "tutorials" and "manuals", that come with the Cadence tools. As useful as these may be, and some of them make good references, they don't provide an effective comprehensive picture to begin with, since they uniformly assume readers already know what "capturing" schematics, and then subsequently designing printed circuit boards, involves. If you are new to EDA tools, understand that Cadence has their own arcane idiosyncratic nomenclature and vocabulary. Engineers using other EDA tool chains will find the chief impediment to quickly acquiring expertise lies with this obtuse language. Relevant Cadence documentation provides little help here - it is among the poorest I have ever read, suffering from semi-literate organization and exposition (*viz*. excessive overuse of "you can" *ad nauseam*; illiterate redundant expressions; absence of pronoun references

and always in the passive voice; the worst problem, though, is disorganized unconnected paragraphing without clear topic sentences composed mostly of copious factoid descriptions rather than helpful integrated explanations). I have come to call this kind of industrial technical writing

Cadence-speak: A descriptive term referring to a style of technical writing noted above.

My remedy to the abysmal Cadence documentation is to rewrite it where necessary and integrate into the tutorial lecture flow of these laboratories.

The approach we will follow in this course differs from those provided with the tools by organizing your first experience around an actual design to better help you quickly gain an overall appreciation and understanding of the *design flow* by taking you through a real example circuit from start to finish. Many details will deliberately be left out, since these can only be appreciated after really grasping the bigger picture first. The first laboratory discusses project organization and creating schematics using Capture. The next laboratory continues on with an introduction to the PCB layout phase using Allegro, where we layout and generate the output, known as "artwork", necessary to making real PCB's. This is followed by a laboratory discussing the details associated with creating PCB footprints and making and working with padstacks, both topics that will be well understood from the previous lab. The last laboratory in this series treats the subject of "design verification" along with "tiling" or "panelizing", where we want to combine different PCB designs together in a single cost-effective submission to a PCB board house. Finally, having gained a solid foundation, the last assignment is your chance to exercise all the skills learned in this class to capture and layout a real custom PCB of your own. I will provide you with candidate engineering designs, but you can propose your own if you like.

3. Introduction to Board-Level Design

The Front-End Phase

Designing a printed circuit having discrete components is basically a two-step process: First, a design must be fully entered into a suitable engineering schematic whose chief purpose is to accurately include all relevant electrical components and interconnections. This front-end phase, usually called *schematic capture* or *design capture*, ideally aims at two objectives: First, besides having the necessary components and interconnections, it should, where possible, *expose the underlying de facto engineering design itself*. This latter point is often unappreciated and misunderstood; tyros consistently disregard it by conflating engineering schematics with *wiring diagrams* [1]. Relatively small projects should always be drawn as understandable self-documenting engineering schematics. This phase, then, will always produce a document, graphical or otherwise, faithfully mapping everything to real tangible devices (ICs, resistors, capacitors etc) and their wired interconnections. But, for example, if the initial design is basically done as a large Verilog HDL project this isn't always possible. Large projects having few devices with many, many pins – like FPGA's are best suited to this mode of capturing. Cadence has a relevant tool specifically for these kinds of designs called *System Connectivity Manager* (SCM).

Cadence offers two major front-end schematic flows: 1) Allegro Design Entry HDL, with or without SCM. 2) Allegro Design Entry CIS. Both flow into the same back-end tool: Allegro PCB Design that functions slightly differently depending upon which front-end tool chain is being used. I want to emphatically state this difference at the outset, since Cadence technical writers nowhere carefully compare and contrast these two front-end flows. The first pre-dates the second by many years and release versions. Much of the documentation, related directory and file organization, are quite specific to this flow. Since the second is based on the much later acquisition of Orcad Capture, the manner in which schematics are created, how they tie into the back-end tool chain, are architecturally different. Without being able to clearly distinguish between these two front-end tool chains, beginners simply cannot extricate themselves from the endless confusion regarding relevant documentation.

Which of these two front-end flows *should* we use? The first is unquestionably the more versatile and powerful tool, and is most useful for large HDL-based designs and those projects requiring sophisticated simulation features.

---

[1] This topic is discussed in, A Note on the Proper Documentation of Engineering Schematics. Please study this for a complete discussion of the differences between *wiring diagrams* and self-documenting *engineering schematics*.

However, it suffers from a poor graphic editor, a *very* inconvenient footprint mapping mechanism requiring that schematic symbols be created simultaneously with their PCB footprints. The second is most useful for relatively small projects, since comparatively it has a superior graphics editor, a cleanly integrated symbolic parts manager, separate from, but easily linked to necessary footprints. Moreover, the "component information system" or CIS manager is capable of keeping track of a sophisticated database of many inventoried components.

The Back-End Phase

The second, or back-end, phase involves taking the captured schematic and implementing it as a practical piece of hardware on a real printed circuit board. This means we must start with a listing of all components and each interconnection (called a *net*) between them derived by first compiling the schematic and then used as input to a companion back-end pcb layout tool. This necessary link between the front and back-ends is classically known as *creating a net-list*, and typically involves one or more output files created by some compilation process. This so-called *net-list*, along with component information, fully encapsulates the captured schematic design. This close relationship between captured schematics represented with compiled netlists and the layout tool was developed to make placing parts and wiring them correctly on a pcb error-free. Each component created for use on the schematic side typically includes hierarchical information pertaining to mechanical facts necessary to actually laying out this part on a circuit board. This would include such things as sockets, pad sizes, etc., and are typically called *decals* or *footprints*.

Advanced topics during this phase might also include any of the following:

- Design for manufacturing (DFM). Basically this means choosing and placing components to make them easily manufacturable using robotic machine processes. This requires extensive knowledge of thermal and soldering reflow techniques.

- High-speed digital or RF analog signal routing. This requires an understanding of distributed signal propagation using PCB traces as transmission lines. This topic is covered in CMPE173.

- EMI and EMC (electromagnetic interference and compatibility) dealing with spurious radiation and susceptibility to interference.

These topics of course are generally beyond the scope of this beginning class. You should, however, be aware of them, since you will undoubtedly encounter them in later more advanced work. Indeed, skilled layout engineers implicitly consider these in every design they do – whether specifically necessary or not. Later, you will find that some of them are encapsulated in the *standards of good engineering practice* discussed in Laboratory 2.

4. Laboratory Topics

The PCB EDA Toolchain – An introductory essay surveying EDA toolchains, and serving as a foundation reference to understand the context of the highly detailed skills necessary to board-level design discussed in the following labs. Refer to this as you go along so as not to lose sight of the big picture.

Laboratory 1, Capture – This lab introduces Orcad Capture CIS, now known as Cadence Design Entry CIS, the essential front-end tool used to "capture" electronic designs as well-drawn, self-documenting engineering schematics. This lab must either be completed or well understood before attempting the next one.

Laboratory 2, PCB Layout – This lab treats the backend phase of PCB design, building seamlessly on work done in Lab-2. Allegro PCB Design XL is used to layout and generate necessary files meant to be sent to a board house for PCB manufacturing. This lab is difficult, requiring close attention to detail throughout. DO NOT be tempted to address it lightly, hoping to skim along as an undisciplined dilettante, trying to "just get through it" by facily doing it in a hurry.

Laboratory 2 (Addenum 1), Footprints and Padstacks – After learning the elementary topics required for PCB design, several are developed in detail. This lab is really a detailed addendum to lab 2, where we now specifically delve into the interesting details of footprints and padstacks, basic objects used to define real mounting pads and through-hole vias on physical printed-circuit boards. I show you how to create two footprints, one for a real surface-mount part and one for a through-hole part.

Laboratory 3, Post-Processing - This lab treats the last phase of the workflow, sometimes called *design verification.* Before sending relevant files to a board-house, they should always be "verified" using a third-party Gerber and drill viewer. I will also show you how to "tile" or "panelize" disparate PCB layouts into a combined, large, single design. We will be using program called Gerbtool.

Laboratory 4, Project Assignment – This is your chance to capture, layout and design a real PCB that we will send out to an external board house as part of a tiled group of designs with other students.

Specialized topics useful for future reference:

Laboratory 3.1, Circuit Cam – This lab covers how to use the LPKF pre-processing program, similar to Gerbtool in Lab 3, that reads Gerber and drill files (Excellon format) and uses them to prepare a special output file that can then be read by the LPKF M60 board router control program.

Laboratory 3.2, Board Master – This lab covers how to use the LPKF M60 board router control program. It accepts as input files prepared by Circuit Cam.