[A5] Conference: New Approaches to Undergraduate Education IV, Santa Barbara, July 26-31, 1992.

METHODS OF DESIGN-ORIENTED ANALYSIS: LOW ENTROPY EXPRESSIONS

R.David Middlebrook, Professor of Electrical Engineering
California Institute of Technology

ABSTRACT

Since Design is the Reverse of Analysis, only analysis that can be "inverted" is worth doing. This is *Design-Oriented Analysis*, and derivation of the results in *Low Entropy Expressions* allows a *Design Feedback Loop* to be closed, so that Specifications can be met more quickly and efficiently.

INTRODUCTION

The premise for the teaching approach outlined in this talk is: *Many engineers are not able to translate a large part of what they thought they learned in college into practical application.*

In the course I've taught for many years at Caltech, I began to change the emphasis of my approach until now the essence is in *the methods and the techniques*, and the subject material is secondary. It's really a course in *howto set up and solve design problems*, using analog circuits as a vehicle, although of course the methods are applicable in other fields as well.

The culmination of the refocused course *Structured Analog Design* was the *assignment of names* to the various methods and techniques that had been developed and refined.

My purpose here is to establish *Design-Oriented Analysis in terms of Low Entropy Expressions* (1,2) as a means by which engineers may reap greater benefit from their academic training.

This paper is a minimally edited transcript of the talk presented at the conference.

DESIGN IS THE REVERSE OF ANALYSIS

Most students feel that their college years constitute a pretty steep ramp of information absorption. They emerge after four or more years, degree in hand, ready to solve the world's design problems, and then *they fall off a cliff.* Because, they get to their first position in a company and suddenly find that what they have to do is *Design*. But, they don't know how to do that, because 80 or 90% of the typical undergraduate curriculum is about *analysis.* Here's the problem; this is the method used to solve it; once you get an equation that says X = a bunch of stuff, that's the "answer". Finished, go on to the next exercise.

When newly minted engineers get into the real world outside, they discover it isn't like that at all, because what they find in front of them is not the statement of a problem, but a *Specification*. That's the *answer*. They have to work back to find what the problem is, and we're all very well aware of the fact that *Design is the Reverse of Analysis.*

Design is the reverse of analysis because the *starting point* of the design problem is the *answer* to the analysis problem, and really this ought to make a lot of difference about how you go about doing the analysis. However, the way it usually happens, you do all the analysis first, then you sit back and look at it and see how you're going to reverse it for design. That's one of the problems that many engineers have difficulty with when they first arrive in industry, and that "falling off the cliff" really involves a *restart* from "Why didn't I hear about this in college?"

Two Impediments

A large part of what new engineers learned and thought they understood very well from undergraduate education turns out not to be much use to them in the real world. Not much use, in the sense that they're not able to translate those things into actual design application. So, they start learning by a different method, the empirical approach, the knob-twiddling approach, let's take a previous design and change a few things. This is the typical approach in which the learning ramp becomes shallower, and the whole process is inefficient in the sense that because that cliff was present in the first place it takes many years, sometimes never, for very much of that academic information to be really translated into practical application. So that's the first of the two problems that I think we have to face.

The second one is that even as far as the analysis part of the problem is concerned, in real life it's at least an order of magnitude more complicated than the problems that engineers worked as students. For good and sufficient reasons, problems that they work as students are very artificial, simplified, *sanitized* to the utmost degree. They have to be, because there's a structure that it has to be fitted into and, in most cases, they have to be set up so there is *one right answer* and all the other answers are *wrong*. That's because most of the papers and exams are graded by teaching assistants who don't know much more about it than the students taking the course.

Recipe for Failure

How does it get this way? The conventional approach for most programs is implicit, and most people don't realize these points explicitly, so I've tried to put some *words* to them just to make a kind of formal structure. The way most of us are taught to tackle engineering problems is, *Put everything into the model, go as far as you can* until you get stuck in the algebra, then look around to see what you can leave out, how you can simplify it, what approximations you may be able to make. That is, *you postpone the approximations as long as possible.* And certainly, if you *do* make an approximation, this is considered some sort of *failure.* You were unable to solve it exactly, so you had to make an approximation, and you'd better be able to justify it on the spot.

Finally, the student syndrome: *The more work you do, the more valuable the result.* Sometimes they feel problems get graded according to weight, instead of actual content. In my opinion, this approach is almost guaranteed to lead to algebraic paralysis. The equations get longer and longer as you go down the page. They start slopping over onto the next line, then you get stuck. This is the typical thing that happens to the new engineer out in industry with his first design problem. That's exactly what he does, he writes down all the equations he can think of, applies the algebraic derivations that he's learned how to do, and the equation simply gets out of hand. He gets bogged down with it, and after some period of flailing around, he gives up. That's where he makes this restart by a totally different, empirical approach, learning how things are "really" done, and a large part of that academic background simply doesn't get used. *It doesn't get translated into the real world.*

In academia, of course there's clear recognition of this problem, and there's a lot of emphasis on how to introduce design into engineering curricula, especially in undergraduate programs. There are a lot of courses now, and labs, labelled *Design* but, unfortunately, it's usually tacked on at the end of the analysis. This is better than nothing, but it doesn't really tackle the problem because once you've got a huge equation that you can't do anything with, it's already too late.

A closely related point has been raised at this conference by Bill Eccles (3), which is that *computers are introduced too soon and too much,* and I agree with him entirely. Computers are a very useful tool, but they are not going to do your thinking for you. In particular, they're not going to help very much with insight into the problem if all you do with a computer is ask it for the answer.
If resorted to *too early* in the design process, use of analysis software can actually *impede* progress.

So this is, I think, the crux of the point we need to think about. I suggest here some possibilities, a paradigm, for improving the teaching of design.

DESIGN-ORIENTED ANALYSIS TO THE RESCUE

The goal is, I believe, that design *can* and *ought* to be integrated throughout the analysis process, and it turns out that when you start looking at analysis from this point of view, it makes a tremendous difference in how you go about doing it. The solution is what I like to call *Design-Oriented Analysis.* Of course that doesn't mean anything standing there as a phrase, anyone could call the solution to a problem any words they like and it doesn't tell you anything, but that's what I want to expand on here.

I believe Design-Oriented Analysis is the Only Way to Go. There's no point in doing any other sort of analysis. If you can't use it backwards for design, it was a waste of time. So let's try to get some sort of definition of *Design,* which is a fuzzy word because it means different things to almost everyone.

The trouble with *Design* in the educational program, I think, is that as students, we don't get to hear from engineers until the mathematicians and the scientists have already been at us for several years. We don't hear from engineers until university, but all the way up through the upper levels of high school we are being taught math and physics and other basic sciences. That's fine, but the mathematicians have always told us from the start that if you want to solve a problem with a certain number of unknowns, you'd better have as many equations as you have unknowns, otherwise you can't solve the problem.

Not Enough Equations...

The difficulty is that *engineers have to solve the problem in the face of not enough equations.*

In fact, not only are there not as many equations as the number of unknowns, there aren't *anywhere near* enough equations.

Where do the equations come from that you *do* know? They come from the Specifications, the *Answer* to the problem. And there's a fairly small number of Specifications, usually. Those are the only equations that you actually have. Those are the only expressions you can write with "knowns" in them, the "knowns" being the answer. The "unknowns" are all the myriad circuit elements and components of the system, as well as how to put them together.

It might be said that engineers start their professional lives from a negative position, having been told by the mathematicians that they are confronting an impossible problem. That's not a very good feeling, even if it's only implicit.

Substitutes for Missing Equations

What is the actual problem that the engineer faces? *Design* is solving a problem in a sense *optimally,* (and of course that's a fuzzy statement too), by finding *substitutes* for all those *missing* equations that correspond to the total number of unknowns, which is the total number of elements in the system. You have to fall back on the next line of defense.

How are you going to find substitutes? They are going to be in the form of *inequalities, approximations, assumptions, tradeoffs,* in fact anything that you can get your hands on. You've got to dredge the river for every possible help you can get in solving the design problem in the face of insufficient information.

Now, that's only one definition of design, and a very limited one of course, but I've found in teaching this stuff for a long time, I've actually had to put *names* and *words* to ideas for students to get hold of them. This came about because for a long time I taught by example; I used a lot of methods, saying we're going to look into the properties of a certain circuit, and I would use some shortcut method or trick that I thought was a neater way to do it, and I just showed them this, without actually emphasizing the method. That was fine, but then when they did homework problems and exams, their minds just automatically reset to the old way of doing it. So that didn't work very well, and finally I realized I had to give *names* to these things, and some of them I want to introduce here.

Low Entropy Expressions

Besides finding substitutes for missing equations, you can improve the efficiency of design in another way Those few equations that you *do* have, can be made to work a lot harder for you; that is, *one* equation can actually give you *more than one* answer.

Here's one of the new words that I introduced to describe that process. The way you can get more than one piece of information from one equation is by writing it in what I call a *Low Entropy* form. The word "entropy" of course is familiar, and that's why I chose it, but this is a different context. Entropy is a measure of disorder; the more disordered the physical system is, the higher its entropy. That's the only aspect of the meaning that I'm borrowing for this purpose. I call a *High Entropy Expression* an equation in which the arrangement of the terms and the symbols conveys no information in itself. It's just a jumble. It's the way it comes out of the algebra after you grind the algebraic crank in the conventional way.

In contrast, a *Low Entropy Expression* is one in which the terms and elements are *ordered* or *grouped* in such a way that their physical origin, that is, where they came from in the circuit, what part of the circuit contributes to this part of the final expression, becomes obvious. That's how you can tell what the relative importance is of the different contributions. The reason that this is an important thing to do is that it's the *only* analysis result that is of any value for design, because for design you have the *Answer*, the left hand side, that's the number you know, the Specification. The *Problem* is to find all that bunch of unknowns on the right hand side, and the more information that you have about the relative importance of those unknowns, the easier it's going to be.

The only analysis result useful is a *Low Entropy Expression*, so that you can change the values in an informed manner, in contrast to just randomly changing all the variables as you do in a worst case kind of computation. Worst case computations are done very well by computer, but that's the *last* step in the design process, and it *should* be the last step Earlier on, when you're trying to find tentative values of a lot of elements and you've got a lot of guesswork in it already, the only way to do that is to have Low Entropy Expressions so you can change an element value knowing what influence it's going to have on the answer, and what other elements are not going to influence the answer.

THE DESIGN FEEDBACK LOOP

So that's what *Design-Oriented Analysis* is. It's analysis that keeps the entropy low all the way through the derivation.

You don't *have* to do it that way; you can go through conventional algebra and get a High Entropy Expression, then you can put a lot of energy into it to lower the entropy. That's consistent with the original definition of entropy: to create order and lower entropy you have to put energy in. In this case, you have to put *mental* energy in to lower the entropy and create the order, and obviously that's inefficient, because what would be much better to do is to keep the entropy low to begin with so you don't lose that information, and that makes a big difference in how you set up the problem. Again, Low Entropy algebra is the *only* kind of analysis worth doing, because deriving a huge equation that is noninvertible for design is simply a waste of time.

Engineers always like block diagrams, so I made one to illustrate the design iteration process, because you keep having to go back and change things. Figure 1 shows a diagram indicating that the way you start a design process is with a very

The Design Feedback Loop:

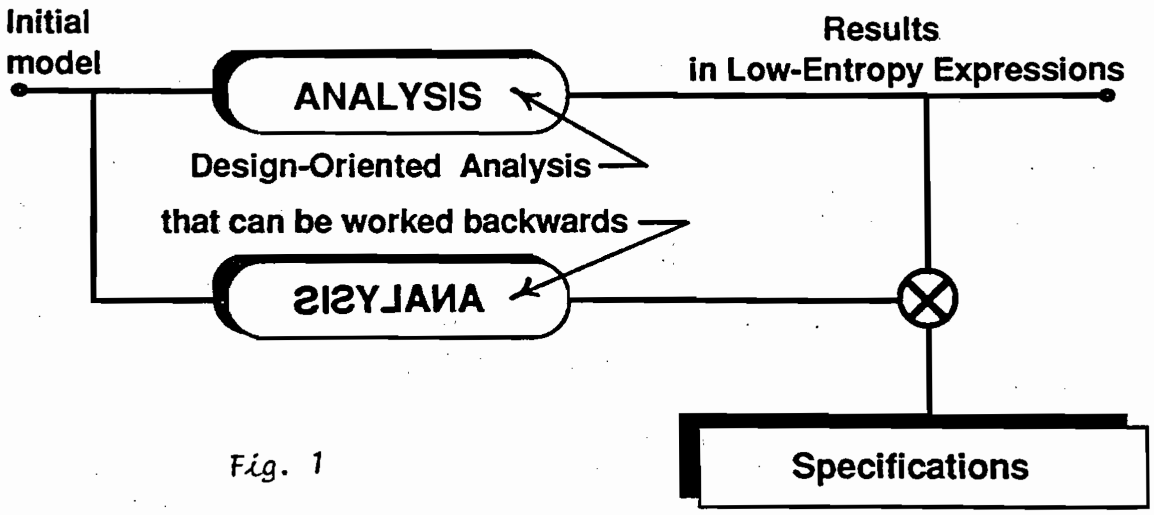


Fig. 1

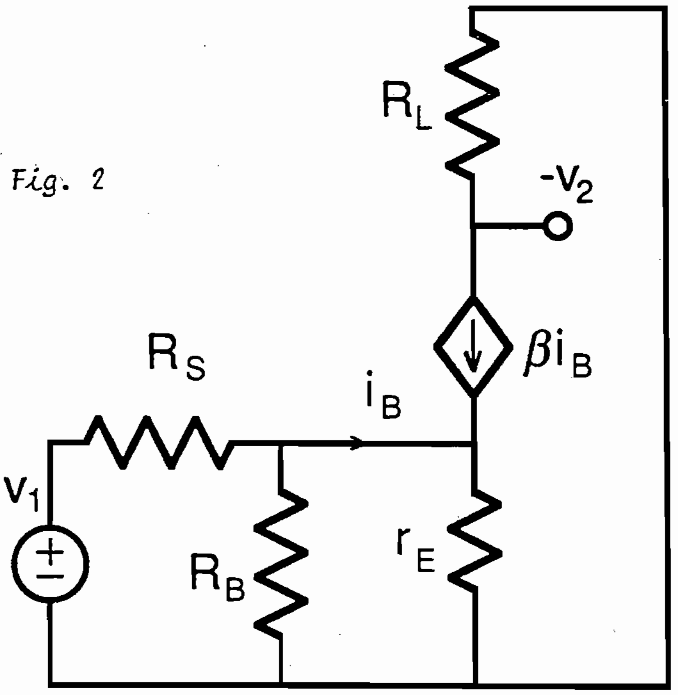
simple model, initially only a block diagram perhaps. You do your analysis and you get a result. Then you have to compare that result with the Specification and see if it matches, and of course most of the time it doesn't. Then you have to go back and change something. But so far, this is an open-loop process, which is the way it's normally done.

I suggest that we want to make this into a closed-loop process. We want to *close a feedback loop around the analysis*, which means we compare the analysis result with the Specification, and if there is an error, a discrepancy, we use that error to work the analysis backwards and change the starting point, change the configuration, change the values. *Being able to close the loop requires you to be able to work the analysis backwards.* This is only possible if the analysis result is a Low Entropy Expression.

Example: Low Entropy Expressions

A very simple example is shown in Fig. 2, a small-signal model of a common emitter transistor amplifier. The result of the conventional analysis for the voltage gain

Fig. 2



$A_m = v_2 / v_1$ is

$$A_m = \frac{\beta R_B R_L}{(1+\beta) r_E R_S + (1+\beta) r_E R_B + R_S R_B} \qquad (1)$$

As typically happens with electronic circuits, the answer is a ratio of sums of products of the various circuit elements. If it's a two-loop circuit, you get sums of products two at a time, which is what this is (the third loop current is a fixed multiple of the second). If it were a five-loop circuit, you would get sums of products five at a time, so the algebra *rapidly* escalates with increasing circuit complexity. That's the direct result of using basic loop or node analysis, which is what everyone is taught to do.

On the other hand, here is a Low Entropy result:

$$A_m = \frac{R_B}{R_S + R_B} \frac{\alpha R_L}{\underbrace{r_E}_{(1)} + \underbrace{(R_S \| R_B)/(1+\beta)}_{(2)}} \tag{2}$$

It's exactly the same equation as Eq. (1), but hardly recognizable. The value of the Low Entropy equation is that there are several additional things you can deduce from it.

First, there's a resistor ratio in front, which is immediately identifiable as a resistive divider in Fig. 2. Second, resistances show up as series/parallel combinations, which means you know that in a series combination, the larger one dominates, in a parallel combination the smaller one dominates. *Useful, extra* information. And third, if you compare terms 1 and 2 in the denominator, you see that the Beta of the transistor, which is the parameter with the biggest tolerance on it, shows up in only one of those two terms: if you know the relative values of those two terms, you know what the dilution factor is for variations of Beta in causing variations of the gain. So that's *three additional* pieces of information , a total of *four* pieces of information from one equation. *You make the equation work harder.*

All I've done in this example is show the benefits of a Low Entropy result over the High Entropy result, but I didn't say how the Low Entropy result is achieved.

One way is to inject the mental energy necessary to manipulate the initial High Entropy result into the Low Entropy version. In the broader spectrum, this corresponds to doing the analysis in the conventional way, then tacking on a design interpretation at the end.

A much better way is to keep the entropy low throughout the derivation, so that the result automatically comes out in a Low Entropy form. The tools to do this are the methods and techniques of Design-Oriented Analysis that constitute my refocused Caltech course, which obviously cannot be described here. Some have been discussed in (4,5,6).

This is only one example, but the point is that equations in Low Entropy forms work harder to make up for missing equations.

RECIPE FOR SUCCESS

Here's what I suggest as a positive approach to Design-Oriented Analysis:

Put only enough into the model to get the answer you need. Your reaction to that statement probably is "Well, of course I wouldn't do a stupid thing like that; who would?" But *almost everyone does*. Everyone puts too much in to begin with, and it's very hard to get that across to students. They fall into the trap every time.

Make all the approximations you can as soon as you can, justified or not. Leave behind you a wake of assumptions and approximations. All you've got to do is go back and check them later. This is where the mathematicians throw up their hands in horror, because they say "You'd better not do that because if you're going to change numbers later, you may invalidate the approximation, so don't do it." Well, I agree with all that statement except the last phrase. *Do it.* That's another way to get useful answers; *you can't lose by trying.*

Finally, *The less work you do, the more valuable the result. You* control the algebra. Make the algebra work *for* you. Don't be a slave to the algebra. Again, this is tough for students to absorb because they are brought up fighting algebra, trying to solve problems that are difficult to them, and it's a constant battle to get the answer any way they can. If they get any sort of answer, they think they've done very well, which is true.But *it's not good enough*. They've got to get into a position where *they* can direct the algebra to do what *they* want.

You see that the above positive approach represents essentially the opposites of the conventional approach, listed previously under the heading Recipe for Failure.

I've been encouraged to offer this approach to educators because of the success of a condensed version of my course, *Structured Analog Design*, in industry. Experienced engineers are extremely receptive to these ideas, because they've already tried it the "hard way," and they *know* that doesn't work. After such a course, I'ver many times had engineers say, "I wish I'd known these things years ago! Why wasn't I taught them in college?"

CONCLUSIONS

The premise is that many engineers are unable to extract sufficient benefit from their academic training when they are faced with real-life design problems. This is ascribed to two causes: first, 80% of conventional curricula emphasize analysis, and design, if present at all,is tacked on at the end; second, on the job, the engineer is immediately faced with a problem that is at least an order of magnitude more complicated than were the simplified, sanitized examples he aced as a student.

We choose to be engineers because we want to apply scientific principles to design new or useful functional applications. Unfortunately, we are programmed in our earlier years to believe that "to solve the problem, you must have as many equations as there are unknowns." This is a recipe for failure as an engineer, since design problems always involve far more unknowns (the system configuration and the element values) than there are equations (representing the Specifications).

An attempt is made to define *Design* as an optimal solution for system configuration and element values in terms of the Specifications. This requires finding substitutes for the missing equations, in two ways. First, we need all the help we can get by invoking the next best thing in terms of inequalities, approximations, assumptions, and tradoffs. Second, we have to make the few equations that we do have work harder.

This leads to the definition of Low Entropy Expressions, which are equations in which the ordering, or grouping, of the various elements illuminates their contribution to the result. An example is given in which one equation reveals four pieces of information that are useful for design.

Analysis techniques that maintain Low Entropy Expressions throughout constitute *Design-Oriented Analysis.*

A *Recipe for Success* is proposed, essentially the opposite points from those typically ingrained.

REFERENCES

(1) R.D.Middlebrook, "Low Entropy Expressions: The Key to Design-Oriented Analysis," Proc. IEEE Frontiers in Education, Twenty-First Annual Conference, Purdue University, September 21-24, 1991, pp. 399-403.

(2) R.D.Middlebrook, "Analog Design Needs a Change of Perspective," Electronic Engineering Times, Technology Trends Supplement, October 7, 1991; p. T5.

(3) William J Eccles, "Don't Hide the Physics," New Approaches to Undergraduate Engineering Education IV, Santa Barbara, July 26-31, 1992.

(4) R.D.Middlebrook, "Null Double Injection and the Extra Element Theorem," IEEE Trans. on Education, vol. 32, no. 3, August 1989; pp. 167-180.

(5) R.D.Middlebrook, "The Two Extra Element Theorem," Proc. IEEE Frontiers in Education, Twenty-First Annual Conference, Purdue University, September 21-24, 1991; pp. 702-708.

(6) R.D.Middlebrook, "Methods of Design-Oriented Analysis: the Quadratic Equation Revisited," Proc. IEEE Frontiers in Education, Twenty-Second Annual Conference, Vanderbilt University, November 11-15, 1992.