Incremental Design for Linear Circuits

Juan Flores
Facultad de Ingenieria Electrica
Universidad Michoacana
Morelia, Michoacan, 58030
juanf@zeus.ccu.umich.mx

Arthur M. Farley
Computer and Information Science Department
University of Oregon
Eugene, Oregon, 97403
art@cs.uoregon.edu

Abstract

In this paper, we present a framework for performing incremental design in the domain of linear circuits (Kerr 1977; Lancaster 1974; Walton 1987). By incremental design, we mean the modification of an existing design to meet additional design goals while not denying certain design constraints. We start with a given circuit and want to modify aspects of its behavior while not changing others. Through means-ends search, we add components to the circuit to achieve the desired behavior without violating given constraints. The means-ends solution is based on a constraint-based model derived from circuit theory.

A given design problem is first solved in terms of a qualitative model of the circuit. The framework we present is also capable of determining numerical values of parameters associated with the components added by the design process. This is accomplished by using the operating conditions of the circuit as input and the values of the parameters as output and running our constraint propagation system to determine the output values. This feature allows us to complete the design process, combining qualitative and quantitative reasoning.

Introduction

The research reported here is a continuation of the work presented by Flores and Farley in (Flores & Farley 1996). In that paper, we consider how a linear circuit can be modeled by a set of constraints, as derived from basic electro-magnetic theory. To produce a constraint-based circuit model, we first decompose a given circuit into parallel-series clusters, as suggested by Liu (Liu & Farley 1990). The resultant circuit structure (clustering) is traversed, and a list of constraints are produced for each element and cluster. Figure 1 shows an example circuit and part of its corresponding constraint-based model. Finally, in (Flores & Farley 1996), a set of applications for the methodology is proposed; among others, design is mentioned as a goal and some initial ideas are presented.

Flores (Flores 1996) then reports on an efficient implementation of constraint propagation, for the particular needs of this reasoning task. We describe how constraints can be represented in a form that allows

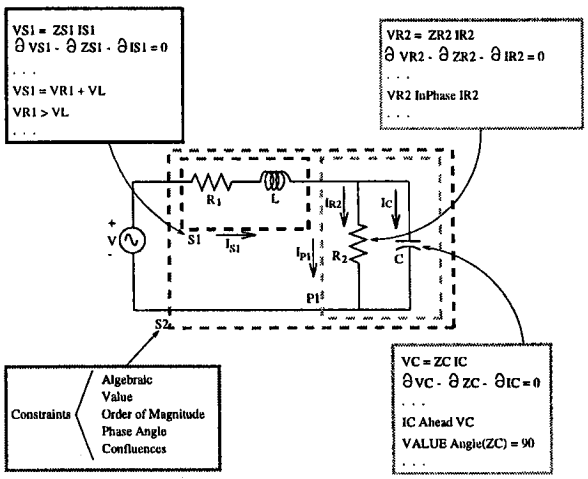

Figure 1: Example circuit and part of its constraint-based model

efficient value propagation, order of magnitude propagation, and mixed propagation. We represent all values as intervals, which offers the user the ability of providing information to the system gradually, as knowledge about the circuit is acquired or as the knowledge becomes more specific. As a result of that earlier research, we developed a software platform, called *QPA*, to model and reason about linear electrical circuits in sinusoidal steady state. A compilation of all these results can be found in Flores' dissertation (Flores 1997).

In this paper, we present the use of a constraint-based circuit model to perform incremental or control design on linear circuits. The primary contribution of the work is a qualitative methodology for changing the structure of a linear circuit to meet certain control goals. The paper is organized as follows. The next section reviews our earlier work on the qualitative representation and reasoning about linear circuits. Next, we express the problem of incremental or control design as a planning problem and defines, based upon first principles, a set of operators that allow us to modify a given circuit to realize certain control goals. After that, we present the incremental design algorithms based on the design

operators. Next, some examples of qualitative designs are presented; these results were obtained with the approach we have developed. We then describe and illustrate how numeric values can be determined for the new circuit elements by placing constraints that express the operating conditions, and propagating those constraints over the interval representation. Finally, the work is concluded.

QPA

The electrical engineering community has been very successful in predicting behavior of linear circuits in steady state. The main tool they use in circuit analysis is the phasor. Phasors are a mathematical transformation that maps sinusoids from the time domain to the frequency domain, allowing us to replace complicated simultaneous differential equations by algebraic simultaneous equations in the complex domain. Besides their power to solve linear circuits, phasors can be expressed in an intuitive graphical form; as *phasor diagrams*. These diagrams allow electrical engineers to have a better understanding of what happens inside a circuit and can be used to produce causal explanation of physical phenomena.

In a circuit excited by a sinusoidal voltage source, of frequency ω, all variables are also sinusoidals oscillating at the same frequency. Each variable $V(t)$ can be expressed as the real part of a complex quantity. That is

$$\begin{aligned} V(t) &= Re(|V|(\cos(\omega t + \angle V) + j\sin(\omega t + \angle V))) \\ &= |V|\cos(\omega t + \angle V) \end{aligned}$$

where $V(t)$ represents a (real) function of time, and V represents its corresponding phasor in the frequency domain. If we represent all variables in a circuit by a phasor, they will rotate at the same angular frequency, as if fastened together. So a phasor diagram can be seen, at any given moment, as a snapshot of the set of rotating phasors that represent all the quantities in the circuit. The solution to the circuit can be obtained by taking the real part of each phasor (i.e. make all phasors rotate at the same frequency as the source and take each phasor's projection over the real axis).

To determine the set of algebraic constraints corresponding to a given circuit, we represent the circuit as a structure of series-parallel clusters. We then can recursively traverse that structure, generating constraints for each cluster or component we encounter. The set of constraints can be partitioned into subsets of several types: Definitions (e.g. $(DEFINITION(= \ V_{R_2} \ (* \ Z_{R_2} \ I_{R_2})))$), Order of Magnitude (e.g. $(> \ I_{R1} \ I_C)$ or $(\gg \ I_{R1} \ I_C)$), Phase Angle (e.g. $(IN\_PHASE \ V_{R_2} \ I_{R_2})$), value (e.g. $(VALUE \ V_C \ (+\angle 90))$), and Confluences (e.g. $(CONFLUENCE \ (+ \ (\partial \ VR)) \ (- \ (\partial \ ZR)) \ (- \ (\partial \ IR)))$).

As described above, our implemented system QPA will generate a set of constraints for each component and cluster of the circuit. The resulting constraints constitute what we call the Basic Circuit Model. Once this has been generated, propagation is used to obtain the transitive closure of the constraints and their implications. For instance, from constraints $(= \ I_{S_1} \ I_{R_1})$ and $(= \ I_{S_1} \ I_L)$ we can derive $(= \ I_{R_1} \ I_L)$.

If the user has further information about features of the circuit, these can be expressed as additional constraints. For instance, consider a capacitor C in parallel with a resistor R to create a cluster P_1. The user can tell QPA that $(< \ Z_R \ Z_C)$; propagation will indicate any implied constraints, such as $(< \ I_C \ I_R)$. The user can also ask if a certain property holds. For example, if the user asks if $(ANGLE \ I_{P_1} \ V_{P_1})$ can be 90 degrees, the system can respond with the following answer "No. You told me that $(< \ Z_R \ Z_C)$, which implies that $(< \ I_C \ I_R)$, and therefore $(VALUE \ (ANGLE \ I_{P_1} \ V_{P_1}) \ (0 \ 45))$." An important aspect of constraint propagation is the interaction between magnitude and phase angle variables. For simple elements, the phase angle is precisely defined, but when components of different kinds are combined, phasor addition may be ambiguous, depending on what we know regarding magnitudes of the elements involved.

All values in QPA are represented as intervals: numbers are point intervals (e.g. $5 = [5,5]$), uncertain values are intervals with open or closed limits, and symbolic values are translated into intervals as well (e.g. $positive = (0,\infty)$). This approach integrates value propagation with order propagation. For instance, given constraints $X = [0,10]$, $Y = [5,15]$, and $(= \ X \ Y)$, we can *refine* the values of X and Y to be both equal to $[5,10]$. This feature allows QPA's solutions to default to the solutions of conventional circuit solvers in the case when all values are precisely defined. We will use this feature in incremental design, taking a qualitative solution to a design problem and making it more precise by determining parameter values to meet more precise design goals.

Confluences represent algebraic constraints about change. For instance, for Ohm's law in a resistor $V_R = Z_R I_R$, we have the qualitative counterpart $\partial V_R - \partial Z_R - \partial I_R = 0$. This confluence indicates, for example, that if Z_R decreases and V_R does not change, I_R increases. First-order reasoning is important if we want to be able to explain or to design changes in a circuit's behavior. Confluences capture the interaction among different variables in the circuit and how changes in one variable can produce changes in other variables. In our parallel cluster, if the user asks "what happens if R decreases?" (expressed as the constraint $(VALUE \ (\partial \ R) \ -)$), the system replies

> "If R decreases, Z_R decreases, which causes I_R to increase. The increase in I_R, in turn, causes I_{P_1}'s magnitude to increase, and I_{P_1}'s phase angle to decrease. ".

QPA also handles order of magnitude constraints. Order of magnitude constraints can be used to simplify a circuit, when appropriate. For example, if af-

ter running propagation it is determined that a current (voltage) is near or at zero, it can substitute that element by an open (short) circuit. Opening an element or cluster is equivalent to removing it from the circuit; short-circuiting is equivalent to removing that element or cluster and to collapsing both nodes into one. After these structural modifications, a new model of the circuit is rebuilt, and propagation on the given constraints are recomputed. This is similar to what we will do in incremental design.

Incremental Design

By *incremental design* we mean the process of modifying the existing design of an artifact to meet further goals under certain constraints. As such, incremental design answers questions about change. As an example, for the circuit of Figure 1, an incremental design goal could be "How can I get the phase angle of cluster S_1 to decrease?". In some cases, given one or more variable parameters, we can solve such questions by adjusting the parameters to meet the given goals. In other cases, however, there exists no adjustment of design parameters that will achieve the given goal(s). Furthermore, we may want to constrain acceptable solutions to a given incremental design problem. For the example of Figure 1, "How can I get the phase angle of cluster S_1 to decrease, without changing V_{S1}?". This again may not be solvable by simple first-order reasoning over given parameters.

One type of incremental design typically addressed in the domain of electrical circuits is that of control design. A *control design problem* can be defined in terms of an existing circuit, a set of goals to be achieved, and a set of constraints to be kept satisfiable. The design goals are specified as desired changes in the circuit's variables, involving either magnitudes, angles, or both. For the circuit of Figure 1, a design goal could be $\partial \angle Z_{S_1} = -$, subject to the design constraint $\partial V_{S_1} = 0$. To solve a control design problem, we must find modifications to the given circuit that entail the goals without violating the design constraints. Based upon our qualitative representation of circuits developed as part of QPA, we find a natural form for stating control design tasks as two sets of qualitative constraints, one representing the design goals and the other the design constraints.

An incremental design problem can be seen to be a planning problem, where the given circuit, represented as a set of constraints, is the initial state, the given circuit plus the design goals and constraints is the goal state, and a set of circuit modification actions are the operators. The problem is to find a sequence of such operators (i.e., a set of modifications to the circuit), such that the resulting circuit satisfies the goals and constraints of the task definition.

Our design operators will be adding an element to the circuit either in parallel or in series with an existing cluster or element. For instance, Figure 2 shows a situation where a capacitor C is short-circuited by placing a resistor R_d in parallel.

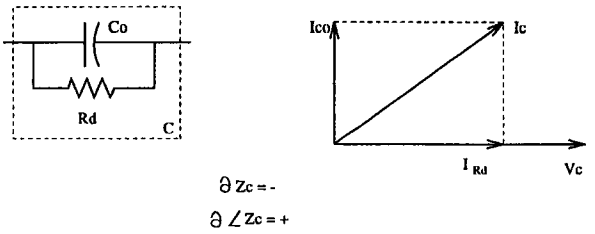

Figure 2: A capacitor short-circuited by a resistor.

The capacitor has been renamed C_0 and the new parallel cluster corresponds to the modified capacitor, now called C. The phasor diagram shows the current before the modification I_{C_0}, the current in the design element I_{R_d}, and the total current I_C. The current's angle and magnitude are drawn relative to the cluster's voltage V. A change in the ratio $V_C/I_C = Z_C$ indicates a decrease in impedance magnitude and an increase in impedance angle.

To define our design operators, we must specify what happens to affected circuit values when they are applied. Some of the operator definitions are given in Figure 3. Each operator definition indicates the changes in impedance Z, both angle and magnitude, that result in the new cluster relative to the original impedance.

Preconditions	Action	Effects
Capacitor(C)	Parallel(C, R_d)	$\partial Z_C = -$ $\partial \angle Z_C = +$
Capacitor(C)	Parallel(C, C_d)	$\partial Z_C = -$ $\partial \angle Z_C = 0$
IndCluster(Cl)	Series(Cl, R_d)	$\partial Z_{Cl} = +$ $\partial \angle Z_{Cl} = -$
IndCluster(Cl)	Parallel(Cl, C_d)	$\partial Z_{Cl} = +$ $\partial \angle Z_{Cl} = 0$

Figure 3: Selected Design Operators

The operator definitions reflect a qualitative application of the standard circuit theory. We consider short and open circuits as special situations. In both cases, the current in the immediately affected element is zero; the difference is that in a short circuit, the voltage is also zero.

Incremental Design Algorithm

In this section, we discuss the pseudo-code for our design algorithm, as presented in Figures 4 and 5. An incremental design task is stated as a set of design goals to be satisfied and a set of design constraints that must be satisfiable. For a given design task, we first select a goal to be achieved and select the design operator(s) that contain that goal as a consequence. This is the means-ends aspect of the search process. When applying an operator, the consequences indicate what changes in variable values the operator makes to the current circuit. To compute all of the ramifications of

```
Design-Step(dtask<C,Rg,Sg,DC,IC,H>, goal)
    if Rg
        rules = select-rules(goal)
        result = nil
        for each (rule ∈ rules)
            init-conds = suspend-values(this-goal,IC)
            new-constraints = assert(init- conds+consequences(rule)+Sg)
            if not(error?(new-constraints))
                rel-constraints = (consequences(rule) + new- constraints)
                                                    ∩ (Rg + Sg)
                new-Rg = Rg − rel-constraints
                new-Sg = Sg + rel-constraints
                new-H = H + <rule,rel-constraints>
                result + = <new-C,new-Rg,new-Sg,init-conds,new-H>
```

Figure 4: Pseudo-Code for Design-Step

```
Incremental-Design(dtask<C,Rg,Sg,DC,H>)
    result = nil
    queue = dtask
    while q
        dt<C,Rg,Sg,DC,H> = remove(q)
        selected-goal = select-goal(Rg)
        new-designs = design-step(dt, selected-goal)
        for each new-dt ∈ new-designs
            if not violates-constraints(sol-dt Sg(new-dt))
                look-for-satisfied-goals(new-dt)
                if not(Rg(new-dt))
                    if |H| <= *max-depth*
                        q + = Design-Step(dt)
                else
                    if not (violates-constraints C, DC)
                        if |result| = *num-designs*
                            return result
                        else
                            result + = dt
                    else
                        q + = violated-constraint-to-goal(dt)
    return result
```

Figure 5: Pseudo-Code for Incremental Design

the operator, we update our circuit model and assert the operator effects in the circuit model, However, this will normally cause inconsistencies in our set of derivative values determined prior to the change.

For instance, if we know that the impedance of a cluster does not change (under normal conditions) and the operator says it will decrease (under the given circuit modification), an inconsistency will be discovered immediately. To solve this problem, we suspend all confluence-related values for the cluster in question and for all its components; i.e. we make it a black box. We can now assert the consequences of the operator and run propagation in QPA without producing inconsistencies.

The asserted operator consequences entail some of the goals but may violate one or more design constraints. The application of each operator reduces the number of initial goals to be achieved. As each operator is applied, it generates a subsequent design task to be considered. These design tasks are placed on a queue of pending tasks, resulting in a breadth-first search for a solution. Each design task keeps a record of which design goals have been satisfied so far. To avoid loops, if a new design step violates a previously satisfied goal, we prune the search tree at that node. At each design step, we record the operator used and its relevant consequences. Relevant consequences are the intersection of the consequences of the operator with design goals and constraints of the incremental design task. This information is used to generate an explanation sequence of the design process.

A design task can be formalized as a tuple <C,Rg,Sg,DC,IC,H>, where C is a circuit, Rg are the remaining goals, Sg the satisfied goals, DC the design constraints, IC the initial conditions, and H the history list. The procedure *Design-Step*, shown in Figure 4, takes as input *dtask*, a design task as formulated above, and returns a list of all design tasks derived by satisfying one of *dtask*'s remaining goals by application of a design operator.

As noted, we implement a breadth-first exploration of the search space. A queue of remaining design tasks starts with only one element, the initial design task. Each application of a *Design-Step* produces a set of new design taks, one for each applicable rule. A design task is removed from the queue; using *Design-Step*, its descendants are computed and then enqueued to be explored later. If all the goals of a design task have been achieved, we then check to see whether any of the design constraints are violated. If any design constraint is violated, the system generates a new goal that would reverse the undesirable effects. We then must continue the search process.

Incremental design is intended to make relatively limited adjustments to a given design to meet new design goals. As such, we limit the search process to a fixed, maximum depth (i.e. the maximum number of elements to be added in the design). The user can indicate how many modifications are to be allowed and also how many solutions to find; the number of solutions is typically one or all, but may be any number. Figure 5 presents the general search algorithm *Incremental-Design* in pseudo code form for incremental design.

Examples

Figure 6 shows an example of the definition of a control design task, together with a transcript of two of the design solutions. To specify a design task, we merely name a circuit, for which a QPA type model (as a set of constraints) is expected to already exist, and provide lists of design goals to be met and design constraints not be contradicted. These lists present partial derivatives of model variables in terms of magnitude and angle each element, being of the form ((P ¡var¿ (¡magnitude¿ L ¡angle¿)). In our example, we ask that the angle of

```
(def-dtask
  :circuit     ex01
  :rem-goals   (((P ZS1) (? L -)))
  :constraints (((P VS1) (0 L ?))))

Solution: Sol1
     ((PARALLEL S1 Cd)
     provides (((P ZS1) (? L -))
               ((P VS1) (? L ?))))
Solution: Sol2
     ((PARALLEL S1 Rd1)
     provides (((P ZS1) (? L -))
               ((P VS1) (- L ?)))
     ((PARALLEL P1 Rd2)
     provides (((P ZP1) (- L ?))
               ((P IP1) (+ L ?))
               ((P IS1) (+ L ?))
               ((P VS1) (+ L ?)))
...
```

Figure 6: Design Task #1 and Solution

variableZ_{S_1} be increased while maintaining the magnitude ofV_{S_1} unchanged. A '?' entry in the problem specification means we do not care what happens to that value; in our example, we do not care what happens to the magnitude of Z_{S_1}.

The first solution to our problem causes Z_{S_1} to decrease, leaving the magnitude undetermined. This causes the voltage V_{S_1} to be undetermined, therefore not violating the constraint. That is, there remains a setting for the inserted capacitor that makes the angle of S_1 decrease, without altering its voltage. The first element of the second solution causes the phase angle of S_1 to decrease, but also makes V_{S_1} decrease. As a result, another resistor is inserted in parallel with P_1, which makes S_1's current and voltage increase. This effect adds to the previous one, making the total outcome ambiguous; this indicates that there again remains a setting for parameters of the design elements that results in the constraint not being violated. Figure 7

shows the circuit diagram for solution *Sol1* of Figure 6.

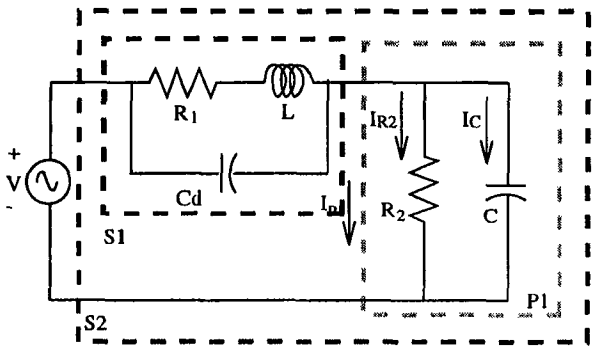

Figure 7: Diagram for Solution 1

Figure 8 presents another control design example, with two goals and one constraint, and part of its solution. It is important to note that every time the design

```
(def-dtask
  :circuit      ex01
  :rem-goals    (((P ZS1) (? L -)))
                 ((P IP1) (+ L ?))))
  :constraints (((P VS1) (0 L ?)))))

Solution:
     ((PARALLEL S1 R)
     provides (((P ZS1) (- L -))
               ((P ZS2) (- L -))
               ((P IS1) (+ L +))
               ((P IP1) (+ L +)))
     ((PARALLEL P1 P)
     provides (((P ZP1) (- L +))
               ((P ZS2) (- L +))
               ((P VS1) (? L ?)))
...
```

Figure 8: Design Task #2 and Solution

algorithm inserts an element, it computes all its consequences. It may be the case that some additional goals and/or constraints can be satisfied by the same operator. In this example we see that inserting a resistor in parallel with S_1 causes $\partial Z_{S_1} = (-\angle -)$ which in turn causes $\partial Z_{S_2} = (-\angle -)$. This makes the currents I_{S_1} and I_{P_1} increase. By trying to meet one design goal, we have satisfied both. However, the design constraint is violated, and an action is needed to ensure that constraint can be satisfied. The system suggests the insertion of a resistor in parallel with P_1, completing the qualitative design for this particular solution.

Parameter Design.

An interesting property of constraint programming and propagation is that any variable can be used as input or output at any time in the computation. a variable can

```
(assert-constraint
  ((= W    60)
   (= VS2 100)
   (= R1   [ 5, 25])
   (= L    [0.1, 2])
   (= R2   [10, 50])
   (= IS1 ([1.583,  1.970]
           L [287.044, 291.736]))
   (= IR2 ([0.386,  0.536]
           L [211.371, 217.320]))
   (= VR2 ([9.707, 13.360]
           L [211.371, 217.320])))
  sol1)

Series cluster: S2 (1, 0)
...
Component1:
   Parallel cluster: S1 (1, 3)
   Component1:
      Series cluster: S1d (1, 3)
      ...
   Component2:
      Capacitor: Cd (1, 3)
      C  = [0.0019, 0.0034]
      ZC = ([4.9680, 8.9924]
            L 270.0000)
      VC = ([9.7073, 13.3605]
            L [211.3713, 217.3201])
      IC = ([1.4858, 1.9540]
            L [301.3713, 307.3201])
Component2:
   ...
```

Figure 9: Example of Parameter Design

even change its role during one execution. This makes the QPA system ideal for situations where a user knows some desired operating conditions and parameters and wants to design an acceptable range of values for other parameters. In this case, the user can express the desired operating conditions as value constraints. QPA computes the range of values that the rest of the parameters may take on to achieve the goal state.

Figure 9 shows an example of parameter design for the circuit in Figure 7. The idea is that after solving a given control design problem qualitatively, the user can choose the desired circuit structure and then ask that the solution be made more precise by providing further, quantitative constraints on the design. In this example, the user provides values for all parameters but C_d, and some desired values for currents and voltages. A range for C_d and some other circuit variables are computed. This can be further refined, as interval propagation allows us the bridge the gap between qualitative and quantitative solutions in stages, as desired.

Conclusions

QPA has been implemented in Allegro Common Lisp. The system is capable of designing solutions to incremental design problems applicable to linear circuits in sinusoidal steady state. To solve such control design problems, we use a means-ends analysis technique, where the states are represented by a constraint-based model of the circuit, and the operators were derived by first principles from circuit theory. The process of incremental design is completed by performing numerical design of the parameters that the planner qualitatively selects. The numeric computation is performed by constraint propagation. Since our representation is based on intervals, we can determine the value of the parameters with as much precision as provided by the user.

An immediate application of this technique can be found in the area of power systems, where QPA can solve such common problems as phase angle correction, power routing, etc. A power system modeler was developed to test QPA on power system design problems. QPA's solutions to control design problems in the area of power systems contained solutions prescribed by engineers and engineering textbooks (Gönen 1988; Grainger & Stevenson 1994).

More work needs to be done in the determining the complexity of the algorithms used in the circuit modeling and design processes. As we limit the depth of search in incremental design to a relatively small depth, while exponential, the search may remain feasible. The derivation of an expression for complexity would yield a good figure of how scalable this approach is for domains with many operators and more complex design structures. Our testing to date has been on relatively small, linear circuits without active elements (e.g., transistors).

Not much work has been done on the analysis, design, and diagnosis for linear circuits in sinusoidal steady state. QPA compiles and extends the work from earlier efforts in qualitative physics in general, applying state of the art techniques to effectively reason about and modify linear circuits. The present work contributes by extending the range of problems that can be solved by qualitative reasoning and by providing a definition and framework for the solution of incremental design problems.

References

Flores, J. J., and Farley, A. M. 1996. Qualitative phasor analysis. In *Proc. 10th Int. Workshop on Qualitative Reasoning About Physical Systems.*

Flores, J. J. 1996. Hybrid representation constraint propagation. In *In Proceedings of the Nineth International Symposium on Artificial Intelligence*, 340–347.

Flores, J. J. 1997. *Reasoning about Linear Circuits in Sinusoidal Steady State.* Ph.D. Dissertation, University of Oregon.

Gönen, T. 1988. *Modern Power System Analysis.* New York: John Wiley and Sons.

Grainger, J. J., and Stevenson, W. D. 1994. *Power System Analysis.* New York: McGraw-Hill.

Kerr, R. B. 1977. *Electrical Network Science.* Englewood Cliffs, NJ: Prentice-Hall.

Lancaster, G. 1974. *DC and AC Circuits.* Oxford: Calendar Press.

Liu, Z., and Farley, A. 1990. Shifting ontological perspectives in reasoning about physical systems. In *Proc. 8th National Conf. on Artificial Intelligence (AAAI-90).* Menlo Park, Cambridge, London: AAAI Press/The MIT Press.

Walton, A. K. 1987. *Network Analysis and Practice.* Cambridge MA: Cambridge University Press.