A SET OF FORTH WORDS LETS YOU DO NETWORK ANALYSIS

Jens Storjohann
Deutsches Elektronen-Synchrotron DESY
Notkestr. 85
D-2000 Hamburg 50
W.-Germany

ABSTRACT

Most basic tasks in electrical network analysis are calculations of driving point impedances of one-ports and the transfer functions of two-ports.

The program NAOMI (Netzwerkanalyse ohne Matrizeninversion / network analysis without matrix inversion) uses a simple language to describe components and networks and to immediately invoke suitable arithmetic operations.

The following example calculates the admittance of a one-port built by connecting two series RC-circuits in parallel.

```
60 HZ IS-FREQUENCY   R1 C1 ++   R2 C2 ++   ||
```

This approach avoids in contrast to systems like SPICE numbering of nodes and storing and inverting large matrices.

INTRODUCTION

Right at the beginning of any instruction in electrical network theory you learn how to calculate the impedance of a one-port (or two-terminal network) which consists of one-ports in series. You have to add their impedances. Similarly you calculate the admittance of one-ports in parallel as the sum of their admittances.

Even quite complicated circuits can be built by connecting components or two-terminal subnetworks in parallel or series. Engineers use this approach very often. But normally they work by symbolic manipulations of rational functions and then evaluate them for the desired frequencies by hand or computer.

The problem itself would be treatable in "pocket-calculator style" if the appropriate types of data and operations were implemented and could be used simply.

Most common network programs expect the numbers of nodes and the components connecting them as input. The user chooses the results to be displayed from a set of options. The program does a topological analysis of the network and calculates a complete description (state vector) of the network behavior by some kind of matrix analysis.

This approach has several disadvantages: The user has no access to the internal working of the program. Therefore he has to be content with the limited set of options included in the program. The topological analysis is done even in cases where the structure could be recognized easily by inspection. Then a lot of state variables are calculated only to boil them down to one single quantity like input current or output voltage. This approach is quite universal: You one can calculate even the time behavior of nonlinear circuits, but you have to pay a penalty in the form of large memory requirements, slow execution, and inflexibility.

The engineer, especially in power engineering, has to do a lot of simple AC analysis of passive linear circuits. He does not want to pay the price for these universal programs. NAOMI was developped with these simple applications in mind.

HOW NAOMI WORKS

The input language ist the first thing you learn about a network analysis program. I decided to let Forth words be the elements of the input language. They can be invoked interactively and exchange data mainly via the stack. To have all "electrotechnical items" represented by tagged data. turned out to be quite comfortable. So a capacitor is represented by a floating point number and a 16-bit number as a tag on top of it. A calculated impedance is represented as a floating point complex number and a tag.

The most important high level words are ++ ,|| , and //. The word ++ ("in series") takes any two one-ports from the stack and leaves the impedance of their series connection as a complex number plus a tag. If the operator's ++ operands are admittances their reciprocal values will be calculated. If they are components like inductors their impedances will be calculated by multiplication with jω. The appropriate decisions how to calculate the impedance as the required standard representation is done in the word ->Z ("make impedance"). It expects to find any one-port i.e. a resistor, capacitor, inductor, impedance, or admittance on stack and leaves its impedance with the appropriate tag. An item of another type e.g. a voltage would cause an error message and execute QUIT. The total impedance is calculated by adding two complex numbers.

The operation of || ("in parallel") is analogous to ++. The one-ports on stack are converted to admittances by the word ->Y ("make admittance") first and then added.

The word // is used to calculate the transfer function of simple voltage dividers and ladder networks which can be looked upon as chained dividers. It accepts a voltage and two one-ports and leaves the divided voltage and the output admittance of the divider. An example is shown. on the application screen.

The user has never to worry whether the impedance or admittance of e.g. an inductor, which should be connected to another circuit, has been calculated. The tagging guarantees that the necessary transformations to suitable representations can be invoked transparently. That means he can always think as if he has real two-terminal networks on the stack which he can solder together with operators like ++.

FURTHER DETAILS

The internal calculations are done with floating point numbers. In my Forth system from Computer One (similar to LMI Forth) running on the Sinclair QL an extensive set of floating point words is implemented, and even the graphics facilities use them.

This program should be usable almost without any knowledge of Forth. Therefore I decided to restrict the "normal" input of parameters to the form <integer> <unit> i.e. 100 KHZ. The operators for the units like HZ, KHZ, MEGAHZ take an integer number from the stack, change it to floating point format and put a tag on the stack. This approach avoids all complications of the input of floating point numbers. The user can use NAOMI words inside colon definitions almost without restrictions. Otherwise one would have had to introduce concepts like state-smart, execute-only or more difficult Forth words like [COMPILE].

The set of floating point words was supplemented with a few words with obvious meanings like F. and F>R. A set of words for floating point complex numbers was written in high level using the floating point words. They are prefixed with FC like in FC*. Their implementations are not listed here.

For ease of use the defining word COMPONENT was made to work like a glorified CONSTANT. In compiling it takes a single tagged floating point number from the stack. The words defined by it then puts this "component" on stack.

The whole system including the complex floating point words occupies about two kilobytes. Extensive use is made of the return stack to store intermediate results.

REMARKS AND FUTURE

Work is under way to implement operations on two-ports using their ABCD matrices and further utilities like delta-wye transformations. My experience showed me that the prospective users want a very extensive system with lots of options already coded because they do not realize that Forth/NAOMI is easily extensible.

Before I wrote the Forth version I had already written NAOMI, including two-ports, in Fortran. There I imitated the Forth style by having a stack in a named COMMON block.

Paul Penfield from MIT embedded a network description and analysis system MARTHA in an APL environment (see his article in R. W. Jensen and L. P. McNamee (eds.), Handbook of Circuit Analysis Languages and Techniques; Prentice Hall, 1976). Both, MARTHA and NAOMI, work in a similar spirit; but I hope that the advantages of the Forth approach will make NAOMI more popular than her older distinguished, but heavyweight sister MARTHA.

```
Screen # 12
 ( NAOMI Version 0.1  z" y" abcd" ... ."wrong  x.      jst10'85)
 (                                                              )
  11 constant z"      99 constant y"       999 constant abcd"
   1 constant ohm"     2 constant farad"     3 constant henry"
   4 constant volt"    5 constant ampere"
   6 constant hertz"   7 constant second"    8 constant ratio"
 : ."wrong ." wrong type on stack" ;
 : x. (  print routine for tagged data: x - )
   case  z"   of fc. ." Ohm"  endof      y" of fc. ." S"   endof
        abcd" of fc. fc. fc. fc. ." ABCD"  endof
         ohm" of f.  ." Ohm"  endof  farad" of  f. ." F"   endof
       henry" of f.  ." H"    endof   volt" of fc. ." V"   endof
      ampere" of fc. ." A"    endof  hertz" of  f. ." Hz"  endof
      second" of f.  ." s"    endof  ratio" of  fc.        endof
      ." no legal tag found" quit
   endcase ;                                                  -->

Screen # 13
 ( NAOMI VERSION 0.1: units: ohm kohm ... ma            jst10'85)
 ( <unit> : n - fn tag   or <unit> : n  - f0 fn tag             )
 : ohm  float ohm"             ;  : kohm  float kilo ohm" ;
 : megaohm  float mega ohm"    ;
 : f  float farad"             ;  : mf  float milli farad" ;
 : myf  float mikro farad"     ;  : nf  float nano farad"  ;
 : pf   float pico  farad"     ;
 : h    float henry"           ;  : mh  float milli henry" ;
 : myh  float mikro henry"     ;
 : hz   float hertz"           ;  : khz  float kilo hertz" ;
 : megahz  float mega hertz"   ;  : ghz  float giga hertz" ;
 : v  float 0 float volt"      ;  : kv  float kilo 0 float volt" ;
 : mv float milli 0 float volt" ;
 : a  float 0 float ampere"  ; : ka float kilo 0 float ampere" ;
 : ma  float milli 0 float ampere" ;
                                                              -->

Screen # 14
 ( NAOMI Version 0.1: omega is-frequency jomf ...       jst10'85)
 (                                                              )
 fvariable omega
 f# 314.1592 omega f! ( omega preset to European line frequency )
 : is-frequency ( fn tag - )
              hertz" = if   f# 6.283184 f* omega f!
                       else ."wrong  quit
                       then ;
 : jomf   ( fn - f0 fn )
      omega  f@ f* 0 float fswap ;
 : 1/jomf ( fn - f0 fn )
       -1 float fswap omega f@ f* f/ 0 float fswap ;
 : component ( loading: fn tag - ;  executing: - fn tag        )
              create [ f#bytes 2/ 2+ ] literal 1 do  ,  loop
              does> dup 1 - swap [ f#bytes ] literal +
              do i @ -2 +loop ;                               -->
ok
```

ok

```
Screen # 15
( Naomi Version 01: ->z ->y                                   jst10'85)
(  ->z "make impedance"  ->y "make admittance" ,                      )
: ->z   (  any tagged one-port - impedance tag )
        case z"      of ( done) endof
             y"      of fc1/z   endof
             farad"  of 1/jomf  endof
             henry"  of jomf    endof
             ohm"    of 0 float endof
                        ."wrong  quit   endcase z" ;
: ->y   (  any tagged one-port - admittance tag )
        case y"      of ( done)  endof
             z"      of fc1/z    endof
             farad"  of jomf     endof
             henry"  of 1/jomf   endof
             ohm"    of 1 float fswap f/  0 float endof
                        ."wrong  quit   endcase y"  ;        -->
Screen # 16
( NAOMI Version 0.1:  ++ ¦¦ //                               jst10'85 )
(                                                                     )
: ++    ( connects two one-ports in series: 1port 1port - 1port)
        ->z drop fc>r  ->z drop fcr>  fc+ z" ;
: ¦¦  ( connects two one-ports in parallel: 1port 1port - 1port)
        ->y drop fc>r  ->y drop fcr>  fc+ y" ;
: //    ( voltage divider: voltage 1pa 1pb - div.-volt. 1pa¦¦1pb)
        ( The divided voltage and the admittance seen from the )
        ( output of the divider is left on stack.              )
        ->y drop fc>r  ->y drop fcdup fcr>
        fc+ fcdup fc>r  fc/ fc>r  volt" =
        if    fcr> fc* volt"
        else  fcr> fcr> ."wrong quit
        then fcr> y" ;


Screen # 18
( NAOMI Version 0.1: application program                      jst10'85)
 10 kohm component r1
  5 kohm component r2
 10 myf  component c1
  5 myf  component c2
: one-port r1 c1 ¦¦ r2 c2 ¦¦ ++ ;
: run1 20 10 do
                    i khz is-frequency  i khz x.
                    one-port ."   " x. cr
                 loop ;
: divider ( ladder network )
     1 v r1 c1 // r2 ++ c2 // ;
: run2 11  1 do
                    i khz is-frequency  i khz x.
                    divider  ."   "  drop fdrop fdrop x. cr
                 loop ;
ok
```