

Translating SPICE Models to VHDL-AMS

Ramesh Mayiladuthurai¹ and Harold W. Carter²
Electronic Design Automation Research Center
Mail Location 30
University of Cincinnati
Cincinnati, OH 45221-0030

Abstract

Analog and mixed-signal circuits involving only a few discrete gates are usually modeled using some version of SPICE to support simulation and behavioral analysis. With the emergence of the VHDL-AMS mixed-signal description language, there exists a need to convert many of these SPICE models into VHDL-AMS models while preserving the function and accuracy observed using SPICE. This paper describes a technique for easily converting SPICE component declarations into VHDL-AMS statements. Explicit mappings from SPICE 3 component types to both structural and behavioral VHDL-AMS forms are presented for all elementary components and sources. Device and transmission line models are not addressed. A brief description is also given of a SPICE-to-VHDL-AMS translator software program which implements the techniques.

1.0 Introduction

The ubiquitous use of SPICE along with libraries of components described in the SPICE language[1] forces any new analog description language to ensure SPICE descriptions can be processed using the new language. Thus, the emerging VHDL-AMS mixed-signal language[3] must provide a semantic relationship with the semantics of the SPICE, and, through the use of an appropriate translator, map SPICE descriptions to VHDL-AMS descriptions. This paper presents the process, mappings, and an implementation approach for such a translator.

SPICE is a electronic component-based declarative language. VHDL-AMS has the capability of modeling circuits at either a structural component-based view or a behavioral view or some mixture of the two. Therefore, the need is to translate SPICE models in a declarative, component-based form into VHDL-AMS models at either a component-based structural form, or a behavioral form, without compromising accuracy.

We present some possible mappings between SPICE component representations and VHDL-AMS representations. Note that since a VHDL-AMS simulator does not yet exist, these mappings have not been verified. Further, the VHDL-AMS language is still evolving although the basic VHDL-AMS concepts and syntactical structures used in the paper are fairly well defined at this time. But be aware that any of these syntactical or underlying semantic forms could possibly change as the VHDL-AMS language proceeds to (hopefully) become a IEEE standard.

Not all of the SPICE language is presented in this paper. Because of space considerations, only basic elements and sources are discussed here. Also, we have chosen to present the specifics of SPICE3[2], a version of SPICE written in C by the University of California, Berkeley.

An excellent treatise by Peter Liegbmann[4] prepared for the IEEE 1076.1 Language Architecture Team on SPICE related issues in VHDL-A provides most of the specific background for this paper. The reader is encouraged to read that white paper, particularly for further details on dependent sources and semiconductor device models.

1. <http://www.ececs.uc.edu/~rmayilad>, rmayilad@ececs.uc.edu

2. <http://www.ececs.uc.edu/~hcarter>, hal.carter@uc.edu

2.0 SPICE 3

The general form of component SPICE declarations is

```
name node1 node2 value
```

where name is a unique name of the component instance, node1 and node2 are integers representing the terminals of the component, and value is either a constant, a sequence of arguments, or a function. An example of a typical SPICE3 model is shown in Figure 1.

```
*CIRCUIT TITLE NETWORK
.PRINT TRAN V(4)
.TRAN 10NS 0.1US 0US 0.1NS
R1 1 2 5
R2 2 0 5
L1 2 3 0.002UH
C1 3 0 0.002UF
C2 3 2 1MF
V1 1 0 PULSE (0V 5V 0NS 1FS 1FS 50NS 100NS)
.END
```

FIGURE 1. Example SPICE3 circuit.

The names of the components begin with a letter which identifies the component type (R for resistance, C for capacitance, etc.). The lines beginning with a period (.) are simulation directives (e.g., .TRAN requests a transient analysis simulation), and a line beginning with an asterisk (*) is a comment.

3.0 VHDL-AMS

VHDL-AMS as a mixed-signal language where additional syntax and semantics have been added to VHDL to support analog modeling and continuous time simulation. At present, VHDL-AMS is time based only, although some consideration is being given by the language designers to incorporate small-signal continuous and discrete frequency semantics. Thus, only transient analysis SPICE models can be translated to VHDL-AMS. The capabilities added to VHDL to create the present draft form of VHDL-AMS are:

1. Two analog objects: Quantities, which are currents and voltages in electrical circuits, and Terminals, which are equipotential nodes in a circuit. Thus, a current quantity is defined as being THROUGH terminal1 TO terminal 2, and a voltage quantity is defined as being ACROSS terminal1, terminal2. The association of a current being a THROUGH variable or voltage being an ACROSS variable is explicitly defined in VHDL-AMS using a NATURE statement. For example, Figure 2 shows a part of a VHDL-AMS model where two electrical nodes, t1 and t2 are declared, and a voltage v and current i2 is also declared. VHDL-AMS keywords are capitalized.

```
SUBTYPE voltage is real;
SUBTYPE current is real;
NATURE electrical IS voltage ACROSS current THROUGH;
TERMINAL t1, t2: electrical;
QUANTITY v ACROSS i1,i2 THROUGH t1 TO t2
```

FIGURE 2. Example VHDL-AMS fragment illustrating analog quantity and terminal declarations.[3]

2. Simultaneous Statements: These are statements with syntax `<expressions> == <expression>`; which are elaborated to differential algebraic equations for simulation. The statements in Figure 4 with “==” in them are simultaneous statements each one representing, in this case, the constitutive equation for a component in the circuit. This example is the behavioral VHDL-AMS model translated from the SPICE model shown in Figure 1.

4.0 Example of VHDL_AMS Structural and Behavioral Model

We present both a complete behavioral and structural model of the VHDL-AMS code as translated from the SPICE model shown in Figure 1. A package of common electrical definitions [3] for both VHDL-AMS models is shown Figure 3. The behavioral VHDL-AMS model is shown in Figure 4 while the structural model is shown in Figure 5.

```

PACKAGE electrical_system IS
    SUBTYPE voltage IS REAL;
    SUBTYPE current IS REAL;
    NATURE electrical IS voltage ACROSS current THROUGH;
    NATURE electrical_array IS array (INTEGER RANGE <> ) electrical;
    ALIAS ground IS electrical'reference;
    ALIAS voltage_vector IS electrical_array'across;
    ALIAS current_vector IS electrical_array'through;
    TYPE real_vector IS ARRAY (INTEGER RANGE <>) OF REAL;
END electrical_system;

```

FIGURE 3. Package electrical_system used by the VHDL-AMS models in Figures 4 and 5.[3]

```

library ieee;
use ieee.std_logic_1164.all;
use work.package_electrical.all;

ENTITY RLC IS
    GENERIC (R1:Real:=5; R2:Real:=5; L1:Real:=0.002e-6;
            C1:Real:=0.002e-6; C2:Real:= 1e-3);
END RLC;

ARCHITECTURE behavior OF RLC IS
    TERMINAL N1, N2, N3;
    QUANTITY vR1 ACROSS iR1 THROUGH N1 TO N2;
    QUANTITY vR2 ACROSS iR2 THROUGH N2 TO ground;
    QUANTITY vC1 ACROSS iC1 THROUGH N3 TO ground;
    QUANTITY vC2 ACROSS iC2 THROUGH N2 TO N3;
    QUANTITY vL1 ACROSS iL1 THROUGH N2 TO N3;
    QUANTITY vV1 ACROSS iV1 THROUGH N1 TO ground;
BEGIN
    r1: vR1 == R1 * iR1;
    r2: vR2 == R2 * iR2;
    l1: vL1 == L1 * iL1'dot;
    c1: iC1 == C1 * vC1'dot;
    c2: iC2 == C2 * vC2'dot;
    r3: vR3 == R3 * iR3;
    v1: vV1 == pulse (0, 5, 0 NS 1 FS 1 FS 50 NS 100 NS);
END behavior;

```

FIGURE 4. Behavioral View of the VHDL-AMS model for the SPICE3 code shown in Figure 1. See Figure for pulse() function in VHDL-AMS. Package electrical_system is from [3].

```

ARCHITECTURE structure OF RLC IS
    SIGNAL N1, N2, N3: ELECTRICAL;
    COMPONENT resistor GENERIC(rR: REAL);
        PORT (rR1, rR2: ELECTRICAL);
    END COMPONENT;

    COMPONENT inductor GENERIC(lL: REAL);
        PORT (lL1, lL2: ELECTRICAL);
    END COMPONENT;

    COMPONENT capacitor GENERIC(cC: REAL);
        PORT (cC1, cC2: ELECTRICAL);
    END COMPONENT;

BEGIN
    r1:resistor GENERIC MAP (rR=>R1) PORT MAP(rR1=>N1, rR2=>N2);
    r2:resistor GENERIC MAP (rR=>R2) PORT MAP(rR1=>N2, rR2=>ground);
    l1:inductor GENERIC MAP (lL=>L1) PORT MAP(lL1=>N2, lL2=>N3);
    c1:capacitor GENERIC MAP (cC=>C1) PORT MAP(cC1=>N3, cC2=>ground);
    c2:capacitor GENERIC MAP (cC=>C2) PORT MAP(cC1=>N3, cC2=>N2);
    v1:vpulse GENERIC MAP(v1=>0, v2=>5, td=>0, tr=>1 fs,
        tf=>1 ns,pw=>50 ns, per=>100 ns);
        PORT MAP (n1, ground);

END structure;

```

FIGURE 5. Structural architecture of the entity shown in Figure 3 and SPICE3 code shown in Figure 1.

5.0 Mapping SPICE to VHDL-A

We present the explicit mapping from SPICE statements to VHDL-AMS code for both structural and behavioral representations. Table 1 contains the mappings for two-terminal passive elements while Table 2 gives similar information for independent sources. Mappings for dependent sources, transmission line models, and semiconductor device models are not shown due to space limitations. The reader is referred to [4] for further information on mapping these particular elements.

The following definitions and notations are used in Tables 1 and 2.

1. n1 and n2 or n+ and n- refer to the component terminals in a SPICE statement. They map to n1 and n2, or nplus and nminus, respectively, in the VHDL-AMS statements. In VHDL-AMS, n1, n2, nplus, and nminus are declared as terminal types.
2. The braces ([and]) are metasymbols used to denote optional syntax.
3. Capital characters or names are literal symbols. They appear in statements exactly as shown.
4. Variables in the SPICE statements are defined in [1]. They usually translate to similar variable names in the VHDL-AMS statements. For example, for the sinusoidal source component, the variables vo, va, freq, td, and theta are defined as the offset and amplitude voltage or current, the frequency, delay, and damping factor, respectively. They are used in exactly the same way in the VHDL-AMS models.
5. ANOW is the current analog time.

TABLE 1. Conversion of elementary elements from SPICE 3 to VHDL-AMS

Item	SPICE Representation	VHDL-AMS Structural Instantiation	VHDL-AMS Behavioral Representation
Resistors	Rname n1 n2 resistance	Rname: resistor generic map (R=> resistance); port map (n1, n2);	quantity re across ri through n1 to n2; begin re == ri * resistance; end;
Capacitors	Cname n1 n2 capacitance [IC=initvoltage]	Cname: capacitor generic map (C=> capacitance, IC=> initvoltage); port map (n1, n2);	quantity ce across ci through n1 to n2; begin ci == capacitance * ce'dot; end;
Inductors	Lname n+ n- inductance [IC=initcurrent]	Lname: inductor generic map (L=> inductance, IC=>initcurrent); port map (nplus, nminus);	quantity le across li through n1 to n2; begin le == inductance * li'dot; end;
Semiconductor Resistors	Rname n1 n2 Mname L=length [W=width] [TEMP=temperature]	Rname: semiresistor generic map (L=>length, W=>width, TEMP=>temperature); port map (n1, n2);	quantity re across ri through n1 to n2; begin constant rnom := RSHEET * (L-NARROW) / (W-NARROW); tdiff := temprature-TNOM; resistance := rnom * (1+ TC1 * tdiff + TC2 * tdiff * tdiff); re == ri * resistance; end;
Semiconductor Capacitors	Cname n1 n2 Mname L=length [W=width] [IC=initcurrent]	Cname: semicapacitor generic map (L=>length, W=>width, CJ=>junction_bottom_cap, CJSW=> junction_sidewall_cap); DEFW=> default_device_width, port map(n1, n2);	quantity ce across ci through n1 to n2; begin constant lactual := L-NARROW; constant wactual := W-NARROW; constant cap := CJ * lactual/wactual + 2*CISW * (lactual + wactual); ci == cap * ce'dot; end;

TABLE 2. Conversion of independent sources from SPICE 3 to VHDL-AMS.

Item	SPICE Representation	VHDL-AMS Structural Instantiation	VHDL-AMS Behavioral Representation
DC Voltage Source	Vname n+ n- [DC] voltage	Vname: vdcsource generic map (voltage); port map (nplus, nminus);	quantity dcv across nminus to nplus; begin dcv == voltage; end;
DC Current Source	Iname n+ n- [DC] current	Iname: idcsource generic map (current); port map (nplus, nminus);	quantity dci across nminus to nplus begin dci == current; end;
AC Voltage Source	Vname n+ n- AC [voltage [phase]]	Vname: vacsource generic map (voltage, phase); port map (nplus, nminus);	Frequency not defined yet by IEEE 1076.1 committee
AC Current Source	Iname n+ n- AC [current [phase]]	Iname: iacsource generic map (current, phase); port map (nplus, nminus);	Frequency not defined yet by IEEE 1076.1 committee
Distortion Voltage Source	Vname n+ n- [DISTOF1[voltage [phase]] [DISTOF1 [voltage[phase]]]	Vname: distvsource generic map (voltage1, phase1); port map (voltage2, phase2);	Distortion not defined yet by IEEE 1076.1 committee
Distortion Current Source	Iname n+ n- [DISTOF1[current [phase]] [DISTOF1 [current [phase]]]	Iname: distvsource generic map (current1, phase1); port map (current2, phase2);	Distortion not defined yet by IEEE 1076.1 committee
Pulse Source (only voltage source shown)	Vname n+ n- PULSE (v1 v2 td tr tf pw per)	Vname: vpulse generic map(v1, v2, td, tr, tf, pw, per); port map (nplus, nminus);	quantity pv across nminus to nplus; begin pv == pulse (v1, v2, td, tr, tf, pw, per); end; (see Fig. 6 for definition of pulse())
Sinusoidal Source Only voltage source shown	Vname n+ n- SIN (vo va freq td theta)	Vname: vsine generic map(vo, va, freq, td, theta); port map (nplus, nminus);	quantity sv across nminus to nplus; variable diff: real; begin diff := ANOW-td; if diff < 0.0 then sv == vo; else sv == vo+ va*exp(-theta*diff)* sin(2*pi*freq*diff); end if; end;

TABLE 2. Conversion of independent sources from SPICE 3 to VHDL-AMS.

Item	SPICE Representation	VHDL-AMS Structural Instantiation	VHDL-AMS Behavioral Representation
Exponential Source (only voltage source shown)	Vname n+ n- EXP(v1 v2 td1 tau1 td2 tau2)	Vname: vexp generic map (v1, v2, td1, tau1, td2, tau2); port map (nplus, nminus);	quantity ev across nminus to nplus; variable vt, vt1: voltage; begin vt := (v2-v1)* (1-exp(td1-ANOW)/tau1); vt1 := v1+vt; if ANOW < td1 then ev == v1; elseif ANOW <= td2 then ev == vt1; else ev == vt1-vt; end;
Piece-Wise Linear Source (only voltage source shown)	Vname n+ n- PWL(t1 v1 [t2 v2 t3 v3 t4 v4 ...])	Vname: vpw generic map (t1, v1, [t2, v2, t3, v3, ...]); port map (nplus, nminus);	quantity pv across nminus to nplus; variable v: voltage; begin if ANOW < t1 then v := v1; elseif NOW < t2 then v := v1+(t2-ANOW)* (v2-v1)/(t2-t1); elseif NOT < t3 then v := v2+(t3-ANOW)* (v3-v2)/t3-t2); else NOT < tn then v := v[n-1]+(tn-ANOW)* (v3-v[n-1])/(t3-t[n-1]) end if; end;

```

function pulse (v1, v2, td, tr, tf, variable pw, per) return real is
variable t: real;
begin
    if ANOW < td then return v1; end if;
    t := (now - td) mod per;
    if t < tr then return v1+t*(v2-v1)/tr;
    elseif t < tr+pw then return v2;
    elseif (t < tr+pw+tf) then return v2+(t-tr-pw)*(v1-v2)/tf;
    else return v1;
    end if;
end pulse;

```

FIGURE 6. VHDL-AMS function for determining the voltage value of a pulse source as a function of time.

6.0 The University of Cincinnati Translator

A translator to convert SPICE models to VHDL-AMS models is currently being created. At present it converts all SPICE code into structural VHDL-AMS syntax. A library of VHDL-AMS components is being created so that, when a VHDL-AMS simulator is available, converted code will simulate properly. At that time, final verification of the correctness of the translation process can be confirmed. .

The first step towards developing a translator is to effectively parse the source language. This means that the source language should be free of syntactic errors. With this objective in mind a LL(2) grammar (left linear with two tokens lookahead) was written for spice. The SPICE grammar is LL(2). PCCTS [5] was used to create the parser from the SPICE BNF productions.

The internal intermediate representation (called Spice Intermediate Representation (SIR)) defines a common view for internal information during the translation process. The SIR specification defines the minimal set of interface constraints needed for effective inter-operability. The SIR uses a collection of objects (structure) instances linked by pointers to represent the required information. The collection of objects represents generalized abstract syntax trees(AST) which are traversed by action routines which generates the equivalent VHDL-AMS code.

7.0 Conclusions

Converting SPICE descriptions to VHDL-AMS descriptions is rather straightforward for simple components and independent sources. By using the mappings described here, one should be able to easily convert the basic elements in a SPICE description to VHDL-AMS descriptions. An automatic translator is being created which should ease the manual process of performing the conversion.

References

1. Vladimirescu, Andrei, *The SPICE Book*, John Wiley & Sons, 1994.
2. T. Quarles, A.R. Newton, D. O. Pederson, A. Sangiovanni-Vincentelli, *SPICE3 Version 3f3 User's Manual*, May 1993.
3. Kenneth Bakalar, Ernst Christen, "Opal Proposal - VHDL-A: Analog and Mixed Signal Extensions for VHDL," unpublished papers submitted to IEEE 1076.1 Working Group, Jan 1996. (Updated sections of this proposal, available via <http://vhdl.org/vi/analog/wwwpages/1076.1-proposal/>)
4. Liebmann, Paul, working paper on SPICE to VHDL-A, Version 1.2, 17 March 1995. (Available via <http://vhdl.org/vi/analog/wwwpages/1076.1-proposal/spice.html>)
5. Parr, T. J., Dietz, H. G., and Cohen, W. E., "PCCTS Reference Manual," Version 1.00, Aug 1991. (Available via <ftp://ftp.uu.net/languages/tools/pccts>)
6. Martin, Dale, and Philip A. Wilsey, "SAVANT: An Extensible Object-Oriented Intermediate for VHDL," VHDL User's Group Conference, Spring 1996, 275-281, Mar 1996.