THE USE OF HARDWARE DESCRIPTION LANGUAGES IN THE DEVELOPMENT OF MICROELECTROMECHANICAL SYSTEMS

Dennis Gibson * *Carla Purdy* † *H. Carter*

ECECS Department
University of Cincinnati

ABSTRACT

We describe how hardware description languages (HDL's) can be used to support the design and simulation of systems incorporating components interacting in multiple energy domains. This methodology encourages emphasis on higher-level system design while providing performance targets for lower-level component design. It can also be made independent of the actual simulation engine(special-purpose mixed domain simulators or separate simulators for different energy domains). This approach supports modular, hierarchical design and allows strategies such as rapid prototyping and component reuse to be applied to multidomain systems, thus speeding the development of such systems for commercial use.

1. INTRODUCTION

Today's state-of-the-art digital design systems typically support many different levels of design activity. At the highest levels in the design process, structural system descriptions and high-level language component descriptions promote system-level thinking, while at lower levels sophisticated simulation tools provide detailed descriptions of physical behavior. Fabrication processes are well-characterized, and parameters from these processes are routinely collected for use in low-level simulations. These differing levels of design and simulation activity are tied together by sophisticated analysis, synthesis and verification tools. Thus digital systems can be rapidly prototyped and modified, and large-scale commercialization of these systems has become a reality. In contrast, the current state of the art for production of multi-energy domain systems is much more primitive. Here we will use the term MEMS to mean any multi-energy domain system, including but not limited to electromechanical systems. MEMS fabrication processes are extremely varied and, in consequence, much less standardized [7]. Robust simulation tools which can handle interacting energy domains efficiently, such as MEMCAD, IntelliCAD, and the CFDRC tools, are just beginning to become available[10], as are some tools for use in university research[9]. MEMS component libraries do exist, but they tend to consist of individual components which must be laboriously integrated into working systems[6][11]. And techniques for macro-level MEMS design are almost nonexistent. Recently, however, researchers have begun to succeed in extending digital design methodologies to the analog domain, with "mixed signal" design and simulation tools being developed. One such tool is VHDL-AMS (VLSI Hardware Description Language with Analog and Mixed Signal extensions)[1]. Here we show, using some simple examples, how this language can in fact be used to support modular design, component reuse, and hierarchical development, not only of mixed signal systems but also of multi-domain, or MEMS, systems. By incorporating VHDL-AMS into emerging MEMS design systems, it should eventually be possible to develop design methodologies similar to current digital design methodologies and hence to more easily achieve commercialization for these much more complex systems. Thus the many benefits which MEMS systems promise[8] will become realizable. Our remarks also apply, of course, to similar HDL's such as Verilog and its extensions.

2. THE VHDL-AMS LANGUAGE

VHDL-AMS is a structured programming language with origins in the Ada language. VHDL-AMS is an extension to the analog domain of VHDL, which was originally designed for simulation of digital circuits and systems and which has also been used extensively in digital circuit synthesis. As in VHDL, a description consists of an entity specifying inputs and outputs, as well as associated "architectures" describing, possibly in different styles and at different levels of abstraction, how the inputs will be transformed into the outputs. VHDL-AMS is, in fact, not limited to the electrical domain. It has the ability to handle any algebraic or ordinary differential equation or any system of such equations. VHDL-AMS does not yet incorporate either finite element methods or distributed parameter calculations. However, for a given problem, it is possible to define a set of elements and their associated equations offline and, using piecewise linear approximations, to deal not only with lumped parameters but also with distributed parameters. In addition, because it must deal with both digital and analog "events", VHDL-AMS can handle both discrete and continuous phenomena. Currently, complex VHDL-AMS SPICE type models for transistors and other electrical components are becoming available[5]. But VHDL-AMS can also be used at much higher levels of abstraction to specify connections between system components. A VHDL-AMS "compiler" can output intermediate code for any simulator or simulators, as long as certain rules are followed. For example, the compilation step of the SEAMS simulator used for this work[12] has options to output C++, VHDL or code for parallel execution.

*Supported by State of Ohio DAGSI program and the United States Air Force, contract no. F33615-96-2-1945. Thanks to Jeremy Law for work on FEA models

†Supported by State of Ohio DAGSI program and the United States Air Force, contract no. F33615-96-2-1945.

3. SOME EXAMPLES

As a simple example of how VHDL-AMS easily handles multiple energy domains, consider a thermistor, used to translate between temperature and electrical resistance. The Steinhart and Hart equation,

$$T = (a + b(\ln R) + c(\ln R)^3)^{-1} \quad (1)$$

gives T as a function of resistance, where a,b,c depend on physical properties of the thermistor. A VHDL-AMS program implementing this calculation is given in Figure 1. Values for a,b,c are taken from [13]. In this example, the behavior of a component has been specified. This component description can be used at the system level , along with other components, to design a multi-component part. It can also be used by the component designer as a specification for the actual component behavior. Thus this description provides a separation between the task of the system designer, i.e., fitting it into a complex system, and the task of the component designer, i.e., making the component with this functionality. Here we assume no other relevant interactions with other components or with the environment.

```
-----------------------------------------------
package electricalsystem is
  nature electrical is real across real through;
  function sin (x: real) return real;
  function exp (x: real) return real;
  function tanh (x: real)  return real;
  function sqrt (x: real) return real;
  function pow (x,y: real) return real;
  function log (x:real) return real;
end package electricalsystem;
use work.electricalsystem.all;
--------------------------------------
entity thermister is
end entity thermister;
--------------------------------------
architecture behavior of thermister is
  quantity t: real;
  constant a: real :=1.4733e-3;
  constant b: real :=2.372e-4;
  constant c: real :=1.074-6;
  quantity denom: real;
  quantity logr: real;
  quantity r:real ;
begin
  denom == a + b*logr + c*(logr*logr*logr);
  t == 1.0/denom;
  logr == log((real(time'pos(NOW))*1.0e-7)+10.0);
end behavior;
--------------------------------------
```

Figure 1. Simple Thermistor Model

This model does not take into consideration any connections to the outside world. This is noted in the entity having an empty port description. This allows a designer to simplify a model to be able to make rapid changes in the model before more complexity is added. A thermistor may be given an interface to the outside world by defining its inputs and outputs in the entity with the port, as shown in Figure 2.

```
-------------------------------------------
--entity describes interface to "outside world"
--here input is resistance, output temperature
-------------------------------------------
entity thermistor is
port(rin: in real; temp: out real);
end entity thermistor;
-------------------------------------------
--architecture displays desired behavior;  for
--given input resistance, a particular output
--temperature must be obtained
-------------------------------------------
architecture simple of thermistor is
   quantity r: real;
   quantity t: real;
   constant a: real := 1.4733E-3;
   constant b: real := 2.372E-4;
   constant c: real := 1.074E-7;
begin
   b1: break t => 0.0, r => 0.0;
   inputtestbench: process
      --process to read in resistances from input
      --file
      file indata: text open read_mode is
                                        "thermist.in";
      variable linebuf: line;
      variable resist:real;
      variable okay: boolean;
      begin
        while (not (endfile(indata))) loop
           readline(indata,linebuf);
           read (linebuf,resist,okay);
           rin <= resist;
WAIT FOR 1 ns;
        end loop;
    end process;
    initialize: process(rin)
       --code to initialize r to rin from file.
...
     end process;
    calctemp: process
       --Place specific model equations here
...
     end process;
end architecture simple;
-------------------------------------------
```

Figure 2. Thermistor Model with Interface

A more complex example would involve interaction between subcomponents encapsulated in one VHDL-AMS description. As an example of such a system, consider the cantilever beam-capacitor system shown in Figure 3. A behavioral approach to designing such a system was presented in[3]. In [4] several tools for designing and modeling this system, where the beam was treated as a simple spring, were discussed. A simple model for the beam may be expressed as in Figure 4 below.

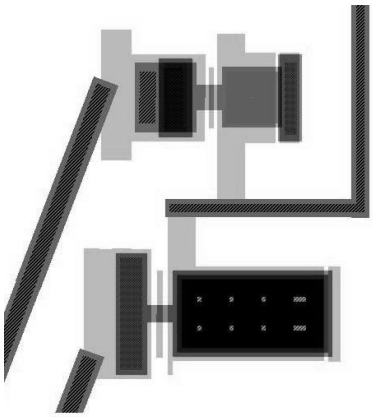

Figure 3. Micro-beam/Capacitor system[3]

```
-----------------------------------
entity cant_beam is
end entity cant_beam;
-----------------------------------
architecture simple of cant_beam is
quantity x: real;
quantity v: real;
quantity a: real;
quantity F: real;
constant Force: real := 1.0e-5;
constant L: real := 80.0e-6;
constant W: real := 20.0e-6;
constant H: real := 2.0e-6;
constant P: real := 2.26E3;
  constant B: real := 3.92e-6;
constant M: real := L*W*H*P;
constant Y: real := 170.0e9;
constant IZ: real:= (W*H*H*H)/12.0;
constant Rigidity: real := Y*IZ;
constant k: real := (3.0*Rigidity)/(L*L*L);
begin
  b1: break v =>0.0, a=>0.0, x=>0.0;
   F == Force;
  spring:  F == M*a + B*v + k*x;
  vel:     v == x'dot;
  accel:   a == v'dot;
end architecture simple;
---------------------------------------
```

Figure 4. Simple Cantilever Beam Model

Once again this is a simple model of the cantilever beam for a given constant force applied. We can calculate the deflection according to the force. Using this model, a more complex model incorporating the electrical domain (the attractive force due to capacitance) is given.

It would be convenient if one could have terminal types which make sense for all the energy domains besides the electrical which VHDL-AMS supplies for us. Recently, work has been done on creating packages which may be incorporated into the models which allow more intuitive interfaces for each energy domain. Figure 5[14] is an example of a capacitve driven cantilever beam using such a set of packages. Note that the port has terminals of type electrical and translational. Translational is a type of mechanical behavior. The beam deflection is controlled by the forces due to capacitance when a voltage is applied. Thus, a higher level of abstraction may be attained by using design strategies of this nature.

```
-----------------------------------------------
USE work.mechanical_systems.all ;
USE work.electrical_systems.all ;

ENTITY sensor IS
  GENERIC (
  M : MASS := 0.16e-9 ;
  B : DAMPING := 4.0e-6 ;
  K : STIFFNESS := 2.6455 ;
  A : real := 2.0e-6*110.0e-6 ;
  d0 : real := 1.5e-6 );
  PORT (TERMINAL proof_mass,ref:translational ;
TERMINAL top_el,mid_el,bot_el:electrical);
END ENTITY sensor ;
-----------------------------------------------
ARCHITECTURE behavioral OF sensor IS
  QUANTITY pos ACROSS force THROUGH proof_mass TO ref;
  QUANTITY Vtm ACROSS Itm THROUGH top_el TO mid_el;
  QUANTITY Vbm ACROSS Ibm THROUGH bot_el TO mid_el;
  QUANTITY vel : VELOCITY ;
  QUANTITY Dtm,Dbm : DISPLACEMENT ;
  QUANTITY Qtm,Qbm : CHARGE ;
  QUANTITY Ctm,Cbm : CAPACITANCE ;
  CONSTANT eps0 : real := 8.85e-12 ;
BEGIN
  -- differential equations for MSD
  vel == pos'dot ;
  force == - K*(pos) - B*(vel) - M*(vel'dot);
  Dtm == d0 + pos ;
  Dbm == d0 - pos ;
 -- equations for capacitive transduction (pickoff)
  Ctm == A*eps0/Dtm ;
  Cbm == A*eps0/Dbm ;
  Qtm == Ctm*Vtm ;
  Qbm == Cbm*Vbm ;
  Itm == Qtm'dot ;
  Ibm == Qbm'dot ;
END ARCHITECTURE behavioral ;
-----------------------------------------------
```

Figure 5. Model of Beam-Capacitor System[14]

The model for Figure 6 brings many of the above models together. the input voltage produces a force on the beam, causing a deflection; the deflection causes a change in capacitance, thereby changing the force on the beam. Thus there is a feedback loop between the electrical component, the capacitor, and the mechanical component, the beam. As we see in Figure 6, this interaction is easily modeled in VHDL-AMS. Note that the force calculation for this cantilever beam system can be done in several ways–as a "spring-mass" system using the previous beam models, or with a finite element method. To this point we have included only the spring solutions. We will go into more detail

in the next section regarding finite element analysis (FEA) models.

```
-----------------------------------------------
entity cantbeamactuator is
port(vin: in real;
deflection, force: out real);
end entity cantbeamactuator;
-----------------------------------------------
architecture choice of cantbeamactuator is
 quantity x: real;
 ....(auxiliary variables)....
 quantity F: real;
 quantity volt: real;
---physical constants (length, width, etc.)
constant B:real :=0.4*sqrt(M*K);
 constant e0: real := 8.85E-12;
begin
-----------------------------------------------
--initialization
-----------------------------------------------
.........
  inputtestbench: process
      file indata: text open read_mode is
            "beaminfo.in";
      variable linebuf: line;
      variable vtemp: real;
      begin
         while (not (endfile(indata))) loop
             readline(indata, linebuf);
             read(linebuf,vtemp);
             vin <= vtemp;
         end loop;
  end process;
  initialize: process(vin)
      begin
         volt := vin;
      end process;
-----------------------------------------------
--force calculation (from capacitor)
-----------------------------------------------
  calc_force: process
      variable area, perm, attractforce: real;
      begin
         area := l*w;
         perm := e0;
         attractforce :=
 (area*perm*volt*volt)/(2.0*x*x);
         f := attractforce;
      end process;
-----------------------------------------------
--input force and output  x value
-----------------------------------------------
.........see Figure 4,5,7 and 8 for options
deflection <= x;
      force <= f;
end architecture choice;
-----------------------------------------------
```

Figure 6.System with choice for beam model

4. VHDL-AMS AND FINITE ELEMENT ANALYSIS

Since VHDL-AMS is an ordinary differential equation solver, it is ideal for solving systems of equations. We demonstrate its effectiveness and ease by modeling the cantilever beam (with the assumption that the beam is uniform). In this case, we only deal with a mechanical beam in which a constant load is applied. Therefore, we are examining the static behavior of the beam. Finite Element Analysis (FEA) on the beam can be done by breaking the beam into multiple elements using the equation:

$$F = KX \tag{2}$$

where F is the vector of forces applied to the assigned elements, K is the stiffness matrix for the beam, and X is the vector of displacements. The K matrix is actually the combination of all the stiffness matrices for each individual element of the beam. Since the endpoints of each element interact with the endpoints of an adjacent element, the K matrix takes this point into consideration. Consider a two element beam (Figure 9) with 2 elements, A and B, where A has endpoints 1 and 2 and B has endpoints 2 and 3. A has its matrix shown as

$$K^A = \begin{bmatrix} k_{11}^A & k_{12}^A \\ k_{21}^A & k_{22}^A \end{bmatrix} \tag{3}$$

And B has its corresponding matrix shown as

$$K^B = \begin{bmatrix} k_{11}^B & k_{12}^B \\ k_{21}^B & k_{22}^B \end{bmatrix} \tag{4}$$

Therefore, the corresponding stiffness matrix for the beam in figure 5 will be

$$K = \begin{bmatrix} k_{11}^A & k_{12}^A & 0 \\ k_{21}^A & k_{22}^A + k_{11}^B & k_{12}^B \\ 0 & k_{21}^B & k_{22}^B \end{bmatrix} \tag{5}$$

We have written a program which generates a VHDL-AMS model for a cantilever beam given the dimensions of the beam and the number of elements. The models are generated rapidly using this program. The program concentrates on filtering out useless terms, such as ignoring the 0 terms in the K matrix and using the boundary condition which assumes that the deflection at the fixed end is 0. In Figure 7 we show a model generated for a beam modeled with only one element:

```
----------------------------------
entity FEABEAM is
end entity FEABEAM;
----------------------------------
architecture behavior of FEABEAM is
  constant F0: real:=1.0e-5;
  constant  L: real := 80.0e-6;
  constant  W: real := 20.0e-6;
  constant  H: real := 2.0e-6;
  constant EZ: real := 170.0e9;
  constant IZ: real := (W*H*H*H)/12.0;
  constant EI: real := EZ*IZ;
  constant L3 :real := L*L*L;
  constant k : real := 3.0*EI/L3;
  constant L2 :real := L*L;
  quantity V1 : real;
```

```
begin
    V1 == F0/k;
end behavior;
---------------------------------
```

Figure 7. One Element Beam Model

The model in Figure 7 was reduced from a two by two matrix to just one element due to the boundary condition that deflection at Node 0 is 0. So column 1 and row 1 of the matrix are eliminated from consideration. Thus the deflection at node 1, called V1, is dependent only on the stiffness of the single element. Figure 8 describes the model generated for a beam partitioned into five finite elements:

```
---------------------------------
entity FEABEAM is
end entity FEABEAM;
---------------------------------
architecture behavior of FEABEAM is
  constant F5: real:=1.0e-5;
  constant  L: real := 80.0e-6/5.0;
  constant  W: real := 20.0e-6;
  constant  H: real := 2.0e-6;
  constant EZ: real := 170.0e9;
  constant IZ: real := (W*H*H*H)/12.0;
  constant EI: real := EZ*IZ;
  constant L3 :real := L*L*L;
  constant k : real := EI/(16.0*L3);
  constant L2 :real := L*L;

  quantity V1: real;
  quantity V2: real;
  quantity V3: real;
  quantity V4: real;
  quantity V5: real;

begin
     V1 == (2.0*V2- V3);
     V2 == 2.0*V1;
     V2 == -2.0*V3 + V4;
     V3 == 2.0*V4 - V5;
     V4 == (k*V5 - F5)/k;
end behavior;
---------------------------------
```

Figure 8. 5 Element Beam Model

The model in Figure 8 results in a system of five equations and five unknowns. Thus, it can be solved quite simply. Thus, FEA may be incorporated into VHDL-AMS designs to aid in the rapid prototyping of systems. In the future, the program will be modified so that the models created will input the parameters via input file. This will aid the designer since the designer will not have to continue to recompile the models each time a set of beam parameters or different number of elements are chosen. As FEA is incorporated into VHDL-AMS, greater flexibility will be given to MEMS designers in the future.

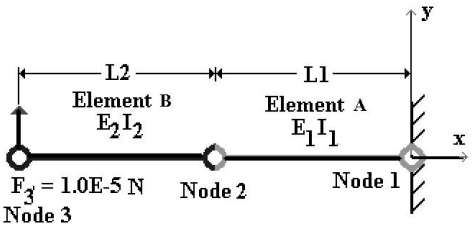

Figure 9. FEA 2 element beam

5. RESULTS OF FEA ON CANTILEVER BEAM

The FEA models generated were simulated on the SEAMS[12] simulator. The results were compared to ANSYS FEA models of an elastic 1-D static beam with the same number of elements. ANSYS was used for the comparison since it has been shown to be highly reliable for these types of measurements. The following tables shows results for a beam with dimensions 80x20x2 microns with a constant load applied to the free end of the beam for both the 3 element and 5 element cases.

.	VHDL-AMS	ANSYS.	Diff.
Node	10^{-6}m	10^{-6}m	%
0	0.0000	0.0000	0.0000
1	0.2509804	0.11155	124.93
2	0.5019608	0.39041	28.57
3	0.7529412	0.75294	0.00547

.	VHDL-AMS	ANSYS.	Diff.
Node	10^{-6}m	10^{-6}m	%
0	0.0000	0.0000	0.0000
1	0.05782588	.042165	37.1419
2	0.1156518	0.15661	26.1466
3	0.1734776	0.32527	46.6665
4	0.4626071	0.53007	12.7271
5	0.7517365	0.75294	0.1598

The errors for the FEA model are somewhat high for beams with fewer elements. However, the error at the end of the beam is negligible. Some of the error may have been introduced with some simplifing assumptions. One may create a more detailed model for FEA; this should be able to reduce the errors at each node to an acceptable level.

6. ADVANTAGES OF THIS APPROACH

The above examples illustrate how VHDL-AMS can be used for system design involving multiple energy domains. The language gives a unified approach to dealing with multiple

domains. VHDL-AMS also allows for the definition of physical types. For example, "time" is a standard VHDL-AMS type. This feature facilitates understanding of domain interactions and also simplifies translations of units between energy domains. VHDL-AMS encourages concentration on system rather than component considerations,encapsulates low-level information,encourages hierarchical, evolutionary design and reuse, and provides component designers with concrete specifications for their work. It is compatible with the use of component libraries which are already being developed. In addition, it provides a comfortable path into the MEMS design area for electrical and computer engineers. It also encourages the development of MEMS tools which interface will with current hardware / software design tools. It encourages decoupling of system design from low-level physical considerations, while providing support for simulations which take physical behavior into account. It should be possible to provide VHDL-AMS interfaces to powerful MEMS simulation systems[10] already under development. VHDL-AMS supports simulation of dynamic system behavior and can model both continuous and discrete events, thereby providing support for the simulation of complex physical systems.

7. POSSIBLE DRAWBACKS

Some limitations of the approach we have described here include the effort required to interface VHDL-AMS to existing simulators, the present lack of graphical interfaces for VHDL-AMS modeling and simulation, and the lack of symbolic computation. Some flaws in VHDL itself have been pointed out[2], which may necessitate some redesign both of VHDL and of VHDL-AMS. But we believe that the advantages listed above are more than sufficient to justify our methodology.

8. CONCLUSIONS AND FUTURE RESEARCH

We have presented examples showing how VHDL-AMS can be used to further the development of commercially viable MEMS systems. We are continuing to develop a library of parts like those presented here. In particular, we are developing a model of interacting arrays of cantilever beams and beam models in which more complicated behaviors (such as fractures, e.g.) can be simulated. In addition, we will expand the FEA capabilities in our programs. Future research will also include extending our modeling techniques to other domains, including the fluidic domain, and intefacing our VHDL-AMS descriptions with domain-specific simulators.

REFERENCES

[1] Design Automation Standards Committee, IEEE Computer Society, IEEE Standard VHDL Language Reference Manual (Integrated with VHDL-AMS Changes), Std. 1076.1, IEEE, 1997.

[2] S.Ghosh and N. Giambiasi, Language barriers in hardware design?, Circuits and Devices, Sept. 1999, 25-40.

[3] D.Gibson, A. Hare, F. Beyette, Jr. and C. Purdy, Design Automation of MEMS Systems using Behavioral Modeling, Ninth Great Lakes Symposium on VLSI, Ann Arbor, Mich. (ed. R.J. Lomax and P. Mazumder), March 1999, pp. 266-269.

[4] D. Gibson and C.Purdy,Extracting Behavioral Data from Physical Descriptions of MEMS for Simulation, Analog Ingegrated Circuits and Signal Processing 20, 1999, 227-238.

[5] V.R. Kasula Srinivas, Modeling Semiconductor Devices using the VHDL-AMS Language, M.S. thesis, University of Cincinnati, 1999.

[6] D.A. Koester, R. Mahadevan, A. Shishkoff, and K.W. Markus, Smart-MUMPS Design Handbook Including MUMPS Introduction and Design Rules (rev. 4), MEMS Technology Applications Center, MCNC, 1996.

[7] K.W. Markus and K.J. Gabriel, MEMS: The Systems Function Revolution, IEEE Computer 32 (10), 1999, 25-31.

[8] Petersen,K.E,."Silicon as a Mechanical Material", IEEE Proceedings, 70 (5), may 1982, 420-457.

[9] Sugar V1.0, http://www-bsac.EECS.Berkeley.edu/ cfm/

[10] S.D. Senturia, Simulation and Design of Microsystems: A 10 Year Perspective, Sensors and Actuators A67, 1998, 1-7.

[11] Tanner Tools User's Manual, Tanner Research, Inc., 1996.

[12] University of Cincinnati, ECECS Dept., Distributed Processing Laboratory, SEAMS Simulator Project, http://www.ececs.uc.edu/ hcarter

[13] YSI Precision Thermistors And Probes, YSI, Incorporated, 1997.

[14] Impact Project, Duke University, http://www.ee.duke.edu/research/IMPACT/.