A Compatibility

This appendix presents some of the practical details that you need to know to be able to use Verilog-A/MS with the simulators available that support it. It starts by describing what constructs to avoid with purely digital models in order to retain compatibility with traditional Verilog-HDL simulators. Then, the issues involved when using SPICE models from Verilog-A/MS are discussed. Finally, it describes specifically how to use Verilog-A and Verilog-AMS in the Cadence simulators: *Spectre* and *AMS Designer*. Spectre was the first simulator to support Verilog-A. It is currently the most popular simulator, and was used to validate each of the Verilog-A models contained in this book. Similarly, AMS Designer was the first simulator to support Verilog-AMS, is currently the most popular simulator that does so, and was used to validate each of the Verilog-AMS models given in this book.

1 Verilog-HDL Compatibility

When describing purely digital modules, it is often desirable to completely avoid the use of the AMS extensions so that the models can be read by a Verilog-HDL simulator without modification. In these cases, one should consciously avoid the following constructs.

1. Explicit parameter type declarations
2. Parameter range limits
3. Analog declarations such as disciplines, natures, branches and ground.
4. Analog processes and the various things associated with them (contributions, analog operators, limiting and stimulus functions, etc.)
5. Analog events, such as *cross*, *timer*, *initial_step* and *final_step*.
6. Analog functions
7. System functions and tasks that were added to support analog or mixed-signal modeling, such as $*abstime*, *analysis*, $*bound_step*, $*discontinuity*, the $*driver_*... functions, $*limexp*, the $*rdist_*... functions, $*temperature*, and $*vt*.
8. The *wreal* wire type

9. Connect modules and connect rules

2 SPICE Compatibility

At some point all circuit simulators such as SPICE will understand Verilog-A, all device models will be available as Verilog-A modules, and model files and netlists will be readily available in Verilog formats; but not today. Until that day, simulators that support Verilog-A/MS must provide the ability to access SPICE primitives from a Verilog description. This section describes how access to SPICE built-in primitives is provided from the Verilog-A/MS language.

2.1 Scope of Compatibility

SPICE is not a single language, but rather is a family of related languages. The first widely used version of SPICE was SPICE2g6 from the University of California at Berkeley. However, SPICE has been enhanced and distributed by many different companies, each of which has added their own extensions to the language and the models. As a result, there is a great deal of incompatibility even among the SPICE languages themselves.

Verilog-A/MS makes no judgment as to which of the various SPICE languages should be supported. Instead, it states that if a simulator that supports Verilog-A/MS is also able to read SPICE netlists of a particular flavor, then certain objects defined in that flavor of SPICE netlist can be referenced from within a Verilog-A/MS structural description. In particular, SPICE models and subcircuits can be instantiated within a Verilog-A/MS module. This is also true for most of the SPICE primitives that are built into the simulator.

2.1.1 Degree of Incompatibility

There are four primary areas of incompatibility between versions of SPICE simulators.

1. The version of the SPICE language accepted by various simulators is different and to some degree proprietary. This issue is not addressed by Verilog-A/MS. So whether a particular Verilog-A/MS simulator is SPICE compatible, and with which particular variant of SPICE it is compatible, is solely determined by the authors of the simulator.
2. Not all SPICE simulators support the same set of component primitives. A particular SPICE netlist may reference a primitive that is unsupported within a given simulator. Verilog-A/MS offers no alternative in this case other than the possibility that if the model equations are known, the primitive can be rewritten as a module.

3. The names of the built-in SPICE primitives, their parameters, or their ports can differ from simulator to simulator. This is particularly true because many primitives, parameters, and ports are unnamed in SPICE. When instantiating SPICE primitives in Verilog-A/MS, the primitives must, and parameters and ports can, be named. Since there are no established standard names, there is a high likelihood of incompatibility cropping up in these names.

   To avoid this, Verilog-A/MS defines a list of names that must be supported for common SPICE primitives when made available within Verilog-A/MS. This list is given in Table 1. However, it is not possible to anticipate all SPICE primitives and parameters that could be supported; so different implementations can end up using different names. This level of incompatibility can be overcome by using wrapper modules to map names.
4. The mathematical description of the built-in primitives can differ. As with the netlist syntax, incompatible enhancements of the models have crept in through the years. Again, Verilog-A/MS offers no solution in this case other than the possibility that if the model equations are known, the primitive can be rewritten as a module.

2.2 Accessing SPICE Objects from Verilog-A/MS

If an implementation of a Verilog-A/MS tool supports SPICE compatibility, it is expected to provide the basic set of SPICE primitives given in Section 2.3 and be able to read SPICE netlists that contain models and subcircuit statements.

SPICE primitives built into the simulator are treated in the same manner in Verilog-A/MS as built-in primitives. However, while the Verilog-A/MS built-in primitives are standardized, the SPICE primitives are not. All aspects of SPICE primitives are implementation dependent.

In addition to SPICE primitives, it is also possible to access subcircuits and models defined within SPICE netlists. The subcircuits and models contained within the SPICE netlist are treated as module definitions.

2.2.1 Case Sensitivity

SPICE netlists are case insensitive, whereas Verilog-A/MS descriptions are case sensitive. From within Verilog-A/MS, a mixed-case name matches the same name with an identical case as if it were defined in a Verilog description. However, if no exact match is found, the mixed-case name will match the same name defined within SPICE regardless of the case.

2.2.2 Examples

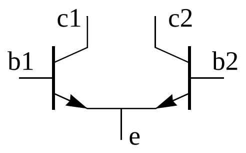
Accessing SPICE models. Consider the following SPICE model file being read by a Verilog-A/MS simulator.

```
.model vertnpn npn    bf=80 is=1e-18 rb=100 vaf=50
+                     cje=3pf cjc=2pf cjs=2pf tf=0.3ns tr=6ns
```

This model can be instantiated in a Verilog-A/MS module as follows

```
module diffPair (c1, b1, e, b2, c2);
    electrical c1, b1, e, b2, c2;

    vertNPN Q1 (c1, b1, e, );
    vertNPN Q2 (.c(c2), .b(b2), .e(e));
endmodule
```



Unlike with SPICE, the first letter of the instance name, in this case *Q1* and *Q2*, is not constrained by the primitive type. For example, they can just as easily be *T1* and *T2*. The ports and parameters of the BJT are determined by the BJT primitive itself and not by the model statement for the BJT. See Table 1 for more details. The BJT has 3 mandatory ports (collector, base, and emitter) and one optional port (the substrate). In the instantiation of *Q1*, the ports are passed by order. With *Q2*, the ports are passed by name. In both cases, the optional substrate port is defaulted by simply not giving it.

Accessing SPICE Subcircuits. As an example of how a SPICE subcircuit is referenced from Verilog-A/MS, consider the following SPICE subcircuit definition of an oscillator.

```
.subckt ecposc (out gnd)
    va vcc gnd 5
    iee e gnd 1ma
    q1 vcc b1 e vcc vertnpn
    q2 out b2 e out vertnpn
    l1 vcc out 1uh
    c1 vcc out 1p ic=1
    c2 out b1 272.7pf
    c3 b1 gnd 3nf
    r1 b1 gnd 10k
    c4 b2 gnd 3nf
    r2 b2 gnd 10k
.ends ecposc
```

This oscillator can be referenced from Verilog-A/MS as:

```
module osc (out, gnd);
    electrical out, gnd;
```