5 Language Reference

1 Basics

Verilog-A/MS is a case sensitive language.

Spaces, tabs, and newlines are considered white space and are ignored except when found in strings.

1.1 Comments

Comments are text added to the model for purposes of documentation. They are ignored by the simulator that implements the model.

Single line comments start with // and end at the end of the line.

```
// this is a single line comment
```

Block comments begin with /* and end with */.

```
/*
 * This is a block comment
 */
```

Block comments may not be nested.

1.2 Identifiers

An ***identifier*** is used to give an object a unique name so it can be referenced. An identifier can be any sequence of letters, digits, dollar signs '$', and the underscore characters '_'. The first character of an identifier cannot be a digit or '$'; it can be a letter or an underscore.

Examples: clk, out_p, bus2, n$12

Escaped identifiers start with the backslash character '\' and end with white space (space, tab or newline). They provide a means of including any of the printable ASCII characters in an identifier. Neither the leading back-slash character nor the terminat-

ing white space is considered to be part of the identifier. Therefore, an escaped identifier \out is treated the same as a non-escaped identifier out.

Examples: \out+, \/x1/n1, \\x1\n1, \{a,b}, \V(p,n)

1.3 Keywords

Keywords are predefined non-escaped identifiers that are used to define the language constructs. The list of reserved keywords for Verilog-AMS is shown in Table 1. Preceding a keyword with an escape character (\) causes it to be interpreted as an escaped identifier.

TABLE 1 *Reserved words in Verilog-AMS*

above	**cross**	**forever**	**ln**	**pulldown**	**time**
abs	**ddt**	**fork**	**log**	**rcmos**	**timer**
absdelay	**deassign**	**from**	**macromodule**	**real**	**tran**
ac_stim	**default**	**function**	**max**	**realtime**	**tranif0**
acos	**defparam**	**generate**	**medium**	**reg**	**tranif1**
acosh	**disable**	**genvar**	**min**	**release**	**transition**
always	**discipline**	**ground**	**module**	**repeat**	**tri**
analog	**driver_update**	**highz0**	**nand**	**rnmos**	**tri0**
analysis	**edge**	**highz1**	**nature**	**rpmos**	**tri1**
and	**else**	**hypot**	**negedge**	**rtran**	**triand**
asin	**end**	**idt**	**net_resolution**	**rtranif0**	**trior**
asinh	**endcase**	**idtmod**	**nmos**	**rtranif1**	**trireg**
assign	**endconnectrules**	**if**	**noise_table**	**scalared**	**vectored**
atan	**enddiscipline**	**ifnone**	**nor**	**sin**	**wait**
atan2	**endfunction**	**inf**	**not**	**sinh**	**wand**
atanh	**endmodule**	**initial**	**notif0**	**slew**	**weak0**
begin	**endnature**	**initial_step**	**notif1**	**small**	**weak1**
branch	**endprimitive**	**inout**	**or**	**specify**	**while**
buf	**endspecify**	**input**	**output**	**specparam**	**white_noise**
bufif0	**endtable**	**integer**	**parameter**	**sqrt**	**wire**
bufif1	**endtask**	**join**	**pmos**	**strong0**	**wor**
case	**event**	**laplace_nd**	**posedge**	**strong1**	**wreal**
casex	**exclude**	**laplace_np**	**potential**	**supply0**	**xnor**
casez	**exp**	**laplace_zd**	**pow**	**supply1**	**xor**
ceil	**final_step**	**laplace_zp**	**primitive**	**table**	**zi_nd**
cmos	**flicker_noise**	**large**	**pull0**	**tan**	**zi_np**
connectrules	**flow**	**last_crossing**	**pull1**	**tanh**	**zi_zd**
cos	**force**	**limexp**	**pullup**	**task**	**zi_zp**
cosh					

1.4 Compiler Directives

The ` character (referred to as a tick, an open quote, or a grave accent) introduces a language construct used to implement compiler directives. The behavior dictated by a compiler directive takes effect as soon as the compiler reads the directive. The directive remains in effect for the rest of the compilation unless a different compiler directive specifies otherwise. A compiler directive in one file can therefore control compilation behavior in multiple description files.

Verilog-AMS supports the following compiler directives.

`default_discipline	`else	`resetall
`default_transition	`endif	`timescale
`define	`ifdef	`undef
	`include	

Defines (*`define*) give a name to a string that can substitute for a string of characters. The name is then referred to as a macro. Any valid identifier, including keywords already in use, can be used as a name. Once defined, the macro is referenced using its name preceded by a tick. Undefines (*`undef*) remove the macro.

Example:

```
`define size 8
electrical [0:`size–1] out;
```

Includes (*`include*) are replaced by the contents of a file. It takes the filename as an argument, which can either be specified with a relative or absolute path to the file. Included files may include other files, etc.

Example: `**include** "disciplines.vams"

Sections of code can be conditionally ignored using the *`ifdef* directive. It takes a macro name as an argument. If the argument is currently undefined, the text that follows is ignored up to a matching `else or `endif and accepted otherwise. If `else is used, then the text between it and the matching `endif is ignored if the argument is defined, and accepted otherwise.

Verilog-AMS supports a predefined macro to allow modules to be written that work with both IEEE 1364-1995 Verilog HDL and Verilog-AMS. The predefined macro is called __VAMS_ENABLE__.

Example:

```
`ifdef __VAMS_ENABLE__
    parameter integer del = 1 from [1:100];
`else
    parameter del = 1;
`endif
```

When the *`resetall* compiler directive is encountered during compilation, all compiler directives are set to their default values. This is useful for ensuring that only those directives that are desired when compiling a particular source file are active. To do so, place *`resetall* at the beginning of each source text file, followed immediately by the directives desired in the file.

The *`timescale* compiler directive defines the time unit and the time precision for the modules that follow it. The time unit and time precision is specified using either 1, 10, or 100 followed by a measurement unit of either s, ms, us, ns, ps, or fs, which represents seconds, milliseconds, microseconds, nanoseconds, picoseconds, or femptoseconds.

Example:

```
`timescale 10ns / 1ns
```

The first value given specifies the units of time and the second specifies the precision. The values affect the way delays are specified and the return value from the $realtime function. Both are rounded to the time resolution and given in multiples of the time unit. Thus, with the specification given in the example above, #55.79 corresponds to a delay of 558ns (55.79 × 10 ns rounded to the nearest 1 ns).

2 Data Types

This section starts with a discussion of the various types of constants and variables available in Verilog-A/MS, and then presents signal types, including a discussion of natures and disciplines.

2.1 Constants

2.1.1 Integers

Underscores are ignored in numbers, so 42_839 is equivalent to 42839.

Examples: 124, +124, –124, 42_839

Except in Verilog-A, integer constants can be expressed in decimal, hexadecimal, octal, or binary. To do so, use *sb*'*fn*; where *s* is an optional sign, either '+' or '–'; *b* is an optional decimal number that indicates the size of the constant in bits; *f* is the base format and is either 'd', 'h', 'o', or 'b' for decimal, hexadecimal, octal, or binary; and *n* is the number in the specified base. In hexadecimal numbers the letters 'a' through 'f' represent the digits 10 through 15. Letters in integer constants can be either lower or upper case.

Examples:

63	*unsized decimal number*
'd63	*unsized decimal number*
'h3f	*unsized hexadecimal number*
'o77	*unsized octal number*
'b11_1111	*unsized binary number*
12'h3f	*12 bit hexadecimal number*
–'h3f	*negative unsized hexadecimal number*

The letters 'x' and 'z' can be given to denote unknown and high impedance digits in all but decimal numbers, and '_' is ignored. Sized constants for which the size is larger than the given number are padded on the left with zeros unless the first digit of the given number is an *x* or *z*, which are padded with the *x* or *z*. The number is truncated on the left if the size is smaller than the given number.

Examples:

12'hx	*a 12 bit unknown hexadecimal number*
64'o0	*a 64 bit octal 0 (zero padded)*
8'hfx	*equivalent to 8'b1111_xxxx*
8'hfffx	*equivalent to 8'b1111_xxxx (truncated)*
8'hx	*equivalent to 8'bxxxx_xxxx (x padded)*

2.1.2 Reals

Real numbers must either include a decimal point or a scale factor. If a decimal point is present, there must be digits on both sides. So .12, 9., 4.eE3, and .2e–7 are not valid numbers. Underscores are ignored in real numbers. Scale factors are given in Table 2.

Examples: 3.14, 0.1, 1.2E12, 1.30e–2, 236.123_763_e–12, 1.3u, 5.46K

Predefined numbers in the form of compiler directives are included in the file *constants.vams* and listed in Table 3. Mathematical constants are denoted with a `M_ prefix and physical constants use the `P_ prefix.

TABLE 2 *Scale factors for real numbers.*

Multiplier	Name	Symbol	Multiplier	Name	Symbol
10^{12}	tera	T	10^{-3}	milli	m
10^{9}	giga	G	10^{-6}	micro	u
10^{6}	mega	M	10^{-9}	nano	n
10^{3}	kilo	K or k	10^{-12}	pico	p
			10^{-15}	fempto	f
10^{d}	exponent	e*d* or E*d*	10^{-18}	atto	a

2.1.3 Strings

Strings are a sequence of characters enclosed in double quotes. Table 4 lists the escape sequences used to enter special characters into strings.

Example: "Hello World!\n"

2.1.4 Vectors

A ***constant vector*** is created using the ***concatenate*** operator, which consists of balanced braces surrounding a sequence of arguments given as expressions. It simply combines its arguments into an array. The individual arguments may be scalars or vectors, and the end result is a vector whose length equals the sum of the lengths of each argument.

Examples:

{4, 8, 12, 16, 20}
{4, 2∗4, 3∗4, 4∗4, 5∗4}
{4.0, 8.0, {12.0, 16.0, 20.0}}

In addition, the ***replicate*** operator can be used to specify a sequence of repeated values. The replication operator is similar to the concatenation operator, except the leading brace is preceded with an integer count and then the whole construct is surrounded with another set of braces. So {0, {2{1, 2}}} is equivalent to {0, 1, 2, 1, 2}

Vectors come in many forms. The examples above are numeric vectors, which can consist of either integers or real numbers. One can also have vectors of bits, nets, branches, instances and registers (referred to as memories).

TABLE 3 *Predefined constants in constants.vams*

Mathematical Constants		
`M_PI	π	3.14159265358979323846
`M_TWO_PI	2π	6.28318530717958647652
`M_PI_2	$\pi/2$	1.57079632679489661923
`M_PI_4	$\pi/4$	0.78539816339744830962
`M_1_PI	$1/\pi$	0.31830988618379067154
`M_2_PI	$2/\pi$	0.63661977236758134308
`M_2_SQRTPI	$2/\sqrt{\pi}$	1.12837916709551257390
`M_E	e	2.7182818284590452354
`M_LOG2E	$\log_2 e$	1.4426950408889634074
`M_LOG10E	$\log_{10} e$	0.43429448190325182765
`M_LN2	$\log_e 2$	0.69314718055994530942
`M_LN10	$\log_e 10$	2.30258509299404568402
`M_SQRT2	$\sqrt{2}$	1.41421356237309504880
`M_SQRT1_2	$1/\sqrt{2}$	0.70710678118654752440
Physical Constants		
`P_Q	charge of an electron	$1.602176462\times10^{-19}$ C
`P_C	speed of light	2.99792458×10^{8} m/s
`P_K	Boltzmann's constant	1.3806503×10^{-23} J/K
`P_H	Planck's constant	6.626076×10^{-34} J-s
`P_EPS0	permittivity of a vacuum	$8.854187817\times10^{-12}$ F/m
`P_U0	permeability of a vacuum	$\pi\times 4.0\times10^{-7}$ H/m
`P_CELSIUS0	0 Celsius	273.15 K

2.2 Variables

Variables can be thought of as named registers that contain a value of a particular type. They are initialized at the beginning of simulation to either zero or unknown as appropriate and cannot be explicitly initialized when declared. They retain their value until changed by way of an assignment statement. As such, they are different from variables in programming languages such as C in that they retain their value even when the flow of execution appears to leave their context (5§6.2.1p197).