SANDSTROM ENGINEERING

System Design

Comparing Verilog to VHDL Syntactically and Semantically

The way to a designer's heart is through HDL.

by Johan Sandstrom

With the advent of Verilog/VHDL co-simulators (Verilog = commercial and VHDL = DoD lines blurring) I thought I should hedge my bets by learning Verilog. I'm quite competent in VHDL, so armed with the Verilog-XL Reference Manual, the Digital Design with Verilog HDL book, and the free Wellspring Verilog simulator, I started my education.

The basics of each language seem common enough, but soon I was into nuances that I couldn't relate to. So, I decided I needed a summary of the language differenceshere it is.

The crux of the article is a cross-reference table indexed by Verilog (.v suffix) and VHDL (.vhd suffix) keywords. I don't show how to create designs using both languages, because Larry Saunders and Yatin Trivedi do a wonderful job. Nor do I show the BNF (Bachus-Naur Format) for language constructs, because the Language Reference Manuals and various "third-party" books cover the languages just fine. This table is merely a pointer into the other language.

Find the keyword of your favorite language in the "Keyword" column, then read along that row to find the other language equivalent keyword/construct/command in the "Equivalent" column. On the same row, in the "Verilog" and "VHDL" columns, are brief code fragments to illustrate a simple keyword usage with the keyword in bold type and names of user-created objects in italics.

I omitted the Verilog built-in compiler/simulator commands (+, -, $) because there are no VHDL equivalent commands, since the various VHDL tools have their own commands. I added the new VITAL (VHDL Initiative Towards ASIC Libraries) primitives that cross-reference to the Verilog primitives for netlisting purposes.

I suppose everybody knows that Verilog is patterned after "C" and VHDL is patterned after Ada. What that basically means is that Verilog is terse and doesn't take its type-checking seriously, whereas VHDL is verbose and strongly typed.

- VHDL says TestBench, and Verilog says TestFixture.
- Verilog is case sensitive, and VHDL is case insensitive.
- VHDL has user-defined enumerated types, and Verilog has `define.
- Verilog uses parameters, and VHDL uses constants.
- The languages have many other differences in character and implementation.

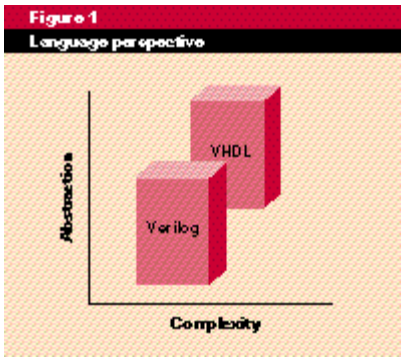

Figure 1. A view of the relative abstraction levels of the two languages.

It would appear that the HDL benefits of Designing-in-English (if this ... else that), robust simulation capability, and logic synthesis can be achieved by either language. Being conversant in both languages allows access to a wide variety of models and tools to achieve maximum design efficiency and verification.

Johan Sandstrom (johan@sandstrom.org) is a system/logic design consultant who started out hand-drawing schematics and has evolved up the abstraction/tool chain.

Rob Antman (rob1a@aol.com) knows both languages and helped me over the rough spots in the table.

Table 1

Verilog/VHDL equivalent keywords

Keyword	Equivalent	Verilog	VHDL
abs.vhd	?: // expression needed	*out1 = (in1 < 0) ? -in1 : in1;*	*out1* <= **abs**(*in1*);
access.vhd	// no counterpart	// no counterpart	type *pointer* is **access** *memory;*
after.vhd	#	#10 *out1 = in1;*	*out1* <= *in1* **after** 10 ns;
alias.vhd	`define	`define *status* word[13]	**alias** *status* : std_logic is *word*(13);
all.vhd	+lib ...	// command line option	use *libname.pkgname*.**all;**
always.v	process	**always** begin ... end	process begin ... end process;
and.v // gate	vitaland	**and (***out1, in1, in2***);**	vitaland(*out1, in1, in2*);
and.vhd	&, &&	*out1 = in1 & in2;*	*out1* <= *in1* **and** *in2;*
architecture.vhd	module	module *name*; endmodule	**architecture** *name* of *entityname* is ... end *name*;
array.vhd	reg[range] memory [range]	reg[0:3] *memory* [0:7]	type *memory* is **array**(0 to 3) of std_logic_vector(0 to 7);
assert.vhd	$display	if (*not condition*) $display("*string*");	**assert** *condition* report "*string*" severity *level*;
assign.v	when ... else	if ... **assign** *out1* = 0; else deassign	when ... *out1* <= '0'; else *out1* <= 'Z';

		out1;	
attribute.vhd	// no counterpart	// no counterpart	**attribute** *capacitance* : *farads*;
begin.v	begin	**begin** *multiple statements* end	*name* **begin** ... end *name*;
begin.vhd	begin	**begin** *multiple statements* end	*name* **begin** ... end *name*;
block.vhd	begin ... end	begin *multiple statements* end	*alu* : **block** ... end **block** *alu*;
body.vhd	// hierarchical path names	*pkgname.definitions*	package **body** *pkgname* is ... end *pkgname*;
buf.v // gate	vitalbuf	**buf (***out1, in1***);**	vitalbuf(*out1, in1*);
buffer.vhd	out	out *out1*;	port(*out1* : **buffer** std_logic);
bufif0.v // gate	vitalbufif0	**bufif0(***out1, in1, control1***);**	vitalbufif0(*out1, in1, control1*);
bufif1.v // gate	vitalbufif1	**bufif1**(*out1, in1, control1*);	vitalbufif1(*out1, in1, control1*);
bus.vhd	tri	tri *name*;	signal *name* : std_logic **bus**;
case.v	case	**case** *(expression) choices* endcase	**case** *expression* is when *choice* => *out1* <= *in1;* end **case;**
case.vhd	case	**case** *(expression) choices* endcase	**case** *expression* is when *choice* => *out1* <= *in1;* end **case;**
casex.v	no direct counterpart	**casex** *(expression) choices* endcase	no direct counterpart
casez.v	no direct counterpart	**casez** *(expression) choices* endcase	no direct counterpart
cmos.v // gate	no standard counterpart	**cmos***(out1, in1, ncontrol, pcontrol)*;	no standard counterpart
component.vhd	// structural instantiation	*inverter u1 (out1, in1)*;	**component***inverter* port(*out1* : out std_logic; *in1* : in std_logic);
configuration.vhd	// no counterpart	// no counterpart	**configuration** *name* of *EntityName* is ... end *name*;
constant.vhd	parameter	parameter *maxsize* = 64;	**constant** *maxsize* : positive := 64;
deassign.v	when ... else	if ... assign *out1* = 0; else **deassign** *out1*;	when ... *out1* <= '0'; else *out1* <= 'Z';
default.v	others	case *(expression)* ... **default** : *out1* = *in2*;	case *expression* is ... when others => *out1* <= *in2*; end case;
defparam.v	generic map	**defparam** *maxsize* = 32;	generic map(*maxsize* => 32);
disable.v	exit, next, return	for ... **disable** *name* ...	loop ... next when ... end loop;
disconnect.vhd	?: // expression needed	*out1* = guard ? *in1* : 'bz;	**disconnect** *GuardedSignalName* : std_logic after 10 ns;
downto.vhd	// [higher:lower]	[15:0]	(15 **downto** 0)
else.v	else	if ... **else if** ... else	if ... then ... elsif ... then ... **else** ... end if;
else.vhd	else	if ... **else** if ... **else**	if ... then ... elsif ... then ... **else** ... end if;
elsif.vhd	else, if	if ... else if ... else	if ... then ... **elsif** ... then ... else ... end if;
end.v	end	begin *multiple statements* **end**	... **end** ... ;

end.vhd	end	begin *multiple statements* **end**	... **end** ... ;
endcase.v	case ... end case;	case *(expression) choices* **endcase**	case *expression* is when *choice => out1 <= in1*; end case;
endfunction.v	function ... end;	function *name*; ... **endfunction**	function *name*(...) return std_logic is ... end *name*;
endmodule.v	entity/architecture	module *name*; ... **endmodule**	entity *name* is ... end *name*; architecture ...
endprimitive.v // gate	VitalTruthTable	primitive *name* ... **endprimitive**	procedure VitalTruthTable(...);
endspecify.v	generic	specify specparem *setup*=10; **endspecify**	entity *name* is generic(*setup* : time := 10 ns); ...
endtable.v // gate	VitalTruthTable	table ... **endtable**	procedure VitalTruthTable(...);
endtask.v	procedure ... end;	task *name*; ... **endtask**	procedure *name*(...) is ... end *name*;
entity.vhd	module	module *name*; ... endmodule	**entity** *name* is ... end *name*;
event.v	signal	**event** *name*;	signal *name* : sync_type;
exit.vhd	disable	for ... disable *name* ...	loop ... **exit** when ... end loop;
file.vhd	$input	$input("*source.txt*");	**file** *source_code* : text is in "*source.txt*";
for.v	for	**for** (i = 1; i <= 10; i = i + 1)	**for** i in 1 to 10 loop
for.vhd	for, repeat	**for** (i = 1; i <= 10; i = i + 1)	**for** i in 1 to 10 loop
force.v	user defined resolution function	if ... **force** *out1* = 0; else release *out1*;	*out1 <= stomp_value*; perhaps in an error_injector component.
forever.v	wait	**forever** @(posedge *clock*) *out1* = *in1*;	wait until rising_edge(*clock*); *out1 <= in1*;
fork.v	user defined synchronization	... **fork** *concurrent_execution* join ...	wait on *sync*'transaction until *sync* = 300;
function.v	function	**function** *name*; ... endfunction	**function** *name(...)* return std_logic is ... end *name*;
function.vhd	function	**function** *name*; ... endfunction	**function** *name*(...) return std_logic is ... end *name*;
generate.vhd	// no counterpart	// no counterpart	*generate_label* : if *expression* **generate** ... end **generate**;
generic.vhd	defparam, specparem	defparam *maxsize* = 32;	entity *name* is **generic**(*maxsize* : natural := 32); ...
group.vhd	// no counterpart	// no counterpart	attribute *rising_delay of c2q* : **group** is 7.2 ns;
guarded.vhd	?: // expression needed	*out1 = guard ? in1* : 'bz;	block (*guard-expression*) ... *out1* <= **guarded** *in1* after 10 ns;
highz0.v // strength	'Z' in std_logic_1164	buf**(highz0)** *out1*;	*out1* <= 'Z';
highz1.v // strength	'Z' in std_logic_1164	buf**(highz1)** *out1*;	*out1* <= 'Z';
if.v	if	**if** ... else **if** ... else	**if** ... then ... elsif ... then ... else ... end **if**;
if.vhd	if	**if** ... else **if** ... else	**if** ... then ... elsif ... then ... else ... end **if**;

impure.vhd	// no counterpart	// no counterpart	**impure** function ...
in.vhd	input	output *out1*; input *in1*;	port(*out1* : out std_logic; *in1* : **in** std_logic);
inertial.vhd	// inertial by default	#10 *out1 = in1*;	out1 <= reject 5 ns **inertial** *in1* after 10 ns;
initial.v	process	**initial** begin ... end	process begin ... wait; end process;
inout.v	inout	**inout** *inout1*;	port(*inout1* : **inout** std_logic);
inout.vhd	inout	**inout** *inout1*;	port(*inout1* : **inout** std_logic);
input.v	in	output *out1*; **input** *in1*;	port(*out1* : out std_logic; *in1* : in std_logic);
integer.v	predefined type	**integer** *name*;	signal *name* : **integer;**
is.vhd	// unnecessary	module *name*; ... endmodule	entity *name* **is** ... end *name*;
join.v	user defined synchronization	... fork *concurrent_execution* **join** ...	wait on *sync*'transaction until *sync* = 300;
label.vhd	// no counterpart	// no counterpart	attribute *location* of *adder1* : **label** is (10, 15);
large.v // charge	enumerated type	trireg**(large)** ***out1*;**	type *strength* is (*small, medium, large*);
library.vhd	+lib ...	// command line option	**library** *libname*;
linkage.vhd	// no counterpart	// no counterpart	port(*out1* : **linkage** std_logic; *in1* : in std_logic);
literal.vhd	// no counterpart	// no counterpart	attribute *area*of others : **literal** is 7;
loop.vhd	while, for, etc.	while *(condition)*	while (*condition*) **loop**
macromodule.v // gate	entity/architecture	**macromodule** *name*; ... endmodule	entity *name* is ... end *name*; architecture ...
map.vhd	// structural instantiation	*inverter u1 (out1, in1)*;	*u1 : inverter* port **map***(out1 => out1, in1 => in1)*;
medium.v // charge	enumerated type	trireg**(medium)** *out1*;	type *strength* is (*small, medium, large*);
mod.vhd	%	*out1 = in1 % in2*;	*out1* <= *in1* **mod** *in2*;
module.v	entity/architecture	**module** *name*; ... endmodule	entity *name* is ... end *name*; architecture ...
nand.v // gate	vitalnand	**nand** *(out1, in1, in2)*;	vitalnand(*out1, in1, in2*);
nand.vhd	~, &	*out1 = ~(in1 & in2)*;	*out1* <= *in1* **nand** *in2*;
negedge.v	falling_edge - std_logic_1164	always @ (**negedge** *clk*) begin ... end	wait until falling_edge(*clk*);
new.vhd	// no counterpart	// no counterpart	*pointer* := **new** *name* ... deallocate*(name)*;
next.vhd	disable	for ... disable *name* ...	loop ... **next** when ... end loop;
nmos.v // gate	vitalbufif1	**nmos***(out1, in1, control1)*;	vitalbufif1(*out1, in1, control1, ... ResultMap*);
nor.v // gate	vitalnor	**nor** *(out1, in1, in2)*;	vitalnor(*out1, in1, in2*);
nor.vhd	~, |	*out1 = ~(in1 | in2)*;	*out1* <= *in1* **nor** *in2*;
not.v // gate	vitalinv	**not** *(out1, in1)*;	vitalinv(*out1, in1*);

not.vhd	~	*out1 = ~in1*;	*out1* <= **not**(*in1*);
notif0.v // gate	vitalinvif0	**notif0***(out1, in1, control1)*;	vitalinvif0(*out1, in1, control1*);
notif1.v // gate	vitalinvif1	**notif1***(out1, in1, control1)*;	vitalinvif1(*out1, in1, control1*);
null.vhd	// no direct counterpart	// no direct counterpart	case *expression* is when choice => *out1* <= null; end case;
of.vhd	// unnecessary	module *name*; ... endmodule	architecture *name* **of** *entityname* is ... end *name*;
on.vhd	@	@(posedge *in1*) ...	wait **on** *in1*;
open.vhd	// no counterpart	// no counterpart	*u1 : inverter* port map(*out1* => **open,** *in1* => *in1*);
or.v // gate	vitalor	**or** *(out1, in1, in2)*;	vitalor(*out1, in1, in2*);
or.vhd	|, ||	*out1 = in1 | in2*;	*out1* <= *in2* **or** *in2*;
others.vhd	default	case *(expression)* ... default : *out1 = in2*;	case *expression* is ... when **others** => *out1* <= *in2*; end case;
out.vhd	output	**output** *out1*; input *in1*;	port(*out1* : **out** std_logic; *in1* : in std_logic);
output.v	out	**output** *out1*; input *in1*;	port(*out1* : out std_logic; *in1* : in std_logic);
package.vhd	// hierarchical path names	*pkgname.definitions*	**package** *pkgname* is ... end *pkgname*;
parameter.v	constant	**parameter** *maxsize* = 64;	constant *maxsize* : positive := 64;
pmos.v // gate	vitalbufif0	**pmos***(out1, in1, control1)*;	vitalbufif0(*out1, in1, control1, ... ResultMap*);
port.vhd	module name (port signals);	module *name (out1, in1)*; ... endmodule	**port**(*out1* : out std_logic; *in1* : in std_logic);
posedge.v	rising_edge - std_logic_1164	always @ (**posedge** *clk) begin ... end*	wait until rising_edge(*clk*);
postponed.vhd	#0	#0 ...	*postponed* process ...
primitive.v // gate	VitalTruthTable	**primitive** *name* ... endprimitive	procedure VitalTruthTable(...);
procedure.vhd	task	task *name*; ... endtask	**procedure** *name*(...) is ... end *name*;
process.vhd	always	always begin ... end	**process** begin ... end **process**;
pull0.v // strength	'L' in std_logic_1164	buf(**pull0**) *out1*;	*out1* <= 'L';
pull1.v // strength	'H' in std_logic_1164	buf(**pull1**) *out1*;	*out1* <= 'H';
pulldown.v // gate	vitalbuf	**pulldown**(pull0) (*out1*);	*out1* <= 'L';
pullup.v // gate	vitalbuf	**pullup**(pull1) (*out1*);	*out1* <= 'H';
pure.vhd	// no counterpart	// no counterpart	**pure** function ...
range.vhd	// no counterpart	// no counterpart	for i in *inputs*'**range** loop ... end loop;
rcmos.v // gate	no standard counterpart	**rcmos***(out1, in1, ncontrol, pcontrol)*;	no standard counterpart
record.vhd	// hierarchical path names	module *record_name* ... endmodule	type *name* is **record** ... end **record**;
reg.v	variable, signal	**reg** *name*;	variable *name* : std_logic;

register.vhd	trireg	trireg name;	signal *name*: std_logic **register;**
reject.vhd	// no counterpart	// no counterpart	*out1* <= **reject** 5 ns inertial *in1* after 10 ns;
release.v	user defined resolution function	if ... force *out1* = 0; else **release** *out1*;	*out1* <= *stomp_value*; perhaps in an error_injector component.
rem.vhd	%	*out1 = in1 % in2*;	*out1* <= *in1* **rem** *in2*;
repeat.v	for	**repeat** (10)	for i in 1 to 10 loop ...
report.vhd	$display	if *(not condition)* $display("string");	assert *condition* **report** "*string*" severity *level*;
return.vhd	disable	task *name*; ... disable *name*;	function *name*(...) **return** std_logic is ... end *name*;
rnmos.v // gate	vitalbufif1	**rnmos***(out1, in1, control1)*;	vitalbufif1(*out1, in1, control1, ... ResultMap*);
rol.vhd	{} // expression needed	*alu_data = {alu_data*[5:0], *alu_data*[7:6]};	*alu_data* <= *alu_data* **rol** 2;
ror.vhd	{} // expression needed	*alu_data = {alu_data*[1:0], *alu_data*[7:2]};	*alu_data* <= *alu_data* **ror** 2;
rpmos.v // gate	vitalbufif0	**rpmos***(out1, in1, control1)*;	vitalbufif0(*out1, in1, control1, ... ResultMap*);
rtran.v // gate	no standard counterpart	**rtran***(inout1, inout2)*;	no standard counterpart
rtranif0.v // gate	no standard counterpart	**rtranif0***(inout1, inout2, control)*;	no standard counterpart
rtranif1.v // gate	no standard counterpart	**rtranif1***(inout1, inout2, control)*;	no standard counterpart
scalared.v	std_logic_vector	tri1 **scalared**[63:0] *name*;	signal *name* : std_logic_vector(63 downto 0);
select.vhd	?: // expression needed	*cond_expr ? true_expr : false_expr*	with *expression* **select** *choices*
severity.vhd	$stop, $finish	$stop(1);	assert *condition* report "*string*" **severity** *level*;
shared.vhd	// no counterpart	// no counterpart	shared **variable** ...
signal.vhd	wire	wire *in1*;	**signal** *in1* : std_logic;
sla.vhd	// sla function needed	*alu_data = sla(alu_data, 2)*;	*alu_data* <= *alu_data* **sla** 2;
sll.vhd	<<	*alu_data = alu_data << 2*;	*alu_data* <= *alu_data* **sll 2**;
small.v // charge	enumerated type	trireg(**small**) *out1*;	type *strength* is (*small, medium, large*);
specify.v	generic	**specify** specparem *setup*=10; endspecify	entity *name* is generic(*setup* : time := 10 ns); ...
specparam.v	generic	specify **specparem** *setup*=10; endspecify	entity *name* is generic(*setup* : time := 10 ns); ...
sra.vhd	// sra function needed	*alu_data = sra(alu_data, 2)*;	*alu_data* <= *alu_data* **sra 2**;
srl.vhd	>>	*alu_data = alu_data* >> 2;	*alu_data* <= *alu_data* **srl** 2;
strong0.v // strength	'0' in std_logic_1164	buf(**strong0**) *out1*;	*out1* <= '0';
strong1.v // strength	'1' in std_logic_1164	buf(*strong1*) out1;	*out1* <= '1';

subtype.vhd	// predefined types	// predefined types	**subtype** *subtypename* is *typename* range ... ;
supply0.v // strength	'0' in std_logic_1164	buf(**supply0**) *out1*;	*out1* <= '0';
supply1.v // strength	'1' in std_logic_1164	buf(**supply1**) *out1*;	*out1* <= '1';
table.v // gate	VitalTruthTable	**table** ... endtable	procedure VitalTruthTable(...);
task.v	procedure	**task** *name*; ... endtask	procedure *name* is ... end *name*;
then.vhd	// unnecessary	if ... else if ... else	if ... **then** ... elsif ... **then** ... else ... end if;
time.v	predefined type	**time** *setup*;	generic(*setup*: **time** := 3 ns);
to.vhd	// [lower:higher]	[0:15]	(0 to 15)
tran.v // gate	no standard counterpart	**tran***(inout1, inout2)*;	no standard counterpart
tranif0.v // gate	no standard counterpart	**tranif0***(inout1, inout2, control)*;	no standard counterpart
tranif1.v // gate	no standard counterpart	**tranif1***(inout1, inout2, control)*;	no standard counterpart
transport.vhd	// inertial only	// no counterpart	*out1* <= **transport** *in1* after 10 ns;
tri.v // net	bus, resolved signal	**tri *out1*;**	signal *out1* : resolved_tri;
tri0.v // net	resolved signal	**tri0** *out1*;	signal *out1* : resolved_tri0;
tri1.v // net	resolved signal	**tri1** *out1*;	signal *out1* : resolved_tri1;
triand.v // net	resolved signal	**triand** *out1*;	signal *out1* : resolved_triand;
trior.v // net	resolved signal	**trior** *out1*;	signal *out1* : resolved_trior;
trireg.v // net	register, resolved signal	**trireg** *out1*;	signal *out1* : resolved_trireg;
type.vhd	// predefined types	// predefined types	**type** *typename ... enumerate*;
unaffected.vhd	?: // expression needed	*#delay_value out1 = condition ? out1 : in1*;	*out1* <= **unaffected** when *condition* else *in1* after *delay_value*;
units.vhd	// no counterpart	// no counterpart	physical type ... **units** ...
until.vhd	wait	wait (*condition*);	wait **until** *condition*;
use.vhd	+lib ...	// command line option	**use** *libname.pkgname.all*;
variable.vhd	reg	reg *name*;	**variable** *name* : std_logic;
vectored.v	enumerated type	tri **vectored**[31:0] *name*;	signal *name : enumerated_type*;
wait.v	wait	**wait** *(condition)*;	**wait** [on] [until] [for] ... ;
wait.vhd	wait	**wait** *(condition)*;	**wait** [on] [until] [for] ... ;
wand.v // net	resolved signal	**wand** *out1*;	signal *out1 : resolved_wand*;
weak0.v // strength	'L' in std_logic_1164	buf(**weak0**) *out1*;	out1 <= 'L';
weak1.v // strength	'H' in std_logic_1164	buf(**weak1**) *out1*;	out1 <= 'H';

when.vhd	case	case (*expression*) choices	case *expression* is **when** *choice* => *out1* <= in1; end case;
while.v	while	**while** *(condition)*	**while** (*condition*) loop
while.vhd	while	**while** *(condition)*	**while** (*condition*) loop
wire.v // net	resolved signal	wire *out1*;	signal *out1 : resolved_wire*;
with.vhd	?: // expression needed	*cond_expr ? true_expr : false_expr*	**with** *expression* select *choices*
wor.v // net	resolved signal	**wor** *out1*;	signal *out1 : resolved_wor*;
xnor.v // gate	vitalxnor	**xnor** (out1, in1, in2);	vitalxnor(*out1, in1, in2*);
xnor.vhd	^~	*out1 = in1 ^~ in2;*	*out1* <= *in1* **xnor** in2;
xor.v // gate	vitalxor	**xor** *(out1, in1, in2)*;	vitalxor(*out1, in1, in2*);
xor.vhd	^	*out1 = in1 ^ in2*;	*out1* <= *in1* **xor** *in2*;

integrated system design October 1995

Home | About Us | Demo | Products | Articles | Methodology | Vocation | Avocation