Name: Kasi Vinay Chowdary Bhogavalli

Netid: kxb220031

ADVANCED DIGITAL LOGIC
EEDG/CE 6301
Spring 2023 Midterm Examination

VERSION Kasi Vinay Chowdary Bhogavalli

Instructions

This is your personalized exam.

You have 1 hour and 15 minutes to work the first **seven** problems. You have 24 hours (due 6 pm on Wednesday to upload the eigth problem.

During the last ten minutes of the exam, you may make free use of both sides of one $8\frac{1}{2} \times 11$ sheet of paper, on which you have written information that you want to use during the examination.

Please upload all answers to eLearning. Use a camera or PDF writer to upload the answers to the proper spot on eLearning. The upload link will expire at the end of the exam.

Please take pictures of your work and upload it to eLearning for the proper question. Valid forms are any reasonable image format: GIF, JPEG, PNG, and even old-fashioned TIFF as well as PDF. If you wish to type out your answer, a text file will also be sufficient. Make sure to upload answers to the proper question.

Upload multiple files if necessary. No archive files (.zip or .tgz) as these can not be efficiently graded on eLearning. There is no mechanism for marking up archives. Points will be deducted for archives.

You may upload multiple times, the last one uploaded will be graded.

Consultation with any person (other than the TAs or Professor) during the examination period constitutes cheating.

If cheating is detected, the incident will be dealt with according to established UT-Dallas procedures. Possible disciplinary actions for academic dishonesty include a failing grade on this examination, a failing grade in this course, or expulsion from the University, depending on the severity of the infraction.

If specific directions are given in a problem, please follow these directions carefully. Full credit will be given only for answers that are correct and that conform to the directions.

Problems—Version kxb220031

1. [10 points] Given the following function:

$$f(w, x, y, z) = \sum m(2, 3, 7, 10, 11, 12)$$

Use Shannon expansion to expand with respect to w, x in the first step. Then expand y, and finally z. This means a 4:1 and two 2:1 multiplexers. Show the block level mux implementation. Draw the logic diagram.

2. [10 points] Please give big O notation for the following functions:
   $f = 1.1x10^9n^6 + n^6 + nlogn$

   $f = \pi^2 + e + ln(1000000000)$

   $f = n^n + n! + 10^{-100}n^{n^n}$

   $f = nlogn + nloglogn + logn$

   $f = 2^n + n^{100000}$

3. [10 pts] Analysis of VHDL behavioral code.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
-- use package
USE work.procmem_definitions.ALL;

ENTITY a) _______ IS
PORT (clk,rst_n : IN std_ulogic;
wen : IN std_ulogic; -- write control
writeport : IN std_ulogic_vector(width-1 DOWNTO 0); -- register input
adrwport : IN std_ulogic_vector(regfile_adrsize-1 DOWNTO 0);-- address write
adrport0 : IN std_ulogic_vector(regfile_adrsize-1 DOWNTO 0);-- address port 0
adrport1 : IN std_ulogic_vector(regfile_adrsize-1 DOWNTO 0);-- address port 1
readport0 : OUT std_ulogic_vector(width-1 DOWNTO 0); -- output port 0
readport1 : OUT std_ulogic_vector(width-1 DOWNTO 0) -- output port 1
);
END a) _______;

ARCHITECTURE behave OF a) _______ IS
  SUBTYPE WordT IS std_ulogic_vector(width-1 DOWNTO 0); -- reg word TYPE
  TYPE StorageT IS ARRAY(0 TO regfile_depth-1) OF WordT; -- reg array TYPE
  SIGNAL registerfile : StorageT; -- reg file contents
  BEGIN
  -- b) ------------------------------------------
  PROCESS(rst_n, clk)
  BEGIN
    IF rst_n = '0' THEN
      FOR i IN 0 TO regfile_depth-1 LOOP
        registerfile(i) <= (OTHERS => '0');
      END LOOP;
    ELSIF rising_edge(clk) THEN
      IF wen = '1' THEN
        registerfile(to_integer(unsigned(adrwport))) <= writeport;
      END IF;
    END IF;
  END PROCESS;
  -- c) ------------------------------------------------
  readport0 <= registerfile(to_integer(unsigned(adrport0)));
  readport1 <= registerfile(to_integer(unsigned(adrport1)));
END behave;
```

(a) What is a good name for this module above?

A. alu

B. register_file

C. ddr_memory

D. lut

E. none of the above

(b) What is good comment for the section of code denoted by b on the previous page?

A. read the data

B. reset and write the data

C. reset and write the data to memory if enabled
D. synchronous reset the data and read to memory
E. none of the above

(c) What is good comment for the section of code denoted by c on the previous page?

A. write two ports synchronously
B. write two ports asynchronously
C. read two ports synchronously
D. read two ports asynchronously
E. none of the above

(d) What does the VHDL code do?

4. [10 pts] Given the following Verilog code:

```
module mystery_module    (
output reg [9:0] out,
input enable,
input clk,              // clock Input
input reset             // reset Input
);
always @(posedge clk)
if (reset) begin
  out <= 0 ;
end else if (enable) begin
  out <= out + 1;
end
endmodule
```

(a) Is the reset signal here synchronous or asychronously updated?

(b) What does this circuit do?

(c) What is the largest value possible the out signal?

5. [15 points]

   Minimize the following function using Quine-Mccluskey:

$$f(w, x, y, z) = \sum m(1, 3, 6, 11, 14) + d(8, 10, 12, 15)$$

6. [15 points]

Given the function:

$$f = a\bar{c}d + ae + ag\bar{h} + b\bar{c}d + be + bg\bar{h} + bh$$

Given the divisor a + b, calculate the quotient and remainder using weak division. Is the divisor cube free?

7. [10 points] Calculate the kernels and co-kernels for the following function:

$$f = ac\bar{e} + dc\bar{e} + gh$$

8. [20 pts] Write the Verilog and the VHDL behavioural models for the following circuit: You are to create a latch with a preset pin synchronous to the clock. The output should be both Q and Qbar. Your latch should react to a level sensitive clock. You must supply a test bench for both Verilog and VHDL to prove that your latch works. In addition, you must take a snapshot of the output waveforms of your simulation. You must upload your code to the place specified on eLearning.

Verilog HDL QUICK REFERENCE CARD

Revision 2.1

()	Grouping	[ ]	Optional
{}	Repeated	|	Alternative
bold	As is	CAPS	User Identifier

1. MODULE

module MODID[({PORTID,})];
  [**input** | **output** | **inout** [range] {PORTID,};]
  [{declaration}]
  [{parallel_statement}]
  [specify_block]
endmodule

range ::= **[**constexpr **:** constexpr**]**

2. DECLARATIONS

parameter {PARID **=** constexpr,};

wire | **wand** | **wor** [range] {WIRID,};

reg [range] {REGID [range],};

integer {INTID [range],};

time {TIMID [range],};

real {REALID,};

realtime {REALTIMID,};

event {EVTID,};

task TASKID;
  [{**input** | **output** | **inout** [range] {ARGID,};}]
  [{declaration}]
begin
  [{sequential_statement}]
end
endtask

function [range] FCTID;
  {**input** [range] {ARGID,};}
  [{declaration}]
begin
  [{sequential_statement}]
end
endfunction

3. PARALLEL STATEMENTS

assign [(strength**1**, strength**0**)] WIRID **=** expr;

initial sequential_statement

always sequential_statement

MODID [**#**({expr,})] INSTID
  (**[**{expr,} | {.PORTID**(**expr**)**,}**]**);

GATEID [**(**strength**1**, strength**0)**] [#delay]
    [INSTID] **(**{expr,}**)**;

defparam {HIERID **=** constexpr,};

strength ::= **supply** | **strong** | **pull** | **weak** | **highz**

delay ::= number | PARID | **(** expr [, expr [, expr]] **)**

4. GATE PRIMITIVES

and (out, in_1, ..., in_N); **nand** (out, in_1, ..., in_N);
or (out, in_1, ..., in_N); **nor** (out, in_1, ..., in_N);
xor (out, in_1, ..., in_N); **xnor** (out, in_1, ..., in_N);

buf (out_1, ..., out_N, in); **not** (out_1, ..., out_N, in);

bufif0 (out, in, ctl); **bufif1** (out, in, ctl);
notif0 (out, in, ctl); **notif1** (out, in, ctl);

pullup (out); **pulldown** (out);

[r]**pmos** (out, in, ctl);
[r]**nmos** (out, in, ctl);
[r]**cmos** (out, in, nctl, pctl);

[r]**tran** (inout, inout);
[r]**tranif1** (inout, inout, ctl);
[r]**tranif0** (inout, inout, ctl);

5. SEQUENTIAL STATEMENTS

;

begin[: BLKID
  [{declaration}]]
  [{sequential_statement}]
end

if (expr) sequential_statement
[**else** sequential_statement]

case | **casex** | **casez** (expr)
  [{{expr,}: sequential_statement}]
  [**default:** sequential_statement]
endcase

forever sequential_statement

repeat (expr) sequential_statement

while (expr) sequential_statement

for (lvalue **=** expr; expr; lvalue **=** expr)
  sequential_statement

#(number | **(**expr**)**) sequential_statement

@ (event [**or** event]) sequential_statement

lvalue [**<**]**=** [**#**(number | **(**expr**)**)] expr;

lvalue [**<**]**=** [**@** (event [**or** event])] expr;

wait (expr) sequential_statement

-> EVENTID;

fork[: BLKID
  [{declaration}]]
  [{sequential_statement}]
join

TASKID[({expr,})];

disable BLKID | TASKID;

assign lvalue **=** expr;

deassign lvalue;

lvalue ::=
  ID[range] | ID**[**expr**]** **!** **{**{lvalue,}**}**

event ::= [**posedge** | **negedge**] expr

6. SPECIFY BLOCK

specify_block ::= **specify**
    {specify_statement}
  endspecify

6.1. SPECIFY BLOCK STATEMENTS

specparam {ID **=** constexpr,};

(terminal **=>** terminal**)** **=** path_delay;

((terminal,} ***>** {terminal,}**)** **=** path_delay;

if (expr) (terminal [**+**|**-**]**=>** terminal**)** **=** path_delay;

if (expr) **(**{terminal,} [**+**|**-**]***>** {terminal,}**)** **=**
    path_delay;

[**if** (expr)] **([posedge**|**negedge**] terminal **=>**
    (terminal [**+**|**-**]: expr**))** **=** path_delay;

[**if** (expr)] **([posedge**|**negedge**] terminal ***>**
    ({terminal,} [**+**|**-**]: expr**))** **=** path_delay;

$setup(tevent, tevent, expr [, ID]**)**;

$hold(tevent, tevent, expr [, ID]**)**;

$setuphold(tevent, tevent, expr, expr [, ID]**)**;

$period(tevent, expr [, ID]**)**;

$width(tevent, expr, constexpr [, ID]**)**;

$skew(tevent, tevent, expr [, ID]**)**;

$recovery(tevent, tevent, expr [, ID]**)**;

tevent ::= [**posedge** | **negedge**] terminal
    [**&&&** scalar_expr]

path_delay ::=
  expr | **(**expr, expr [, expr [, expr, expr, expr]**)**

terminal ::= ID[range] | ID**[**expr**]**

7. EXPRESSIONS

primary

unop primary

expr binop expr

expr **?** expr **:** expr

primary ::=
literal | lvalue | FCTID**(**{expr,}**)** | **(** expr **)**

7.1. UNARY OPERATORS

+, -	Positive, Negative
!	Logical negation
~	Bitwise negation
&, ~&	Bitwise and, nand
|, ~|	Bitwise or, nor
^, ~^, ^~	Bitwise xor, xnor

7.2. BINARY OPERATORS

Increasing precedence:

?:	if/else
||	Logical or
&&	Logical and
|	Bitwise or
^, ^~	Bitwise xor, xnor
&	Bitwise and
==, != , **===, !==**	Equality
<, <=, >, >=	Inequality
<<, >>	Logical shift
+, -	Addition, Subtraction
*, /, %	Multiply, Divide, Modulo

7.3. SIZES OF EXPRESSIONS

unsized constant		32
sized constant		as specified
i *op* j	+,-,*,/,%,&,|,^,^~	max(L(i), L(j))
op i	+, -, ~	L(i)
i *op* j	===, !==, ==, !=	
	&&, ||, >, >=, <, <=	1
op i	&, ~&, |, ~|, ^, ~^	1
i *op* j	>>, <<	L(i)
i ? j : k		max(L(j), L(k))
{i,...,j}		L(i) + ... + L(j)
{i{j,...k}}		i * (L(j)+...+L(k))
i = j		L(i)

8. SYSTEM TASKS

* indicates tasks not part of the IEEE standard but mentioned in the informative appendix.

8.1. INPUT

$readmemb("fname"**,** ID **[,** startadd **[,** stopadd**]]);**
$readmemh("fname"**,** ID **[,** startadd **[,** stopadd**]]);**
***$sreadmemb(**ID**,** startadd**,** stopadd **{,** string**});**
***$sreadmemh(**ID**,** startadd**,** stopadd **{,** string**});**

8.2. OUTPUT

$display[defbase]**(**[fmtstr,] {expr,}**);**
$write[defbase] **(**[fmtstr,] {expr,}**);**
$strobe[defbase] **(**[fmtstr,] {expr,}**);**
$monitor[defbase] **(**[fmtstr,] {expr,}**);**
$fdisplay[defbase] **(**fileno, [fmtstr,] {expr,}**);**
$fwrite[defbase] **(**fileno, [fmtstr,] {expr,}**);**
$fstrobe(fileno, [fmtstr,] {expr,}**);**
$fmonitor(fileno, [fmtstr,] {expr,}**);**
fileno = **$fopen(**"filename"**);**
$fclose(fileno**);**

defbase ::= **h** | **b** | **o**

8.3. TIME

$time	"now" as TIME
$stime	"now" as INTEGER
$realtime	"now" as REAL
$scale(hierid**)**	Scale "foreign" time value
$printtimescale[(path**)]**	Display time unit & precision
$timeformat(unit#**,** prec#**,** "unit"**,** minwidth**)**	Set time %t display format

8.4. SIMULATION CONTROL

$stop	Interrupt
$finish	Terminate
***$save(**"fn"**)**	Save current simulation
***$incsave(**"fn"**)**	Delta-save since last save
***$restart(**"fn"**)**	Restart with saved simulation
***$input(**"fn"**)**	Read commands from file
***$log[(**"fn"**)]**	Enable output logging to file
***$nolog**	Disable output logging
***$key[(**"fn"**)]**	Enable input logging to file
***$nokey**	Disable input logging
***$scope(**hiername**)**	Set scope to hierarchy
***$showscopes**	Scopes at current scope
***$showscopes(1)**	All scopes at & below scope
***$showvars**	Info on all variables in scope
***$showvars(**ID**)**	Info on specified variable
***$countdrivers(**net**)**	>1 driver predicate
***$list[(**ID**)]**	List source of [named] block
$monitoron	Enable $monitor task
$monitoroff	Disable $monitor task
$dumpon	Enable val change dumping
$dumpoff	Disable val change dumping
$dumpfile("fn"**)**	Name of dump file
$dumplimit(size**)**	Max size of dump file
$dumpflush	Flush dump file buffer
$dumpvars(levels [**{,** MODID | VARID**}]**)	Variables to dump
$dumpall	Force a dump now
***$reset[(0)]**	Reset simulation to time 0
***$reset(1)**	Reset and run again
***$reset(0|1,** expr**)**	Reset with reset_value
***$reset_value**	Reset_value of last $reset
***$reset_count**	# of times $reset was used

8.5. MISCELLANEOUS

$random[(ID**)]**	
***$getpattern(**mem**)**	Assign mem content
$rtoi(expr**)**	Convert real to integer
$itor(expr**)**	Convert integer to real
$realtobits(expr**)**	Convert real to 64-bit vector
$bitstoreal(expr**)**	Convert 64-bit vector to real

8.6. ESCAPE SEQUENCES IN FORMAT STRINGS

\n, **\t**, ****, **\"**	newline, TAB, '\', "'
****xxx	character as octal value
%%	character '%'
%[w.d]**e**, **%**[w.d]**E**	display real in scientific form
%[w.d]**f**, **%**[w.d]**F**	display real in decimal form
%[w.d]**g**, **%**[w.d]**G**	display real in shortest form
%[0]h, **%[0]H**	display in hexadecimal
%[0]d, **%[0]D**	display in decimal
%[0]o, **%[0]O**	display in octal
%[0]b, **%[0]B**	display in binary
%[0]c, **%[0]C**	display as ASCII character
%[0]v, **%[0]V**	display net signal strength
%[0]s, **%[0]S**	display as string
%[0]t, %[0]T	display in current time format
%[0]m, **%[0]M**	display hierarchical name

9. LEXICAL ELEMENTS

hierarchical identifier ::= {INSTID **.**} identifier

identifier ::= letter | _ { alphanumeric | **$** | _}

escaped identifer ::= \{nonwhite}

decimal literal ::=
[**+**|**-**]integer [**.** integer] [**E**|**e**[**+**|**-**] integer]

based literal ::= integer ' base {hexdigit | **x** | **z**}

base ::= **b** | **o** | **d** | **h**

comment ::= **//** comment newline

comment block ::= **/*** comment ***/**

Qualis Design Corporation
Elite Consulting and Training in High-Level Design

Phone: +1-503-670-7200 FAX: +1-503-670-0809
E-mail: info@qualis.com com
Web: http://www.qualis.com

Page 13
Figure 3: VHDL 1

QUALIS DESIGN CORPORATION

VHDL QUICK REFERENCE CARD

REVISION 1.1

()	Grouping	[ ]	Optional
{}	Repeated	|	Alternative
bold	As is	CAPS	User Identifier
italic	VHDL-1993		

1. LIBRARY UNITS

[{use_clause}]
entity ID **is**
  [**generic (**{ID : TYPEID [**:=** expr];}**);**]
  [**port (**{ID : **in** | **out** | **inout** TYPEID [**:=** expr];}**);**]
  [{declaration}]
[**begin**
  {parallel_statement}]
end [***entity***] ENTITYID;

[{use_clause}]
architecture ID **of** ENTITYID **is**
  [{declaration}]
begin
  [{parallel_statement}]
end [***architecture***] ARCHID;

[{use_clause}]
package ID **is**
  [{declaration}]
end [***package***] PACKID;

[{use_clause}]
package body ID **is**
  [{declaration}]
end [***package body***] PACKID;

[{use_clause}]
configuration ID **of** ENTITYID **is**
for ARCHID
  [{block_config | comp_config}]
end for;
end [***configuration***] CONFID;

use_clause::=
  library ID;
  [{**use** LIBID.PKGID.**all;**}]

block_config::=
  for LABELID
    [{block_config | comp_config}]
  end for;

comp_config::=
  for all | LABELID **:** COMPID
    (**use entity** [LIBID.]ENTITYID [**(** ARCHID **)**]
      [[**generic map (** {GENID **=>** expr ,} **)**]
       port map ({PORTID **=>** SIGID | *expr* ,}**)**];
    [**for** ARCHID
      [{block_config | comp_config}]
    end for;]
    end for;) |
    (**use configuration** [LIBID.]CONFID
      [[**generic map (**{GENID **=>** expr ,}**)**]
      port map ({PORTID **=>** SIGID | *expr*,}**)**];)
  end for;

2. DECLARATIONS

2.1. TYPE DECLARATIONS

type ID **is (** {ID,} **);**

type ID **is range** number **downto** | **to** number;

type ID **is array (** {range | TYPEID ,}**)**
  of TYPEID | SUBTYPID;

type ID **is record**
  {ID : TYPEID;}
end record;

type ID **is access** TYPEID;

type ID **is file of** TYPEID;

subtype ID **is** SCALARTYPID **range** range;

subtype ID **is** ARRAYTYPID**(** {range,}**)**

subtype ID **is** RESOLVFCTID TYPEID;

range ::=
  (integer | ENUMID **to** | **downto**
   integer | ENUMID) | (OBJID'[**reverse_**]**range**) |
  (TYPEID **range <>**)

2.2. OTHER DECLARATIONS

constant ID : TYPEID **:=** expr;

[***shared***] **variable** ID : TYPEID [**:=** expr];

signal ID : TYPEID [**:=** expr];

file ID : TYPEID (**is in** | **out** string;) |
  (*open read_mode* | ***write_mode***
   | ***append_mode is*** *string*;**)**

alias ID : TYPEID **is** OBJID;

attribute ID : TYPEID;

attribute ATTRID **of** OBJID **| others | all :** class
  is expr;

class ::=
  entity | **architecture** | **configuration** |
  procedure | **function** | **package** | **type** |
  subtype | **constant** | **signal** | **variable** |
  component | **label**

component ID [***is***]
  [**generic (** {ID : TYPEID [**:=** expr];} **);**]
  [**port (**{ID : **in** | **out** | **inout** TYPEID [**:=** expr];}**);**]
end component [*COMPID*];

[***impure***] **function** ID
  [**(** {[**constant** | **variable** | **signal**] ID :
    in | **out** | **inout** TYPEID [**:=** expr];}**)**]
  return TYPEID [**is**
begin
  {sequential_statement}
end [***function***] ID];

procedure ID[({[**constant** | **variable** | **signal**] ID :
    in | **out** | **inout** TYPEID [**:=** expr];}**)**]
[**is begin**
  [{sequential_statement}]
end [***procedure***] ID];

for LABELID | **others** | **all :** COMPID **use**
  (**entity** [LIBID.]ENTITYID [**(** ARCHID **)**]) |
  (**configuration** [LIBID.]CONFID)
    [[**generic map (** {GENID **=>** expr,} **)**]
     port map ({PORTID **=>** SIGID | *expr*,} **)**];

3. EXPRESSIONS

expression ::=
  (relation **and** relation) |
  (relation **or** relation) |
  (relation **xor** relation)

relation ::= shexpr [relop shexpr]

shexpr ::= sexpr [*shop sexpr*]

sexpr ::= [**+|-**] term {addop term}

term ::= factor {mulop factor}

factor ::=
  (prim [** prim]) | (**abs** prim) | (**not** prim)

prim ::=
  literal | OBJID | OBJID'ATTRID | OBJID({expr,})
  | OBJID**(**range**)** | **(**{[choice [**|** choice}] **=>**] expr,**)**
  | FCTID({[PARID **=>**] expr,}**)** | TYPEID'**(**expr**)** |
  TYPEID**(**expr**)** | **new** TYPEID[**'(**expr**)**] | **(** expr **)**

choice ::= sexpr | range | RECFID | **others**

3.1. OPERATORS, INCREASING PRECEDENCE

logop	**and** | **or** | **xor**
relop	**=** | **/=** | **<** | **<=** | **>** | **=>**
shop	***sll*** / ***srl*** / ***sla*** / ***sra*** / ***rol*** / ***ror***
addop	**+** | **-** | **&**
mulop	***** | **/** | **mod** | **rem**
miscop	****** | **abs** | **not**

4. Sequential Statements

wait [**on** {SIGID,}] [**until** expr] [**for** time];

assert expr
  [**report** string] [**severity note** | **warning** | **error** | **failure**];

report *string*
  [severity note** | **warning** | **error** | **failure];

SIGID **<=** [**transport**] | [***reject*** *TIME* ***inertial***]
  {expr [**after** time]};

VARID **:=** expr;

PROCEDUREID[({[PARID **=>**] expr,})];

[*LABEL*:] **if** expr **then**
  {sequential_statement}
[{**elsif** expr **then**
  {sequential_statement}}]
[**else**
  {sequential_statement}]
end if [*LABEL*];

[*LABEL*:] **case** expr **is**
{**when** choice [{| choice}] **=>**
  {sequential_statement}}
end case [*LABEL*];

[LABEL:] [**while** expr] **loop**
  {sequential_statement}
end loop [LABEL];

[LABEL:] **for** ID **in** range **loop**
  {sequential_statement}
end loop [LABEL];

next [LOOPLBL] [**when** expr];

exit [LOOPLBL] [**when** expr];

return [expression];

null;

5. Parallel Statements

[LABEL:] **block** [***is***]
  [**generic (** {ID : TYPEID;} **);**
    [**generic map (** {GENID **=>** expr,} **);**]]
  [**port (** {ID **: in | out | inout** TYPEID } **);**
    [**port map (** {PORTID **=>** SIGID | *expr,*})];]
  [{declaration}]
begin
  [{parallel_statement}]
end block [LABEL];

[LABEL:] [***postponed***] **process** [**(** {SIGID,})]
  [{declaration}]
begin
  [{sequential_statement}]
end [***postponed***] **process** [LABEL];

[LBL:] [***postponed***] PROCID({[PARID **=>**] expr,});

[LABEL:] [***postponed***] **assert** expr
  [**report** string] [**severity note** | **warning** | **error** | **failure**];

[LABEL:] [***postponed***] SIGID **<=**
  [**transport**] | [***reject*** *TIME* ***inertial***]
  [{expr [**after** time]} / ***unaffected*** **when** expr
  else}] {expr [**after** time]} | ***unaffected***;

[LABEL:] [***postponed***] **with** expr **select**
  SIGID **<=** [**transport**] | [***reject*** *TIME* ***inertial***]
    {{expr [**after** time]} |
     unaffected **when** choice [{| choice}]};

LABEL: COMPID
    [[**generic map (** {GENID **=>** expr,})]
     port map ({PORTID **=>** SIGID,} **)**];

LABEL: ***entity*** *[LIBID.]ENTITYID [(ARCHID)]*
    *[[**generic map (** {GENID **=>** expr,} **)***]
     port map (*{PORTID **=>** SIGID,}* ***)];***

LABEL: ***configuration*** *[LIBID.]CONFID*
    *[[**generic map (** {GENID **=>** expr,} **)***]
     port map (*{PORTID **=>** SIGID,}* ***)];***

LABEL: **if** expr **generate**
  [{parallel_statement}]
end generate [LABEL];

LABEL: **for** ID **in** range **generate**
  [{parallel_statement}]
end generate [LABEL];

6. Predefined Attributes

TYPID'**base**	Base type
TYPID'**left**	Left bound value
TYPID'**right**	Right-bound value
TYPID'**high**	Upper-bound value
TYPID'**low**	Lower-bound value
TYPID'**pos(**expr**)**	Position within type
TYPID'**val(**expr**)**	Value at position
TYPID'**succ(**expr**)**	Next value in order
TYPID'**prec(**expr**)**	Previous value in order
TYPID'**leftof(**expr**)**	Value to the left in order
TYPID'**rightof(**expr**)**	Value to the right in order
*TYPID'**ascending***	*Ascending type predicate*
*TYPID'**image(**expr**)***	*String image of value*
*TYPID'**value(**string**)***	*Value of string image*
ARYID'**left[(**expr**)]**	Left-bound of [nth] index
ARYID'**right[(**expr**)]**	Right-bound of [nth] index
ARYID'**high[(**expr**)]**	Upper-bound of [nth] index
ARYID'**low[(**expr**)]**	Lower-bound of [nth] index
ARYID'**range[(**expr**)]**	'left down/to 'right
ARYID'**reverse_range[(**expr**)]**	'right down/to 'left
ARYID'**length[(**expr**)]**	Length of [nth] dimension
*ARYID'**ascending[(**expr**)]***	*'right >= 'left ?*
SIGID'**delayed[(**expr**)]**	Delayed copy of signal
SIGID'**stable[(**expr**)]**	Signals event on signal
SIGID'**quiet[(**expr**)]**	Signals activity on signal
SIGID'**transaction[(**expr**)]**	Toggles if signal active
SIGID'**event**	Event on signal ?
SIGID'**active**	Activity on signal ?
SIGID'**last_event**	Time since last event
SIGID'**last_active**	Time since last active
SIGID'**last_value**	Value before last event
*SIGID'**driving***	*Active driver predicate*
*SIGID'**driving_value***	*Value of driver*
*OBJID'**simple_name***	*Name of object*
*OBJID'**instance_name***	*Pathname of object*
*OBJID'**path_name***	*Pathname to object*

7. Predefined Types

BOOLEAN	True or false
INTEGER	32 or 64 bits
NATURAL	Integers >= 0
POSITIVE	Integers > 0
REAL	Floating-point
BIT	'0', '1'
BIT_VECTOR(NATURAL)	Array of bits
CHARACTER	7-bit ASCII
STRING(POSITIVE)	Array of characters
TIME	hr, min, sec, ms, us, ns, ps, fs
DELAY_LENGTH	*Time => 0*

8. Predefined Functions

NOW	Returns current simulation time
DEALLOCATE(ACCESSTYPOBJ**)**	Deallocate dynamic object
FILE_OPEN(*[status], FILEID, string, mode****)***	*Open file*
FILE_CLOSE(*FILEID****)***	*Close file*

9. Lexical Elements

Identifier ::= letter { [underline] alphanumeric }

decimal literal ::= integer [**.** integer] [**E**[**+**|**-**] integer]

based literal ::=
  integer **#** hexint [**.** hexint] **#** [**E**[**+**|**-**] integer]

bit string literal ::= **B**|**O**|**X** **"** hexint **"**

comment ::= **--** comment text

Qualis Design Corporation
Beaverton, OR USA
Phone: +1-503-531-0377 FAX: +1-503-629-5525
E-mail: info@qualis.com