Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2013-06-04
Sal (1) Om tentan går i flera salar ska du bifoga ett försättsblad till varje sal och ringa in vilken sal som avses	T2
Tid	14-18
Kurskod	TDTS01
Provkod	TEN1
Kursnamn/benämning **Provnamn/benämning**	Datorstödd elektronikkonstruktion Skriftlig tentamen
Institution	IDA
Antal uppgifter som ingår i tentamen	13
Jour/Kursansvarig Ange vem som besöker salen	Zebo Peng
Telefon under skrivtiden	0702582067, 013-28 2067
Besöker salen ca kl.	15:45
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Liselotte Lundberg, 013-281278, liselotte.lundberg@liu.se
Tillåtna hjälpmedel	Engelsk ordbok
Övrigt	
Vilken typ av papper ska användas, rutigt eller linjerat	rutigt
Antal exemplar i påsen	

TEKNISKA HÖGSKOLAN I LINKÖPING
Institutionen för datavetenskap (IDA)
Zebo Peng och Petru Eles

Tentamen i kursen

TDTS 01 Datorstödd elektronikkonstruktion

(Examination on TDTS01 Computer Aided Design of Electronics)

2013-06-04, kl. 14-18

Hjälpmedel:

Engelsk ordbok.

Supporting material:

English dictionary.

Poänggränser:

Maximal poäng är 40.
För godkänt krävs 20 poäng.

Points:

Maximum points: 40.
You need 20 points to pass the exam.

Jourhavande lärare (Teacher on duty):

Zebo Peng, tel. 070 258 2067 / 013-28 2067

Lycka till (Good Luck)!

Note: You can give the answers in English or Swedish.

1. a) Describe briefly Gajski's Y-chart. What domains of information of an electronic design are captured by the Y-chart?

   b) Use the Y-chart to illustrate a typical top-down design process. What are the most important tasks performed during such a design process? Why?

   (3 p)

2. a) What is the design productivity gap for integrated circuits? Why is this gap very large and is growing?

   b) Describe one technique that can be used to address the problems related to the large design productivity gap.

   (3 p)

3. Consider the following VHDL code:

```
entity EXAM is
   port (A, B, C, D, E, F, G: in INTEGER;
        X, Y: out INTEGER);
end EXAM;

architecture HIGH-LEVEL of EXAM is
begin
   X <= F*(A+B)+ C*D+D*E*G;
   Y <= (A*B+C*D)*G+F;
end HIGH-LEVEL;
```

   a) Draw a data flow graph to capture the above design.

   b) Derive a list schedule, assuming that there are one adder and two multipliers. You can propose a priority function which is appropriate for this purpose. Illustrate, at least in a step, how the proposed priority function is used.

   c) Is your schedule, generated with the list scheduling algorithm, an optimal one for this particular example? Why?

   (4 p)

4. a) Define the concept of vertical microcodes.

   b) One problem with the vertical microcode is that it may not support the concurrent operations specified in the original controller. Why do we have this problem?

   c) Describe the different methods that can be used to address the problem stated in (b).

   (3 p)

Note: You can give the answers in English or Swedish.

5. a) What is a heuristic algorithm? What are the motivations of using such an algorithm?
b) Describe the basic principles of the Genetic algorithms.
c) Describe the three genetic operators that are used to generate new solutions in the next generation. Illustrate the operators with simple examples.

(4 p)

6. a) What is the Branch-and-Bound technique? Describe the main features of this technique.
b) Illustrate the Branch-and-Bound technique with an example.

(3 p)

7. a) Why is it difficult to test modern chip which is implemented with mixed technologies?
b) Describe one technique which can be used to deal with testing of mixed technologies efficiently.

(3 p)

8. a) What is the basic principle of the scan technique?
b) What are the main advantages of using the full scan method?
c) Discuss the different overheads which are associated with the scan technique.

(3 p)

The VHDL Part:

9. The VHDL simulation cycle.
a) Describe the successive steps of the cycle.
b) What do we call a delta cycle? When does such a cycle appear?

(3 p)

10. The generate statement. When is it useful?
Give an example showing the efficiency of using the generate statement.

(2 p)

11. We have discussed the following design units a VHDL model is composed of: entity declaration, architecture body, and configuration declaration.

Explain which aspect of the model (or of a part of the model) does each of them capture. (What information regarding the model and its simulation do they carry?)
Illustrate by an example for each of them, considering a very simple circuit.

(3 p)

Note: You can give the answers in English or Swedish.

12. Consider that we are at simulation time 100ns and the driver of a signal *S* has the following content:

0	10	25
100 ns	115 ns	155 ns

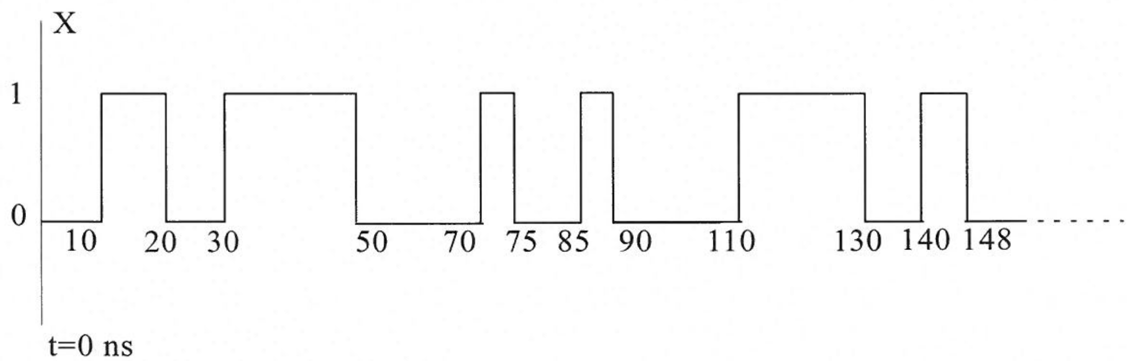
The following two signal assignments are performed, one after the other, at the current simulation time of 100ns:

S <= 18 **after** 20 ns, 2 **after** 45 ns, 5 **after** 65 ns, 10 **after** 110 ns, 25 **after** 135 ns;
S <= **reject** 40 ns **inertial** 5 **after** 80 ns;

Indicate the content of the driver for signal S
a) after the first signal assignment above;
b) after the second signal assignment above.

(3 p)

13. Consider the signal X having the waveform as follows:



Draw the output waveform (Z) if X is applied at the input of a buffer element specified as:
a) Z <= transport X after 15 ns
b) Z <= X after 15 ns
c) Z <= reject 7 ns inertial X after 25 ns

(3 p)

QUALIS
DESIGN CORPORATION

VHDL QUICK REFERENCE CARD

REVISION 1.0

()	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 [RESOLVFCTID] TYPEID [range];

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

Also available: 1164 Packages Quick Reference Card
Verilog HDL Quick Reference Card

© 1995 Qualis Design Corporation