Försättsblad till skriftlig tentamen vid Linköpings universitet

(fylls i av ansvarig)

Datum för tentamen	2010-08-23
Sal	TER4
Tid	8-12
Kurskod	TDTS01
Provkod	
Kursnamn/benämning	Datorstödd elektronikkonstruktion
Institution	IDA
Antal uppgifter som ingår i tentamen	13
Antal sidor på tentamen (inkl. försättsbladet)	7
Jour/Kursansvarig	Petru Eles
Telefon under skrivtid	0703681396 / 281396
Besöker salen ca kl.	10:00
Kursadministratör (namn + tfnnr + mailadress)	Madeleine Häger Dahlqvist 282360
Tillåtna hjälpmedel	Engesk ordbok
Övrigt (exempel när resultat kan ses på webben, betygsgränser, visning, övriga salar tentan går i m.m.)	

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

Tentamen i kursen

TDTS 01 Datorstödd elektronikkonstruktion

(Examination on TDTS01 Computer Aided Design of Electronics)

2010-08-23, kl. 8-12

Hjälpmedel:
Engelsk ordbok.

Poänggränser:
Maximal poäng är 40.
För godkänt krävs 20 poäng.

Supporting material:
English dictionary.

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

Jourhavande lärare (Teacher on duty):
Petru Eles, tel. 070-368 13 96 / 013-28 13 96

Lycka till (Good Luck)!

Exam for TDTS 01 Computer Aided Design of Electronics, 2010-08-23

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

1. What is the basic idea of the Capture-&-Simulate paradigm for electronic design? What are advantages and disadvantages of this design approach? How does this design paradigm deal with the increasing complexity of electronic systems?

(3 p)

2. a) Describe the force-directed scheduling algorithm. Use a simple example to illustrate the basic idea of this algorithm.

b) What are the advantages and disadvantages of the force-directed scheduling algorithm, respectively?

(3 p)

3. a) Define the clique partitioning problem.

b) Which problems of high-level synthesis can be formulated as clique partitioning problems? How?

c) Describe a technique that can be used to solve the clique partitioning problem.

(3 p)

4. a) Describe the microcode-based approach for control-unit synthesis.

b) What are the differences between horizontal microcodes and vertical microcodes? What are their respective features?

(3 p)

5. a) What are the basic ideas and principles of the Simulated Annealing (SA) algorithm?

b) What are the main advantages of using the SA algorithm, as compared with other optimization heuristics?

b) Identify an optimization problem in high-level synthesis, formulate it formally, and discuss how it can be solved with the SA technique.

(5 p)

6. a) What are the different components of the test costs for electronic systems?

b) Describe methods that can be used to reduce the different test costs.

(3 p)

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

7. a) What are the Ad Hoc DFT techniques?
   b) Discuss one Ad Hoc DFT technique in details and use a simple example to illustrate the advantages of this technique.

   (3 p)

8. a) Discuss the store-and-generate technique in the context of BIST. What can it be used for?
   b) Give an example to show how this technique works.

   (3 p)

The VHDL Part:

9. There is a great difference between the way signal values and variable values are updated in VHDL. What is the difference? Explain how a new value is attached to a signal, according to the VHDL simulation semantics.

   (3 p)

10. Component configuration.
    a) What is component configuration? Why is it needed?
    b) We have discussed three ways to solve component configuration. Which are these three alternatives and how do they work?

    (3 p)

11. What is a resolved signal? Why do we need a resolution function attached to such a signal?
    Imagine you have an one bit bus to which several processes write. If one single process is writing to the bus, the bus carries the value written by that process. If zero, two or more processes are writing to the bus, the bus carries the value '0'.
    Declare the signal representing the bus and specify the resolution function (give the VHDL code).

    (3 p)

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

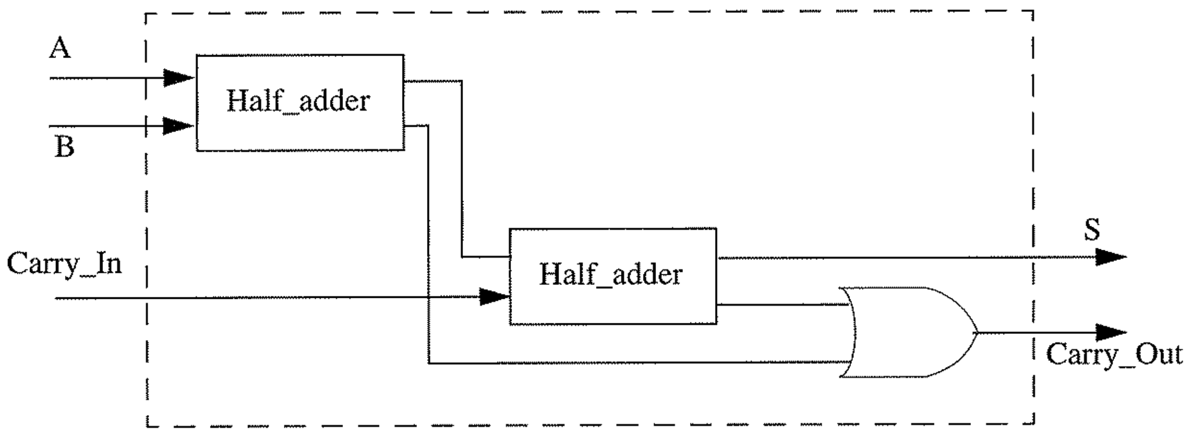
12. Concurrent signal assignment and sequential signal assignment look very similar in their syntax. When is such an assignment interpreted as a sequential and when as a concurrent one?

Write an architecture body which is equivalent to the one below, using no process statements but only signal assignments.

```
architecture EXAM of TENTA is
        signal S: BIT;
begin
        OR_GATE: process
        begin
                S<=X or Y after 1 ns;
                wait on X,Y;
        end process;
        INVERTER: process
        begin
                Z<=not S after 0.5 ns;
                wait on S;
        end process;
end EXAM;
```

(2 p)

13. In the figure below you see how a full adder is built out of two half adders and an OR gate.



a) Give the entity declarations corresponding to the half adder, the OR gate and the full adder.

b) Give the architecture body corresponding to a structural specification of the full adder.

(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