Försättsblad till skriftlig tentamen vid Linköpings universitet

(fylls i av ansvarig)

Datum för tentamen	2009-08-19
Sal	TER1
Tid	8-12
Kurskod	TDTS01
Provkod	TEN1
Kursnamn/benämning	Datorstödd elektronik konstruktion
Institution	*IDA*
Antal uppgifter som ingår i tentamen	13
Antal sidor på tentamen (inkl. försättsbladet)	7
Jour/Kursansvarig	Zebo Peng
Telefon under skrivtid	070-258 2067
Besöker salen ca kl.	ca kl. 10.00
Kursadministratör (namn + tfnnr + mailadress)	Madeleine H Dahlqvist 282360, madha@ida.liu.se
Tillåtna hjälpmedel	Engelsk-svensk eller Svensk-engelsk ordbok.
Övrigt (exempel när resultat kan ses på webben, betygsgränser, visning, övriga salar tentan går i m.m.)	
Vilken typ av papper ska användas, rutigt eller linjerat	
Antal exemplar i påsen	10st

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

Tentamen i kursen

TDTS 01 Datorstödd elektronikkonstruktion

(Computer Aided Design of Electronics)

2009-08-19, kl. 8-12

Hjälpmedel (Supporting material):

Engelsk-svensk eller svensk-engelsk ordbok. (You are only allowed to bring an English/Swedish dictionary to the examination.)

Poänggränser (Points):

Maximal poäng är 40. För godkänt krävs 20 poäng. (The examination gives maximally 40 points. You need 20 points to pass.)

Jourhavande lärare (Teacher on duty):

Zebo Peng, tel 070 258 2067

Lycka till (Good Luck)!

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. 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 <= C*D+D*E*G+F*(A+B);
   Y <= (A*B+C*D)*G;
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 choose a priority function which is appropriate for this purpose. Illustrate at least in a step how the priority function is used.

(4 p)

3. Give a complete ILP formulation of the resource-constrained scheduling problem.

(3 p)

4. a) What problems of high-level synthesis can be solved by a clique partitioning algorithm? How?

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

(3 p)

5. 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)

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

6. Write the pseudo-code of a Simulated Annealing algorithm, and discuss the basic idea of the technique. (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 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)

<u>The VHDL Part:</u>

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)

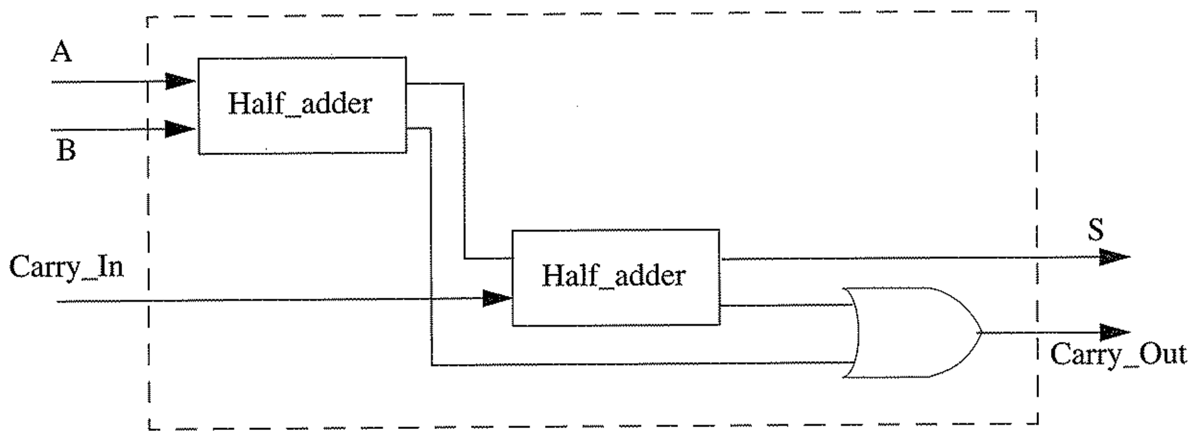
10. What is special about guarded signals?
We have guarded signals of class register and bus. What is the difference between them?
(3 p)

11. 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)

12. 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)

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