E3.06
AC5
ISE3.5

IMPERIAL COLLEGE LONDON

✓ (C changed marks totals — out of 20)

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2008

MSc and EEE/ISE PART III/IV: MEng, BEng and ACGI

VHDL AND LOGIC SYNTHESIS

CORRECTED COPY
(EXAM NOTES 5.7)

Wednesday, 14 May 10:00 am

Time allowed: 3:00 hours

There are FOUR questions on this paper.

Question 1 is COMPULSORY
Answer question 1 and any TWO of questions 2-4
Question 1 carries 40% of the marks, questions 2-4 each carry 30% of the marks.

Any special instructions for invigilators and information for candidates are on page 1.

Examiners responsible	First Marker(s) :	T.J.W. Clarke, T.J.W. Clarke
	Second Marker(s) :	C. Bouganis, C. Bouganis

Special Information for Invigilators: **none.**

Information for Candidates

VHDL language reference can be found in the booklet VHDL Exam Notes.

Unless otherwise specified assume VHDL 1993 compiler.

All packages that must be explicitly referenced with USE, e.g. IEEE.numeric_std, must be explicitly noted in each answer: semantically correct LIBRARY and USE statements may be omitted where this is done.

The Questions

Question 1 is COMPULSORY, and carries 40% of the total marks

1.

a) Describe briefly, giving examples, two distinct ways in which a VHDL process intended to represent combinational logic can compile correctly (possibly with warnings) but either simulate or synthesise incorrectly.

[4]

b) Write a synthesisable VHDL architecture for the entity *demult* in *Figure 1.1* which implements a demultiplexor such that output $x(n)$ is high if and only if input y is '1' and input *addr* has unsigned binary value n. How would you use this entity to implement a 512 output demultiplexor?

[4]

c) Write a synthesisable VHDL architecture for the entity *mul* in *Figure 1.2* which implements as combinational logic the equation:

$$a = b(c+d^2)$$

where a,b,c,d are the integer equivalent values of the similarly named ports of *mul*.

[4]

d) Write an entity *count* and synthesisable architecture for a hardware module whose 6 bit clocked output x and one bit combinational outputs *max*, *min* are controlled by *negative* edge triggered clock *clk* and a two bit input m as in *Figure 1.3*. All ports of your module must be of type *std_logic* or *std_logic_vector*.

[8]

```
ENTITY demult IS
GENERIC( K: INTEGER);
PORT(
      addr: IN  std_logic_vector(K-1 DOWNTO 0);
      y   : IN  std_logic;
      x   : OUT std_logic_vector(2**K-1 DOWNTO 0)
);
END demult;
```

Figure 1.1

```
ENTITY mul IS
PORT(
      a   : OUT unsigned(17 DOWNTO 0);
      b,c: IN unsigned(5 DOWNTO 0);
      d   : IN signed(3 DOWNTO 0)
);
END mul;
```

Figure 1.2

x	*max*	*min*
0	0	1
1-62	0	0
63	1	0

x denotes the unsigned integer equivalent value of port *x*.

m	x_{n-1}	x_n
00	*don't care*	x_{n-1}
01	≠0	x_{n-1} - 1
01	0	0
10	≠63	x_{n-1} + 1
10	63	63
11	*don't care*	0

$\boldsymbol{x_n}$ denotes the value of ***x*** in the *n*th cycle of *clk*.

Figure 1.3

Students must answer TWO out of Questions 2-4. Each question carries 30% of total marks.

2.

a) Using controllability factoring at node $m1$, derive a lower critical path time implementation of the circuit in *Figure 2.1* using the minimum number of two input NAND gates, two input NOR gates, and a single two input multiplexor. Note that invertors are not available, and must therefore be constructed from gates.

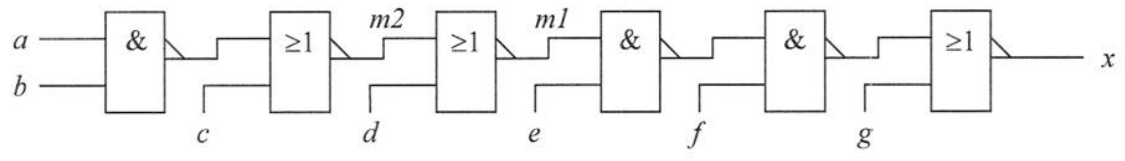
[8]

b) Using the fact that $m1 = \overline{m2 + d}$, or otherwise, repeat part a, factoring the circuit at node $m2$.

[8]

c) You may assume that the delay through each of the gates, and the multiplexor, is one time unit. Calculate the maximum critical path time from any input to the output for the original circuit, and the two factored circuits.

[4]



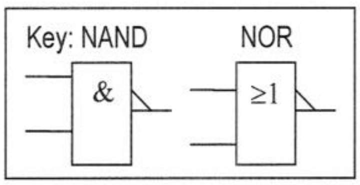


Figure 2.1

3. An *n* digit BCD (binary coded decimal) counter consists of *n* digit counters cascaded as in *Figure 3.1*. Each digit counter is a 4 bit unsigned binary counter with count sequence:

0->1->2->...->8->9->0

On positive *clk* edges, if *cin* is '1', the digit counter progresses to the next value in the sequence, otherwise its output is unchanged. Independently of the clock, if *cin* is '1', and the digit counter output is 9, *cout* will be '1', otherwise it will be '0'.

a) Write synthesisable VHDL architecture for entity *bcd12* in *Figure 3.2* which implements a 12 digit BCD counter using structural VHDL and a separate entity *digit_counter* (which you must write). It is not necessary to write the architecture of *digit_counter*. Credit will be given for compact solutions.

[6]

b) Write a single behavioural and synthesisable architecture which implements entity *bcd12* as a 12 digit BCD counter. Credit will be given for compact solutions.

[10]

c) Describe, with reasons, a modification to the *bcd12* entity which will allow proper simulation both pre-synthesis and post-synthesis.

[4]

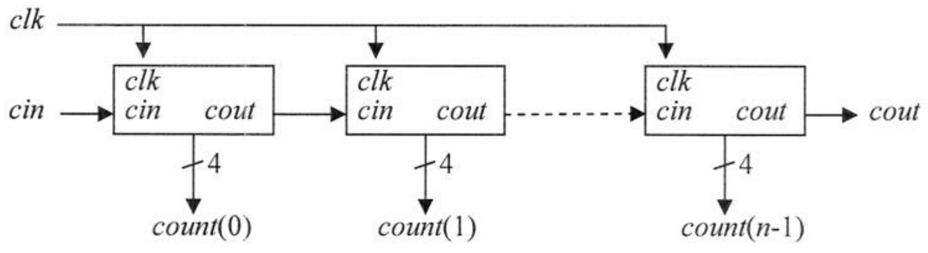


Figure 3.1

```
PACKAGE bcdpack IS
  TYPE digits IS ARRAY (0 TO 11) OF std_logic_vector(3 DOWNTO 0);
END PACKAGE bcdpack;

USE WORK.bcdpack;

ENTITY bcd12 IS
  PORT(
    count    : OUT digits;
    clk, cin: IN  std_logic;
    cout     : OUT std_logic
    );
END bcd12;
```

Figure 3.2

4.

a) Describe precisely under what circumstances an FSM implemented using a VHDL enumeration type might have different behaviour when simulated pre-synthesis and post-synthesis, and suggest a solution to this problem.

[5]

b) A hardware timing circuit has connections as in entity *timing* of *Figure 4.2* where *clk* is a 100MHz clock and *n,m* represent signed binary numbers. The required behaviour is as in *Figure 4.1*. Implement this circuit as a synthesisable VHDL architecture, using a state machine of no more than 4 states and counter of appropriate width.

[15]

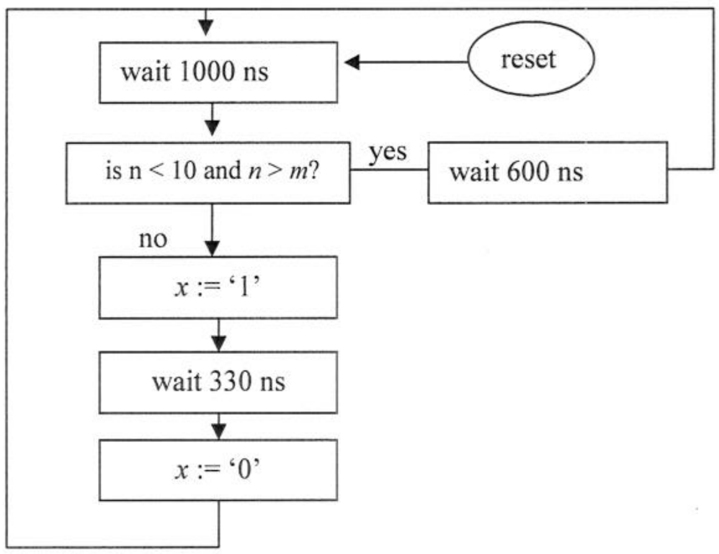


Figure 4.1

```
ENTITY timing IS
PORT (
      clk, reset     : IN  std_logic;
      n, m           : IN  std_logic_vector(9 DOWNTO 0);
      x              : OUT std_logic
    );
END timing;
```

Figure 4.2

VHDL & Logic Synthesis Exam Notes 2008

S.1

VHDL Sequential (Behavioural) Statements - inside PROCESS body

Meta-language	
[]	*optional part*
*	*0 or more repetitions*
{}	*grouping brackets*

- See also the dataflow statements which can also occur inside a PROCESS - previous slide
- Sequential statements (except WAIT) take zero simulation time to execute

```
variable := value ; -- variable assignment
NULL; -- empty statement
WAIT [ ON signal ]  [ UNTIL condition ] ;
WAIT FOR time ;

IF condition THEN statements
[ ELSIF condition THEN statements ]
[ ELSE statements ]
END IF ;

FOR var IN range LOOP
  for-statements
END LOOP;

CASE var IS
     WHEN case1 => statements
     [ WHEN case2 => statements ]
     [ WHEN OTHERS => statements ]
END CASE ;
```

S.2

VHDL Dataflow statements

(Sequential also)

```
signal <= [ TRANSPORT ] value [AFTER time ];

subprogram( para1 [, para2 [ , ... ] ] );

[ ASSERT condition ] REPORT message-if-false
    [ SEVERITY level ] ;

level ::= note | warning | error | failure
```

(Dataflow only)

{} *meta-language grouping brackets - not VHDL*

```
signal <=  value WHEN condition
        [ ELSE value WHEN condition ]
          ELSE value ;

WITH sel SELECT
   signal <=   value WHEN sel-case
           [ , value WHEN sel-case ]
              [ , value WHEN OTHERS ] ;

label :FOR var IN range GENERATE
      dataflow-statements
END GENERATE;

IF condition GENERATE
      dataflow-statements
END GENERATE;

label :  {ENTITY entity-name} | component-name
    GENERIC MAP( gen-map [ , gen-map ] )
    PORT MAP( port-map  [, port map] ) ;

[ label : ] PROCESS [ ( sensitivity-list ) ]
          process-declarations
    BEGIN sequential statements
    END PROCESS [ label ];

[ label : ] BLOCK local-declarations
    BEGIN dataflow-statements
    END [ label ] ;
```

S.3

Signal Attributes & Design Units

Signal Attributes

SIGNAL x has type T

Expression	Type	Description
x'EVENT	Boolean	TRUE if event on x
x'DELAYED(*del*)	T	x delayed by time *del*
x'LAST_VALUE	T	value of x before last or current event
x'LAST_EVENT	TIME	elapsed time from last event
x'STABLE(*tim*)	Boolean	FALSE if event on x within time *tim*

Design Units

```
ENTITY myentity IS
GENERIC( signal-list );
PORT( port-signal-list );
END myentity;

ARCHITECTURE archname OF entityname IS
declaration-statements
BEGIN
dataflow-statements
END archname;

PACKAGE mypackage IS
declarations
END [mypackage];

PACKAGE BODY mypackage IS
function and procedure body definitions
END PACKAGE BODY mypackage;
```

S.4

VHDL array syntax

unconstrained_array_type ::= STD_LOGIC_VECTOR | SIGNED | UNSIGNED | etc
range ::=   low TO high | high DOWNTO low | array_signal 'RANGE

TYPE my_type IS ARRAY range OF base_type;
SUBTYPE my_subtype IS unconstrained_array_type(range) ;

SIGNAL | VARIABLE | CONSTANT name : unconstrained_array_type(range);

my_array(range) -- array slice on LHS or RHS
my_array(index) -- array element on LHS or RHS
(val1, value2, value3) -- array value using element values specified via position on RHS
(index1=>val1, index2=>val2, ..., OTHERS=>valn) -- array value on RHS

array'LEFT	array'LOW
array'RIGHT	array'HIGH
array'LENGTH	
array'RANGE (see definitions of range above)	

VHDL Declarations

Declarations
SIGNAL sname : stype [:= init_val];
CONSTANT cname: ctype := init_val;
VARIABLE vname: vtype [:= init_val];
SHARED VARIABLE vname: vtype [:= init_val];
FILE fname: ftype [OPEN file_open_kind IS file_name_string];
TYPE tname : tspec;

Types

INTEGER	1, 123, -3456
NATURAL	<non-negative integer>
REAL	1.21, -0.033
CHARACTER	'*', '0',
STRING	"my string"
BOOLEAN	FALSE, TRUE

TIME 10.1 ns, 11 fs, 10 min (units fs, ps, ns, us, ms, s, min, hr) -- physical time constant
INTEGER RANGE low TO high -- fixed range integer type
TYPE enumeration_type IS (value_name-1, value_name-2, ..., value_name-n); --enumeration type

VHDL Operators

Logical (not is unary)	**and, or, nand, nor, xor, xnor, not** S X S -> B, B X B -> B (unary S->S, B->B)
Relational	=,/= (any type) <,<=,>,<= (scalar or discrete types)
Shift	**sll,srl,sla,sra,rol,ror** V X I -> V Shift Left/Right Logical/Arithmetic Rotate Left/Right
Addition	+, - N->N, N X N -> N (all same type)
Multiply	*,/ N X N->N (all same type) **mod, rem** Integer
Misc **: exponentiation **abs**: absolute value	** N X N->N (all same type) a ** b = a^b **abs** N->N (both same type)
Concatenation	**&** V X V -> V, V X S -> V, S X V -> V

Key to Types
V: std_logic_vector
N: integer or real
I: integer
S: std_logic
B: Boolean

Logical, Relational, Shift, Additive, Concatenation ops are synthesisable on vectors and fixed range integers

NUMERIC_STD Operators

Operation V= Unsigned or Signed (all same)	Integer operand width	smaller operand width	result width	Function
>,<,>=,<=,=,/= VXV→B VXI→B IXV→B	as other (vector) operand	extended to be same as larger	n/a (NB - result is Boolean NOT std_logic)	Comparison – operations are signed or unsigned as determined by types
+,- VXV→V VXI→V IXV→V	as other (vector) operand	extended to be same as larger	max(width(op1), width(op2))	Arithmetic +,-. Signed, Unsigned same except for extension of smaller operand which is signed or unsigned. Result may be truncated
* VXV→V VXI→V IXV→V	as other (vector) operand	unchanged	width(op1) + width(op2)	Arithmetic *, different Signed and Unsigned. Result can never be truncated (unlike +,-).
mod, rem VXV→V VXI→V IXV→V *(synthesis may be a problem)*	as other (vector) operand	unchanged	width (op2)	Both **mod** & **rem** are identical remainder operation for positive operands, in which case result is positive. **a mod b** has result same sign as **b** **a rem b** has result same sign as **a** If x = a **mod** b or x = a **rem** b then x = a + kb for some integer k and |x| < |b|
/ as **mod, rem** *(synthesis may be a problem)*	as other (vector) operand	unchanged	width(op1)	integer quotient, signed or unsigned. /0 is error.