IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2011

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

VHDL AND LOGIC SYNTHESIS

Friday, 13 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) : G.A. Constantinides, G.A. Constantinides

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 marks.

1.

a) Write a compact VHDL process without using table lookup which implements the combinational logic function q of x and y shown in Figure 1.1.

[8]

b) Write an architecture for entity *count* in Figure 1.2 such that y changes to 0 when *reset* is 1 and y counts upwards by 1 as a two's complement signed number from -7 to 17: -7,-6, ..., 16,17. After 17 the next y value is -7 again and the sequence repeats. All changes are on the positive edge of *clk*.

[8]

c) Architecture *deltadelay* in Figure 1.3 drives signals x, y, z, q, w from a. For each driven signal indicate all transitions and corresponding delta delays caused by a 0→1 transition at delta 0 on a, assuming that before this transition a is stable.

[8]

d) Figure 1.4 shows entity *fsm*. Write an architecture for *fsm* which implements the state diagram in Figure 1.5, assuming that p,q inputs and y output are unsigned numbers, and synchronously setting the state equal to a on reset.

[8]

e) Figure 1.6 shows a testbench *tb* which drives internal signals *clk*, x, y, z. Draw a timing diagram showing the *std_logic* value of these signals in the first 40ns of the simulation (delta delays not required). Comment on whether process *P1* could be made synthesisable by rewriting the code, and if so how.

[8]

x(4)	x(3)	x(2)	x(1)	x(0)	q(4)	q(3)	q(2)	q(1)	q(0)
0	0	0	0	0	y(4)	y(3)	y(2)	y(1)	y(0)
1	X	X	X	X	0	0	0	1	1
0	1	X	X	X	0	0	0	0	0
0	X	1	X	X	0	0	0	0	0
0	X	X	1	X	0	0	0	0	0
0	X	X	X	1	0	0	0	0	0

Figure 1.1

```
ENTITY count IS
PORT(
  reset, clk: IN std_logic;
  y: OUT std_logic_vector(6 DOWNTO 0));
END ENTITY count;
```

Figure 1.2

```
ENTITY dl IS
   PORT (    u: IN std_logic;
             v : OUT std_logic);
END dl;

ARCHITECTURE behav OF dl IS
BEGIN
    v <= u;
END behav;

ARCHITECTURE deltadelay OF test IS
    SIGNAL x,y,z,w,q,a: std_logic;
BEGIN
    x <= a;
    y <= x xor a;

    P1: PROCESS(y)
    BEGIN
       z <= y;
       q <= y;
    END PROCESS P1;

    E1: ENTITY WORK.dl
        PORT MAP(u=>a,v=>w);
END deltadelay;
```

Figure 1.3

```
ENTITY fsm IS
PORT(
    reset, clk: IN std_logic;
    p,q:        IN std_logic_vector(3 DOWNTO 0);
    y:          OUT std_logic_vector(1 DOWNTO 0));
END ENTITY fsm;
```

Figure 1.4

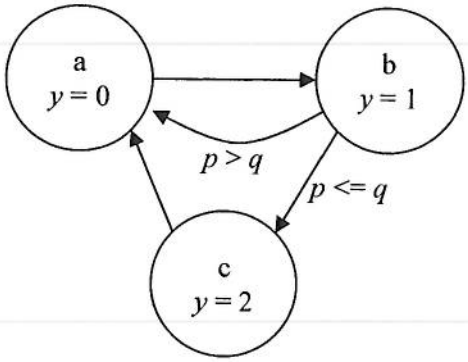

Figure 1.5

```
ENTITY tb IS
END ENTITY tb;

LIBRARY IEEE; USE IEEE.std_logic_1164.ALL;
ARCHITECTURE behav OF tb IS
        SIGNAL clk: std_logic := '0';
        SIGNAL x: std_logic := '0';
        SIGNAL y, z: std_logic;
BEGIN
        clk <= not clk AFTER 10 ns;
        x <= not x AFTER 25 ns;

        P1: PROCESS
        BEGIN
                WAIT UNTIL clk'EVENT and clk='1';
                y <= x;
                WAIT UNTIL clk'EVENT and clk='0';
                z <= y;
        END PROCESS P1;

END ARCHITECTURE behav;
```

Figure 1.6

Answer two of questions 2 - 4. Each question carries 30% of marks.

2. Figure 2.2 displays a block diagram of structural hardware using the saturated arithmetic entities defined in Figure 2.1. The three arithmetic entities *add_sat*, *mult_sat* and *power_sat* are denoted in blocks *B1-B5* by labels +, * and **Pow** respectively. Generic inputs are specified by numbers. The VHDL bit numbers on a signal *z* are indicated *z*(*a*:*b*) for the range *z*(*a* DOWNTO *b*). **All unconnected block input signals should be assumed connected to 0.** Input and output connections to the hardware are specified by signal lines that intersect the bounding box, and the output *s* is driven as stated in Figure 2.2.

   a) Assuming that the three entities implement combinational unsigned integer $p+q$ (*add_sat*), $p*q$ (*mult_sat*) and p^n (*power_sat*) functions respectively, what, ignoring overflows, is the multivariate polynomial implemented by this hardware?

   [5]

   b) You may assume that the entities in Figure 2.1 have synthesisable implementations which you need not write. Write a VHDL entity and synthesisable architecture which implements the Figure 2.2 hardware with all ports appropriately chosen vectors.

   [25]

```
ENTITY mult_sat IS -- unsigned multiplication with arithmetic saturation
        GENERIC(    w_in, w_out: INTEGER);
        PORT(       p,q: IN unsigned(w_in-1 DOWNTO 0);
                    x: OUT unsigned(w_out-1 DOWNTO 0);
                    saturated: OUT std_logic );
END ENTITY mult_sat;

ENTITY add_sat IS -- unsigned addition with arithmetic saturation
        GENERIC(    w_in, w_out: INTEGER);
        PORT(       p,q: IN unsigned(w_in-1 DOWNTO 0);
                    sum: OUT unsigned(w_out-1 DOWNTO 0);
                    saturated: OUT std_logic );
END ENTITY add_sat;

ENTITY power_sat IS -- unsigned integer exponentiation with arithmetic saturation
        GENERIC(    w_in, w_out, n: INTEGER);  -- n is exponent
        PORT(       p: IN unsigned(w_in-1 DOWNTO 0);
                    x: OUT unsigned(w_out-1 DOWNTO 0);
                    saturated: OUT std_logic );
END ENTITY power_sat;
```

Figure 2.1 VHDL Entities using **IEEE.numeric _std**

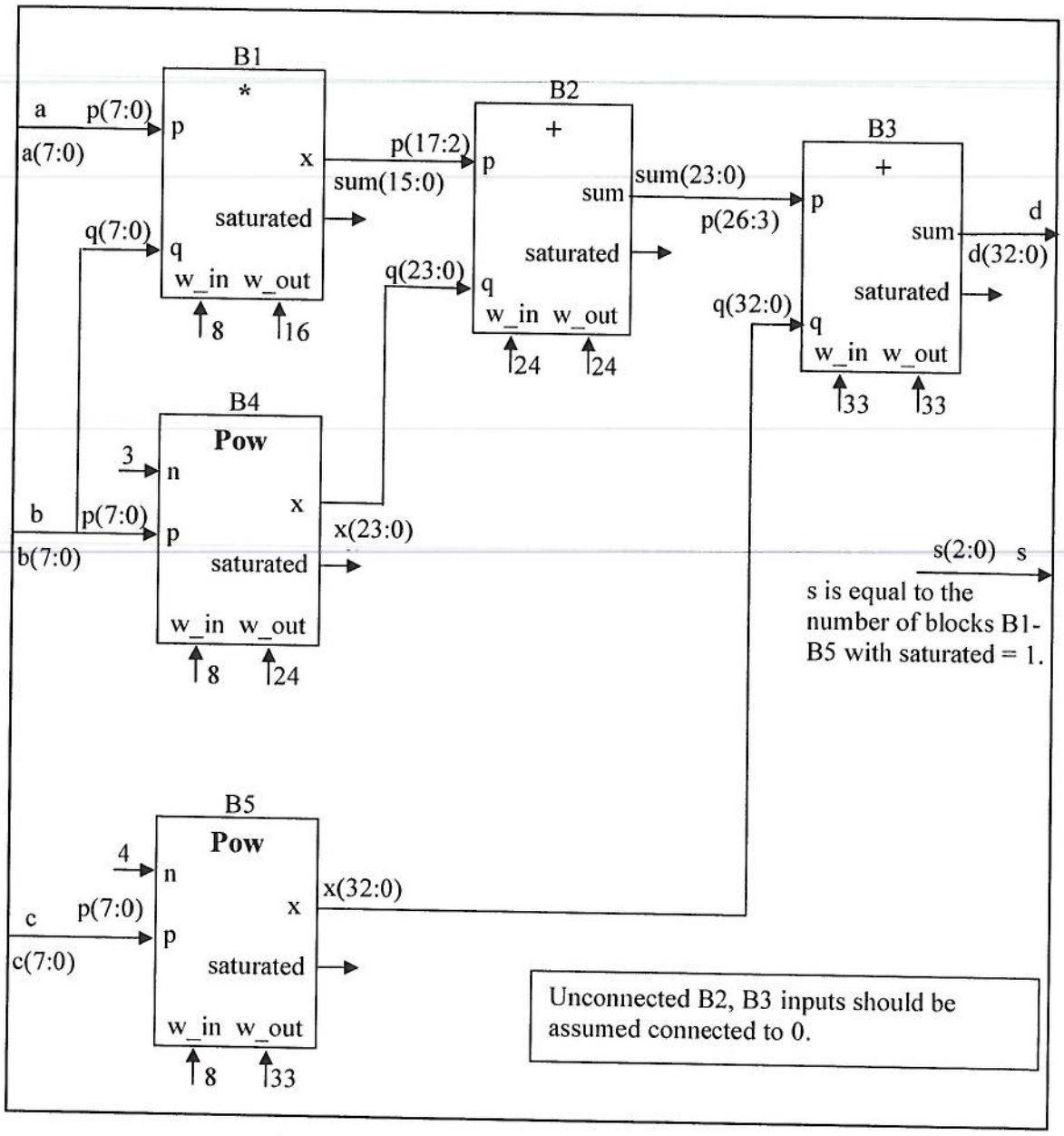

Figure 2.2

3. This question relates to the implementation of entity *power_sat_signed* as defined in Figure 3.3. The combinational outputs *x* and *saturated* of *power_sat_signed* are defined in Figure 3.1 where X and P are the signed numbers represented by ports *x* and *p* respectively.

a) Explain why this is the appropriate definition for the integer power function X^n in digital hardware.

[3]

b) Write VHDL which ensures that *w_in*, *w_out*, *n* satisfy the equations in Figure 3.2 wherever the entity *power_sat_signed* is instantiated.

[6]

c) Write a VHDL synthesisable FOR loop which implements X^n, for any positive value of *n*. Efficiency is not required. What would be the consequences of using your code with a very large value of *n*?

[6]

d) Using your answers to the above, or otherwise, write a synthesisable VHDL architecture which implements *power_sat_signed* for all non-negative values of *n*. Note that a zero length FOR loop will not compile, and therefore you may need to use one or more IF GENERATE statements.

[15]

condition	X	*saturated*
$P^n > 2^{w\_out-1}-1$	$2^{w\_out-1}-1$	1
$P^n < -2^{w\_out-1}$	-2^{w_out-1}	1
$-2^{w\_out} \leq P^n \leq 2^{w\_out-1}-1$	P^n	0

Figure 3.1

$w\_out \leq n*w\_in$
$n \geq 0$
$w\_out > 0$
$w\_in > 0$

Figure 3.2

```
LIBRARY IEEE;
USE IEEE.numeric_std
ENTITY power_sat_signed IS
    GENERIC(    w_in, w_out, n: INTEGER); -- n is exponent
    PORT(       p: IN signed(w_in-1 DOWNTO 0);
                x: OUT signed(w_out-1 DOWNTO 0);
                saturated: OUT std_logic );
END ENTITY power_sat_signed;
```

Figure 3.3 VHDL Entities

4. The testbench architecture *zz* in Figure 4.1 implements 10 independent tests of entity *dut*. Arrays *testin* and *testout* are defined in package *testpack*.

a) Explain, with reference to the delta delay timings on which data is read and changed, why this architecture will correctly test an entity *dut* clocked synchronously by clk.

[5]

b) In test *i* (*i* = 0 to 9) what are the timing and data input conditions under which *dut* is tested, and what is the required output? Give your answers for data in terms of values of *testin* & *testout*. What assumption about *dut* is necessary for each test to have a definite result independent of the other tests?

[5]

c) The algorithm implemented by *dut* is known to use different equations for the cases $xin \geq 0$ and $xin < 0$, and to have maximum output +32767 at $xin = 1000$. Provide, with justification, a suitable definition of *testin* for ad-hoc testing. Write your answer in VHDL as package *testpack*, omitting *testout*.

[10]

d) Suppose that the simulation takes 1ms per simulated clock cycle. Discuss the relative merits of random sample, exhaustive, ad hoc testing for the case illustrated in Fig 4.1, and also in the case that *xin*, *xout* are both 32 bit.

[10]

```
USE IEEE.numeric_std.ALL;
ENTITY dut IS
  PORT(xin,xout: signed(15 DOWNTO 0));
END dut;

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
LIBRARY WORK;
USE WORK.testpack.ALL;

ARCHITECTURE zz OF tb IS
    SIGNAL clk,a: std_logic := '0';
    SIGNAL x, y, z: std_logic;
    SIGNAL xin_i, xout_i, r:
        signed(15 DOWNTO 0);
BEGIN

    clk <= not clk AFTER 100 ns;

    I1: ENTITY WORK.dut
        PORT MAP( xin=>xin_i,
                  xout=>xout_i );

Pmain: PROCESS
BEGIN
   FOR num in testarray'RANGE LOOP
      WAIT FOR 150 ns;
      WAIT UNTIL clk'EVENT and clk='1';
      a <= '1';
      xin_i <= testin(num);
      WAIT UNTIL clk'EVENT and clk='1';
      a <= '0';
      FOR i IN 0 To 9 LOOP
         WAIT UNTIL clk'EVENT and clk='1';
      END LOOP; -- 0 to 9
      r <= xout_i;
      WAIT FOR 0 ns;
      ASSERT r = testout(num) REPORT "Test " &
          INTEGER'IMAGE(num) & " failed."
          SEVERITY failure;
   END LOOP; -- numtest
   ASSERT false REPORT "test ended"
      SEVERITY failure;
END PROCESS Pmain;

END ARCHITECTURE zz;
```

Figure 4.1

END OF QUESTIONS

VHDL EXAM NOTES 2011

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 ;
```


(Sequential also) VHDL Dataflow statements

```
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 ] ;
```


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;
```


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) Also in *numeric_std* for V X V -> V etc
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

Vector lengths a,b:

	length result
a + b	-> max(a,b)
a * b	-> a+b
a mod b	-> b

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

VHDL Text File I/O

TYPE *file_type* IS FILE OF *element-type*;

FILE *file_object*: *file_type* OPEN *file_open_kind* IS *file_name_string;*

–"Automatic" file open & close in object declaration

--TEXT file access uses Package STD.TEXTIO

--Use statement required (but no library statement, since STD is always available)

--Defines TEXT file type, LINE linebuffer type

TYPE SIDE IS (right, left);

TYPE FILE_OPEN_KIND IS (read_mode, write_mode, append_mode);

TYPE FILE_OPEN_STATUS IS (open_ok, status_error, name_error, mode_error);

file_open(VARIABLE status: OUT FILE_OPEN_STATUS;
FILE *f*: TEXT: name: IN STRING;
mode: IN FILE_OPEN_KIND := read_mode);

file_close(FILE f: TEXT);

endfile(FILE f: TEXT) RETURN boolean;

readline(FILE *f*: TEXT; *line*: OUT LINE);
– read the next line from f into line

writeline(FILE *f*: TEXT; *line*: INOUT: LINE);
– write a line from line to f, clearing line

read(*line*: INOUT LINE; *value*: OUT <*any-type*>);

write(*line*: INOUT LINE;
value: IN <*any-type*>;
justified: IN SIDE := right;
field: IN WIDTH := 0);

<*any-type*> ::= BIT, BIT_VECTOR, CHARACTER, INTEGER, REAL, TIME, STRING

Functions & Procedures

Meta-language

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

```
PACKAGE mypack IS
   <function header>;
END PACKAGE mypack;
```

<function header> ::=
[IMPURE] FUNCTION *myfunc*[(<par> { ; <par> }*)]
RETURN *rtype*;
<par> ::= [<pspec>] *pname* : [<fmode>] *ptype* ;
<pspec> ::= FILE | SIGNAL | VARIABLE | CONSTANT
-- omit <pspec> for value IN parameter
-- may omit <pspec> for VARIABLE OUT mode
<fmode ::= IN | OUT | INOUT – default mode is IN
– functions can only have IN parameters

```
PACKAGE BODY mypack IS
  <function_header> IS
    <function-declarations>
  BEGIN
    <function-statements>
  END FUNCTION;
END PACKAGE BODY mypack;
```

<function-statement>::= <sequential_statement> |
RETURN *expression*;

- PROCEDURE as for FUNCTION but
  - No RETURN *rtype* in header
  - WAIT allowed in body
  - OUT, INOUT parameters are allowed
- FUNCTIONs & PROCEDUREs can be defined in package body without duplicating header in package - in this case their scope is restricted to package body.
- Functions with zero parameters have no brackets in header or function call.

Built-in functions

now -- returns current simulation time
'POS -- CHARACTER -> ASCII code
'VAL -- ASCII code -> CHARACTER

<scalar_type_name>'IMAGE(<value>)
-- <value> -> printable string

EE3-06
EEG AC5

Answers 2011

Question 1.

The 5 parts of this question test basic VHDL writing & simulation skills & understanding.

a)

marks will be deducted for less compact answers.

```
PROCESS(x,y)
BEGIN
  q <= "00000";
  IF x(0)='1' THEN
    q(1 DOWNTO 0) <= "11";
  END IF;
  IF x = "00000" THEN
    q <= y;
  END IF;
END PROCESS;
```

[8 marks]

b)

```
ARCHITECTURE behav OF count IS
  SIGNAL y_i: unsigned(6 DOWNTO 0);
BEGIN
P1:PROCESS
  BEGIN
    WAIT UNTIL clk'EVENT and clk='1';
    y_i <= y_i+1;
      IF reset = '1' THEN y_i <= (OTHERS=>'0');
    END IF;
    IF y_i = to_unsigned(17,7) THEN
      y_i <= to_unsigned(-7,7);
    END IF;
  END PROCESS P1;
  y <= std_logic_vector(y_i);
END behav;
```

[8 marks]

c)

	delta of 1st transition 0->1	delta of 2nd transition 1->0
x	1	n/a
y	1	2
z	2	3
q	2	3
w	1	n/a

1 mark for each transition correct

[8 marks]

d)

```
ARCHITECTURE behav OF fsm IS
TYPE state IS (a,b,c);
SIGNAL ss,ss_next: state;
BEGIN
  F1: PROCESS
  BEGIN
    WAIT UNTIL clk = '1';
    IF reset = '1' THEN ss <= a;
    ELSE ss <= ss_next; END IF;
  END PROCESS F1;

  F2: PROCESS(p,q,ss)
  BEGIN
    CASE ss IS
          WHEN a =>     ss_next <= b;
                        y <= "00";
          WHEN b =>      IF unsigned(p) > unsigned(q)
                        THEN ss_next <= a;
                        ELSE  ss_next <= c;
                        END IF;
                        y <= "01";
          WHEN c => ss_next <= a; y <= "10";
          END CASE;
  END PROCESS F2;
END ARCHITECTURE behav;
```

[8 marks]

e)

	0-5	5-10	10-15	15-20	20-25	25-30	30-35	35-40	**marks**
clk	0	0	1	1	0	0	1	1	**1**
x	0	0	0	0	0	1	1	1	**1**
y	U	U	0	0	0	0	1	1	**2**
z	U	U	U	U	0	0	0	0	**2**

[6 marks]

Code could be made synthesisable by splitting P1 into two separate processes, each clocked by different clk edge, and driving y from the positive edge process, z from the negative edge process.

[2 marks]

Question 2.

Long, but straightforward, test of ability to write structural code

a) d = 32ab+8b^3+c^4

[5 marks]

b)

```
LIBRARY IEEE; USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
ENTITY polynomial IS
PORT( a,b,c:IN unsigned(7 DOWNTO 0);
      d: OUT unsigned(32 DOWNTO 0);
      s: OUT unsigned(2 DOWNTO 0));
END polynomial;

ARCHITECTURE struct OF polynomial IS
-- b1-b6 are data outputs from blocks
  SIGNAL b2, b4, p_b3: unsigned(32 DOWNTO 0);
  SIGNAL b1: unsigned(15 DOWNTO 0);
  SIGNAL b5: unsigned(31 DOWNTO 0);
  SIGNAL p_b2: unsigned(23 DOWNTO 0);
  SIGNAL sat: std_logic_vector(5 DOWNTO 1); -- sat outputs
 BEGIN

  p_b2 <= "000000" & b1 & "00";
  p_b3 <= resize(a,30) & "000";

PROCESS(sat) -- add the sat(i) outputs
  VARIABLE sum: unsigned(2 DOWNTO 0);
BEGIN
  sum := "000";
  FOR i in sat'RANGE LOOP
    sum := sum + unsigned ('0','0',sat(i));
  END LOOP;
  s <= sum;
END PROCESS;

IB1: ENTITY WORK.mult_sat
   GENERIC MAP(w_in=>8,w_out=>16)
   PORT MAP(p=>a,q=>b,x=>b1,saturated=>sat(1));


IB2: ENTITY WORK.add_sat GENERIC MAP(w_in=>24,w_out=>24)
   PORT MAP(p=>p_b2,q=>b4,sum=>b2,saturated=>sat(2));

IB3: ENTITY WORK.add_sat
   GENERIC MAP(w_in=>33,w_out=>33)
   PORT MAP(p=>p_b3,q=>b5,sum=>d,saturated=>sat(3));

IB4: ENTITY WORK.power_sat
   GENERIC MAP(w_in=>8,w_out=>24, n=>3)
   PORT MAP(p=>b,x=>b4,saturated=>sat(4));

IB5: ENTITY WORK.power_sat
   GENERIC MAP(w_in=>8,w_out=>33, n=>4)
   PORT MAP(p=>c,x=>b1,saturated=>sat(5));

END ARCHITECTURE struct;
```

[25 marks]

Question 3.

The code written here (can be) short, but the question involves quite difficult conceptual issues.

a) The numbers are limited to the max or min two's complement value for the output bus and are thus always as close as is possible to the correct value even with overflow.

[3 marks]

d) [15 marks]

```
ARCHITECTURE behav OF power_sat_signed IS
   SIGNAL s: unsigned(n*w_in DOWNTO 0); -- long enough for result+1,
to keep sign
BEGIN

--b) [6 marks]
   ASSERT n >= 0 and w_out <= n*w_in and w_out >0 and w_in > 0
   REPORT "n or wout invalid";
   SEVERITY failure;
-- end of b)

   IF n /= 0 GENERATE
   P1: PROCESS(p)
      VARIABLE x1: unsigned(n*w_in DOWNTO 0);
   BEGIN
      saturated <= '0';
      x1 := to_unsigned(1,w_out);

--c) [6 marks]
        FOR i IN n DOWNTO 1 LOOP
           x1 := resize(x1 * p, n*w_in+1);
        END LOOP;
-- this uses n multipliers so is expensive for large n
-- end of c)
        IF x1 >= 2**(w_out-1) THEN
           saturated <= '1';
            x <= to_unsigned(-(2**(w_out-1)),w_out);
        END IF;
        IF x1 < -2**(w_out-1) THEN
           saturated <= '1';
            x <= to_unsigned(-(2**(w_out-1)),w_out);
        END IF;
      END PROCESS P1;

   END GENERATE;

   IF n = 0 GENERATE
      saturated <= '0';
      x <= to_signed(1,x'length);
   END GENERATE;

END ARCHITECTURE behav;
```

Question 4.

This question is quite easy but requires independent thinking and tests ability to reason about testbenches and test strategies.

a) The wait on clock edge statements will result in code after executing in same delta as clk + edge, therefore dut output is read on this delta BEFORE any clk clocked outputs from dut have time to change in new cycle. Dut input is driven in clk edge + 1 delta. This is OK because the hold time for all dut f-fs clocked by clk will be met, and in a synchronous system any small delay on inputs is acceptable.

[5 marks]

b) For ith test data in = testin(i), data out = testout(i) for test to pass. xin is set to input data during single cycle where a is pulsed 1 and stays equal to this throughout the test. The output is checked on the clock edge 10 cycles after a goes to 0. All inputs & outputs change on positive edge of clock. For tests to be independent the dut must have no internal state preserved from one test to the next - e.g. it is reset by a=1.

[5 marks]

c) we need to test corner cases, 0,-1,1 & 1000 (since output might overflow). and max & min values: +32767, -32768. The remaining 4 values don't matter very much, choose two positive & two negative roughly even throughout range: 10000,20000,-10000,-20000. Testin() must be set with these 10 values in any order.

[6 marks]

```
PACKAGE testpack IS
TYPE intarray IS ARRAY (0 TO 9) OF INTEGER;
CONSTANT testin : intarray := (0,1,-1,1000,32767,-32768,10000,20000,-10000,-20000);
END PACKAGE testpack;
```

[4 marks]

d) 1ms *2^16 = 64 seconds => exhaustive testing is no problem and easier than anything else - ad hoc testing is not needed. 1ms * 2^32= 50 days. Too long for sensible exhaustive testing therefore tests should be both ad hoc (to catch corner cases) and random (to catch rest. Probably a few 1000 random samples, with ad hoc samples added, would give very good test coverage.

[10 marks]