E3.06
AC5
ISE3.5

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2004

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

VHDL AND LOGIC SYNTHESIS

Monday, 26 April 10:00 am

Time allowed: 3:00 hours

Corrected Copy

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 carry equal marks

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

Examiners responsible First Marker(s) : T.J.W. Clarke
Second Marker(s) : G.A. Constantinides

This page is intentionally left blank

Special Information for Invigilators: none.

Information for Candidates

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

Unless otherwise specified assume VHDL 1993 compiler.

*Library functions from the VHDL package **utility_pack** used in the coursework may be used freely in your implementations.*

1.

a) Explain why the VHDL process *P1* in Figure 1.1 does not synthesise to combinatorial (also known as combinational) logic. State under what circumstances it will synthesise without error, and describe the behaviour of the synthesised hardware under these circumstances.

[8]

b) Write a VHDL entity and synthesisable architecture for a 4 bit positive edge triggered counter with count sequence: 0, 1, 2, ..., 12, 0, represented by a 4 bit *std_logic_vector* output. The counter must have an input *down* which controls the count direction. If *down* = '0' the count sequence will be as above, otherwise it will be the reverse sequence. Your counter should also have a synchronous input *rst* which sets the output to 0 when *rst* = '1'.

[8]

c) In process *P2* of Figure 1.2 determine the delay, both physical time and, if relevant, simulation delta time, between events on *clk*, and corresponding events on *a,b,c,d*. You may assume that *clk* changes in $\Delta(0)$.

[8]

d) Figure 1.3 specifies the next state table for a system with inputs *a,rst*, and internal state *s* and *cnt*, where *cnt* is an 10 bit number and s can have one of the values *x*, *y*, *z*.
If input *rst* is '1' this should asynchronously set *s* to state *y*, and set *cnt* to 0.
The output *b* is '1' in state *y* when *ctr* = 0, and should be '0' at all other times.
Implement this as a VHDL architecture for the entity *system* in Figure 1.4 using a finite state machine with state *s* together with an appropriate counter, and positive edge triggered clock *clk*.

cnt

[8]

e) Write a VHDL entity and architecture that implements a combinatorial *and* gate logic block with an open array *std_logic_vector* input, and single *std_logic* output equal to '1' if and only if all of its inputs are '1'.

[8]

```
P1 : PROCESS(clk, din)
 BEGIN
   IF clk = '0' THEN
     dout <= NOT din;
   END IF;
 END PROCESS P1;
```

Figure 1.1

```
P2: PROCESS(clk, a)
  VARIABLE x : STD_LOGIC;
BEGIN
  x := clk;
  b <= a;
  a <= x;
  c <= clk AFTER 10 ns;
  d <= b AFTER 10 ns;
END PROCESS P2;
```

Figure 1.2

current			next		output
s	*a*	*cnt*	*s*	*cnt*	*b*
x	0	c	x	c+1	0
x	1	c	y	c	0
y	0	c	y	c	c = 0
y	1	c	z	c	0
z	0	c	z	c-1	0
z	1	c	x	c	0

Figure 1.3

```
ENTITY system IS
  PORT( a, clk, rst : IN  STD_LOGIC;
        b           : OUT STD_LOGIC
        );
END system;
```

Figure 1.4

2 This question concerns two different implementations of an equality checker, in which a set of n pairs of 8 bit inputs a_i and b_i (i =1,2,...,n) are compared and two outputs are generated: *errs*, which counts the number of pairs which differ ($a_i \neq b_i$), and *biterrs* which sums the total number of non-equal bits in a bitwise comparison of a_i and b_i over all pairs. Thus the sequence a = (7,0,1), b=(7,15,1) would result in *errs*=1 and *biterrs*=4.

a) The hardware block *checker* has a positive edge triggered clock *clk*. Input pairs a_i, b_i are presented in successive clock cycles on inputs *a*,*b* as shown in Figure 2.1. Outputs *err* and *biterr* must be valid during the cycle after the *n*th input is presented. An input *sens* is used to show when the pairs are presented: it is '1' while this happens, and '0' during the cycle in which *err*, *biterr* are output, as illustrated in Figure 2.1. You may also assume that *sens* is '0' for at least one cycle before the inputs are presented. Write a VHDL entity and architecture to implement *checker*, using 16-bit vectors for the outputs *err*, *biterr*.

[15]

b) The combinatorial hardware block *parchecker* has two arrays *a* and *b* of 8 bit inputs, and outputs *err* and *biterr*. The value of *n* is a constant equal to the length of the arrays *a* and *b*. Outputs *err* and *biterr* represent unsigned integers. Determine the minimum bit-widths for *err* and *biterr* respectively consistent with all values of *n* less than 16. Write a VHDL package *parchecker_pak* containing the constant *n* set to a value of 8 and an appropriate array type definition for *a* and *b*. Then write a VHDL entity and architecture using this package that implements *parchecker*, using your previously determined bit-widths.

[15]

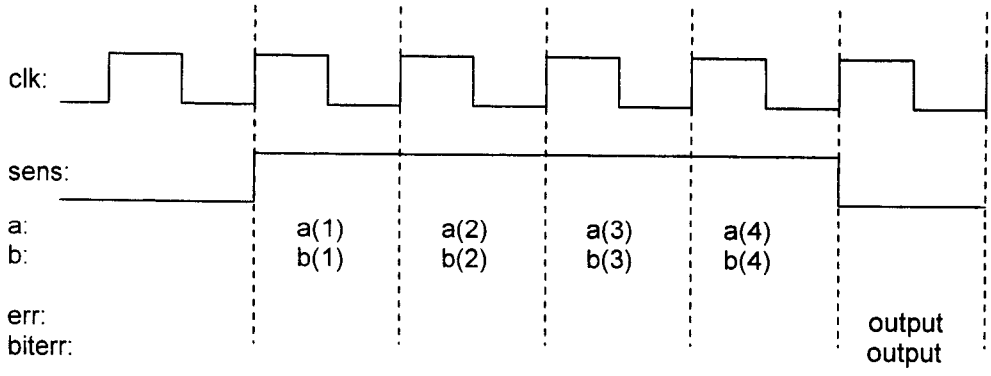


Figure 2.1

3. Figure 3.1 shows a combinatorial hardware block composed of two combinatorial sub-blocks *F* and *G*. The output *x* of block *G* is known to have the following dependence on its input *a*. If inputs *b* and *c* are equal then *x* = *a*, otherwise *x* is independent of *a*.

a) Using transduction or otherwise, show, with an appropriate circuit, how sub-blocks *F* and *G* may be rearranged together with some extra hardware to reduce delay from *p* to *r*. You should assume that the delays from *a* or *b* to *x* in *F*, and *a,b,c,d* to *x* in *G* are much longer than a single gate delay.

[12]

b) VHDL entities *fblock* and *gblock* in Figure 3.2 represent blocks *F,G*. Write a synthesisable VHDL architecture for entity *blocks* in Figure 3.2 to implement your solution to part (a) which incorporates *F* and *G* as subentities, and where *s* is known to be –11, represented in two's complement binary. You may assume that all inputs and outputs have type *std_logic_vector*. You should assume that the blocks F and G will be flattened during synthesis optimisation, and explain your choice of input *a* to *gblock*.

[12]

c) Assume that the simulation delay through either *F* or *G* from any input to any output is 10Δ. Determine the simulation delay from *p* to *r* in your solution to part (b).

[6]

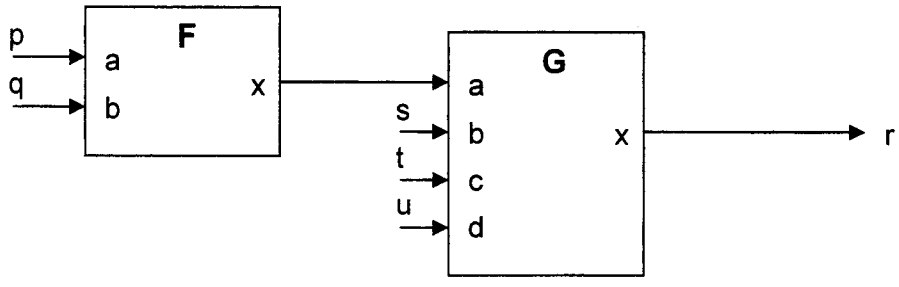


Figure 3.1

```
ENTITY fblock IS
  PORT(a, b : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
       x    : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END fblock;

ENTITY gblock IS
  GENERIC( b : IN  STD_LOGIC_VECTOR(7 DOWNTO 0));
  PORT(    a : IN  STD_LOGIC_VECTOR( 15 DOWNTO 0);
         c, d : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
            x : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END gblock;

ENTITY blocks IS
  PORT( p, q, t, u : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
        r          : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END blocks;
```

Figure 3.2

4.

a) Write a synthesisable VHDL entity and architecture to implement the arithmetic function:
$s = a + b + 4*d - c$
Where a,b,d are 8 bit unsigned numbers, c is an 8 bit signed number, the output s is a signed number, and all synthesised arithmetic units are the optimal width.

[10]

b) How long would it take to test exhaustively your answer to part (a) in a VHDL testbench, given that testing the output for a single set of inputs is known to take 10us? Without exhaustive testing, what corner cases, if any, would it be desirable to check?

[5]

c) You are given a VHDL function:

```
FUNCTION RANDOM(hi, low: INTEGER)RETURN INTEGER;
```

which generates a pseudorandom integer in the range *low* to *hi* whenever it is called. Write a VHDL testbench, using this function, which performs 1,000,000 random sample tests on your answer to part (a), and stops, printing out appropriate diagnostics, on discovering any error.

[15]

VHDL & Logic Synthesis

SOLUTIONS 2004

VHDL & Logic Synthesis: Solutions

Question 1.

a)

dout is not assigned a value if *clk* = '1', therefore it can't represent combinatorial logic. This will synthesise correctly to a negative clocked transparent latch with inverted output, but only if the synthesis program and target architecture supports synthesis of latches.

[8]

b)

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY udcnt4 IS
 PORT( clk, rst, down : IN STD_LOGIC;
       q          :   STD_LOGIC_VECTOR(3 DOWNTO 0)
       );
END udcnt4;


ARCHITECTURE synth OF udcnt4 IS
 SIGNAL q_int : STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN

 P1 : PROCESS
 BEGIN
  WAIT UNTIL clk'EVENT AND clk = '1';
  IF down = '0' THEN
   IF q_int = conv_std_logic_vector(12, 4) THEN
    q_int <= "0000";
   ELSE
    q_int <= UNSIGNED(q_int)+1;
   END IF;
  ELSE
   IF q_int = conv_std_logic_vector(0, 4) THEN
    q_int <= "1100";
   ELSE
    q_int <= UNSIGNED(q_int)-1;
   END IF;
  END IF;
  IF rst = '1' THEN
   q_int  <= "0000";
  END IF;
 END PROCESS;

END synth;
```

c) [8]

a: clk + 1 delta
b: clk + 2 delta
c: clk+10ns
d: clk+ 10ns

[8]

d)

```
ARCHITECTURE synth OF system IS
 TYPE fsmtype IS (x, y, z);
 SIGNAL s   : fsmtype;
 SIGNAL cnt : STD_LOGIC_VECTOR(9 DOWNTO 0);
BEGIN
 P1       : PROCESS(clk, rst, s, cnt, a)
 BEGIN
  IF rst = '1' THEN
    cnt <= (OTHERS => '0');
    s   <= y;

  ELSIF clk'EVENT AND clk = '1' THEN

    CASE s IS
     WHEN x => IF a = '1' THEN s <= y; ELSE cnt <= UNSIGNED(cnt)+1; END IF;
     WHEN y => IF a = '1' THEN s <= z; END IF;
     WHEN z => IF a = '1' THEN s <= x; ELSE cnt <= UNSIGNED(cnt)-1; END IF;
    END CASE;
  END IF;

 END PROCESS P1;

 P2 : PROCESS(s, cnt)
 BEGIN
  IF (s = y) AND (cnt = conv_std_logic_vector(0, 8))
  THEN b <= '1';
  ELSE b <= '0';
  END IF;
 END PROCESS P2;

END synth;
```

[8]

e)

```
ENTITY andgate IS
 PORT( x : IN  STD_LOGIC_VECTOR;
       y : OUT STD_LOGIC);
END parity;

ARCHITECTURE synth OF andgate IS
BEGIN
 P1         : PROCESS(x)
  VARIABLE z : STD_LOGIC;
 BEGIN
  z  := '1';
  FOR i IN x'RANGE LOOP
   z := z AND x(i);
  END LOOP;
  y <= z;
 END PROCESS p1;
END synth;
```

[8]

Question 2

a)

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;


ENTITY checker IS
 PORT( clk, sens   : IN    STD_LOGIC;
       a, b        : IN    STD_LOGIC_VECTOR(7 DOWNTO 0);
       err, biterr : BUFFER STD_LOGIC_VECTOR(15 DOWNTO 0)
       );
END checker;

ARCHITECTURE synth OF checker IS
BEGIN
 P1         : PROCESS
  VARIABLE w : STD_LOGIC_VECTOR(7 DOWNTO 0);
  VARIABLE b : STD_LOGIC_VECTOR(15 DOWNTO 0);
 BEGIN
  WAIT UNTIL clk'EVENT AND clk = '0';
  IF sens = '0' THEN
   err   <= (OTHERS => '0');
   biterr <= (OTHERS => '0');
  ELSE
   w    := a XOR b;
   IF NOT (w = conv_std_logic_vector(0, 8)) THEN
    err  <= UNSIGNED(err)+1;
   END IF;
   b    := biterr;
   FOR i IN w'RANGE LOOP
    IF w(i) = '1' THEN
     b := UNSIGNED(b)+1;
    END IF;
   END LOOP;
   biterr <= b;
  END IF;
 END PROCESS p1;
END synth;
```

[15].

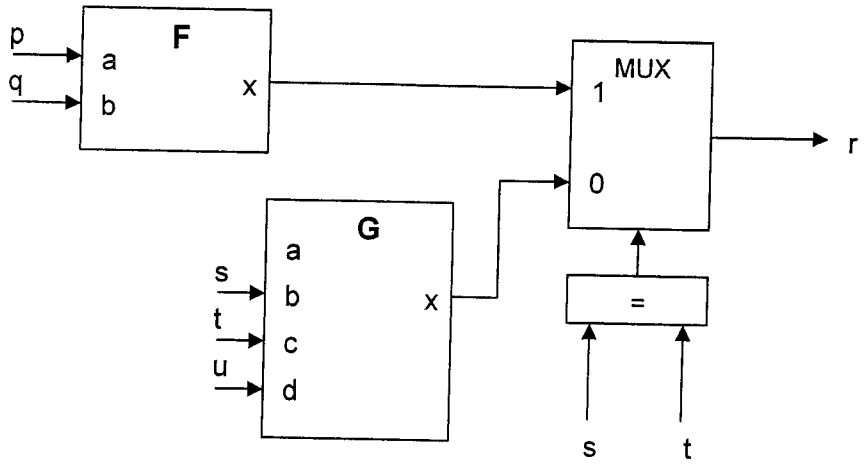
b)

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

PACKAGE parchecker_pak IS

 CONSTANT n : INTEGER := 8;

 TYPE at IS ARRAY (1 TO N) OF STD_LOGIC_VECTOR(7 DOWNTO 0);

END;


LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE WORK.parchecker_pak.ALL;

ENTITY parchecker IS
 PORT( a, b      : IN    at;
       err : BUFFER STD_LOGIC_VECTOR( 3 DOWNTO 0 );
       biterr: BUFFER STD_LOGIC_VECTOR( 6 DOWNTO 0 )
       );
END parchecker;

ARCHITECTURE synth OF parchecker IS
BEGIN
 P1        : PROCESS(a, b)
  VARIABLE w  : STD_LOGIC_VECTOR(3 DOWNTO 0);
  VARIABLE be : STD_LOGIC_VECTOR(6 DOWNTO 0);
 BEGIN
  w                       := (OTHERS => '0');
  be                      := (OTHERS => '0');
  FOR i IN a'RANGE LOOP
   IF NOT (a(i) = b(i)) THEN w    := UNSIGNED(w)+1; END IF;
   FOR j IN a(i)'RANGE LOOP
    IF a(i)(j) /= b(i)(j) THEN be := UNSIGNED(be)+1; END IF;
   END LOOP;
  END LOOP;
  err   <= w;
  biterr <= be;
 END PROCESS p1;

END synth;
```

[15]

Question 3

a)



[12]

b)

```
ARCHITECTURE synth OF blocks IS

 SIGNAL   fi, gi : STD_LOGIC_VECTOR(15 DOWNTO 0);
 CONSTANT indef  : STD_LOGIC_VECTOR(15 DOWNTO 0) := (OTHERS => '-' );


BEGIN

 I1 : ENTITY fblock PORT MAP(x    => fi, a => p, b => q);
 I2 : ENTITY gblock GENERIC MAP(b => conv_std_logic_vector(-11, 8))
  PORT MAP( a  => indef, c => t, d => u, x => gi);
 P1 : PROCESS(t, gi, fi)
 BEGIN
  IF conv_std_logic_vector(-11, 8) = t THEN
   r <= fi;
  ELSE
   r <= gi;
  END IF;
 END PROCESS P1;

END synth;
```

[12]

Note that input *a* of *gblock* is don't care, so this is wired to '-', to allow optimisation of *gblock*.

c)

Ports have zero delay, multiplexor has 1 delta delay, therefore total max delay is 11 delta. This answer will depend on the precise implementation chosen for b).

[6]

Question 4.

a)

```
ENTITY arith IS
 PORT(a, b, c, d : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
    s        : OUT STD_LOGIC_VECTOR(11 DOWNTO 0)
    );
END arith;

ARCHITECTURE synth OF arith IS
 SIGNAL p : STD_LOGIC_VECTOR(8 DOWNTO 0);
 SIGNAL q : STD_LOGIC_VECTOR(10 DOWNTO 0); -- can be negative
BEGIN
 p <= ("0" & UNSIGNED(a))+UNSIGNED(b);
 q <= ("00" & UNSIGNED(p))-SIGNED(c);
 s <= (SIGNED(q(10 DOWNTO 2))+UNSIGNED("00" & d))& q(1 DOWNTO 0);
END synth;
```

Note the care needed to do sign extension correctly on signed/unsigned quantities, and to add 1 bit on change from signed to unsigned. In the computation for s, q(10:2) is correctly sign-extended by the + operator – d is extended by two bits to force the correct width of addition. Could alternatively have sign extended q(10:2) by 1 bit.

[10]

b)

10us * 2^{32} = 10^4 seconds.

Corner cases are max and min value of a,b,d, unsigned
0, 255.
Max & min values of c, signed -128 ... 127

Should test every combination (16 tests).

[5]

c)

I assume library function i2s is available, defined as:
FUNCTION i2s(i: INTEGER) RETURN STRING IS
BEGIN
  RETURN INTEGER'IMAGE(i);
END FUNCTION;
Students will not be penalised if they use this without definition – but could equivalently give answer using INTEGER'IMAGE(), which they have learnt.

```
ENTITY testbench IS
END testbench;


ARCHITECTURE randsample OF testbench IS

 SIGNAL a_i, b_i, c_i, d_i : STD_LOGIC_VECTOR(7 DOWNTO 0);
 SIGNAL s_i              : STD_LOGIC_VECTOR(10 DOWNTO 0);

BEGIN

 dut : ENTITY WORK.arith PORT MAP(a => a_i, b => b_i,
                      c => c_i, d => d_i, s => s_i);

 p1   : PROCESS
  VARIABLE na, nb, nc, nd, n : INTEGER;
 BEGIN
  FOR i IN 1 TO 1000000 LOOP
   na := random(0, 255);
   nb := random(0, 255);
   nc := random(-128, 127);
   nd := random(0, 255);

   a_i <= conv_std_logic_vector(na, 8);
   b_i <= conv_std_logic_vector(nb, 8);
   c_i <= conv_std_logic_vector(nc, 8);
   d_i <= conv_std_logic_vector(nd, 8);
   WAIT FOR 10 ns;
   n := na +nb +4*nd-nc;
   ASSERT n = conv_integer(SIGNED(s_i)) REPORT
    "Bad test : a = "&i2s(na)&", b = "&i2s(nb)&", c = "&i2s(nc)&"d = "&i2s(nd)
    SEVERITY failure;
 END LOOP;
END PROCESS p1;

END ARCHITECTURE randsample;
```

[15]