E3.06
AC5
ISE3.5

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2006

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

Corrected Copy
None

VHDL AND LOGIC SYNTHESIS

Friday, 28 April 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 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.*

The Questions

Question 1 is COMPULSORY

1. a) Determine the precise behaviour of process `P1` in Figure 1.1. Is this process synthesisable?

[4]

b) Figure 1.2. shows an entity *count* for a counter with count output *pout*, control inputs *limit* and *set*, and carry output *cout*. If *set* is '0' the counter must count up with an increment of 1 on every positive edge of *clk* until it reaches a count of *limit*. It must then reset with 0 being the next count. If set is '1' the counter must set *pout* equal to *limit* independently of *clk*. Write a synthesisable architecture for *count*.

[4]

c) In process `P2` of Figure 1.3 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 $\Delta(0)$.

[4]

d) Write a synthesisable VHDL architecture for entity *jiggle* in Figure 1.4 in which *z* is a combinational function of *x* and *y*:

$z(i) = x(i)$, when i is even,

$z(i) = y(size - i - 1)$ xor $y(i)$ when i is odd.

You may find it useful to note that, for non-negative integers a and b, **a MOD b** in VHDL is the integer remainder when a is divided by b.

[4]

e)

(i) Write an entity *sq* with *n* bit *std_logic_vector* input port *x*, bits numbered from n-1 (MSB) to 0 (LSB) and n^2 bit *std_logic_vector* port *y*, bits numbered from n^2-1 (MSB) to 0 (LSB). It must be possible to input & output data on each of the bits $y(i)$. Hint - use an integer generic.

(ii) Write an architecture *testsq_arch* in which two copies of *sq* are used, the two *y* ports are connected together, and the two *x* ports are connected to a 10 bit vector of zeros. You need not write an architecture for *sq*.

[4]

```
P1:PROCESS
BEGIN
  WAIT UNTIL rst = '0';
  FOR i IN 0 TO 99 LOOP
    clk <= '0';
    WAIT FOR 10 ns;
    clk <= '1';
    WAIT FOR 5 ns;
  END LOOP;
  WAIT UNTIL rst = '1';
END PROCESS P1;
```

Figure 1.1

```
ENTITY count IS
PORT(
     clk, set: IN std_logic;
     limit: IN std_logic_vector(15 DOWNTO 0);
     cout: OUT std_logic;
     pout: OUT std_logic_vector(15 DOWNTO 0)
);
END count;
```

Figure 1.2

```
P2:PROCESS(clk,a,b)
  VARIABLE xv : std_logic;
BEGIN
  xv := clk;
  a <= xv;
  b <= clk;
  b <= not clk;
  c <= not b;
  d <= b AFTER 20 us;
END PROCESS P2;
```

Figure 1.3

```
ENTITY jiggle IS
GENERIC( size: INTEGER);
PORT(
     x,y: IN std_logic_vector(size-1 DOWNTO 0);
     z: OUT std_logic_vector( size-1 DOWNTO 0)
);
END jiggle;
```

Figure 1.4

Students must answer TWO out of Questions 2-4.

2. A synchronous first-in-first-out memory (FIFO) may be designed from a *size* word synchronous dual-port RAM and two counters p and q each of length *counter_length*. An entity *fifo* for the FIFO is given in Figure 2.1. The operation of the FIFO is controlled by two inputs *inputitem* and *outputitem* as shown in Figure 2.2. When $p = q$ it is assumed that the FIFO is empty. A '1' input on *reset* will synchronously reset the FIFO to the empty state. All operation is synchronous with the positive edge of *clk*. Note that Figure 2.2 specifies the value of *dataout* in the *current* clock cycle, and the values of p, q and *mem*[p] in the *next* clock cycle. The notation *mem*[x] represents the RAM location with address x.

   a) Write a synthesisable architecture to implement the FIFO. Your architecture should use ASSERT statements to ensure that *size* is a multiple of 4, and if not terminate with an appropriate error message. You may assume that overflow and underflow conditions never occur.

   [16]

   b) Implement two additional *std_logic* outputs *nearlyfull* and *nearlyempty* which are '1' when the number of FIFO items stored, n, satisfies $n \geq (3/4)size$ and $n < size/4$ respectively. You may assume that *std_logic* output ports *nearlyfull* and *nearlyempty* have been added to the *fifo* entity.

   [4]

```
ENTITY fifo IS
  GENERIC(size : INTEGER; counter_length : INTEGER);
  PORT (
      clk, inputitem, outputitem, reset : IN  STD_LOGIC;
      datain  : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
      dataout : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
    );
END fifo;
```

Figure 2.1

		Next cycle			Current cycle
inputitem	**outputitem**	**p**	**q**	**mem[p]**	**dataout**
0	0	p	q	mem[p]	High impedance
0	1	p	(q+1) mod size	mem[p]	mem[q]
1	0	(p+1) mod size	q	datain	High impedance
1	1	p	q	mem[p]	datain

Key: mem[x] = *RAM location with address* x

Figure 2.2

3. Six m bit numbers $x(0)$ to $x(5)$ can be added using a tree of *csadd* blocks *CS0 - CS3* as in Figure 3.1. The bit range of each bus connection is indicated by a:b on the connection where a,b are the MSB and LSB bit numbers. Unconnected bit positions on ports, such as $p(0)$ of CS3 should be set to '0'.

Each *csadd* block has three n bit inputs p, q and r and two n bit outputs s and c with bit values as shown in Figure 3.2. The value of n is m or m+2 in each instance of *csadd* as indicated in Figure 3.1. The n bit outputs c and s represent the set of carries and sums formed by adding the three inputs separately at each bit position:

$c(i+1)$ is '1' if two or more of $p(i)$, $q(i)$, $r(i)$ are 1.

$s(i)$ is '1' if an odd number of $p(i)$, $q(i)$, $r(i)$ are 1.

Note that *csadd c* ports have bit numbering one greater than that of ports p, q, r and s. The output y is calculated by using an m+3 bit adder, labelled *ADD* in Figure 3.1.

a) Write a synthesisable architecture for block *csadd* in Figure 3.2.

[5]

b) Write a package *cspack* containing appropriate constants and/or types needed to define the ports of the hardware block in Figure 3.1. Using *cspack* write an entity *csaddtree* for this block.

[5]

c) Write an architecture for *csaddtree* using instances of *csadd* together with other combinational logic as appropriate.

[10]

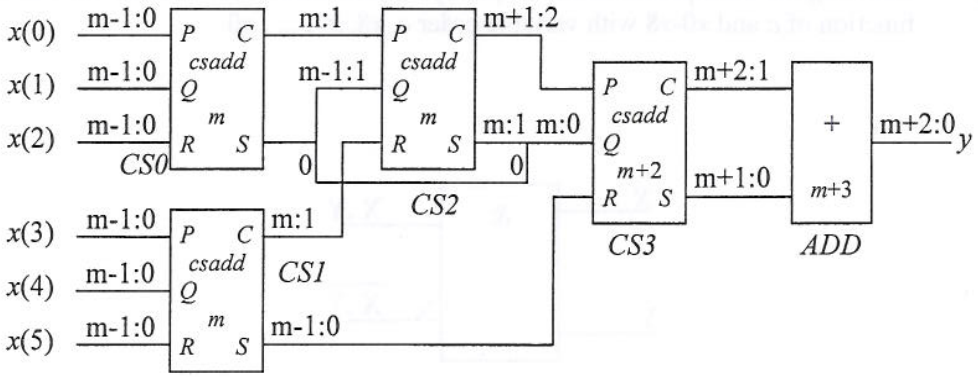


Figure 3.1

```
ENTITY csadd IS
  GENERIC( n : INTEGER);  -- number of bits in this block
  PORT(
    p, q, r  : IN  STD_LOGIC_VECTOR( n-1 DOWNTO 0);
    c        : OUT STD_LOGIC_VECTOR(n DOWNTO 1);  -- carry bits
    s        : OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0)  -- sum bits
    );
END csadd;
```

Figure 3.2

4. A "sea of gates" FPGA architecture consists of 2-input AND/NAND blocks, with both inverting and non-inverting outputs as shown in Figure 4.1. These may be connected together arbitrarily to make combinational logic. Figure 4.2 shows a critical path for a node y after device-dependent synthesis to this architecture. One step in the synthesis algorithm uses transduction about point M to shorten the length of this critical path.

(a) Indicate the circuit for y after transduction, using a multiplexor and give a Boolean expression for the multiplexor control input.

[10]

(b) Using the available block in Figure 4.1, write the circuit diagram of an implementation of the multiplexor control input such that the number of blocks from any input to the output is minimised.

[3]

(c) You may assume that every AND/NAND block in the target architecture introduces one unit of delay between inputs and both non-inverting and inverting outputs. You may further assume that a multiplexor may be implemented with 2 units of delay from data inputs to data output, and 2 units of delay from control input to output. What is the worst case delay from any of $x0$, ... , $x8$ to y before and after the transduction?

[3]

(d) Denoting the multiplexor control input by c, write down the ROBDD for y as a function of c and $x0$-$x8$ with variable order c, $x8$, $x7$, ... , $x0$.

[4]

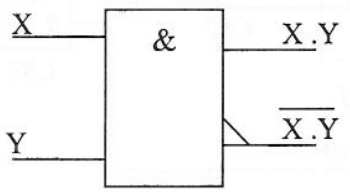


Figure 4.1

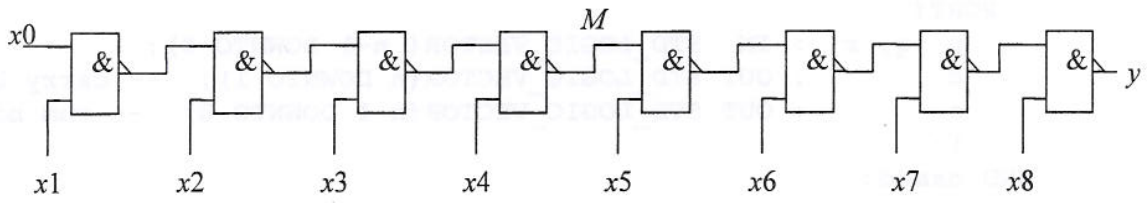


Figure 4.2

12.4.06
EJ.06 /EJC 7.5/ AC5

Question 1 is COMPULSORY, and constitutes 40% of marks, 72 minutes time.

Solution to Question 1.

a) Not synthesisable. Behaviour:

1) waits till rst='0'.

2) repeats 100 times, sets clk to 0, wait 10ns, set clk to '1', wait 5ns

3) waits till rst is '1', and then repeats from 1)

[4]

b)

```
ARCHITECTURE synth OF count IS
  SIGNAL po : STD_LOGIC_VECTOR(15 DOWNTO 0);
BEGIN
  pout          <= po;

  P1: PROCESS(clk,set)
  BEGIN
    IF set = '1' THEN
      po        <= limit;
    ELSIF clk'EVENT AND clk = '1' THEN
          IF po = limit THEN
            po <= (OTHERS => '0');
          ELSE
            po <= UNSIGNED(po)+1;
          END IF;
    END IF;
  END PROCESS P1;
END ARCHITECTURE synth;
```

[4]

c) a : 1 delta, b : 1 delta, c : 2 delta, d : 20us + 0 delta

[4]

d) *could implenent this using FOR GENERATE but this is more cumbersome*

```
ARCHITECTURE synth OF jiggle IS

BEGIN  -- synth

  P1 : PROCESS(x, y)
  BEGIN
    z       <= x;
    FOR i IN 0 TO size LOOP
      IF i MOD 2 = 1 THEN
        z(i) <= y(size-i-1) XOR y(i);
      END IF;
    END LOOP;
  END PROCESS P1;

END synth;
```

[4]

e)

```
ENTITY sq IS

  GENERIC (
    n :       INTEGER
    );
  PORT (
    x : IN    STD_LOGIC_VECTOR(n-1 DOWNTO 0);
    y : INOUT STD_LOGIC_VECTOR(n*n-1 DOWNTO 0)
    );
END sq;

ARCHITECTURE testsq_arch OF testsq IS
  SIGNAL y_int : STD_LOGIC_VECTOR(9 DOWNTO 0);
BEGIN  -- testsq

  I1 : ENTITY sq GENERIC MAP(10) PORT MAP(x => (OTHERS => '0'), y => y_int);
  I2 : ENTITY sq GENERIC MAP(10) PORT MAP(x => (OTHERS => '0'), y => y_int);

END testsq_arch;
```

[4]

Students must answer two questions from questions 2-4, each question caries 30% of marks and takes 54 minutes.

Solution to Question 2

This questions tests ability to write behavioural VHDL code.

3.

```
ARCHITECTURE synth OF fifo IS

  TYPE memtype IS ARRAY (0 TO size-1) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
  SIGNAL mem  : memtype;                    -- the RAM
  SIGNAL code : STD_LOGIC_VECTOR(1 DOWNTO 0);
  SIGNAL p, q : STD_LOGIC_VECTOR( counter_length-1 DOWNTO 0);
BEGIN
  ASSERT (size/4)*4 = size REPORT "size is not a multiple of 4 in fifo instance";

  code <= (inputitem, outputitem);

  P1 : PROCESS
  BEGIN
    WAIT UNTIL clk'EVENT AND clk = '1';
    CASE code IS
      WHEN "01" =>
         q <= conv_std_logic_vector(conv_integer(UNSIGNED(p)+1) MOD size, counter_length);
      WHEN "10" =>
        mem(p) <= datain;
        p      <= conv_std_logic_vector(conv_integer(UNSIGNED(p)+1) MOD size, p'LENGTH);
      WHEN OTHERS => NULL;
    END CASE;
  END PROCESS P1;

  P2 : PROCESS(datain, code, mem)
  BEGIN
    CASE code IS
      WHEN "01"   => dataout <= mem(conv_integer(UNSIGNED(q)));
      WHEN "11"   => dataout <= datain;
      WHEN OTHERS => dataout <= (OTHERS => 'Z');
    END CASE;
  END PROCESS P2;

  P3 : PROCESS(p, q)
  BEGIN
    nearlyempty <= '0'; nearlyfull <= '0';

    IF (UNSIGNED(q)-UNSIGNED(p)) < conv_signed(size/4, counter_length) THEN
      nearlyempty   <= '1'; END IF;
    IF (UNSIGNED(q)-UNSIGNED(p)) >= conv_unsigned(3*size/4, counter_length) THEN
      nearlyfull <= '1';
    END IF;
  END PROCESS P3;

END ARCHITECTURE synth;  -- of fifo
```

Solution to Question 3

This question tests ability to understand structural descriptions and write code for structural VHDL.

a)

```
ARCHITECTURE synth OF csadd IS
BEGIN
  P1 : PROCESS(p, q, r)
  BEGIN
    s <= p XOR q XOR r;
    c <= (p AND q) OR (p AND r) OR (q AND r);
  END PROCESS p1;
END synth;
```

b)

```
PACKAGE cspack IS
  CONSTANT mconst : INTEGER := 10;       -- change as necessary
  TYPE xtype IS ARRAY (0 TO 5) OF STD_LOGIC_VECTOR(mconst-1 DOWNTO 0);
END package cspack;

ENTITY csaddtree IS
  PORT(
    x : IN  xtype;
    y : OUT STD_LOGIC_VECTOR(mconst+2 DOWNTO 0)
    );
END ENTITY csaddtree;
```

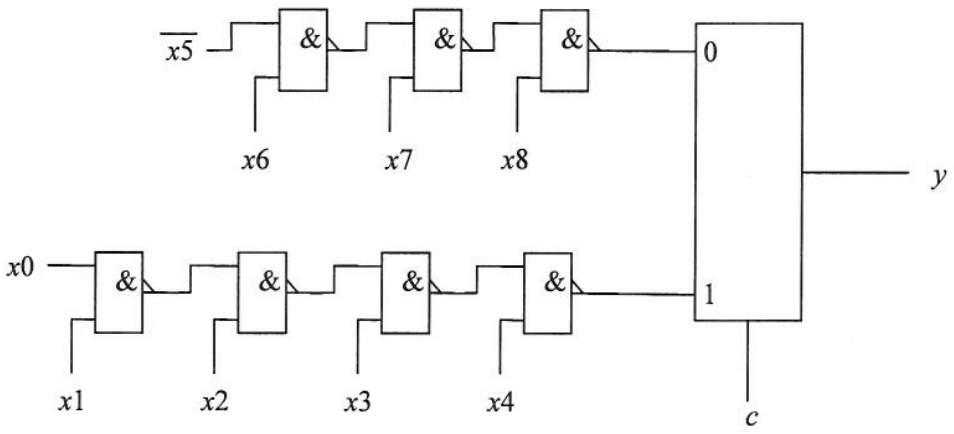
c)

```
ARCHITECTURE synth OF csaddtree IS
  SIGNAL s1, s2          : STD_LOGIC_VECTOR(mconst-1 DOWNTO 0);
  SIGNAL c1, c2, s3, q3  : STD_LOGIC_VECTOR(mconst DOWNTO 1);
  SIGNAL c3              : STD_LOGIC_VECTOR(mconst+1 DOWNTO 2);
  SIGNAL s4, q4, r4, p4  : STD_LOGIC_VECTOR(mconst+2 DOWNTO 1);
  SIGNAL c4              : STD_LOGIC_VECTOR(mconst+1 DOWNTO 0);
BEGIN
  CS0:ENTITY csadd GENERIC MAP(mconst) PORT MAP( x(0), x(1), x(2), c1, s1);
  CS1:ENTITY csadd GENERIC MAP(mconst) PORT MAP( x(3), x(4), x(5), c2, s2);

  q3 <= '0'& s1(mconst-1 DOWNTO 1);
  q4 <= '0' & s3 & s1(0);
  r4 <= "00" & s2;
  p4 <= c3 & "00";

  CS2 : ENTITY csadd GENERIC MAP(mconst) PORT MAP( c1, q3, c2, c3, s3);
  CS3 : ENTITY csadd GENERIC MAP(mconst+2) PORT MAP( p4, q4, r4, c4, s4);

  y <= UNSIGNED(c4 & '0')+UNSIGNED('0' & s4);

END ARCHITECTURE synth;
```

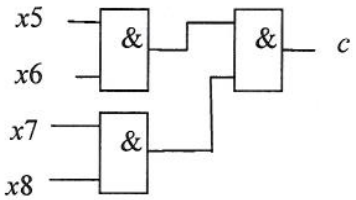
Solution to Question 4

This question is a new application of taught algorithms

a) c = x8.x7.x6.x5



b)



c) Before, 8. After, 6 (from x0,x1).

d)