E3.06
AC5
ISE3.5

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2003

MSc and EEE/ISE PART III/IV: M.Eng., B.Eng. and ACGI

VHDL AND LOGIC SYNTHESIS

Tuesday, 6 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 carry equal marks

Corrected Copy

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) : P.Y.K. Cheung

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.

The Questions

1.

(a) Write down a VHDL architecture for the entity *variable_count* shown in Figure 1.1, which implements a divide by 10 counter when *sel* is '0', and a divide by 11 counter when *sel* is '1'. The counter should be negative edge triggered from clock *clk*, with synchronous reset *rst*.

[6]

(b) Describe how the *input* and *output* signals of a combinational VHDL process may be derived from inspection of the process source code. State the conditions that must be satisfied by a VHDL process if it is to synthesise correctly to combinational logic.

[5]

(c) Write a synthesizable architecture for entity *adder* in *Figure 1.2* such that if a, b, c, d, x are interpreted as signed numbers, the output x is equal to $a+b+c+d$, and pos = '1' if and only if x is strictly greater than 0.

[6]

(d) The architecture in *Figure 1.3* is part of a testbench and generates signals *a,b,c,d*. Draw a dimensioned timing diagram showing the waveforms and physical times of events on signals *a,b,c,d* for the first 65ns of the simulation.

[6]

(e) Under what circumstances is the simulation delta of an event non-zero? Explain, giving a synthesisable VHDL example, how delta delays in simulation can mean that a synthesizable hardware description simulates incorrectly.

[6]

(f) Consider the testing of entity *adder* defined in part (c) of this question. Explain, with reference to this, the term *exhaustive testing*. Give reasons why exhaustive testing of this entity is impracticable and suggest a better test strategy.

[5]

(g) Write a synthesizable VHDL architecture for the entity in *Figure 1.5* that implements the FSM in *Figure 1.4* when reset is '0', and synchronously resets to state *s1* when reset is '1'.

[6]

```
ENTITY variable_count IS
PORT(
      clk, sel, rst: IN std_logic;
      count:  INOUT std_logic_vector( 3 DOWNTO 0)
);
END ENTITY variable_count;
```

Figure 1.1

```
ENTITY adder IS
PORT (      a,b,c,d: IN std_logic_vector(7 DOWNTO 0);
              x: OUT std_logic_vector(9 DOWNTO 0);
            pos: OUT std_logic
      );
END adder;
```

Figure 1.2

```
ARCHITECTURE behave OF testbench IS
  SIGNAL a,b,c,d: std_logic := '0';
BEGIN
      P3: PROCESS
      BEGIN
        a <= '1'; WAIT FOR 10 ns;
        b <= not b; c <= b; WAIT FOR 10 ns;
        d <= '1'; WAIT FOR 10 ns;
        d <= '0'; a <= '0';
      END PROCESS P3;
END ARCHITECTURE behave;
```

Figure 1.3

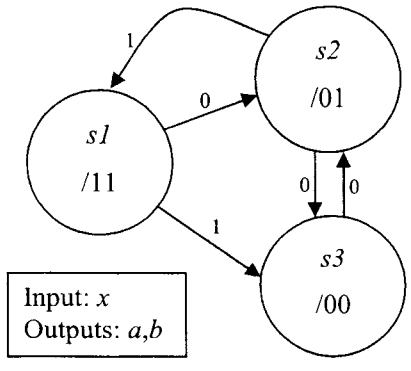


Figure 1.4

```
ENTITY fsm IS
PORT( clk, x, reset: IN
std_logic;
          a,b: OUT std_logic
);
END fsm;
```

Figure 1.5

2. The clocked computational element *sort_element* shown in *Figure 2.1* has clock *clk*, and data inputs *in_h*, *in_l*. It has three modes of operation on the clock edge, summarised in *Figure 2.2*. When input *mode* = "00" the values in internal data registers *data_h* and *data_l* are swapped if and only if *data_l* > *data_h*. When *mode* = "01" and *data_h* > *in_h*, *data_h* is loaded with *in_h*. Similarly, when *mode*="01" and *in_l* > *data_l*, *data_l* is loaded with *in_l*. Finally when mode="10" *data_l* is set to *data_h*, and *data_h* is set to *in_h*. Under all other conditions *data_l* and *data_h* are unchanged.

   In modes "00" and "01", combinational output *unchanged* is '1' during a clock cycle at the end of which the value in both registers will be unchanged, otherwise it is '0'. Outputs *out_h* and *out_l* contain the current values of *data_h* and *data_l* at all times.

   Operation of *sort_element* is don't care whenever *mode* = "11".

   The 16 bit data path contains numbers in sign and magnitude representation, in which the most significant bit is '1' if the number is negative, and the bottom 15 bits contain the absolute value of the number. Zero may be represented by either a '1' or a '0' sign bit.

(a) Write the body of a synthesizable VHDL function *sign_mag_greater* with header as below that returns value TRUE if its sign and magnitude parameter *a* is greater than its sign and magnitude parameter *b*.

```
FUNCTION sign_mag_greater(
        a: std_logic_vector(15 DOWNTO 0);
        b: std_logic_vector(15 DOWNTO 0)) RETURN BOOLEAN;
```

[6]

(b) Using function *sign_mag_greater*, write an entity and synthesizable VHDL architecture for block *sort_element* shown in *Figure 2.1* & *Figure 2.2*.

[12]

(c) A hardware sort engine is designed with 64 *sort_element* instances as in *Figure 2.3*. The block *sorter* is implemented using the entity *sorter* in *Figure 2.4*. In this block data is transferred to and from the 64 *sort_element* instances using cycles with mode = "10", during which *in_h* of the top *sort_element* is connected to *shift_in*. In all other modes *in_h* is connected to a constant *kmax* = "0111111111111111". The bottom *sort_element* has *in_l* always connected to the constant *kmin* = "1111111111111111". The combinational output *a* is '1' only when all of the *unchanged* outputs are '1'.

Write a synthesizable VHDL architecture for the entity *sorter*.

[12]

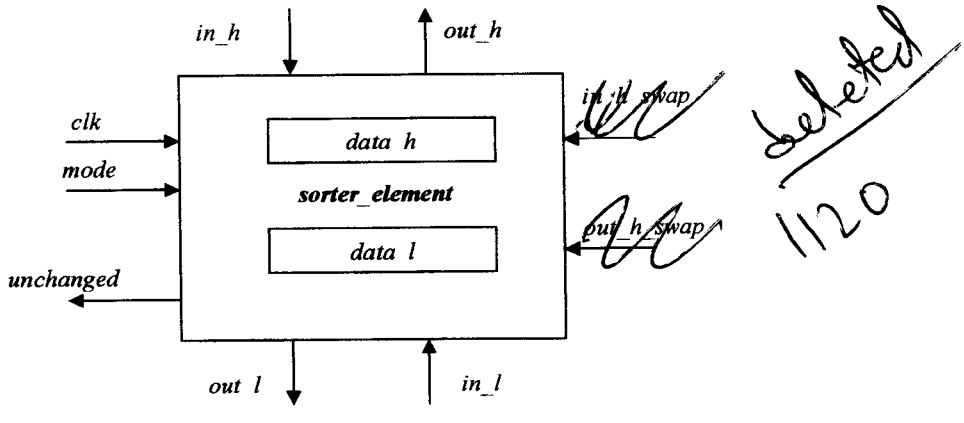


Figure 2.1

Mode "00" and Mode "10"

mode	data_h < data_l	data_h	data_l
00	No	data_h	data_l
00	Yes	data_l	data_h
10	X	in_h	data_h

Mode "01"

in_l > data_l	data_l
No	data_l
Yes	in_l

Mode "01"

in_h < data_h	data_h
No	data_h
Yes	in_h

Figure 2.2

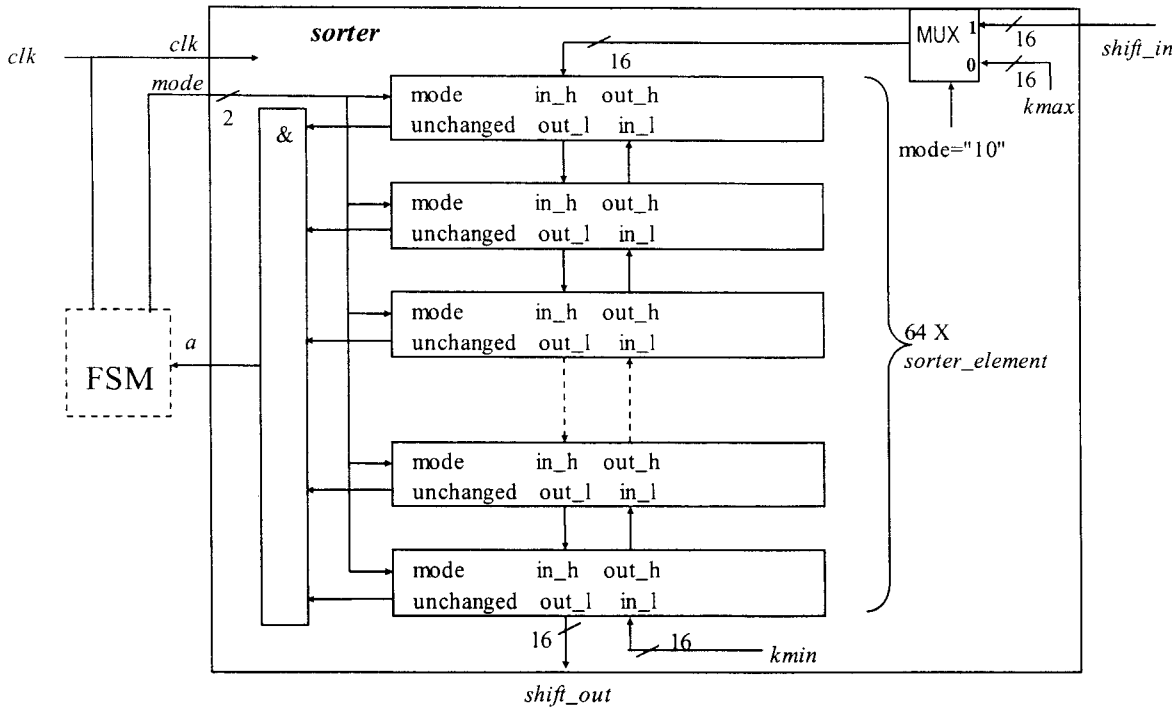


Figure 2.3

```
ENTITY sorter IS
PORT(       clk: IN std_logic;
            mode: IN std_logic_vector(1 DOWNTO 0);
            a: OUT std_logic;
            shift_in: IN std_logic_vector( 15 DOWNTO 0);
            shift_out: OUT std_logic_vector( 15 DOWNTO 0)
      );
END sorter;
```

Figure 2.4

3. In this question ⊕ represents Boolean operation XOR.

(a) Calculate the OBDD for the Boolean expressions $x1 \oplus (x2 \oplus (x3 \oplus x4))$ using variable order ($x1$, $x2$, $x3$, $x4$), and derive the corresponding ROBDD. Write down the sizes of the OBDD and ROBDD, and determine the sizes of OBDD and ROBDD in the case of an n-variable XOR. Hence show that ROBDDs can be much smaller than the corresponding OBDD.

[12]

(b) Write a synthesisable VHDL architecture for the entity *parity* given in *Figure 3.1*, such that the output z is '1' if and only if an odd number of the inputs $y(i)$ are '1'.

[6]

(c) Write a synthesizable fully structural VHDL architecture for the entity *big_parity* in *Figure 3.2*, using multiple instances of *parity*, as shown in *Figure 3.3*. VHDL definitions of the constants c, k in *Figure 3.3* are as follows:

```
CONSTANT c: INTEGER = m/10;
CONSTANT k: INTEGER = m - c*10;
```

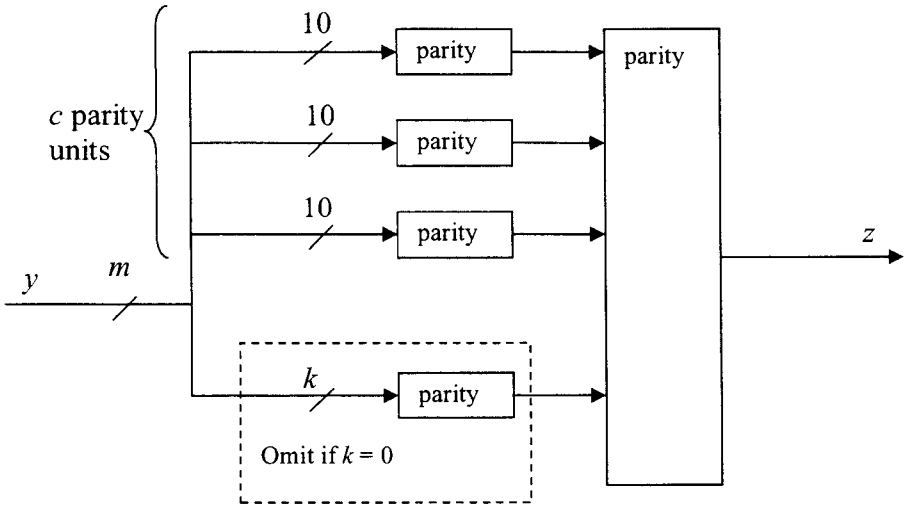
[12]

```
ENTITY parity IS
GENERIC( n: NATURAL);
PORT(
      y: IN std_logic_vector(n-1 DOWNTO 0);
      z: OUT std_logic
);
END ENTITY parity;
```

Figure 3.1

```
ENTITY big_parity IS
GENERIC( m: NATURAL);
PORT(
      y: IN std_logic_vector(m-1 DOWNTO 0);
      z: OUT std_logic
);
END ENTITY big_parity;
```

Figure 3.2



big_parity

Figure 3.3

4. This question relates to the design and implementation of a testbench for the fully synchronous entity *testable* defined in *Figure 4.1*. This entity has positive edge triggered clock *clk*. The two 15 bit unsigned inputs *x1,x2* are read when *start* is '1' (for a single cycle). After some number of cycles of computation the unsigned 16 bit outputs *y1*, *y2* become valid, during a single cycle, in which output *done* is '1'. On initial power up outputs are undefined for the first 1000 clock cycles. Providing that *start* is held '0' for this period, subsequent operation will be as specified. The entity calculates outputs as follows:

$y1 = fa(x1) + fb(x2)$

$y2 = fa(x1) - fb(x2) + 2^{14}$

where *fa*, *fb* are two constant functions with integer values in the range 0 to $2^{15} - 1$.

(a) You are given a package *test_pkg* containing behavioural VHDL functions *fa_behave* and *fb_behave* with headers as in *Figure 4.2* that evaluate to the correct values for functions *fa* and *fb*, and therefore can be used to determine the desired output of *testable*. Write a testbench *tb1* for *testable* that will test that the outputs *y1*, *y2* of *testable* are correct for a sequence of input pairs (*x1*, *x2*) read from a file `"stimulus_file"`. The inputs may be read either from a VHDL record typed file or from a text file. In the latter case you must specify the required input format.

[15]

(b) A testbench *tb2* for *testable* is proposed that will check that every possible output of constituent functions *fa* and *fb* is correct, using at most 2^{16} tests. Determine an appropriate set of tests and discuss the merits of *tb2* when compared with both random and exhaustive testing of *testable*.

[5]

(c) Suppose that it is known that functions *fa* and *fb* are implemented in *testable* using ROM lookup tables, where the top 9 bits of their arguments (*xhigh*) are used to index the table, and the bottom 7 bits (*xlow*) control linear interpolation of the result:

$fa(x) = (roma(xhigh)*(128 - xlow) + roma(xhigh + 1)*xlow)/128$

$fb(x) = (romb(xhigh)*(128 - xlow) + romb(xhigh + 1)*xlow)/128$

Determine the minimum number of tests necessary to test all values of the two tables *roma*, *romb*, specifying the test set that you propose. Assume that access to *roma*, *romb* is not directly available, so all testing must use *testable*. Describe an efficient test strategy for *testable*, including these tests, explaining the purpose of any additional tests that you propose.

[10]

```
ENTITY testable IS
PORT(
      clk, start: IN std_logic;
      done: OUT std_logic;
      x1,x2: IN std_logic_vector(14 DOWNTO 0);
      y1,y2: OUT std_logic_vector( 15 DOWNTO 0)
);
END ENTITY testable;
```

Figure 4.1

```
FUNCTION fa_behave(x: INTEGER) RETURN INTEGER;
FUNCTION fb_behave(x: INTEGER) RETURN INTEGER;
```

Figure 4.2