IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE
UNIVERSITY OF LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2000
MSc EXAMINATIONS
EEE and ISE PART III, B. Eng., M.Eng. and ACGI

VHDL & LOGIC SYNTHESIS

Tuesday, May 2: 10.00 - 13.00

There are SIX questions. *Answer FOUR*.

All questions carry equal marks.

Special information for invigilators: Students may bring any written or printed aids into this examination.

Information for candidates: None.

Examiners responsible: T. J. W. Clarke & P. Y. K. Cheung

1. *Figure 1a* depicts the design of a crossover switch which can connect 8 output lines to 8 input lines in any desired permutation. The switch is made up of an 8X8 matrix of switching elements. Each element is described in VHDL by a procedure with header:

```
PROCEDURE switch_element(
                  SIGNAL a: std_logic_vector;
                  s: IN INTEGER;
                  en: BOOLEAN;
                  SIGNAL i: IN std_logic;
                  SIGNAL o: OUT std_logic
                 );
```

The element operates as a tri-state buffer. The output, `o`, is high impedance when the element is off, and equal to its input, `i`, when the switch is on. The element is controlled by inputs `a`, `s` and `en`. When the unsigned binary value of `a` is equal to the binary number on `s`, and `en` is TRUE, the element will be on. Otherwise it will be off.

In the crossover switch there are 8 rows of elements, as shown in *Figure 1*. Each row connects to a common output line, `y(j)`. The matrix of elements has 8 control inputs `c(0)`, ..., `c(7)`, each 3 bits wide. The `a` input of each element is connected to the appropriate control input. The `s` input of each element is hardwired to a number equal to the index, `j`, of the corresponding output line `y(j)`. During normal operation, with `enable = '1'`, the values on the control inputs determine how signals pass from the `x` inputs to the `y` outputs. If `enable='0'` all elements are switched off.

(a) Write the procedure `switch_element.`

[7 marks]

(b) Write a VHDL entity declaration `crossover` which describes *Figure 1*.

[6 marks]

(c) Using `switch_element` write a structural implementation of your `crossover` entity.

[7 marks]

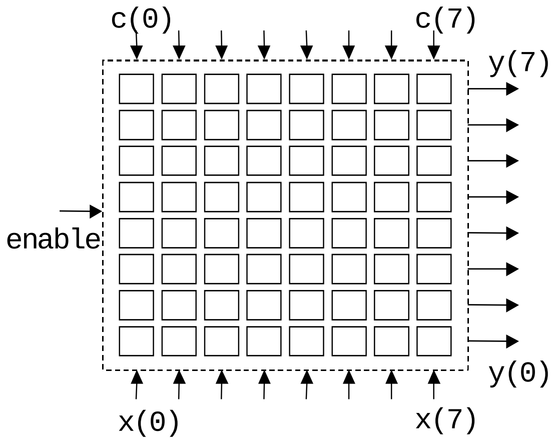


Figure 1

Questions 2 and 3 relate to the VHDL entity `register_file` *specified below.*

```
ENTITY register_file IS
  GENERIC(
    number_of_registers: NATURAL;
    word_size: NATURAL;
    tsetup, thold, tdelay: TIME
  );
  PORT(
    din: IN std_logic_vector( 1 TO word_size);
    dout: OUT std_logic_vector( 1 TO word_size);
    addr: IN INTEGER RANGE 0 to number_of_registers-1;
    cs, write_en, clk, reset: IN std_logic
  );
END register_file;
```

The register file has `number_of_registers` registers, each of `word_size` bits. Each register is positive edge triggered, with clock `clk`. The timing of the register file is defined by generics `tsetup`, `thold` and `tdelay`. Input `write_en` controls writing of data, and must be stable for a time `tsetup` before and `thold` after the active clock edge. If `write_en` is '1', the data on `din` is written into register `addr`. In this case it is an error for `addr` or `din` to change between `tsetup` before, and `thold` after, the active clock edge. If `write_en` is '0' no write occurs. The `reset` input operates asynchronously of `clk`, and forces all registers to zero when it is '1'.

The register file output, `dout`, is independent of the clock. If `cs` is '1' the output will be driven according to the value of register `addr`. If `cs` is '0' the output will be high impedance. The output `dout` has a propagation delay of `tdelay` from `addr`, `cs`, or `reset`. When new data is written into a register, there is a propagation delay to `dout` of `thold`+`tdelay` from the clock edge which caused the write.

2. *This question relates to the VHDL entity* `register_file` *specified above.*

   Write a synthesisable architecture for `register_file`. Your architecture must also simulate accurately the timing specification for `dout`, and provide warning messages if the input setup and hold timing constraints are not met.

   [20 marks]

3. *This question relates to the VHDL entity* `register_file` *specified on the previous page.*

(a) A set of stimulus vectors is defined by the recurrence:

$s_0 = 1$

$s_{i+1} = (31*s_i + 71)$ modulo 256

Write a VHDL function `ram_test_data` that has parameter *i* and returns an 8 bit binary vector with unsigned value equal to s_i.

[5 marks]

(b) Using your function, write a VHDL testbench for `register_file`, with 10 registers, each of 8 bits, and with setup time = 10 ns, hold time = 5ns, and `tdelay`=20ns. The testbench must reset the register file, and then successively write s_i into register *i* for *i* = 0 – 9. After each write operation the testbench must check the values of all 10 registers, and print an error if any of them are incorrect. The testbench must also check that `dout` is high impedance if `cs` = '0'. The testbench should provide stimulus generators and monitors so that:

(i) The setup and hold times required by the register file are met.

(ii) Any correct implementation of the register file with the specified propagation delay will pass the test.

(iii) An `addr` to `dout` propagation delay 1ns or more longer than the specification will result in a testbench warning error.

(iv) Other than as above, output delay times, and input setup and hold times, need not be explicitly tested.

At the end of the test the testbench must print out “Test completed”.

[15 marks]

4. A timebase module for a digital storage oscilloscope is shown in *Figure 2*. The module controls input of data to the oscilloscope memory. It has an input clock *clk*, and all operation is synchronous with the negative edge of this clock. The module outputs are *x_address* : a 10 bit X axis address, and *sample_now*. A *clk* cycle with *sample_now* = '1' will instruct the rest of the oscilloscope circuitry to input a new data sample to the ram location specified by *x_address*. The value of *x_address* starts at 0 and counts up to 1023. During this process *sample_now* will therefore be '1' for exactly 1024 cycles. The timebase logic determines the length of time between successive samples using the divide by *n* counter shown in *Figure 2*. Clock signals are omitted from this figure.

   The timebase module has two possible sampling rates, *a* and *b* cycles/sample. It is controlled by the three trigger signals *start_a*, *start_b*, *end_b*. A '1' on *start_a* enables sampling at a rate of *a* cycles/sample. Samples continue at rate *a* until *start_b* is '1'. Sampling then continues at rate *b* until *end_b* is '1', at which time the sampling rate returns to *a*. If 1024 samples have been taken, at any stage in this progression, sampling is disabled and the module enters a hold-off state for a time of *h*a* clock cycles. At the end of this time *x_address* is reset to 0 and sampling will be enabled again by the next ‘1’ on *start_a*.

   A Finite State Machine controls the operation of the timebase and has 5 states: *waiting*, *first_a_sweep*, *b_sweep*, *second_a_sweep*, *holdoff*.

   A VHDL Entity for the timebase module is shown below:

```
ENTITY timebase IS
PORT(
  reset, clk, start_a, start_b, end_b: IN std_logic;
  a, b, h: IN integer range 0 to 1023;
  x_address: OUT std_logic_vector(9 DOWNTO 0);
  sample_now: OUT std_logic
);
END timebase;
```

(a) Write a state diagram for the state machine in block FSM of *Figure 2*. Note that in general the outputs of this block may depend on its inputs, as well as the current state.

[8 marks]

(b) Write a synthesisable VHDL architecture for entity timebase.

[12 marks]

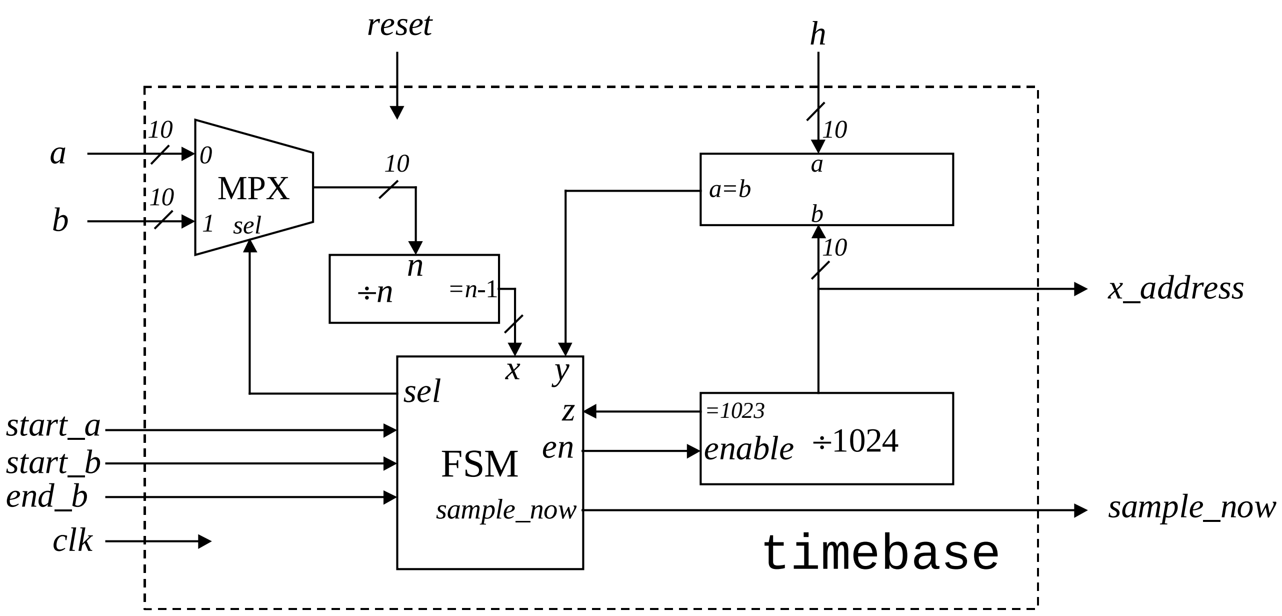

reset
h
a
b
10
10
0
1
MPX
sel
10
n
÷n
=n-1
10
a
a=b
b
10
x_address
x
y
sel
z
en
FSM
sample_now
start_a
start_b
end_b
clk
=1023
enable
÷1024
sample_now
timebase

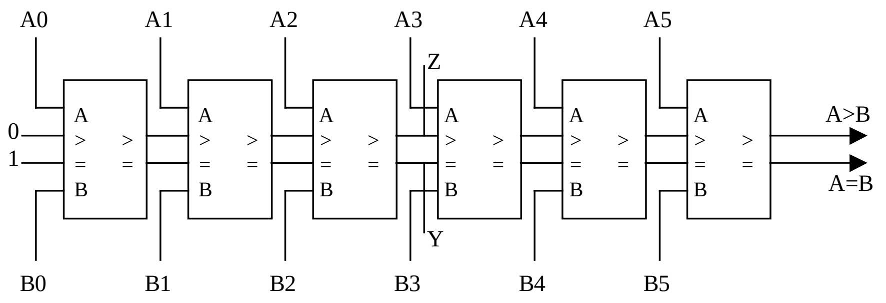
Figure 2

5.

(a) A 2 bit binary adder has inputs *A*0:1 and *B*0:1 and sum output *S*0:1, where 1 is the most significant bit. Write down an ordered binary decision diagram which represents the boolean expression for *S*1 with variable order (*A*1,*B*1,*A*0,*B*0), and simplify it to get a ROBDD. You need not provide any proof of your simplification.

[8 marks]

(b) *Figure 3* shows a circuit for a 6 bit binary comparator which compares words *A*0:5 and *B*0:5, where 5 is the most significant bit. Each comparison block has the truth table shown in *Figure 4*. You may assume that the delay through each comparison block is equal to 2 units from any input to any output. Use controllability factoring, applied at point *Z* for output *A*>*B*, and point *Y* for output *A*=*B*, to find an equivalent circuit with smaller maximum delay. You may use any combination of comparison blocks and 2 input multiplexors. You may also use the following gates: invertors, 2 input *xor*, 2 or 3 input *and* and *or*. Give the new circuit, and its maximum propagation delay from any input to output, assuming that a multiplexor has 2 units delay from any input to output, and all other gates have 1 unit delay.



[12 marks]

Figure 3

A_i	B_i	$>_{i-1}$	$=_{i-1}$	$>_i$	$=_i$
0	0	a	b	a	b
0	1	X	X	0	0
1	0	X	X	1	0
1	1	a	b	a	b

Figure 4

6. A floating point adder has entity:

```
ENTITY fp_adder IS
GENERIC(
  e_size: INTEGER := 8;
  m_size: INTEGER := 16
);
PORT(
  a, b: IN std_logic_vector( 0 TO e_size+m_size-1);
  c: OUT std_logic_vector( 0 TO e_size+m_size-1);
  clk: std_logic
);
END fp_adder;
```

The adder operates on floating point numbers represented by an exponent of `e_size` bits and a mantissa of `m_size` bits. The mantissa is a two's complement signed integer. The exponent is an unsigned integer. The exponent and mantissa are held in bits `(0 TO e_size-1)`, and `(e_size TO e_size+m_size-1)`, respectively, of the vectors `a`,`b`,`c` above.

The adder generates its output in three pipelined stages: *swap*, *align* and *add*. Each stage performs some arithmetic or logical function and stores its outputs in a positive edge triggered flip-flop clocked by `clk`, as shown in *Figure 5*. The numbers on multiple-bit data paths indicate their width, where *e* = `e_size` and *m* = `m_size`.

The *swap* stage contains a block ">" which performs unsigned comparison of the two exponents, and controls two multiplexors MPX. This stage swaps *a* and *b* if necessary to ensure that the exponent at *aa* is no greater than that at *bb*. The *align* stage block "`-`" calculates the difference in exponents, and block "ASR" has an output *q* equal to the arithmetic right shift of input *x* by input *n*. This stage shifts the mantissa of *aa* by an amount equal to the difference between the two exponents, so that both mantissas are scaled by the exponent of *bb*. Finally the *add* stage uses block "+" to add the two mantissas, and generate an *m*+1 bit signed result. The most significant *m* bits of this result are used for the mantissa of *c*, and the exponent of *c* is generated from that of *bb* by adding 1 in the "+1" block.

Write a synthesisable VHDL architecture for `fp_adder`.

[20 marks]

[Note to candidates: In *arithmetic right shift* the sign bit is shifted into vacated bit positions **]**

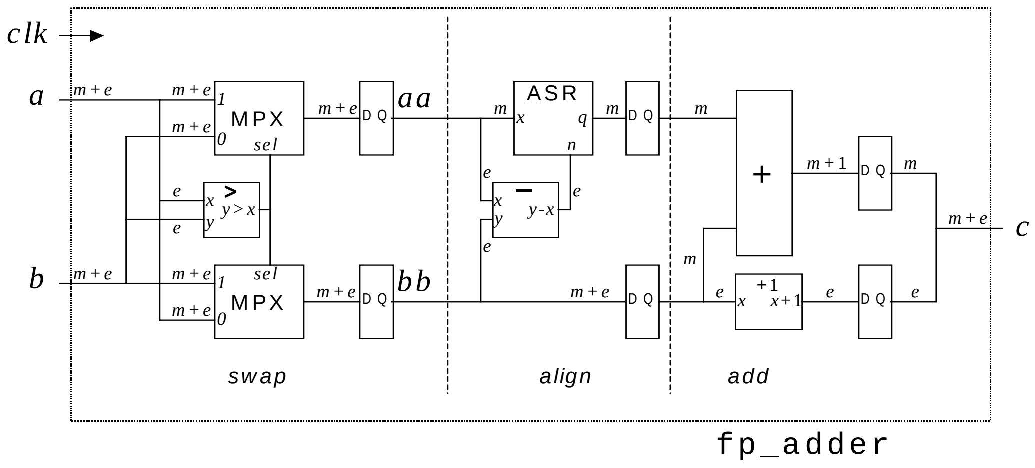


Figure 5