COMPUTER SCIENCE TRIPOS Part IB

Wednesday 4 June 2014 1.30 to 4.30 pm

COMPUTER SCIENCE Paper 5

*Answer **five** questions.*

*Submit the answers in five **separate** bundles, each with its own cover sheet. On each cover sheet, write the numbers of **all** attempted questions, and circle the number of the question attached.*

You may not start to read the questions printed on the subsequent pages of this question paper until instructed that you may do so by the Invigilator

STATIONERY REQUIREMENTS
Script paper
Blue cover sheets
Tags
Rough work pad

SPECIAL REQUIREMENTS
Approved calculator permitted

1 Computer Design

A novice SystemVerilog programmer has written the following decimal counter module which should zero the `decimal_count` on reset and then, when enabled, increment modulo 10 the `decimal_count` on every positive clock edge.

```
module count_decimal_wrong(
    input  logic clk;
    input  logic reset;
    input  logic enable;
    output logic decimal_count);

   always_comb @(posedge clk or reset)
     if(enable)
       begin
          decimal_count = decimal_count+1;
          if(decimal_count>9)
            decimal_count = 0;
       end
     elsif(reset)
       decimal_count = 0;

endmodule // count_decimal_wrong
```

(*a*) What bugs exist in the code and how can they be rectified? [10 marks]

(*b*) SystemVerilog synthesis tools use a Boolean optimiser to simplify the implementation logic.

(*i*) Why are *don't care* terms useful for Boolean optimisation? [3 marks]

(*ii*) How could the SystemVerilog be modified to introduce *don't care* terms for unreachable states above 9? [3 marks]

(*iii*) For a modern FPGA with 6-input look-up tables (LUTs), will Boolean optimisation result in fewer resources being used for the corrected `count_decimal`? Justify your answer. [4 marks]

2 Computer Design

(*a*) For RISC processors like ARM and MIPS, how is the work needed to undertake a subroutine call split between hardware and software? [6 marks]

(*b*) Operating system calls use some form of software exception. How is a software exception similar to a subroutine call, and how do they differ? [8 marks]

(*c*) Subroutine calls cause a control hazard. What is a control hazard and how are they handled on MIPS and ARM processors? [6 marks]

3 Computer Design

(*a*) What is a data cache and how does it differ from an instruction cache? [4 marks]

(*b*) What statistical properties of memory access do caches exploit to deliver improved performance? [4 marks]

(*c*) What impact will an operating-system-managed context switch have on cache hit rate? Justify your answer. [4 marks]

(*d*) Modern desktop and server processors support simultaneous multithreading (also called *hyperthreading*). When will there be a performance benefit in scheduling two non-interactive applications on the same hyperthreaded processor core so that they run in parallel rather than running sequentially, one job one after the other? [4 marks]

(*e*) For level-1 data caches using a snoopy cache coherency protocol, is a write-back or a write-through policy more likely to be used? [4 marks]

4 Computer Networking

(*a*) What is the difference between routing and forwarding? [2 marks]

(*b*) Routing algorithms can be either *link-state* or *distance-vector*. Define these two terms and explain the trade-offs between them. [6 marks]

(*c*) You are required to design a topology discovery protocol for a network of switching nodes interconnected by links. There are n nodes, l links, the maximum degree of any node is k and there is a path between any two nodes of not more than d hops. All links are bi-directional.

Each node has a unique identifier of four bytes which it knows.

(*i*) Describe a protocol for a node to learn about its immediate neighbours. You should specify the format of your messages and the size of any message fields. [4 marks]

(*ii*) Using the characteristics of the network described above, design a protocol for distributing this information across the network. You should specify the format of your messages and the size of any message fields. [8 marks]

5 Computer Networking

(*a*) Below is an excerpt from the DNS record for a fictitious corporation, Lemon:

Name	Type	Value	TTL (seconds)
`lemon.co.uk`	A	91.45.20.24	86400
`lemon.co.uk`	NS	`grove.lemon.co.uk`	86400
`lemon.co.uk`	NS	`tree.lemon.co.uk`	86400
`lemon.co.uk`	MX	`stem.lemon.co.uk`	60
`grove.lemon.co.uk`	A	91.45.23.22	86400
`tree.lemon.co.uk`	A	91.45.23.23	86400
`orchard.lemon.co.uk`	A	91.45.23.82	86400
`stem.lemon.co.uk`	A	91.45.23.85	86400
`www.lemon.co.uk`	CNAME	`orchard.lemon.co.uk`	86400

(*i*) If you type `http://www.lemon.co.uk` into your web browser, to which IP address will your web browser connect? [1 mark]

(*ii*) If you send email to `support@lemon.co.uk`, to which IP address will the message get delivered? [1 mark]

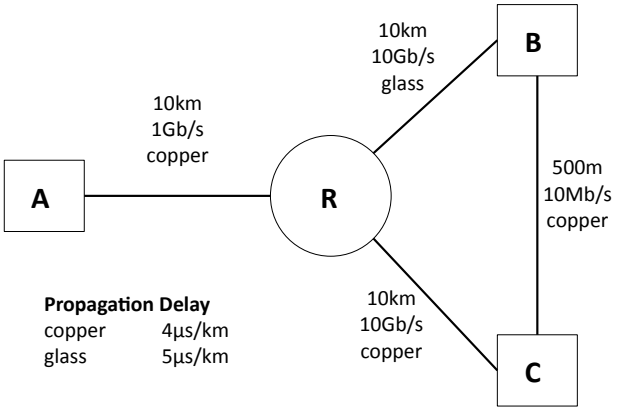
(*iii*) The TTL field refers to the maximum amount of time a DNS server can cache the record. Most of the TTLs in this record were chosen to be 86400 seconds (1 day). What is the trade-off between choosing a shorter or a longer time? Why was the MX record specifically chosen to have a 60 second TTL? [4 marks]

(*iv*) Explain why the Internet DNS uses caching. [2 marks]

(*v*) Comment on how the provision of name servers for `lemon.co.uk` affects the availabilty of the name service. [2 marks]

(*vi*) Outline two strategies to improve availability of the DNS server for the `lemon.co.uk` domain. [2 marks]

[continued ...]



(*b*) Consider the scenario shown above. Host A is sending tiny packets to hosts B and C. R is a store-and-forward switch with an average arrival rate of 10Gb/s and a buffer that contains, on average, 8MBytes of packet data. Delays due to the packet size and packet-processing are negligible.

Little's Law tells us that the average amount of buffered data equals the product of the arrival rate and the average delay experienced.

(*i*) What is the average delay that packets will incur going through the switch? [3 marks]

(*ii*) Compute the latency of the shortest path between each pair of end-nodes: A to B, A to C, and C to B. [3 marks]

(*iii*) Without changing the network propose a solution to decrease the delay between A and B. [2 marks]

6 Computer Networking

(a) Consider an unreliable message service where messages of a fixed size are sent between known endpoints. Outline the *minimum* set of additional features offered by a reliable byte-stream delivery service. [3 marks]

(b) A researcher notes that the message service, *fritter*, resembles a datagram service. It is prone to delivery delays of up to 1 second, message re-ordering and message loss. *Fritter* permits a 140-byte message to be relayed between any two users and each message is delivered without data-corruption.

You are asked to implement a *Stop-and-wait ARQ* to provide a unidirectional reliable byte-stream delivery service between two fritter users. Assume this is the only service between the two fritter users.

(i) Provide a labelled diagram illustrating the format for a *fritter* message that could be used by a reliable, byte-stream, delivery service. Justify your answer. [3 marks]

(ii) Draw and label the Finite State Machine that implements the sender portion of the Stop-and-wait ARQ. Your function will be called as *reliable_send()* while the fritter message receive and message send functions are *fritter_rcv()* and *fritter_send()* respectively. You may assume that the argument to the *reliable_send()* function does not exceed 100 bytes per function call. [8 marks]

(iii) Users assert that the performance using your Stop-and-wait ARQ is terrible for large transfers. Explain why they are correct. [2 marks]

(iv) Describe an appropriate enhancement to the ARQ that will improve performance. Given the constraints of a small *fritter* message size, justify why your particular ARQ enhancement is best suited to the *fritter* application. [4 marks]

7 Concurrent and Distributed Systems

Correct handling of time is critical to the correctness (and performance) of distributed systems such as network file systems and databases. Consider the following two approaches.

(*a*) Physical clock synchronisation

(*i*) Define *clock skew* and *clock drift*. [2 marks]

(*ii*) A client running Cristian's Algorithm observes a local clock time of 1399157100.00s at the start of its RPC, and 1399157100.10s at the end of its RPC. The RPC returns a server timestamp of 1399157100.05s. What client-server clock skew will the algorithm calculate? Justify your answer. [2 marks]

(*iii*) Using Cristian's Algorithm as the underlying primitive, propose a time synchronisation algorithm that measures and compensates for clock drift. [2 marks]

(*b*) Distributed logical clocks

The *make* build tool relies on file-system timestamps to determine whether an object file is older than a source file: if so, the object file is rebuilt; if not, then a rebuild is avoided. However, it is common practice to store source code in a distributed file system and object files in a local temporary directory. The Network File System (NFS) stores file creation and modification times based on the server clock, whereas the local file system uses a local clock.

(*i*) Describe the two failure modes *make* may experience if client and server clocks are out of sync. [2 marks]

(*ii*) One solution is to use NTP to synchronise client and server clocks. Describe two reasons why this might work poorly in practice. [2 marks]

(*iii*) The NFS developers decide that *Lamport Clocks* may be able to solve the problem, as they track the *happens-before* relationship. During a particular run, *make* finds a source logical timestamp s is less than the object logical timestamp o. Explain why it is problematic to use Lamport Clocks to conclude that the object file should not be recompiled. [4 marks]

(*iv*) Explain why *Vector Clocks* might be more suitable than Lamport Clocks for this problem. [2 marks]

(*v*) Explain what a Vector-Clock-based *make* should do if there is no defined *happens-before* relationship between source vector s and object vector o. [4 marks]

8 Concurrent and Distributed Systems

(*a*) Monitors are a programming primitive linking data with two synchronization types: mutual exclusion and condition synchronisation. Which is provided implicitly; which is provided explicitly? [1 mark]

(*b*) Describe two ways in which Monitors and Conditional Critical Regions differ. [2 marks]

(*c*) The object-oriented programming style encouraged by Monitors has many benefits as the number of data types and locks increases in the system.

(*i*) Placing all data in a single Monitor may improve program correctness. Explain why this might have undesirable performance effects. [1 mark]

(*ii*) One problem that can arise when using multiple locks is *deadlock*, which can be prevented by imposing a partial order on locks. Describe the implications this has for code structure when using Monitors. [2 marks]

(*iii*) Explain why Java's Monitor feature does not necessarily impose this code structure. [2 marks]

(*d*) Condition variables allow condition satisfaction to be signalled between threads. Explain the difference between Hoare's *signal-and-wait* and Mesa's *signal-and-continue* in terms of mutual exclusion and scheduling. [4 marks]

(*e*) Consider the (incorrect) pseudocode on the next page:

(*i*) Describe and justify minimal modifications to this code, referencing line numbers, in order to make it correct in the presence of Hoare *signal-and-wait* semantics. [4 marks]

(*ii*) Describe and justify minimal modifications to this code, referencing line numbers, in order to make it correct in the presence of Mesa *signal-and-continue* semantics. [4 marks]

[continued ...]

```
 1:  monitor ProducerConsumer {
 2:    int in, out, buf[N];
 3:    condition notfull, notempty;
 4:
 5:    procedure produce(item) {
 6:      if ((in-out) == N)
 7:        wait(notfull);
 8:      buf[in % N] = item;
 9:      if ((in-out) == 0)
10:        signal(notempty);
11:      in = in + 1;
12:    }
13:
14:    procedure int consume() {
15:      if ((in-out) == 0)
16:        wait(notempty);
17:      item = buf[out % N];
18:      if ((in-out) == N)
19:        signal(notfull);
20:      out = out + 1;
21:    }
22:
23:    /* init */ { in = out = 0; }
24:  }
```

9 Concurrent and Distributed Systems

Java Remote Method Invocation (RMI) is a Remote Procedure Call (RPC) system that allows invocation of methods on objects in remote Java Virtual Machines (JVMs).

(*a*) (*i*) Describe what it means for a class to be labelled as *serializable*. [2 marks]

(*ii*) Describe what it means for a class to be a *remote class*. [2 marks]

(*iii*) Describe the function of RMI's *object registry*. [2 marks]

(*b*) RPC implementations are unable to implement idealised *exactly-once* semantics, instead providing *all-or-nothing*, *at-most-once*, or *at-least-once* semantics.

(*i*) Describe the server-side state obligation for each of *all-or-nothing*, *at-most-once*, and *at-least-once* semantics. [3 marks]

(*ii*) Describe the classes of distributed filesystem RPCs in which it is safe to use *at-least-once* instead of *all-or-nothing* semantics. [2 marks]

(*iii*) A distributed filesystem stores object replicas on multiple servers for fault tolerance. Successful reads must be made to at least Q_r servers, and writes must be made to at least Q_w servers. The client submits requests to all servers simultaneously using a "MultiRPC". Describe when MultiRPC must resend (a) reads and (b) writes due to packet loss. [2 marks]

(*c*) Java is a garbage-collected language: when an object is unreachable, it will be freed. Cycles can exist in which otherwise disconnected objects reference one another, preventing reference counts from reaching zero. Cycle detection addresses this problem: execution is suspended and the object-reference graph is searched for memory that can be recovered. *Distributed garbage collection* faces many of the same challenges: cycles may exist between objects on different servers, which cannot be detected by local garbage collectors in individual JVMs.

(*i*) How does RMI track distributed references to a local object that has been exported to a remote client? [2 marks]

(*ii*) A developer implements a distributed garbage collector (GC) for RMI. The GC queries all nodes for graphs of local and remote references, identifying disconnected cycles spanning multiple JVMs. JVMs are notified of remote references that can be safely released to break cycles. Unfortunately, this does not work: when the system is busy, live references get broken, causing user-visible failures. Describe a scenario in which this algorithm, without additional synchronisation, may lead to incorrectness. [5 marks]

END OF PAPER