COMPUTER SCIENCE TRIPOS Part IB

Wednesday 7 June 2017 1.30 to 4.30

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.*

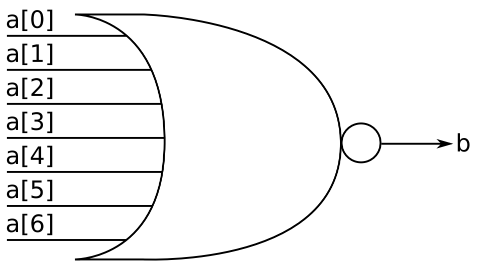
You may not start to read the questions printed on the subsequent pages of this question paper until instructed that you may do so by the Invigilator

STATIONERY REQUIREMENTS	SPECIAL REQUIREMENTS
Script paper	*Approved calculator permitted*
Blue cover sheets	
Tags	

1 Computer Design

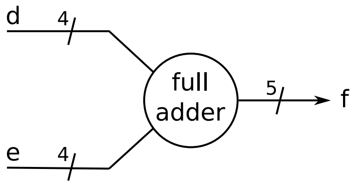
(*a*) Write one or more lines of SystemVerilog that correspond to a complete module for each of the following circuits. Aim for simplicity and explain any subtleties of your implementation.

(*i*)



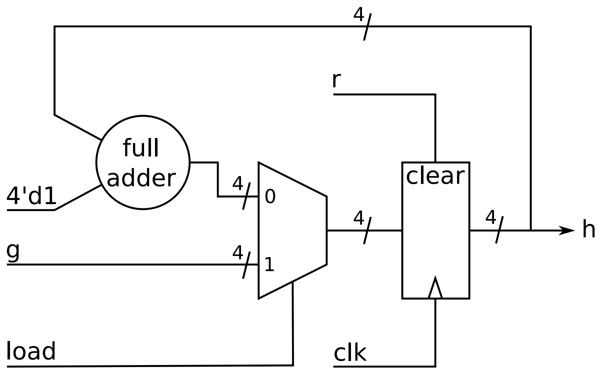
[4 marks]

(*ii*)



[4 marks]

(*iii*)



[4 marks]

(*b*) Consider the following SystemVerilog module:

```
module gcd(
           input  logic        clk,
           input  logic        rst,
           input  logic        start,
           input  logic [15:0] Ain,
           input  logic [15:0] Bin,
           output logic [15:0] answer,
           output logic        done
           );

   logic [15:0] a, b;
   always_ff @(posedge clk or posedge rst)
     if(rst)
       begin
         a <= 0;
         b <= 0;
         answer <= 0;
         done <= 0;
       end
     else
       if(start)
         begin
           a <= Ain;
           b <= Bin;
           done <= 1'b0;
         end
       else if(b==0)
         begin
           answer <= a;
           done <= 1'b1;
         end
       else if(a>b)
         a <= a-b;
       else
         b <= b-a;
endmodule
```

(*i*) For inputs `Ain=21` and `Bin=15`, complete the following state transition table (where `X` means undefined or unknown values): [6 marks]

input			current state		next state			
`Ain`	`Bin`	`start`	`a`	`b`	`a'`	`b'`	`done'`	`answer'`
21	15	1	X	X	21	15	0	X
21	15	0	21	15				

(*ii*) What values of `Ain` and `Bin` will cause the module to fail to terminate (i.e. `done` will never go high)? [2 marks]

2 Computer Design

Consider the following statements about processors. Why are they fallacies?

(*a*) A benchmark is a typical program that accurately predicts performance. [4 marks]

(*b*) Instructions per second is an accurate measure to compare performance among computers. [4 marks]

(*c*) Peak performance tracks observed performance. [4 marks]

(*d*) You can design an ideal processor suitable for all applications. [4 marks]

(*e*) Adding more pipeline stages always improves performance. [4 marks]

3 Computer Design

A shared-memory multicore processor contains three cores, each with a private L1 cache connected to a bus along with a shared L2 cache. Coherence is maintained through a basic MSI cache coherence protocol.

(*a*) What is a shared-memory multicore processor? [2 marks]

(*b*) Contrast the memory hierarchy described above against one containing multiple private L2 caches (with the same total L2 cache space). [6 marks]

(*c*) Describe whether each of the following scenarios represents a valid combination of coherence states across the L1 caches for data at a particular memory address. For example, [M, M, M] represents the data being in state M in each of the three L1 caches in the processor.

(*i*) [M, M, M]

(*ii*) [S, S, S]

(*iii*) [I, I, I]

(*iv*) [M, S, I]

[8 marks]

(*d*) Describe the disadvantages of the basic MSI protocol in each of the following scenarios.

(*i*) A core modifies private data that it has already read.

(*ii*) A core reads data that is already in another core's L1 cache.

[4 marks]

4 Computer Networking

(*a*) Two network-architecture approaches are commonplace: *hop-by-hop*, and *end-to-end*.

Briefly compare these two approaches with reference to the encryption of a web-page and the use of WPA2 (or similar schemes) to secure WiFi. [4 marks]

(*b*) Consider four components that constitute delay for a packet network: queueing delay, processing delay, propagation delay, and transmission delay.

(*i*) Order these delays by magnitude, giving typical values for each and making clear your justification. [8 marks]

(*ii*) (A) Describe circumstances where processing delay for one packet type varies significantly from the mean processing delay of a packet. [2 marks]

(B) Estimate what such a difference in delay might be. State your assumptions and show your working. [2 marks]

(*iii*) Packet-network delays of a typical datacentre may vary significantly from those found in the Internet.

Discuss why this should be the case, with particular reference to the Bandwidth-Delay product and discuss the implications for network architectures in datacentres where we wish to minimise delay. [4 marks]

5 Computer Networking

(*a*) A router vendor is interested in building an ultra low-cost router. Analysis reveals that the total build-costs are 90% for linecards and 10% for control processors. The router vendor focuses on reducing linecard costs.

Discuss which of the following strategies would help the vendor achieve this goal. Keep in mind that high-speed memory is expensive.

(*i*) Develop a faster implementation of Dijkstra's algorithm. [2 marks]

(*ii*) Convince the IETF to eliminate IPv4 fragmentation support. [2 marks]

(*iii*) Convince the IETF to abolish multi-homing. [2 marks]

(*iv*) Convince operating systems vendors to implement packet "pacing" so end hosts will not generate bursts of back-to-back packets. [2 marks]

(*b*) You are running a TCP-based movie streaming service called MeTube. Several of your users are complaining about poor performance. You call in your team of three engineers to examine the problem. They all observe that the 10Gbps access link to the MeTube server is only 25% utilised, and that the server's CPU is only 15% utilised. Each engineer then independently reports back with the following conclusions and suggestions for improvements.

Consider each of the engineers' conclusions and discuss why it is correct or not.

(*i*) Engineer Phil notices that a number of packets are being retransmitted multiple times. Phil recommends the TCP implementation be modified reducing both the timeout period and the fast retransmit threshold from 3 to 2 dupAcks (duplicate Acknowledgments). While this may lead to even more retransmissions, Phil concludes this is OK as the server's access link is only 25% utilised. With these changes, users will receive lost packets faster and hence will definitely see improved performance. [4 marks]

(*ii*) Engineer Leslie runs traceroute from the MeTube server to impacted users and finds that a router R1 along the path repeatedly drops her traceroute messages. Leslie concludes that the high packet drop rates at R1 are the cause of their performance problems. [4 marks]

(*iii*) Engineer Chris notes that users reporting problems have IP addresses belonging to the Classless Inter-Domain Routing (CIDR) block allocated to Tiny-Telco. She believes the problem may be due to high loss rates involving Tiny-Telco's network. Chris recommends they contact Tiny-Telco and ask them to diagnose and fix the problem. [4 marks]

6 Computer Networking

(*a*) Describe *on-off* flow control. In what circumstances is it appropriate? [4 marks]

(*b*) Describe the operation of *window-based* flow control. [4 marks]

(*c*) What happens if a flow using window-based flow control passes through a highly-loaded switch or router? You may assume the switch or router does not participate in the flow control protocol. [4 marks]

(*d*) How is this addressed in the TCP protocol? [4 marks]

(*e*) What are the advantages and disadvantages of having Internet routers participate in window-based flow control of every TCP connection? [4 marks]

7 Concurrent and Distributed Systems

SimplisticFS is an in-memory filesystem, implemented as a directed and (mostly) acyclic graph in which nodes represent *directories* (which contain names and pointers to other nodes, including a special entry named `‘‘..’’` that points back to its parent node) or *files* (leaf nodes that contain data). The in-memory structure, with the exception of `‘‘..’’` entries in directories, is therefore a tree. SimplisticFS does not support hard links, and the `‘‘..’’` of the root node points back to itself. Path lookups start at the root node, and pathnames with multiple segments, separated by `‘‘/’’`, are implemented as lookup operations on successive nodes. For example, opening `‘‘/foo/bar’’` will look up `‘‘foo’’` in the root, and then `‘‘bar’’` relative to the `foo` node.

Fine-grained locking adds a read-write lock to each node to ensure safe concurrent access. Operations for reading a directory entry (e.g., to list the contents or look up a child), or for reading file data will: acquire the node lock for read; perform the operation; and then release it. Operations for mutating a directory entry, (e.g., to add or remove a child node of a directory), or for modifying file data will: acquire its lock for write; perform the operation; and then release it. The lock implementation permits read recursion, write recursion, and race-free lock upgrades from read to write by threads.

(*a*) For (*i*) files and (*ii*) directories, explain, giving examples, how using a read-write lock may improve performance compared to mutual exclusion. [4 marks]

(*b*) The developers discover that compound operations, such as recursive path lookup, suffer from race conditions. They decide to adopt *strict two-phase locking* across compound operations to resolve this problem.

(*i*) Define *strict 2-phase locking* and describe how to apply it. [4 marks]

(*ii*) Explain why deadlock cannot occur prior to this change. [2 marks]

(*iii*) The new strategy suffers deadlocks when files are removed under high load. Given that removing a file requires a write lock on its parent directory, give an example of how a deadlock might occur. [4 marks]

(*c*) Moving from a “giant lock” to this finer-grained model focused on files and directories improves performance for some but not all workloads.

(*i*) Describe and explain an example of a workload in which file-granularity locking is unlikely to eliminate lock contention. [2 marks]

(*ii*) Propose a more granular locking strategy to improve parallelism with respect to (*c*)(*i*). Describe the potential performance benefit with respect to that workload, and additional overhead that might be incurred. [4 marks]

8 Concurrent and Distributed Systems

The developers of *FictionalOS* have an implementation of the *Network File System version 3 (NFSv3)* without support for distributed locking. As a result, applications experience race conditions when operating concurrently on files in NFS. Rather than using the hideously complex NFS distributed locking protocol, the OS developers decide to develop their own simpler locking mechanism ("How hard can it be?"). They add two new NFS *remote procedure calls* (RPCs) that can be used on file nodes in NFS: `NFS_LOCK` and `NFS_UNLOCK`, which lock and unlock nodes, respectively. These are implemented through simple atomic operations on the in-memory `node` data structure representing a file on the NFS file server:

```
nfs_lock(node) {
  atomic {
    if (node->lock_held != 0)
      return (FAILURE);
    node->lock_held = 1;
    return (SUCCESS);
  }
}
```

```
nfs_unlock(node) {
  atomic {
    node->lock_held = 0;
    return (SUCCESS);
  }
}
```

SunRPC retransmission ensures reliable delivery when packets are lost. If a client receives `FAILURE`, it will issue new RPCs each second until it receives `SUCCESS`.

(*a*) Explain why, when server reboots, concurrent applications writing to files across multiple nodes may suffer data races despite acquiring suitable locks, and describe a solution to this problem. [4 marks]

(*b*) (*i*) Define *at-least once* RPC semantics. [1 mark]

(*ii*) Explain why *at-least once* RPC semantics may cause trouble for each of the `NFS_LOCK` and `NFS_UNLOCK` RPCs. [4 marks]

(*c*) Explain why the polling nature of the `NFS_LOCK` RPC, as described, makes it difficult to implement reliable server-side deadlock detection. [4 marks]

(*d*) (*i*) Define *priority inheritance* and explain what problem it solves. [2 marks]

(*ii*) Describe the changes to the `NFS_LOCK` and `NFS_UNLOCK` RPCs necessary to implement priority inheritance. Include any new RPC arguments and return values. Explain the changes (and limitations) this imposes on software implementations. [5 marks]

9 Concurrent and Distributed Systems

This pseudocode, executing in process P_j, employs buffering to impose ordering:

```
receive(M from Pi) {         // Message M received from process Pi
  S = getSeq(M);             // Extract sequence number S
  if (S == nextSeq(Sji)) {   // If S is the next sequence number:
    deliver(M);              //   Deliver M to current process (Pj)
    Sji = flush(HBQ, Sji);   //   Deliver backlog from HBQ; update Sji
  } else holdback(HBQ, M);   // Else: Hold back M for future delivery
}
```

(*a*) Explain what ordering model(s) this pseudocode implements. [2 marks]

(*b*) Write pseudocode (with comments) for the following functions, to be used on the sender (P_i) or receiver (P_j), which accept M (a message), and S (a sequence number): [8 marks]

Receiver	`receive_reliably(M)`	Reliably receive M from P_i.
Sender	`send_reliably(M)`	Reliably send M to P_j.
Sender	`process_ack(S)`	Handle a received ACK for S from P_j.
Sender	`timeout(S, M)`	Process a timeout for S and M.

As needed, employ the following additional utility functions:

`drop(M)`	Drop received M without delivering.
`setSeq(M, S)`	Set sequence number S on message M.
`transmit_msg(M)`	Transmit message M to P_j.
`transmit_ack(S)`	Transmit an ACK with sequence number S to P_i.
`sched_timeout(S, M)`	Schedule `timeout(S, M)` to run in 5 ms.
`cancel_timeout(S)`	If scheduled, cancel timeout for S.

(*c*) Define the *happens-before* relationship. [2 marks]

(*d*) The pseudocode above imposes ordering on pair-wise communications. Assuming reordering but no message loss, write pseudocode (with comments) for the following functions supporting *causal ordering* for group communications: [8 marks]

Receiver	`receive_causally(M)`	Causally receive from the group.
Sender	`send_causally(M)`	Causally send to the group.

As needed, employ the following additional utility functions:

`getVec(M)`	Retrieves the version vector from a message.
`setVec(M, V)`	Set vector V on message M.
`testVec(LV, RV)`	Returns whether vector RV only differs from LV in that it has exactly one entry one greater than the corresponding entry in `LV`.
`updateVec(V)`	Returns V with the local vector entry incremented.
`transmit_group(M)`	Transmits message M to the entire group.

END OF PAPER