UNIVERSITY OF ALBERTA

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ECE 315 – Computer Interfacing

Midterm Examination

Instructor:	B. F. Cockburn
Exam date:	October 23, 2013
Exam duration:	50 minutes
Aids permitted:	A paper or electronic copy of the course lecture slides can be freely consulted. The Internet cannot be accessed by any device. Electronic calculators of any kind are permitted. The model solutions for this year's Assignments #1 and 2 can be consulted.

Instructions:

1. Please enter your printed name, signature and I.D. number on this page.
2. Verify that this booklet contains 6 pages (including the cover page).
3. Neatly enter your answers in the spaces provided.
4. Use the reverse sides of the pages for rough work.
5. Take into account the marks per question when budgeting your time.

Student name: _____Model______,__Solutions___
Family name Given name

Signature: ____________________________________

Student I.D.: ____________________________________

Question	Time	Worth	Mark	Subject
1.	8	15		Basic Concepts
2.	12	25		Multitasking Software
3.	18	35		Hardware Interfaces
4.	12	25		TCP/IP Networking
Total	**50 mins**	**100**		—

Question #1 (Basic Concepts)

Summarize the similarities and differences of the following pairs of concepts:

(a) Global vs. local variables [5 marks]

A **global variable** is a variable that is outside the scope of any function, including main(), and is accessible by default from any function. However, access to a global variable is lost inside a scope that defines a local variable with the same name as the global variable. A **local variable** is a variable that is defined and thus only accessible within a limited "scope". In C, scopes can be formed in several ways: typical scopes include the bodies of functions and within control constructs (e.g. for and while loops). Variables defined inside the main() function are local to the scope of main(), and are actually not global variables. In a MicroC/OS environment we are already within the main() function, but it is common to refer to variables defined outside of all tasks as "global variables."

Similarities: Local and (unshadowed) global variables can be read and written in the same way by code in the local scope. **Differences:** A local variables cannot be accessed by code outside the local variable's scope. (Note that "static" local variables exist outside the local scope, but they remain inaccessible outside that scope.)

(b) Blocked task vs. interrupted task [5 marks]

A **blocked task** is a task that is not executing because it requires some condition to go from false to true. The task was removed by the kernel from the running state and put into the blocked state on a queue of tasks currently blocked on the condition. An **interrupted task** is a task that is not executing because the hardware interrupt handling mechanism has decided to recognize and service an active interrupt-producing condition. The currently running task is then temporarily blocked while the interrupt service routine (ISR) executes. In a multitasking system, at the end of the ISR there will be a check to see if (a) the interrupted task should be restored, (b) a newly unblocked state should replace the interrupted task.

Similarities: In both cases, the task is not currently executing. A blocked task is replaced by another running task, while an interrupted task is replaced by the ISR. **Differences:** A blocked task cannot execute because some required condition is false. An interrupted task was executing before the interrupt, and will be restored to the running state when the ISR finishes executing (unless a context switch has occurred).

(c) The & and * operators in C [5 marks]

The **& operator** returns a reference to the following object (e.g, variable, data structure, function), which is the address of the first byte of that object. The *** operator** returns the object at the given following reference. Not that * and & cancel out when used together, so *&var_name has the same as var_name.

Similarities: Both operators involve the association between an object and its address. **Differences:** The & operator goes from the object to its address, while the * operator goes from a pointer value (a reference) to the corresponding object.

Question #2 (Multitasking Software)

(a) An idle task is automatically provided in the MicroC/OS pre-emptive multitasking kernel. What important purpose is played by having an idle task in a real-time multitasking system? What kinds of useful work can be completed by the idle task?

[10 marks]

The idle task provides a convenient mechanism for absorbing the idle time left over from the foreground tasks that are accomplishing the real-time system functions. There must be some idle time in a real-time system so that the timing of the foreground tasks is predictable, and so that those tasks do not fall behind in their workload.

The idle task can be used to perform work that does not need to meet hard real-time constraints. Such work might include calculating and displaying the system work load, backing up files, performing system diagnostics, etc.

(b) Assume that you have been given a microcomputer with a kernel that has a hardware timer that produces interrupts at a fixed frequency (e.g., 200 Hz). The kernel provides a timer interrupt service routine (ISR) that changes the running task to the next task on a list of ready-to-run tasks. How would you modify this system to provide two classes of tasks: high priority and low priority. The high priority tasks are to have three times as much execution time compared to low priority tasks.

[15 marks]

The timer ISR could maintain a list of the ready-to-run tasks in which each task entry also records two associated priority levels (high and low priority), and an index that shows which priority is currently being used for that task. Thus each task would be assigned two priorities, and there would still be no more than one task for every system priority. The ISR would give three consecutive time slots to every high-priority task, and one time slot to every low-priority task. A system function would need to be provided that would be used to update the list of tasks to show which of the two reserved priorities for that task was being used.

Question #3 (Hardware Interfaces)

(a) Briefly define what is meant by a noise margin at a digital hardware interface. What is the main purpose of having noise margins?

The high noise margin is the difference between the lowest produced “high” output signal and the lowest acceptable input “high”. Similarly, the low noise margin is the difference between the highest acceptable input “low” and the highest produced output “low”. The noise margins are the high and low noise margins together.

The noise margins produce an allowance for random noise or an unexpected signal offset that might be picked up when a signal is transmitted over a communications medium. The noise margins allow a greater proportion of digital highs and lows to be detected from a slightly noisy signal without detection errors.

(b) The laboratory microcomputer uses a Micrel Ethernet PHY chip in combination with the Freescale MCF5234 Microcontroller Unit (MCU). What would be some of the main reasons why the MCF5234 was likely not designed to include the functions that are provided by the PHY? What are the main advantages and possible disadvantages to having a separate PHY chip?

Advantages: Omitting the PHY function from an MCU, like the MCF5234, allows the MCU to benefit from a standardized chip that can be optimized and mass-produced at the lowest possible cost by another company. The engineering cost of including an on-chip PHY could be omitted. If an MCU, like the MCF5234, provides a standard MAC-PHY interface, then the MCU can be easily interfaced to work with different standard PHYs. Thus the PHY could easily be upgraded while using the same MCU.

Disadvantages: The MCU without PHY might be less desirable for some designs because an extra external PHY chip is required to implement the network interface. The assembly cost would be a bit higher (more pins to solder, more board area required), system power would be a bit higher (more driven pin signals) and system reliability would be a bit lower (more pin connections to fail). If the PHY were to be integrated, then the per-unit micro-computer system assembly cost would be a bit lower, and system reliability would be slightly higher (due to fewer pin connections).

Question #3 (Hardware Interfaces, cont'd)

(c) Interrupts can be used to implement software that operates a hardware interface that loads data into a transmitter circuit. What normal operating condition would cause transmitter interrupts to occur? What basic actions would the interrupt service routine (ISR) need to perform to handle transmit interrupts? Why is interrupt-driven software an efficient way of triggering transmitter service operations versus the alternative of busy waiting on a status bit in a readable transmitter register?

There are several possible causes of interrupts in transmitter circuits. One interrupt that would be produced in normal operation is the event when the transmitter is ready to be loaded with more data to be transmitted. This could occur when the transmitter data buffer is completely empty, or only partially empty but ready to accept more data to be transmit.

The interrupt service routine (ISR) for the event described above would need to: (1) Determine if more data is available to be transmitted. If no data is available, then further transmitter interrupts need to be disabled. (2) Data is available, and so the transmitter data buffer is loaded with the next data to be transmitted.

Interrupt-driven software is more efficient than busy waiting software that repeatedly reads a status bit because useless reads on the status bit are avoided. The interrupt-triggered hardware automatically waits for the important events, and only after the events of interest have occurred will a hardware interrupt signal be produced and the CPU notified. The CPU is freed up to deal with other work for most of the time.

Question #4 (TCP/IP Networking)

(a) The Internetworking Protocol (IP) provides a connectionless datagram delivery service, with no guarantee of delivery. IP has the right to break up datagrams into smaller datagrams, or to assemble a larger single datagram from component datagrams. Briefly explain why it is convenient to define a service like IP, with the above characteristics, when attempting to get a variety of different underlying hardware networks to work together as one network.

[12 marks]
Different data networks can have different maximum allowed packet lengths. It is therefore convenient that IP has the flexibility to create data payloads of different lengths that can be sized to efficiently fit within the payload size limitations of the underlying physical layer networks. This flexibility helps IP to connect together different networks into a single "internetwork". The absence from IP of a guarantee of data delivery to the destination node is the weakest possible quality of service, which any underlying network can provide.

(b) The correctness of the data that is transported by TCP/IP is protected by means of different checksums that are computed by both TCP and IP. Briefly explain how these two checksums succeed in protecting the payload as well as the TCP header and IP header against bit errors during transmission. You do not have to explain how the CRC calculation is performed.

[4 marks] The IP checksum protects against errors that might occur in the IP header only. However, this checksum does not provide protection over the IP payload (which is made up of the TCP header plus the TCP payload).

[5 marks] The TCP checksum protects against errors in the TCP header and the TCP payload (which contains the application data).

[4 marks] The IP checksum is recomputed by each IP entity that handles each IP datagram along its path to the destination node.. If the recomputed checksum fails to matchh, the datagram is discarded. Similarly, the TCP checksum is recomputed at the far end of a TCP/IP connection, and if this fails to match, the datagram is discarded. TCP/IP will eventually detect a missing datagram, and will keep on resending a datagram until all of the payload information reaches the destination intact. Thus the IP and TCP checksums, taken together, protect against errors anywhere in the TCP/IP packets.