UNIVERSITY OF ALBERTA

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ECE 315 – Computer Interfacing

Final Examination

Instructor:	B. F. Cockburn
Exam date:	April 27, 2017, 2:00 pm
Exam duration:	120 minutes
Aids permitted:	A copy (paper or electronic) of the lecture slides can be freely consulted. A two-sided 8.5” × 11” formula or summary sheet can be consulted. No Internet access is permitted using any kind of device. Electronic calculators (all kinds) and an English dictionary are permitted.
Instructions:	1. Enter your printed name, signature and I.D. number on this cover page. 2. Verify that this booklet contains 9 pages (including this cover page). 3. Neatly enter your answers in the spaces provided. 4. Use the reverse sides of the pages for extra space or rough work.

Student name: ____________________,________________
Last name **First name**

Signature: ______________________________________

Student I.D.: ______________________________________

Question	Time	Worth	Mark	Subject
1.	15	12		Fundamental Concepts
2.	15	12		Multitasking Software
3.	15	12		Multiple Nonpreemptive Loops
4.	15	12		MicroC/OS
5.	15	13		Serial Communications
6.	15	13		Stepper Motors
7.	15	13		Microcomputer Busses & DMA
8.	15	13		TCP/IP Networking
Total	**120 mins**	**100**		---

Question #1 (Fundamental Concepts)

In your own words briefly explain each of the following concepts. Be sure to explain why each concept is important in embedded microcomputer systems.

(a) Deadlock

(b) Semisynchronous interfaces

(c) Direct memory access

Question #2 (Multitasking Software)

(a) Partitioning the application software into tasks is usually believed to increase the modularity of the software system. Briefly argue why modularity can be effectively increased by using tasks. Be sure to begin by listing some of the key properties that are associated with high modularity, and then consider how multitasking enhances (or decreases) each of those properties.

(b) Briefly summarize some of the major disadvantages that can arise when partitioning the application software into multiple tasks.

Question #3 (Multiple Nonpreemptive Loops)

(a) When the software architecture of a real-time embedded system uses multiple nonpreemptive loops, it is recommended that the timing of those multiple loops be produced by a single hardware timer and that the periods of those loops be harmonically related. Why, given that a typical modern microcontroller has four or more independently programmable interrupt-producing timers, would it still be preferable to only use one of those timers to control the timing of all of the preemptive loops? Are there an disadvantages to using only one of the hardware timers?

(b) A desirable property of an embedded real-time system is that it be designed so that it fails gracefully when subjected to too much work load, when critical real-time constraints are on the verge of not being met. Similarly, in battery-powered systems, it may be desirable to slow down the CPU's clock (but not the hardware timers) to save power when the battery is going low. How could the multiple nonpreemptive loop architecture be modified to offer performance that can be allowed to degrade in a controlled manner so that very important polling operations are still completed on time, with the processing time devoted to less important operations being reduced using software means.

Question #4 (MicroC/OS)

(a) MicroC/OS has proven to be a popular real-time kernel for embedded systems for a number of reasons. A major reason for its success has been its portability, which is enhanced by the fact that it was implemented in well-structured and well-documented code written in ANSI C. What aspects of the software architecture of MicroC/OS, apart from the choice to use ANSI C, make MicroC/OS a portable kernel? What software changes would be required in order to provide an implementation of MicroC/OS on a new microcomputer?

(b) When a semaphore is created and initialized in a MicroC/OS system, an OS_SEM object is instantiated before all tasks, and then initialized either before all tasks or (preferably) by the UserMain task. Why is it necessary for OS_SEM to be instantiated outside all tasks? Why is it preferable for semaphores and message queues to be initialized inside UserMain?

Question #5 (Serial Communications)

(a) Briefly explain how the use of noise margins and differential signaling are two simple ways of enhancing a receiver's ability to receive a correct digital signal despite the addition of random noise during the communication of the signal from the transmitter.

(b) Briefly explain the purpose of the CRC-32 cyclic redundancy calculation in the Ethernet standard. How is this calculation used to improve the quality of the communication provided to the applications that are using an Ethernet connection?

Question #6 (Stepper Motors)

(a) When a microcontroller is used to control a stepper motor with an open loop control architecture, why is it important to produce step signals that start out at a slow step rate and that speed up only gradually through a series of faster and faster step rates until the fastest step rate (the slew rate) is reached? What would likely go wrong if there were too few different step rates?

(b) Briefly explain the difference between open loop and closed loop control for stepper motors. What are the major advantages of closed loop control over open loop control?

Question #7 (Microcomputer Busses & DMA)

(a) Multiplexed address/data busses are fairly common. For example, they are used in the Freescale FlexBus and the PCI expansion bus. Briefly describe the major advantages and disadvantages of such multiplexed busses. What are they considered reasonably choices for busses of moderate performance that might transfer bulk data in longer blocks.

(b) Briefly explain the difference between single-address and dual-address direct memory access (DMA).   Why are data transfers in single-address mode usually driven by DMA request signals originating in a peripheral, whereas in dual-address DMA the transfer is usually initiated when the CPU writes to a DMA control register.

Question #8 (TCP/IP Networking)

(a) TCP uses a window-based flow control method for ensuring that TCP datagrams are eventually received correctly at the far end of a TCP connection. What is the "window" in this method? What factors should be considered when deciding how big to make the window size?

(b) Whenever the TCP entity in a networked node attempts to transmit a TCP segment, it keeps a copy of the segment and then associates with it a timer with a specified time-out. What is the purpose of this timer and how might one go about choosing a good time-out value? What should happen to the copy of the segment and the timer object when all of the bytes in the segment have been acknowledged by the receiver?