UNIVERSITY OF ALBERTA

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ECE 315 – Computer Interfacing

Midterm Examination

Instructor:	B. F. Cockburn
Exam date:	March 7, 2016
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. A set of study notes of reasonable size (10 pages max.) 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		Microcomputer Hardware
3.	18	35		Multitasking Software
4.	12	25		Communication Interfaces
Total	**50 mins**	**100**		—

Question #1 (Basic Concepts)

Carefully explain the differences among the following pairs of concepts:

(a) Hardware interrupt priority level vs. task priority in MicroC/OS

[5 marks] The hardware interrupt priority level (IPL) is recorded in the 3 IPL bits of the CPU’s status register (SR). In M68000 and ColdFire systems, interrupt signals must be assigned (by the hardware) to one of 7 possible levels of increasing priority, from 1 up to 7. IPL=0b000 means that no interrupt is active. If IPL < 0b111, then the CPU will find and begin to execute the interrupt service routine of any newly active interrupt that has a higher priority level than the level encoded in the IPL bits. When this happens, the IPL bits are automatically increased to the level of the new higher priority interrupt. If IPL == 0b111, then the CPU will respond to a further level 7 interrupt, but the IPL won’t be changed.

In MicroC/OS there are 63 task priorities ranging 1 (the highest priority) to 63 (the lowest priority). MicroC/OS enforces the constraint that the highest priority ready-to-run (i.e., unblocked) software task is always running on the CPU. This condition is violated only if multitasking is locked off. At most one task can be assigned to each priority level.

(b) Binary semaphore vs. counting semaphore

[5 marks] Both kinds of semaphore comprise an integer count and a queue that records all of the tasks that are currently blocked on the semaphore. In binary semaphores the count value is initialized to 0 and 1; in counting semaphores the count can be any signed value that fits in the count-sized field. A binary semaphore is used to block tasks, often at the beginning of a critical section. The binary semaphore count is initialized to 0 (or 1) if the first task to pend to the semaphore is to be blocked and added to the queue (or not blocked and allowed to proceed, with the count reduced to 0). A counting semaphore is initialized to the number of available resources in a pool. Pend operations to a counting semaphore cause the count to be decremented. Post operations to either kind of semaphore cause the count value to be incremented. If the count increases from 0 to 1, then the highest priority blocked task is unblocked.

(c) Rate monotonic fixed priority vs. deadline monotonic fixed priority

[5 marks] A rate monotonic fixed priority policy assigns higher priorities to tasks that must execute more frequently. A deadline monotonic fixed priority policy assigns higher priorities to tasks that have a sooner hard deadline to meet.

Question #2 (Microcomputer Hardware)

(a) Briefly explain the similarities and differences between a central processing unit (CPU), a microcontroller unit (MCU), and a digital signal processor (DSP).

[6 marks] A central processing unit (CPU) is the circuit that executes the instructions in a stored program. An MCU contains one or more CPUs. A DSP is a special kind of CPU.

[6 marks] A microcontroller unit (MCU) is a silicon chip that contains one or more CPUs and DSPs, as well as other support circuits, such as peripheral interfaces, times, direct memory access controllers, etc. that are used to create the core of an embedded system. An MCU is larger than either a CPU or DSP.

[6 marks] A digital signal processor (DSP) is a special kind of CPU that has features that make it especially efficient at doing the numerical operations that are common in signal processing, such as filters (convolution calculations). The special features might include extra registers, parallel datapaths, instructions that specify parallel numerical calculations, etc. One or more DSPs may be present in an MCU.

(b) Microprocessors are usually designed and marketed as families of parts that have the same user mode programming model, with the same available set of data registers, address registers, program counter, status register, etc. In the case of the 68000-series of CPUs, which includes the ColdFire v.2, the user mode programming model has been mostly unchanged for over 35 years. What is the reason for keeping the user mode programming model the same across all of the members of a microprocessor family even if the underlying microprocessor hardware implementations have vastly different numbers of transistors organized with very different architectures.

[7 marks] Keeping the user mode programming model the same across all of the members of a microprocessor family ensures that user software will be upwardly compatible to the new members of the same microprocessor family. This increases software portability, and keeps customers happier and loyal since they can preserve their existing software investment going forward. (This is less of a concern when most software is compiled from a high-level language.)

Also, keeping the user mode programming model the same (and relatively simple) allows the underlying implementation to be modified and improved without requiring changes to the customer’s user mode software.

Question #3 (Multitasking Software)

(a) Partitioning embedded system software into states is convenient for many communications interfaces since the behaviour of those interfaces is defined with respect to finite state machines. But there are other good reasons for using state-driven software in embedded controllers. How could one exploit the state-driven architecture to digitize input signals (at possibly different sampling rates) from many different analog sensors located throughout a factory?

[12 marks] Sensors that are to be sampled at the same frequency and at the same times can be controlled (the signals sampled and digitized) by one task that is partitioned into states that are scheduled to run at the required frequency. Perhaps only one combined sample and digitize state is required, or perhaps separate sample and digitize states are required. (The digitization hardware may require the controller to give separate sample and digitize commands at different times by different software states.) Different state-driven tasks could be used for different groups of signals that are sampled and digitized at different frequencies. The state-driven kernel architecture allows the control of the timing to be accomplished and enforced elegantly by the kernel, keeping the state-driven software simple.

(b) The MicroC/OS pre-emptive multitasking operating system uses task priorities to make it easier for the system designer to ensure fast response times and deterministic behaviour despite variations in the system workload. Briefly explain how having at most one task per priority level, and kernel-enforced pre-emption among tasks, makes it easier to ensure fast response and determinism.

[12 marks] Because of kernel-enforced task pre-emption, any lower priority tasks will simply not get any CPU time as long as a task at a given priority level is executing. This makes it easier to ensure fast response times and deterministic behaviour for the given task. However, it is still possible for interrupts to occur and be serviced, and an interrupt service routine could unblock a task at a higher priority level which will pre-empt the given task. Only the task at the highest priority level is truly immune from the activities of lower priority tasks.

The limitations of the priority scheme and kernel-enforced pre-emption mean that it is especially important to carefully select which tasks are assigned to which priorities. Also, it always helps timing accuracy to have a lightly loaded system.

Question #3 (Multitasking Software, cont'd)

(c) MicroC/OS uses a layered software architecture to enhances the portability of the operating system across different microcomputer platforms. In particular, there is a processor-specific “MicroC/OS Port” layer that takes care of most differences between platforms. What CPU-related and timer-related operating system functions would need to be handled by low-level software in the port layer? Why would it likely be desirable to use assembly language in some portions of the port?

[4 marks] MicroC/OS has a routine that performs context switching, where the context of one task is saved in a task control block (TCB) and the CPU is loaded up with the saved context of a second TCB. The context switching routine needs to be able to rapidly access CPU registers, such as the data registers, the address registers, the status register, and the program counter. This is often most easily done using an assembly language program. But even if the context switch routine is expressed in a high-level language, the routine is going to be CPU-specific and thus must be in the port layer.

[3 marks] Details of the interrupt service routines are also likely to be CPU-specific. For example, the interrupt priority masking mechanism is not the same for all CPUs. These differences could be hidden inside an INTERRUPT macro, and this macro would thus have to be in the port layer.

[4 marks] A hardware timer is required in MicroC/OS to support the blocking time delay function. (A time-out timer is also required by the TCP/IP stack.) With respect to the timer, there will be system-level differences in how the timer is controlled through memory-mapped registers. The bit definitions and timer capabilities will differ across systems, and so the timer driver software should also be in the port layer.

Question #4 (Communication Interfaces)

(a) In microcomputers there are several standard methods for transferring data among different locations in the CPU's memory map, including RAM locations and interface registers. Most communication interfaces provide mechanisms where the interface can interrupt the CPU when relevant events occur. In the transmit direction, briefly describe what condition(s) would normally be used to advance the process of transmitting more and more data out the communication interface. What actions would need to be performed in the transmit interrupt service routine?

[5 marks] In the transmit direction, the transmitter hardware must monitor when data has been transmitted so that the CPU can be alerted (using an interrupt) so that the CPU can load more data into the transmitter's FIFO or data register. Similarly, if the transmitter hardware has been disabled (or re-enabled) for some reason by an external (e.g. handshake) signal, the transmitter should alert the CPU using an interrupt.

[3 marks] To start a new transmission, the CPU must enable transmit interrupts in the transmitter hardware. This would often be done by the message send system call or trap routine.

[7 marks] In the transmit interrupt service routine, the CPU must transfer data from a buffer (containing the data to be transmitted) into the transmitter's data register or FIFO. If this action does not automatically remove the interrupt condition, that condition must be cleared by the CPU to turn off the interrupt request signal. If the CPU determines that no more data needs to be transmitted, it must reconfigure the transmitter so that it no longer produces interrupts.

(b) In the Universal Asynchronous Receiver and Transmitter (UART) design that we considered in class, there were two data registers in a sequence in the transmit direction, and four data registers (organized as a first-in first-out buffer) in a sequence in the receive direction. What would be the advantage to having more registers in the receive direction than in the transmit direction?

[4 marks] The CPU has less control over the flow of data in the receive direction compared to the transmit direction. The data in the receive direction is being sent by another CPU.

[6 marks] To reduce the risk of being overwhelmed with a burst of data in the receive direction, it is wise to have additional data registers in the receive path so that extra data can be stored temporarily and thus avoid data loss. It is still useful to have extra data registers in the transmit direction so that the transmit software can load one (or possibly more) bytes of data into the hardware, and then go onto other activities.