UNIVERSITY OF ALBERTA

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ECE 315 – Computer Interfacing

Final Examination

Instructor: B. F. Cockburn
Exam date: December 11, 2013
Exam duration: 120 minutes
Aids permitted: A copy (paper or electronic) of the lecture slides can be freely consulted.
Model solutions for the Fall 2013 assignments and midterm can be consulted.
No other model solutions are to be consulted.
No Internet access is permitted using any kind of device.
Electronic calculators (all kinds) are permitted.

Instructions: 1. Enter your printed name, signature and I.D. number on this cover page.
2. Verify that this booklet contains 10 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: **_____Model_____,___Solutions_____**
Last name **First name**

Signature: **_____**

Student I.D.: **_____**

Question	Time	Worth	Mark	Subject
1.	15	12		Fundamental Concepts
2.	15	12		Multitasking Systems
3.	23	20		Real-time Event Handling
4.	15	12		MicroC/OS
5.	15	12		The MOD5234 Ethernet Interface
6.	22	20		Microcomputer Busses
7.	15	12		Computer Networking
Total	**120 mins**	**100**		---

Question #1 (Fundamental Concepts)

In your own words briefly define each of the following concepts. Be sure to explain why each concept is important in Computer Interfacing.

(a) ARPANET

[4 marks] The ARPANET was the computer network that grew and developed into today's Internet. It started as an experimental military-sponsored packet-based data network involving four universities in California and Utah. Over time it grew to include thousands of universities, military organizations, government bodies, and eventually companies. The original NCP protocol for the ARPANET was improved until it took the form of the current TCP/IP protocol stack. In short, the ARPANET was an important early step in the Internet Revolution which is of central importantance to modern Computer Engineering.

(b) Supervisor mode

[4 marks] "Supervisor mode" is a less restricted mode of operation for a central processing unit (CPU). The restricted "User mode" has greater portability across CPUs, and is intended for most application software. In the ColdFire series CPUs, supervisor mode occurs when the S bit in the status register has value "1"; "user mode" occurs when the S bit has value "0". Software executing in supervisor mode can access additional registers and can use additional instructions compared to user mode software. The operating system and interrupt/exception handling routines execute in supervisor mode since they are trusted software that must have unrestricted ability to access system features. User mode programs are used for other kinds of software that do not need to have full capabilities or that are not as highly trusted or as stable as system software.

(c) Rate-monotonic fixed priorities

[4 marks] Rate-monotonic fixed priorities are used in priority-based multitasking systems to ensure that tasks that must execute more frequently are given higher priorities. This way they are more likely to meet their more challenging real-time deadlines. Less frequently executing tasks can usually perform successfully using lower priorities. Rate-monotonic priorities are only one system-level strategy for ensuring that a real-time system will meet all of its real-time deadlines.

Question #2 (Multitasking Systems)

Partitioning a real-time software system has both possible advantages and disadvantages. Briefly comment on the suitability of a multitasking system architecture with respect to the following design challenges: (a) the ability to allocate the design work to multiple teams of engineers; (b) the ease of ensuring that the system will respond with a predictable delay time when handling external events; (c) the ability to debug the causes of response-time errors; and (d) the ability of designers to enhance the system with new external interfaces.

(a) [3 marks] A multitasking architecture should be an advantage, and thus a suitable choice, when partitioning a system across multiple teams, providing the high-level partitioning is done properly. The task boundaries could correspond naturally to team boundaries, thus minimizing the need for team-to-team interactions. Although there are complications with managing inter-task interactions, these disadvantages can be managed using standard techniques.

(b) [3 marks] A pre-emptive multitasking architecture should be an advantage, and thus a suitable choice, when ensuring predictable delay when handling external events. Each event can be handled in a straightforward way using one or a small number of tasks and hardware interrupts. Task priorities can be selected to ensure fast enough response. There are complications with managing inter-task interactions, but these disadvantages can be managed using standard techniques.

(c) [3 marks] A multitasking architecture can be both a disadvantage and an advantage when debugging the causes of response time problems. It can be difficult sometimes to recreate complex task interactions that led to response time problems. However, simplifying the task interactions should help make a system more diagnosable. Each external event should be handled in a straightforward way using one or a small number of tasks. Task priorities can be selected to ensure fast enough response. There are complications with managing inter-task interactions, but these disadvantages can be managed using standard techniques.

(d) [3 marks] A multitasking architecture should make it easier to add new interfaces if they can be handled by newly added tasks that are relatively independent of the existing tasks. Minimal changes should then be required to the pre-existing tasks. The new tasks will need to be assigned suitable priorities to ensure that all real-time constraints continue to be met. There are complications with managing inter-task interactions, but these disadvantages can be managed using standard techniques such as suitably chosen priorities.

Question #3 (Real-time Event Handling)

(a) Externally generated events are handled by computers using a variety of different mechanisms that have different advantages and disadvantages. For example, the NetBurner MOD5234 microcomputer that we used in the laboratory exercises can respond to externally produced signals using: (1) eTPU channel inputs; (2) the microcontroller unit's (MCU's) external interrupt pins; or (3) general-purpose inputs to the MCU that are polled for changes in state by software running on the CPU. Why would a designer ever prefer to use the polling technique for monitoring a system input signal instead of connecting the signal to an eTPU channel or an MCU external interrupt? When would using an eTPU channel be preferred over an external interrupt or a scanned general-purpose input for handling an input signal?

[5 marks]
The polling technique would be preferred if some signal (e.g. a status bit) needs to be checked on a regular basis. Polling is also likely to be satisfactory when checking for user inputs or other slowly changing inputs since computers act so much faster compared the reaction time of humans. Polling is a simple software-based method that would not deplete the limited number of external interrupt pins or eTPU channels.

[5 marks]
An eTPU channel would be preferred for the most challenging hard real-time input signals. The eTPU offers the fastest and most deterministic response of all of the input handling approaches available on the MCU. Ideally, one of the existing eTPU channel functions can be used to handle the input signal. If the appropriate signal is detected by an eTPU channel, then the CPU can be rapidly informed using the existing eTPU interrupt mechanism.

Question #3 (Real-time Event Handling, cont'd)

(b) A cooperative multitasking software architecture would appear to be very efficient in theory because the number of context switches between tasks can be minimized. (Recall that time spent doing context switches is essentially wasted time.) However, a major weakness of this architecture is the difficulty of ensuring that (1) external events will be handled rapidly and deterministically, and (2) that generated output signals will have accurately controlled timing. Briefly explain how the state-driven code architecture together with interrupts can be used to provide the efficiency of cooperative multitasking together with much better real-time event handling.

[5 marks, high efficiency]
In the state-driven code architecture, context switches occur only when the running code in each state decides to give up the CPU. The software in each state can run efficiently to a desired completion point without being pre-empted by another task. The number of task context switches is thus directly controlled and can be minimized. This comes close to matching the efficiency achieved with cooperative multitasking. But the state-driven architecture also means that applications need to be partitioned into suitable fast-executing states so that the CPU is shared appropriately. However, if this can be done, then the outputs produced by states will have excellent timing since state transitions can be deterministically aligned with desired time intervals.

[5 marks, faster event handling using interrupts]
Real-time response using state-transition-controlled polling may ensure adequate real-time response for slowly changing inputs. But for faster inputs, interrupts offer a better solution. An interrupt will cause an interrupt service routine to start running regardless of the state software (assuming that the interrupt is not masked by the execution by an even higher priority interrupt service routine).

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?

[3 marks]
The portability of MicroC/OS has been enhanced by (1) making minimal assumptions about the hardware platform, and (2) minimizing and containing the processor-specific software to a "port" layer. In (1), only a compiler-supported CPU and a hardware timer are assumed. In (2), the port will contain CPU-specific routines, like the task context switching routine, and routines that control the hardware timer to produce the "tick clock" that is used in some MicroC/OS functions. (If a network interface is provided, then specialized code will be required to operate the networking hardware. However, the TCP/IP stack and communications driver are not really a part of MicroC/OS.)

[3 marks]
The "port" layer of software would need to be modified in order to provide an implementation of MicroC/OS on a new microcomputer. A context switching routine would need to be provide that rapidly saves the CPU register contents in one Task Control Block (TCB), and then loads the CPU registers with the saved contents of the registers in the TCB of the task that is about to be restarted. The port would also need to implement the required framework for interrupt service routines. Finally, the port would need to provide the driver software for hardware timer so that the "tick clock" feature is provided for the MicroC/OS routines that require it.

(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?

[6 marks]
It is desirable to place as many initialization actions as possible inside the UserMain() task. Space for shared objects, like semaphores and message queues, can be saved by creating named instantiations of those objects before any task, including UserMain(). Initialization of those objects can be done safely before all tasks, including UserMain(). However, it might be more appropriate to initialize those objects inside UserMain() before UserMain() has created the application tasks. In this way, all initialization actions would be gathered together in one place inside UserMain().

Question #5 (The MOD5234 Ethernet Interface**)**

(a) When the PHY in the MOD5234 detects the arrival of an Ethernet frame, it first asserts high a carrier sense (CRS) output signal to the FEC block. Then just before the first received data bits are sent up to the FEC, the receive data value (RXDV) signal is also asserted high. Four parallel streams of received data bits are sent up to the FEC along with a recovered receive clock (RXC). Briefly explain what the FEC block in the MOD5234 does with the received data bits when it receives them from the PHY.

The FEC block receives the four parallel streams of data bits from the PHY, and writes them into the Rx partition of a FIFO within the FEC block. The data bits in the Rx FIFO are then transferred using Direct Memory Access to one or more equal-sized Receiver Buffers in the CPU's memory map. These buffers are pointed to by Rx Buffer Descriptor records in the Rx Buffer Descriptor Ring (a circular data structure).

Each time a new Rx Buffer is filled up (or after the last buffer has been written with the last received data), the E bit in the buffer descriptor is cleared to 0 and a status bit is written to 1 in the Ethernet Interrupt Status Register. The Receive Buffer Interrupt (RXB) bit is written to 1 after each buffer (except the last one) is filled; the Receive Frame Interrupt (RXF) bit is set to 1 when the last buffer has been fully written. If the corresponding mask bits are set to 1, an interrupt signal will be sent back to the CPU. When the last buffer is filled, the total frame length field in its buffer descriptor is written with the correct value.

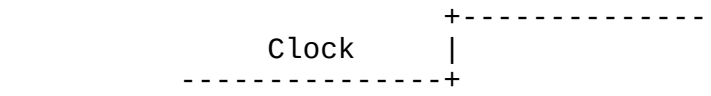
(b) How is the CPU alerted to the arrival of a fully received new Ethernet frame? Where does the CPU retrieve the payload of the received Ethernet frame?

The Receive Frame Interrupt (RXF) bit in the Ethernet Interrupt Status register is set to 1 when the last buffer has been fully written by the FEC DMA controller. If the corresponding mask bit is set to 1, an interrupt signal will be sent back to the CPU. When the last buffer is filled, the total frame length field in its buffer descriptor is written with the correct value.

The CPU retrieves the payload of the received Ethernet frame in the Receive Buffer Registers pointed to by the buffer descriptors in the Receive Buffer Descriptor Ring.

Question #6 (Microcomputer Busses)

(a) What are the set-up and hold times with respect to a clock for the digital signals on a databus? Use a timing diagram to illustrate your answer. What are the underlying physical causes of the set-up and hold times?



The set up time is the minimum time *before* the active clock edge (say the rising edge) when a data signal to be latched by the clock must be valid and stable.

The hold time is the minimum time *after8 the active clock edge when a data signal to be latched by the clock must be valid and stable.

If either the set up or hold time constraints is violated, the state of the latched value may be incorrect. This is because memory elements require time to change their state, the time to move charge around within the storage element and over the wiring, which has finite capacitance. Memory elements often have feedback loops that need to time to stabilize in the new stored state.

(b) Parallel communications interfaces have to accommodate possible timing skew among the data signals on the data bus. Briefly explain what is meant by timing skew in this context. How is timing skew managed using the new USB 3.0 and PCIe standards?

Skew in the context of a data bus is the maximum variation that can be expected in the arrival of the different data signals at the parallel data receiver.

The USB 3.0 standard includes one differential data path from the earlier USB standards. It adds two more parallel differential pairs to help gain the additional data bandwidth. The PCIe standard also uses multiple parallel “lanes”, where each lane is a high-speed serial link with its own timing recovery circuit. The bits that are received over these parallel links are detected separately (thus avoiding the timing skew problem), and the bits then multiplexed together in the digital domain.

Timing skew is managed by separately recovering the clock and data for the multiple serial links. Each link has a separated timing recovery circuit. The recovered data is then multiplexed together in digital form.

Question #6 (Microcomputer Busses, cont'd)

(c) The General-Purpose Interface Bus (GPIB) exploits the electrical properties of bus drivers with tri-state outputs as well as drivers with open-collector outputs. What is the major advantage of drivers with tri-state outputs? Which GPIB bus signals benefit from being driven by tri-state drivers? What key property of open-collector drivers is exploited in the Not Ready for Data (NRFD*) and Not Data Accepted (NDAC*) signals in the GPIB?

Drivers with tri-state outputs have the ability to drive the output to "1", to "0" or to high impedance (no active drive).

The 8 data bus signals need to be driven by tri-state drivers to allow different active "talkers" to drive data onto the data bus at different times.

Drivers with open-collector outputs are only able to drive the output to "0", or to leave the output in high impedance. An external resister is required to ensure that the output signal is pulled up to "1" if is not being actively down to "0". This property is exploited because the outputs of multiple open-collector drivers can be connected to form a logic gate operation. If any one driver drives to "0", the output is "0"; otherwise, the output is pulled to "1", This is logic AND operation. This logic operation is used in the "Not Ready For Data (NRFD)" and "Not Data Accepted (NDAC)" signals on the GPIB. If any "listener" is "Not ready" it can pull the common wire signal (NRFD or NDAC) low. All listeners have to be "ready" for the common signal to rise to "1".

Question #7 (Computer Networking)

(a) Port numbers are used in the header fields of the UDP and TCP transport layer protocols. Briefly explain the purpose of the port numbers. Why must the common port numbers be standardized in networked server nodes?

[2 marks] Port numbers are used to associate TCP/IP connections with software applications on the server and client. Port numbers are also useful for security enforcements, say in firewalls.

[4 marks] When a server host receives a request to set up a TCP/IP connection, the port number is used to rapidly identify the server process that will handle the application byte-stream on the new connection. The port numbers for standard applications must be standardized on servers since the client side needs to know which port number to use on the server. The port number on the client side is not important since it will be discovered by the server after the connection is made.

(b) Briefly explain how TCP/IP recovers from the situation where two transmitted IP datagrams are received in the reversed order at the destination node.

[3 marks] When two IP datagrams are received in the reversed order, TCP/IP will **detect** the problem using the sequence numbers. TCP/IP unloads the payload bytes from arriving IP datagrams and loads these into a circular buffer. Incrementing sequence numbers are associated with each byte. An IP datagram that arrives unexpectedly early (as a result of the reversed order) will create a gap of empty locations in the circular buffer. This gap will be detected because the sequence number in the datagram will lie beyond the last filled position in the buffer.

[3 marks] TCP/IP will **recover** from the situation by either (1) discarding the too-early IP datagram, or (2) leaving the data bytes in the correct position in the circular buffer in the hope that the missing bytes will eventually arrive in the missing IP datagram(s).