UNIVERSITY OF ALBERTA

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ECE 315 – Computer Interfacing

Final Examination

Instructor: B. F. Cockburn
Exam date: April 14, 2016, 2:00 pm
Exam duration: 120 minutes
Aids permitted: A copy (paper or electronic) of the lecture slides can be freely consulted.
Model solutions for the Winter 2016 assignments & midterm can be consulted.
No other model solutions are to be 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 11 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		Watchdog Timer
3.	13	11		Context Switching in MicroC/OS
4.	12	11		Work Load Monitor
5.	15	12		The TCP/IP Protocol Stack
6.	15	12		The Enhanced Time Processing Unit
7.	20	18		Microcomputer Busses & DMA
8.	15	12		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) The dereferencing operator in C

[2 marks] The dereferencing operator in C is the prefix operator *. If var is a pointer variable that contains a reference (essentially an address) to an object (e.g., a variable, an array, a struct, a function, etc.), then *var identifies the object itself. *var can be used on the left side of an assignment statement as the destination, or it can be used to provide an object value into an expression.

[2 marks] The dereferencing operator * is important because pointers are widely used, along with *, in embedded systems to refer to data structures and to access hardware registers. Pointers, used along with *, are an efficient way of accessing objects in the computer's memory map, but errors affecting pointers can easily lead to serious system failure.

(b) Determinism

[2 marks] Determinism refers to the amount of predictability there is in the response time of an embedded system, which is the time between when an input changes value and the time when the system responds by changing the value of appropriate outputs. If a system has greater determinism, then there will be less variance in the response time.

[2 marks] Determinism is a desirable property in embedded systems. Greater determinism makes it easier to ensure that an embedded system design will satisfy the required real-time response time requirements.

(c) The MAC-PHY interface

[2 marks] In a communications interface, the Media Access layer (the MAC) interacts with the Physical layer interface (the PHY) through the MAC-PHY interface. The MAC-PHY interface has been standardized so that MACs and PHYs made by different companies can be used together.

[2 marks] The presence of a standardized MAC-PHY interface means that embedded systems can take advantage of inexpensive PHYs that can be mass produced by specialized companies, thus reducing the cost of the embedded system.

Question #2 (Watchdog Timer)

(a) In both preemptive multitasking and cooperative multitasking, it is possible for system performance to be disrupted if one task starts to consume too great a proportion of the CPU's time. A watchdog timer is a hardware timer that produces interrupts, where executing the interrupt service routine causes a system reboot. Briefly explain how a watchdog timer can protect against the situation where one task consumes too much of the CPU's time. Be sure to explain how the watchdog timer must be controlled in normal operation. (You do not need to reference specific registers or bits in the MCF5234.)

[6 marks] A watchdog timer ensures that the system will be rebooted if the free-running timer counter is ever allowed to down-count to zero. In normal operation, the counter should never reach zero because the system software includes a task (possibly the idle task) that sufficiently frequently services the watchdog timer and causes it to reload its counter with the starting value. A watchdog timer is normally used to ensure that if the system software gets stuck or hangs, it is likely that the failure will allow the watchdog timer to reset the system. Hopefully the system reset will restore the system's functionality. If the concern is that a particular task will get stuck and will cause system failure, the watchdog timer time-out period can be set to be sufficiently short that a slow to react task will cause a system reset. The service routine for the watchdog timer might be sophisticated enough to only kill the task that was running when the timer expired, but this is likely to be an unrealistic strategy since killing one task may let the system carry on in an unstable or incomplete mode.

(b) Briefly explain whether or not round-robin time slicing avoids the problem of one task attempting to take up too much of the CPU's time. Briefly list some of the major weaknesses of time slicing in a hard real time embedded controller.

[3 marks] Round-robin time slicing can indeed avoid the problem of one task taking up too much of a CPU's time. The context switching times that are guaranteed by time slicing are controlled by interrupts coming from an independent hardware timer. These interrupts will occur regardless of how the system software behaves. The context switch routine can swap out the time-hogging task and allow other tasks to execute.

[3 marks] A major weakness of time slicing in a hard real time embedded controller is that the time slicing steps will add uncertainty to the input-to-output response time of the controller. Response times will depend, in a complex way, with the relationship between the input changes and the time slicing times. In short, the controller will become less deterministic. It is thus harder to meet any hard real-time constraints.

Question #3 (Context Switching in MicroC/OS)

Recall that in a pre-emptive multitasking software environment like MicroC/OS, the highest priority ready-to-run task must execute on the CPU. A consequence of this rule is that when a task calls MicroC/OS functions, it is possible that a context switch will occur to a new task. Consider each of the following functions, and for each, describe the conditions that would cause a context switch if the function were called.

- OSTimeDelay()

  [2 marks] If the running task calls OSTimeDelay() with a delay greater than zero, then the kernel will immediately block that task on the tick counter for the requested positive number of ticks. The kernel then causes a context switch to the highest priority ready-to-run task. If the requested delay was zero, then the calling task should carry on unaffected.

- OSQPost()

  [3 marks] When the running task calls OSQPost(), it will cause a pointer to be added into the specified message queue. This action could unblock a higher priority task that was previously waiting on the empty queue. In this scenario, the kernel causes a context switch from the task that called OSQPost() to the now unblocked higher priority task. However, if the queue is full, the error code OS_Q_FULL is returned and the task is not blocked and a context switch does not occur.

- OSSemPend()

  [3 marks] When the running task calls OSSemPend(), the semaphore count is decremented by 1. If the new count is negative, the task will be blocked and the kernel will do a context switch to the highest ready-to-run task. If, however, the count is zero or greater, the calling task will carry on executing to the statement after the call to OSSemPend().

- USER_EXIT_CRITICAL()

  [3 marks] The calling task will be switched off the CPU if the call to USER_EXIT_CRITICAL() allows a pending interrupt to be serviced, which by its execution unblocks a higher priority, and now ready-to-run task. However, if no interrupts were pending or if any enabled interrupts to not unblock a higher priority task, the calling task will carry on executing.

Question #4 (Work Load Monitor)

MicroC/OS provides an idle task that runs at the lowest possible priority, level 63. It is often useful when evaluating the performance of an embedded system to have a task that determines and displays the system work load, in other words, the percentage of the CPU's time that is not spent executing the idle task. Propose a design that uses the idle task and a hardware timer to create a work load monitor. Be sure to specify what features would be required in the timer. Also specify, in general terms, how the work load would be calculated and displayed via a serial port 10 times per second.

[11 marks]
The idle task software will be assumed to be an endless loop containing a sequence of one or more operations. Each operation should have fixed number of operations, so that their execution time does not vary. One of those operations is to increment a 64-bit (long) integer counter variable. As part of calibration step, the idle task calls USER_ENTER_CRITICAL to turn off all interrupts so that the increase in the hardware counter value over a reference wall clock interval, say 10 seconds, can be measured. The expected number of counter increments per second can then be calculated by dividing the counter value increase by the measurement interval. This calibration is only done once, and then the counter increments per second value can be hard-coded in the idle task software.

The system workload can be calculated by taking 100% and then subtracting the percentage time that is spent in the idle task. The idle task percentage is simply 100 times the increase of the count in 10 ms divided by the expected increase in the count for 10 ms in a system that just has the idle task running, as determined in the calibration step.

The hardware timer is configured to produce hardware interrupts every 100 ms. The timer interrupt service routine calculates a new system workload for the past 10 ms, and then posts this value to a message queue. An output task pends on this message queue. When a new workload message appears, the output task enables interrupts for the serial output interface. The serial output interrupt service routine transfers the next character from the latest message into the serial data output register. When the last character in a message has been output, the interrupt service routine for the serial output disables further serial output interrupts.

Question #5 (The TCP/IP Protocol Stack)

(a) The functionality provided by the protocol combination TCP/IP is partitioned into the separate TCP and IP protocols. Briefly justify why, given modern communications technology, it is appropriate to delegate to the end-to-end protocol TCP (as opposed to IP) the problems of data flow control and of recovering from lost or severely delayed datagrams.

[6 marks]
Modern communications technology provides inherently reliable transport from end-to-end. It is relatively rare for data to be lost in transit.

Flow control can be done using end-to-end TCP if it can be safely assumed that buffering along the route of the path is sufficient that data will not be lost as a result of node-to-node communication. In that case, control of flow control can be safely delegated to the end-to-end TCP protocol.

Recovery from lost or severely delayed datagrams should be a relatively rare occurrence, and so it too can be handled by the end-to-end protocol. The window-based flow control mechanism used by TCP should be efficient since most of the time, datagrams will not be lost or greatly delayed.

(b) Now briefly justify why it is appropriate to delegate to the node-to-node protocol IP (as opposed to TCP) the problems of (1) adapting the size of datagrams to fit the underlying physical networks, (2) detecting and removing old datagrams that are lost in the network, and (3) assigning sequence numbers to the datagrams.

[6 marks]
(1) Adapting the size of datagrams to fit the underlying physical network is a local, node-level problem. Different parts of the network will likely have different packet size constraints.

(2) Removing old datagrams is most easily done by the first node that receives the datagram. The age of a datagram is easily tracked on a node-to-node basis. Using an end-to-end protocol would be awkward.

(3) Assigning sequence numbers could be done on a datagram basis, but that would prevent datagram from being easily split into smaller datagrams while in transit. But the problem is easily solved by assigning the sequence numbers to the data bytes. They the datagrams can be just as easily labelled by the sequence number of the first data byte that they happen to carry.

Question #6 (The Enhanced Time Processing Unit)

(a) Briefly describe how the scheduler in the eTPU partitions the available execution time on the microengine among threads of three different active priorities.

[4 marks] The scheduler recognizes threads of three different active priorities: high, middle, and low. To respect the priorities while also enforcing fairness, the scheduler follows a seven slot rotation, where there are four high priorities slots, two middle priority slots, and one low priority slot. The order is: high, middle, high, low, high, middle, and then high. Within each priority level, the active threads are serviced in round robin order. One a thread starts executing in a slot, it can run as long as necessary, regardless of its priority level.

(b) Briefly explain whether or not the eTPU scheduling scheme guarantees that every active thread (that is, every thread that is either being serviced or is waiting to be serviced) gets a certain percentage of the available time on the scheduler. Is the eTPU scheduling scheme more similar to cooperative multitasking or to round-robin time slicing? Briefly justify your answer.

[2 marks] The scheduling scheme only guarantees a chance for every thread to start executing. The scheme does not guarantee a percentage of execution time on the scheduler to the different tasks.

[6 marks] The eTPU scheduling scheme has elements of both cooperative multitasking and time slicing. Similar to cooperative multitasking, once each thread starts to run it can run for as long as it needs. The threads must not abuse the privilege. Like in pure cooperative multitasking, the threads must cooperate to ensure that the sequencer's time is shared effectively. The scheme is more complicated than conventional cooperative multitasking because of the different priorities. But those priorities just affect the order and (depending on the priority) the number of time slots given to each thread. Similar to time slicing, there is a guaranteed rotation among the different priority levels, and also a guaranteed rotation among the threads within each priority level. But there is no independent source of timing that determines when each time slot/slice begins.

Question #7 (Microcomputer Busses & DMA)

(a) Briefly describe how the TS, TIP and TA signals are used to indicate and to control the duration of a read or write bus transaction on the MCF5234 external bus. What makes this microcomputer bus both bilateral and semisynchronous? What are the major advantages of such a bus over a synchronous bus or an asynchronous bus? Illustrate your answer with a timing diagram.

[1 mark] The transfer start (TS) signal is asserted low for the first full clock cycle of a read or write bus transaction.

[1 marks] The transfer in progress (TIP) signal is asserted low for the entire duration of the bus transaction. If a read transaction is followed immediately by a write transaction as one atomic transfer, then the TIP signal stays asserted low until the end of the last clock cycle of the last write bus transaction. TIP indicates when the CPU (or another bus master) is busy using the bus.

[1 mark] The transfer acknowledge (TS) signal is asserted low for the clock cycle that contains the moment of the actual data transfer, in the read or write direction. The data transfer occurs on the rising edge of the clock at the very *end* of the clock cycle flagged by TS.

[2 marks] The MCF5234 bus is bilateral because both the master and slave determine how each bus transfer proceeds, and how long it takes to complete. The bus is also semisynchronous because it uses the rising and falling edges of the clock as timing points, and bus transfers take a variable number of whole clock cycles to complete.

[1 mark] The semisynchronous bus has more timing flexibility than a synchronous bus because bus transfers do not have to take exactly the same number of clock cycles. The semisynchronous bus is simpler to design and optimize than an asynchronous bus because of the presence of the clock signal.

Question #7 (Microcomputer Busses & DMA, cont'd)

(b) The MCF5234 has four general-purpose direct memory access (DMA) controllers. These four controllers support transfers using both the single-address and dual-address modes. Briefly explain what these two modes are. Give a simple example of where one might use each of these two modes in a microcomputer.

[3 marks] The single-address DMA mode involves either (1) transferring data from an input port (e.g., from a hard disk) to a buffer region in memory, or (2) transferring data from a buffer region in memory to an output port (e.g., to an output port that goes to a USB drive). The single-address in both cases is the address of the current location in the buffer.

[3 marks] The dual-address mode involves moving data from one buffer region in memory (using one address) to another buffer region in memory (using a second address). As an example, the data could be moved from a buffer in regular CPU memory over to a memory-mapped graphics frame buffer.

(c) A DMA transfer can be initiated in three possible ways: (1) a DMA request signal from either the eTPU or off-chip hardware is asserted, (2) a DMA request from an on-chip subsystem is asserted, or (3) software writes a 1 to the START bit in the DMA Channel Control Register. Briefly describe how the DMA Request Control Register is used to select between DMA request sources (1) and (2) for a particular DMA control register. What prevents DMA requests from the hardware from occurring at the same time as DMA requests of type (3)?

[3 marks] Bits 19, 18, 17 & 16 in the DMAREQC register are used to select external or eTPU DMA request signals instead of the internal (on-chip) DMA request signals for DMA controllers 3, 2, 1 & 0, respectively. If the external/internal enable bit for a DMA controller is cleared to 0 (selecting the internal DMA request signals), then a four-bit field in the DMAREQC register is used to select one internal source of DMA requests from the four DMA timers, the receivers of the three UARTs, and the transmitters of the three UARTs, as defined in the MCF5234 datasheet.

[3 marks] DMA requests from the CPU software are issued by writing a 1 to the START bit of the corresponding DMA Channel Control register (DCRn). The Enable External Request (EEXT) bit in the DCRn must be set to 1 to allow hardware DMA requests from the eTPU and the three externally controlled DREQn signals. The CPU software can be used to disable the internally caused DMA requests originating from the DMA timers and the UARTs. Thus request conflicts can be avoided.

Question #8 (TCP/IP Networking)

(a) Many interactions over the TCP/IP-based Internet occur following the client-server model. A client starts an Internet connection by (1) creating a socket and initializing the TCP entity to the CLOSED state, (2) sending a SYN segment to the desired server, (3) going from the CLOSED state to the SYN_SENT state in the TCP protocol, and (4) waiting for a SYN+ACK segment to be received from the server. What state transition(s) occur and what segment(s) are exchanged to cause the TCP entity to go from the SYN_SENT state to the connection ESTABLISHED state?

[6 marks] There are two possible paths from the SYN_SENT state to the ESTABLISHED state. (1) If the server sends back a segment with the SYN and ACK bits set, then the TCP entity sends an ACK segment to the server and goes directly to the ESTABLISHED state. (2) If the server sends back a segment with the SYN bit set and the ACK bit cleared, then the client sends an ACK segment and enters the SYN_RCVD state. Only when it then receives an ACK segment from the server will the client change states to the ESTABLISHED state.

(b) A server node gets ready to participate in an Internet connection by (1) creating a socket, (2) binding the socket to a port number, (3) going from the CLOSED state to the LISTEN state in the TCP protocol, and (4) blocking until a connection request is received from a client. Briefly explain the purpose of the port number. Then explain what events must then occur to cause the server to go from the LISTEN state to the connection ESTABLISHED state.

[3 marks] The 16-bit port number is used by the client to select the application within the server that should handle the connection. For example, port 80 is used to select the HTTP server program on the server node.

[3 marks] The server moves from the LISTEN state to the ESTABLISHED state via the SYN_RCVD state. When in the LISTEN state the server receives a SYN segment, it will respond by sending a segment with the SYN and ACK bits set, and it then go to the SYN_RCVD state. When in the SYN_RCVD state the server receives an ACK segment for its SYN, then it will go to the ESTABLISHED state.