ECE 315 - Computer Interfacing

Assignment #2 Solutions

Due: In the ECE 315 assignment box at 15:45 on Monday, Feb. 24, 2020

---

1. Like many other contemporary microcontroller units (MCUs), the MCF54415 provides the ability to enter a low-power mode, which would be attractive in applications where the system is battery-powered and where one wants to avoid wasting the limited stored energy when the MCU does not need to be running. Consult the on-line reference manual for the MCF54415 and briefly explain how the *wait*, *doze* and *stop* modes save power while preserving different degrees of MCU functionality. How does one cause the MCU to enter each of the three modes? How does one get the MCU to exit from the modes and go back to executing software in normal operation?

   ```
   [10 marks]
   The details of the low-power modes are provided in chapter 9 of the MCF5441X Reference Manual.
   The MCU enters a low-power mode when the STOP machine language instruction is executed by the
   CPU. The particular low-power mode is selected by the Low-power Mode Select (LPMD) bits in the
   Wake-up Control Register (WCR). In all three low-power modes, the clock signal to the CPU is
   stopped to stop the dynamic power consumption of the CPU. Note the CPU will still dissipate
   static power due to leakage currents.

   In the wait low-power mode, only the CPU's clock is stopped. The clock still goes to all of the
   peripheral blocks in the MCU. Those peripheral blocks can produce interrupt signals that may be
   high enough in priority to wake up the CPU.

   In the doze low-power mode, the CPU as well as some of the peripherals (consult the reference
   manual to find out which ones) have their clocks turned off to save dynamic power. The
   peripherals that are still clocked continue to function and they can produce interrupts that
   might wake up the CPU.

   In the stop low-power mode, the clock going to the CPU and all peripherals is stopped to obtain
   maximum savings in dynamic power consumption.

   The MPU exits the low-power mode if an interrupt request is received at a higher priority level
   than the PRILVL mask (bits #2, 1, 0) that was loaded previously in the Wake-up Control Register
   (WCR). The low-power modes are also exitted if a hardware reset signal is applied to the MCU.
   ```

2. In the lectures we discussed (1) multiple nonpreemptive loops with interrupts, and (2) periodically-scheduled state-driven code. What are the major similarities and differences between these two software architectures?

   [10 marks]
   Major similarities:

   - Both architectures can be used to set up timer-triggered polling using one or more (harmonic) loop periods.
   - Both architectures could use a graceful degradation strategy where the frequency of the polling could be reduced in a controlled manner.

   Major differences:

   - The state-driven code architecture is more flexible than the nonpreemptive loop architecture. The state machines that are implemented by each task do not have to be simple loops: they can have two or multiway branches when going from one state to the next state.
   - The state-driven code architecture is more modular and flexible because new state machines can be easily added to an existing system. In the nonpreemptive loop, the single thread must be modified to insert the new state machine.

3. What are the major advantages to designing the software in a real-time embedded system using tasks that execute in a multitasking software architecture? When would it be more appropriate to choose a single-threaded bare-metal

software architecture instead of a multitasking architecture?

[10 marks]
There are several advantages to designing the software of a real-time embedded system using a multitasking architecture: (1) The multitasking architecture naturally supports partitioning the software into simpler tasks that are easier to design and debug. (2) The multitasking environment provides many useful built-in features like semaphores and message queues. (3) Priority-based pre-emptive multitasking architectures provide a simple for allocating higher priority, and hence better real-time performance, for some tasks at the expense of lower priority tasks. (4) It is relatively straightforward to extend the software system by adding new tasks that can co-exist with any pre-existing tasks.

A single-threaded bare-metal software architecture is appropriate for simpler real-time embedded systems. The architecture is especially appropriate if all of the internal event timing can be triggered by a single hardware timer-generated execution loop. This architecture can be justified if it is very unlikely that new functions will need to be added later, functions that required new looping frequency(ies) that differ greatly from that of the original timing loop. By using the simpler single-threaded bare-metal architecture, debugging will be simpler and the worst-case real-time performance will be easy to guarantee.

4. Briefly explain how a co-operative multitasking software architecture could be implemented in MicroC/OS. Be sure to provide some idle time in your design to provide timing flexibility. You do not have to provide a full implementation in C for your proposed design, but you do need to propose a design that could be readily implemented using the existing features of MicroC/OS.

[20 marks]
In a MicroC/OS environment, the co-operating application tasks could be controlled using semaphores, where each task X is controlled by its own dedicated global semaphore, say SemX. The execution order of the tasks would be arranged in a daisy chain ring: for example, task A followed by task B, followed by task C, followed by task D, followed again by task A, etc. The tasks (e.g., A, B, C and D) would pend on semaphores (e.g., SemA, SemB, SemC and SemD, respectively).

A major design choice is how to post to the various semaphores. One might consider getting the tasks to post to their own semaphores. For example, the semaphores (e.g. SemA, SemB, SemC and SemD) could be posted by previous task in the ring (task D, task A, task B and task C, respectively). But such a design would cause additional problems since the priorities of the posting task cannot always be higher (or always lower) than the next task in the ring.

A more modular approach is to have one low priority task, say the scheduler task, that posts to the semaphores in the desired execution order for the ring. The scheduler task would be higher in priority than the idle task, but lower in priority than all of the cooperating application tasks. The scheduler task would have an endless execution loop in which each application task semaphore is posted in the desired order. As soon as a semaphore is posted by the scheduler, a context switch occurs to the higher-priority application task. The scheduler task executes again when the application task blocks on its own semaphore.

The cooperating application tasks and the associated semaphores would be created and initialized by the UserMain task. All of the applications tasks could be initialized so that they are each blocked on their semaphore. Then UserMain would then disable itself (say by calling OSTaskDelete) to get out of the way of the scheduler task. The scheduler task would then post to the first semaphore to unblock the first application task in the ring.

Advantages:

- The design is very modular because the application tasks only access their own semaphore, and are shielded from the semaphores used by the other tasks.
- The execution order of the application tasks is controlled in one place by one task, the scheduler task.

Disadvantages:

- The design suffers from the fundamental problem of all co-operative multitasking systems in that the mechanism for fairly sharing the execution time of the CPU is distributed across all of the application tasks. There is no central mechanism that can enforce fairness in how the CPU time is shared. Each application task is free to hog the CPU time.
- Also, the real time response of the application taks would likely be relatively poor. However, the effective real-time response of the system could be improved by using interrupt service routines.

5. In a pre-emptive multitasking system there are typically two systems of priorities: one for the tasks and one for the hardware interrupts. What are the different purposes of the two systems of priorities? In which ways do the two systems of priorities interact?

   [10 marks]
   The **priorities for the tasks** are used to provide the system designer with a simple way of controlling the quality of the real-time response that is experienced by the various tasks in the software system. All of the tasks will get enough time to execute and complete their function (otherwise the system will be failing to do its job), but higher priority tasks will in general have shorter and more predictable response times since when two (or more) ready-to-run tasks are competing to get time on the CPU, the highest priority task always gets awarded the CPU. The designer can choose to assign higher priorities to the tasks that are more important or that must run more frequently (and thus have tighter real-time deadlines). The task priorities are used only when there is a pre-emptive multitasking operating system installed on the CPU.

   The **priorities for the hardware interrupts** are used to quickly decide which interrupts, if any, should be promptly dealt with by executing the corresponding interrupt service routine (ISR). An interrupt handling mechanism is provided for most modern CPUs, and therefore interrupt handling can be present even in a bare metal system, where there is no operating system. Hardware interrupt features are built into the hardware of many peripheral systems in modern microcontrollers and microcomputers to allow fast and efficient interrupt-driven event handling.

   Interaction between the two kinds of priorities: Hardware interrupts can be used completely independently of the task priorities, with no direct interaction between the two systems of priorities. However, it is common for the activities of hardware interrupts to have some effects on the execution of tasks. For example, the execution of an ISR might be required to allow a task to be unblocked (e.g, the ISR is used to receive a message, place the message on queue, and then post to a semaphore flag that will unblock a task that processes the message. Conversely, task execution can cause interrupts to be disabled selectively (according to interrupt priority) on a temporary basis. High priority tasks may then be able to more rapidly affect the interrupts than lower priority tasks.

6. Consider the design of an asynchronous serial data connnection that must reliably communicate data packets that are 1500 bytes long. The nominal bit rate is to be 115200 bits per second. Reliable transport is assumed to be possible only if the sampling time stays within the middle 75% of the bit time. The receiver clock operates at 32 times the frequency of the transmitter clock, and the timing recovery scheme allows the receiver to select the best of 32 possible phases when choosing a divided-down receiver clock to be used to approximate the transmitter clock. How accurate must the transmitter and receiver clock frequencies be, in parts per million, in order to ensure reliable communication of the data packets?

   [10 marks]
   Packet length = 1500 bytes = 12000 bits

   Packet length w.r.t. perfect transmitter clock = (12000 bits)/(115200 bits/s) = 104.16667 ms (from the middle of the first bit to the middle of the last bit)

   Consider the worst case situation at the receiver, and assume that all of the frequency error is in the receiver clock. First, the first bit is sampled 17/32 = 0.53125 along the length of the transmitted bit because the timing recovery circuit just barely missed catching a clock edge at 16/32 along first bit. Second, the receiver clock is actually a bit too slow, and when it samples the last bit, it samples that bit at the last safe instant, which is 0.75 + (1-0.75)/2 = 0.875 along the length of the last bit.

   Packet length w.r.t. worst-case receiver clock = (12000 + (0.875 - 0.53125) bits)/(Rx_freq in bit/s) = 104.16667 ms

   Thus Rx_freq = 12000.34375 bits / 104.16667 ms = 115203.296 bits/s

   Worse-case relative difference in clock frequencies = (115203.296 - 115200)/115200 = 0.000028614 = 28.614 ppm

   The relative error can actually be split equally between the transmitter and receiver clocks, and so each clock must be accurate to 14.307 parts per million. This would allow the worst-case to be tolerated, where one of the clocks is too fast while the other is too slow.

7. Consult the documentation of the Micrel KS8721BT PHY and note that Ethernet 10BASE-T uses a Manchester line

code while Ethernet 100BASE-TX uses a 4B5B line code. Using reputable Internet resources determine what the purpose of these codes code is, and provide the tables that define what these codes are. What do you think is the main advantage of using the the 4B5B code instead of the Manchester code?

[10 marks]
The purpose of these line codes is to ensure that no matter what the transmitted data is, the transmitted bit signals will have a sufficient number of 0-to-1 and 1-to-0 transitions to ensure that the bit timing information will be present in the signal so that the receiver will be able to recover and reconstruct the transmitter clock. To achieve that goal, groups of data bits are encoded with slightly larger codewords that, by construction, have the desired signal transitions.

The Manchester code is applied to a digital bit stream by taking the exclusive-OR (XOR) between the digital data and the associated bit clock, where it is assumed that the rising edge of the clock is aligned with the starting times of the bits. The resulting signal has a rising edge wherever the bit is a "1", and a falling edge wherever the bit is a "0". Thus the signal contains both the required timing information (in the edges) and the bit information (in the direction of the edges). The cost of Manchester encoding is that the effective bit rate on the one combined is twice that of the original data signal.

The 4B5B line code replaces adjacent groups of 4 data bits with thecorresponding 5-bit codewords. There are several possible 4B5B linecodes. The code that is used by 100BASE-TX is as follows: 0000 =>11110 1000 => 10010 0001 => 01001 1001 => 10011 0010 => 10100 1010=> 10110 0011 => 10101 1011 => 10111 0100 => 01010 1100 => 110100101 => 01011 1101 => 11011 0110 => 01110 1110 => 11100 0111 =>01111 1111 => 11101 Note that each of the 5-bit codes contains at least one 0-to-1 or 1-to-0 signal transition. Thus not matter what the data bit sequence it, the corresponding codeword sequence will contain signal transitions that will communicate the clock timing. The cost of the 4B5B line codes is that the bit rate increases by24%.

The main advantage of the 4B5B code over the Manchester code is that the bit rate increases by only 25% instead of 100%.

8. Briefly define each of the following terms in computer networking: (a) duplex point-to-point link; (b) fully connected topology; (c) collision; (d) cross-over cable; and (e) daisy-chain configuration.

[10 marks]

a. A **point-to-point link** is a connection that is devoted to allowing two nodes to communicate directly with each other without passing through any intervening nodes. A **duplex** point-to-point link allows simultaneous communication in both directions between the two directly connected nodes.

b. A **fully connected topology** with N nodes is a network architecture in which all Nx(N-1) possible pairs of nodes are provided with duplex point-to-point links. A fully connected topology provides the maximum possible communication bandwidth among the N nodes, at the cost of fairly large number of links.

c. A **collision** is the situation where two (or more) nodes attempt to transmit at the same time onto the same communication medium (e.g, an Ethernet cable). During such a collision the signals interfere and no useful communication can occur. A protocol must be in effect that resolves the collision so that only one of the transmitting nodes proceeds to use the medium, and all of the other potential transmitting nodes wait their turn to use the medium next. The original Ethernet protocol specified that after collision, each of the transmitting nodes would wait some random period before trying again to transmit.

d. A **cross-over cable** is used to connect nodes that have conflicting driving nodes and receiving nodes. Cross-over cables, as suggested by the name, include wires that cross over. For example, the cross-over connection prevents two transmitters from driving each other, and two receivers from receiving the same nonexistent signal; the cross-over connection ensures that the transmitter output on each node has a connection to the receiver input of the opposite node. The cross-over connections can also be used to avoid conflicts in the handshake lines that perform hardware flow control, like the request-to-send (RTS) outputs and the clear-to-send (CTS) inputs. Cross-over cables are widely used in RS-232 direct connect data terminal equipment units to each other. Cross-over cables used to be widely using in Ethernet connections, but the need for such cables in Ethernet has disappeared in the recent Ethernet standards through the use of electronic autonegotiation.

e. A **daisy chain configuration** is a configuration where a sequence of electronic devices are connected in a loop with the first device driving signals to the second device, which drives signals to the third device, etc, down to the second-last device in the loop which drives the last device in the loop. The daisy chain connection allows all of the devices in the chain to be accessed by a host device using the same number of connections as a single device. All of the devices in the daisy chain will appear to the host to be one large device.