NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

EXAMINATION FOR
Semester 1 AY2006/2007

CS2271 – Embedded Systems

Nov/Dec 2006 **Time Allowed: 2 Hours**

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains **FOUR** questions and comprises **FOURTEEN (14)** printed pages, including this page. The weightage for each question is as indicated, and the total is 50 marks.
2. Answer **ALL** questions within the spaces provided in this booklet.
3. This is an Open Book examination.
4. Please write your Matriculation Number below.

MATRICULATION NO: _____

This portion is for examiner's use only

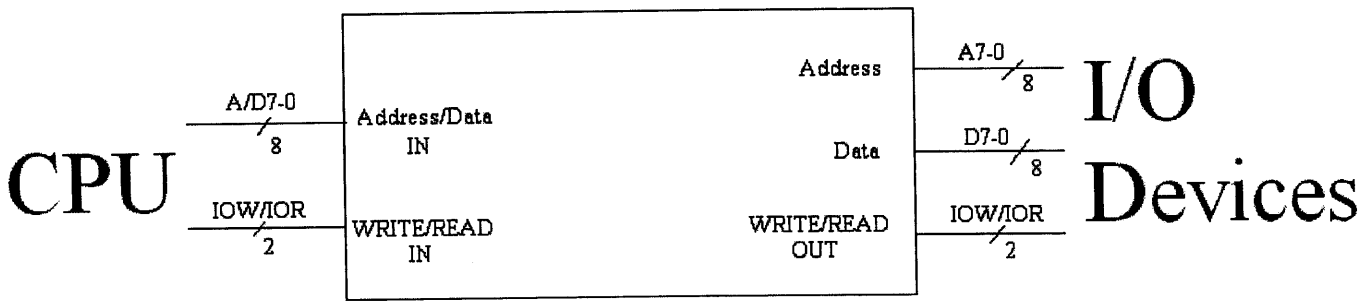
Question	Marks	Remarks
Q1	/20	
Q2	/10	
Q3	/10	
Q4	/10	
Total		

Question 1 (Handel-C) (20 marks)

An 8-bit "split-bus" CPU transmits data and addresses over the same 8-bit bus, with address being transmitted in the first cycle and data in the next. The particular CPU we are considering has internal memory that can be written to/read from by external devices, first by placing the address of the memory location in one cycle, and the data in the next. The CPU uses memory-mapped I/O.

The diagram below shows an Input-Output (I/O) bridge that interfaces with an 8-bit split-bus CPU and up to four I/O devices. The primary purpose of this bridge is to de-multiplex the addresses and data from the CPU into a separate 8-bit address bus and 8-bit data bus, and to multiplex addresses and data from I/O devices to the CPU onto the split bus. You may assume that the device that wish to communicate with the CPU would arbitrate amongst themselves first before accessing the I/O bridge. Hence the I/O bridge does not need to arbitrate among devices.

The CPU and I/O devices assert READ when wanted to read the device/CPU, and WRITE when wanting to write. Otherwise the two lines are de-asserted, and the I/O bridge is essentially idle.



a. Write down the Handel-C interface statement(s) and all related declarations for this I/O Bridge. You are free to name the pins as you wish ("P1', "P2", etc). (6 marks)

b. The function *reqFromCPU* reads from the CPU and outputs to the device. Complete the code below (4 marks):

```
void reqFromCPU()
{
```

```
}
```

c. The function *reqFromDevice* reads data/addresses from the device and outputs to the CPU. Complete its code below (4 marks):

```
void reqFromDevice
{
```

```
}
```

d. Write the main routine and any remaining routines you need to complete your I/O bridge. The internal Handel-C clock runs at ½ the external clock rate (3 marks).

e. Based on your code above, sketch the timing diagram, with respect to the external clock, showing the transfer of data from the I/O device to the CPU (3 marks).

Question 2 (Hardware / HDL Design) (10 marks)

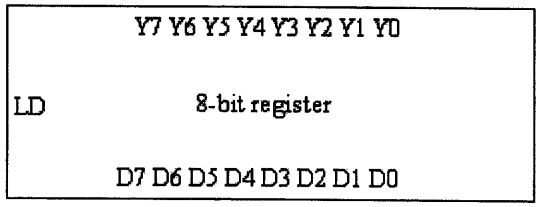
With reference to the I/O bridge in question 1, we will now build a circuit that outputs the contents of the 8-bit data bus onto two 7-segment LED displays. This circuit will be activated when the CPU writes to memory location 0x1F (remember that the CPU in question 1 has memory-mapped I/O).

a. Design a circuit using AND, OR and NOT gates to produce a "1" when the address bus from the I/O bridge contains the address 0x1F, and "0" otherwise. (2 marks)

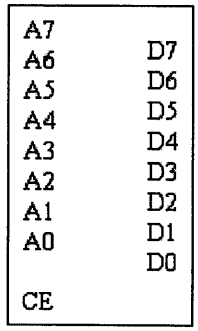
b. In implementing this circuit, we require a register that stores the values on the data bus. Explain why. (1 marks).

c. Given standard logic gates and the following components (you DO NOT need to design them!):

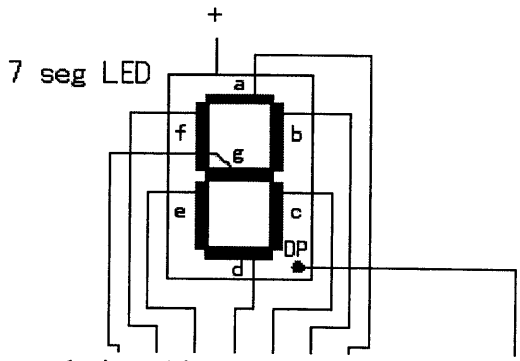
i. An 8-bit register that latches the contents of the input lines D7-0 when LD=1, and retains the previously latched value when LD=0. The contents of the register are output on lines Y7-0.



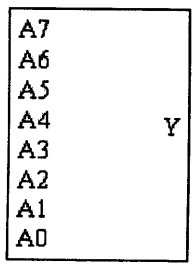
ii. Two 256x16 bit PROMs. The contents of the address location provided by A7-0 are output on D7-0 if CE=1. If CE=0, D7-0 are all 0. These PROMs are used to drive the 7-segment LEDs. Both PROMs will contain identical contents.



iii. Two 7-segment LEDs.



iv. The circuit you designed in part a, abstracted into a black box shown below:



Design a circuit to connect the two 7-segment LEDs to the I/O bridge in Question 1. You do not need to draw the bridge; you only need to show what lines are required. (4 marks)

d. Derive the first 9 entries of the PROM programming table for the PROMs in part c. Note that you can use the same programming table for both PROMs, so show only 1 PROM. You can leave your answers in binary. (3 marks)

Question 3 (Scheduling) (10 marks)

In this question, 3 tasks are running on a microprocessor and are scheduled using Rate Monotonic Scheduling. The tasks have the following characteristics, where T_{Cpu} is the amount of CPU time used by a task each time it is run, and T_{Period} that elapses between runs.

Task	T_{CPU} (ms)	T_{Period} (ms)
P1	2	10
P2	2	12
P3	2	6

a. Explain the term "schedulable'. Show an example of a set of tasks that is not "schedulable" under RMS. (3 marks).

b. Are the tasks given above schedulable under RMS? If yes, derive an unrolled schedule for the tasks. If not, explain why. (5 marks)

c. Compute the CPU utilization (2 marks).

Question 4 (Real-Time Programming) (10 marks)

In this question we will be implementing the interrupt handler for an RTOS. Some non-reentrant routines will be provided to you to do this at a later part of the question.

a. Although it is trivial to make the RTOS the highest priority 'task' in a system, re-entrancy is still an issue with RTOS routines. Explain briefly why. (1 mark)

We will now implement the part of the RTOS that captures and processes interrupts. Tasks register their interests in an interrupt, and some part of the RTOS written by another member of your team stores this information in a table starting at location 0x1C00, in the form of pointers to tasks blocked on this IRQ. The first entry of the table contains the pointer to the queue of tasks interested in IRQ0, the second entry contains the pointer to the queue of tasks interested in IRQ1, etc. The queues are sorted by task priorities. The RTOS also maintains a queue of ready tasks at memory location 0x1D00.

b. Write the fragment of code in ECPU assembly that sets up the ECPU's interrupt vector table to vector to the part(s) of the RTOS that correctly implement the interrupt system above (4 marks).

c. You are provided with the following routines, none of which are re-entrant. You can use these routines by setting up the registers as shown in the "Inputs:" column, and calling the RTOS entry point at 0x1F00. The "Outputs:" column shows what the routines return, if anything.

Based on your answer and information given in part b, implement the RTOS code that will check for tasks that are waiting for a particular interrupt, and wake up those tasks. You may also use a new ECPU instruction called "CLI" or Clear Interrupt. CLI pops the ECPU interrupt stack but does not return. Thus, executing "CLI" followed by "RET" is equivalent to executing "RTI". (5 marks)

Function	**Inputs:**	**Outputs:**
Data Structures		
Enqueue a task. Tasks are sorted by priority.	R0=0 R1 = Address of queue R2 =Task Number. R3 =Pointer to Task Descriptor.	None
Dequeue a task. Highest priority task is removed.	R0=1 R1 = Address of queue	R2 = Task Number. -1 if queue is empty. R3 = Pointer to Task Descriptor
Task Scheduler		
Run scheduler. If the task at the head of the ready queue has a higher priority than the current task, the current task is added back to the ready queue, and the task at the head of the queue is removed and run.	R0=2 R1=Address of ready queue.	R1=Task number of task picked for running R2 = Pointer to Task Descriptor
Disable scheduler. Suspends all scheduling activities.	R0=3	None

~~~~~ END OF QUESTIONS ~~~

Page is intentionally left blank.