NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

EXAMINATION FOR
Semester 1 AY2007/2008

CS2271 – Embedded Systems

Nov/Dec 2007 Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains **FOUR** questions and comprises **EIGHTEEN (18)** printed pages, including this page. The weightage for each question is as indicated and total 60 marks. Note that the marks distribution is not equal across questions.
2. Answer **ALL** questions within the spaces provided in this booklet. If you need more space, you may write on the back of the page, but please indicate P.T.O. on your answer, or whatever is written on the back will be ignored.
3. This is not an Open Book examination. You are permitted a single 2-sided cheat-sheet.
4. Please write your Matriculation Number below.

MATRICULATION NO: ______________________________

This portion is for examiner's use only

Question	Marks	Remarks
Q1	/ 6	
Q2	/30	
Q3	/10	
Q4	/14	
Total	/60	

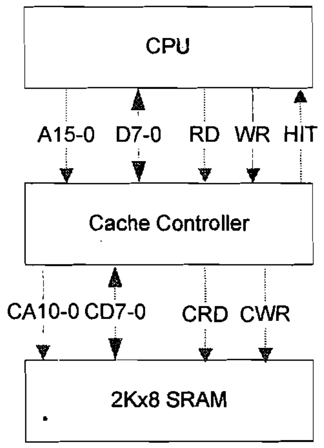
Question 1 Programmable Logic Devices **(6 Marks)**

a. Describe, in 100 words or less, what an FPGA is, and how the hardware resources provided on an FPGA work together to very quickly implement complex logic systems like CPUs and controllers. (3 marks).

b. “Handel-C is much better suited to all forms of circuit design than ABEL (Advanced Boolean Expression Language)”. Would you agree or disagree? Explain why. (3 marks).

Question 2 HDL Design **(30 marks)**

A cache is a small, fast static-RAM based memory that stores portions of main memory for quick access. Since not all of main memory can be stored in cache, a *tag* is used to identify the actual memory location that the data came from. In this question, we will implement a two-way set associative cache controller that interfaces between a CPU and a single 2Kx8 static RAM chip as shown in the diagram below.



This is a 2-way set associative cache, meaning that when the controller receives an address from the CPU, the controller maps the address to *n* sets, and each set consists of 2 blocks which are fully-associative (see later what this means).

For the purpose of this paper, we will look only at how to implement the "cache write" portion of the cache controller on FPGA. The CPU and SRAM are separate units connected to the FPGA through the FPGA's pins. Note that the cache here may be slightly different from what you may have learnt in other courses (e.g. it doesn't have a dirty bit, and we will leave out writing to the main memory).

Cache Write

To write to the cache, the CPU places the address to write to on A_{15}-A_0, the data to be written on D_7-D_0, then asserts the WR line. The cache controller maps A_9-A_0 to one of its 1024 sets, then uses bits A_{15}-A_{10} to check the tags stored in the two blocks of the set. If A_{15}-A_{10} matches at least one of the tags, the cache controller asserts the HIT line to the CPU informing the CPU that there has been a write-hit.

Note that a cache set consists of two blocks, and each block consists of a tag which is stored on the cache controller, and a data byte, which is stored on the SRAM.

If there has been a write-miss, the cache controller chooses one of the two blocks in the set (how is up to you. Some suggestions include "round robin" where each block takes turns to be written to, or least-recently-used where the block that was least recently read/written is used), and stores A_{15}-A_{10} into the tag portion of the table.

Regardless of hit/miss, the cache controller copies bits A_9-A_0 onto lines CA_{10}-CA_1, and sets CA_0 to 0 if the A_{15}-A_{10} matches the tag stored in the first block of the set, or 1 if it matches the second block of the set. The cache controller then copies over D_7-D_0 to CD_7-CD_0, and asserts the CWR line to write to the SRAM.

Timing Requirements

External and Internal Clocks:

The CPU, FPGA and SRAM are connected to a 40 MHz clock. In addition the FPGA's internal 20 MHz clock is derived from this clock signal.

CPU Write Cycle:

The CPU places the address onto the A_{15}-A_0 address bus and asserts the WR line together. It then places the data onto the D_7-D_0 data bus 50ns later. The data is held for at least 50ns by the CPU, together with the address and WR lines.

SRAM Write Cycle:

The SRAM requires the address to be placed by the cache controller onto the CA_{10}-CA_0 address bus, while the controller simultaneously asserts the CWR line. The controller should wait for 50 ns, before placing the data to be written to the SRAM onto the CD_7-CD_0 data bus, and hold the data, address and CWR lines for at least 75 ns.

a. Write, in Handel-C, the necessary `interface` statements to communicate with the CPU. You can freely name the pins that you require (e.g. "P1", "P2", etc). To save time, for this paper you are allowed to use the shorthand notation "P1"... "P4" to mean "P1", "P2", "P3", "P4", etc. (5 marks)

b. Similarly, write the interfaces to the SRAM. (5 marks)

c. Write a function (or module) with the following function header definition (note that all definitions are given in pseudocode, not in Handel-C):

   *boolean findtag(unsigned address, unsigned *slot)*

   This function uses the address to map to a global 1024-entry tag table, then searches the 2 blocks within that entry for the tag, derived from the most significant 6-bits on the address.

   The function returns TRUE if there has been a tag match, and FALSE otherwise. If the function returns TRUE, then *slot* contains 0 if there was a tag match in the first block of the set, and 1 if in the second block of the set. (Hint: It's easiest to implement this using a global 2D array of 1024x2 entries. Show your declaration for this array, if you are using one.) (6 marks)

d. Write the function *cpu_read* which reads the address and data from the CPU, subject to the timing constraints in Figure 2. This function has the following header definition:

*void cpu_read(unsigned *addr, unsigned *data)*

Where *addr* and *data* contain the address and data read from the CPU respectively. (4 marks)

e. Write the function *sram_write* that writes to the SRAM, subject to the timing constraints in Figure 2. This function has the following header definition.

   void sram_write(unsigned addr, unsigned data)

   Where *addr* and *data* are the address and data to be written onto the SRAM respectively. (4 marks).

f. Write a main routine that:

i) Monitors the WR signal. When asserted, it reads the address and data placed by the CPU.
ii) It then checks for a cache hit, and asserts the HIT line accordingly.
iii) If there is a cache miss, it will use A_9-A_0 to map to one of the 1024 entries in the tag table, then chooses one of the two blocks of that entry and writes the tag into that block.
iv) Generate the address to write to the SRAM, and use sram_write to write the data to the SRAM.

You should also write in any other code (e.g. clock declaration) that you might need to make this into a full Handel-C program. (6 marks)

Question 3 Real Time Operating Systems and Scheduling **(10 marks)**

a. Describe, briefly, what can go wrong with the following code, and under what circumstances. Assume that all RTOS calls are not interrupt-aware (i.e. the calls cannot differentiate between being called by task code vs. being called by an ISR) (3 marks).

```
OSSema A=1, B=0;  // Semaphores. Assumed initialized correctly.

// ISR
interrupt void myisr()
{
      !! Read device
      releaseSema(B);
}

void taskA()
{
      while(1)
      {
            !! some stuff here
            takeSema(A);
            !! Perform long computation
            releaseSema(A);
      }
}

void taskB()
{
      while(1)
      {
            !! some stuff here
            takeSema(B);
            !! Perform long computation.
      }
}

void taskC()
{
      while(1)
      {
            !! some stuff here
            takeSema(A);
            !! Perform short computation
            releaseSema(A);
      }
}

void main()
{
      !! Add tasks and start up OS.
}
```

(Page is intentionally left blank)

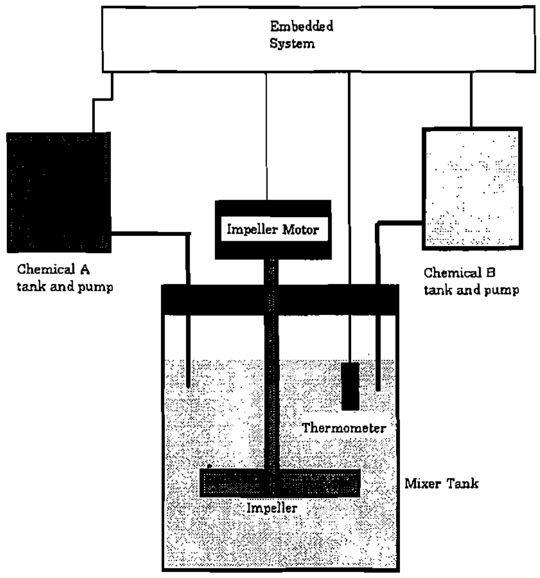
(Page is intentionally left blank)

b. Explain why critical instant analysis gives a more reliable assessment of the schedulability of a set of tasks than the Liu and Leyland criterion. (3 marks).

c. "Given a set of tasks, as long as the CPU utilization is under 100%, the task set is guaranteed to be schedulable under earliest-deadline first". Would you agree with this statement? If yes, briefly explain why, and if no, construct a counter example to disprove the statement. (4 marks).

Question 4 Low Level Device Programming (14 marks)

You have been assigned to program a controller that measures the temperature of a chemical mixer tank. The full system is shown in the figure below. Your controller needs to maintain a temperature of between 20°C and 25°C. The thermometer must be polled every 5 ms. Assume that there is a timer that interrupts the system on IRQ0 every millisecond.



When the temperature of the mixture falls below 20°C, you will need to turn on the pump for chemical A, and the rate of flow is determined by:

$\text{Rate}_A = 0.315\,(1.5T - 20) + 0.21$, where T is the measured temperature.

When the temperature goes above 25°C, you will need to turn on the pump for chemical B. The rate of flow is given by:

$\text{Rate}_B = 0.91(T - 25)$, T is again the measured temperature.

In both cases, the rate of flow must be adjusted as the temperature rises (first case) or falls (second case), and the impeller must be turned on at the start, and turned off at the end. Assume that the mixer tank is sufficiently large so that it never overflows.

The embedded system uses memory-mapped I/O and has the following I/O ports.

Purpose	I/O Port	Semantics
Thermometer (Read)	0x3b0	Temperature in Degrees C, correct to 3 decimal places.
Impeller (Write)	0x3b4	0 to turn off impeller, non-zero value to turn on.
Pump A (Write)	0x3b8	Rate of flow, given as a floating point number.
Pump B (Write)	0x3bc	Rate of flow, given as a floating point number.

You may find it useful to access each I/O port by using a pointer of an appropriate type. E.g.

```
int *imp = 0x3b4;
*imp=1;  // Turn on impeller.
...
*imp=0;  // Turn off impeller.
```

All programming is to be done in C (not Ċ!!). You may assume that your CPU is sufficiently fast to execute your control programs in under 5 ms. That is, you are guaranteed to be able to read the thermometer at least once every 5 ms.

a. Draw a structure diagram like the one found on the RTOS case study. To refresh your memory, the diagram from the case study is attached in the Appendix (Note: This diagram has nothing to do with the question, and is attached just as a reference) (3 marks)

b. Write the interrupt service routine for the timer. Use the following C prototype. (3 marks)

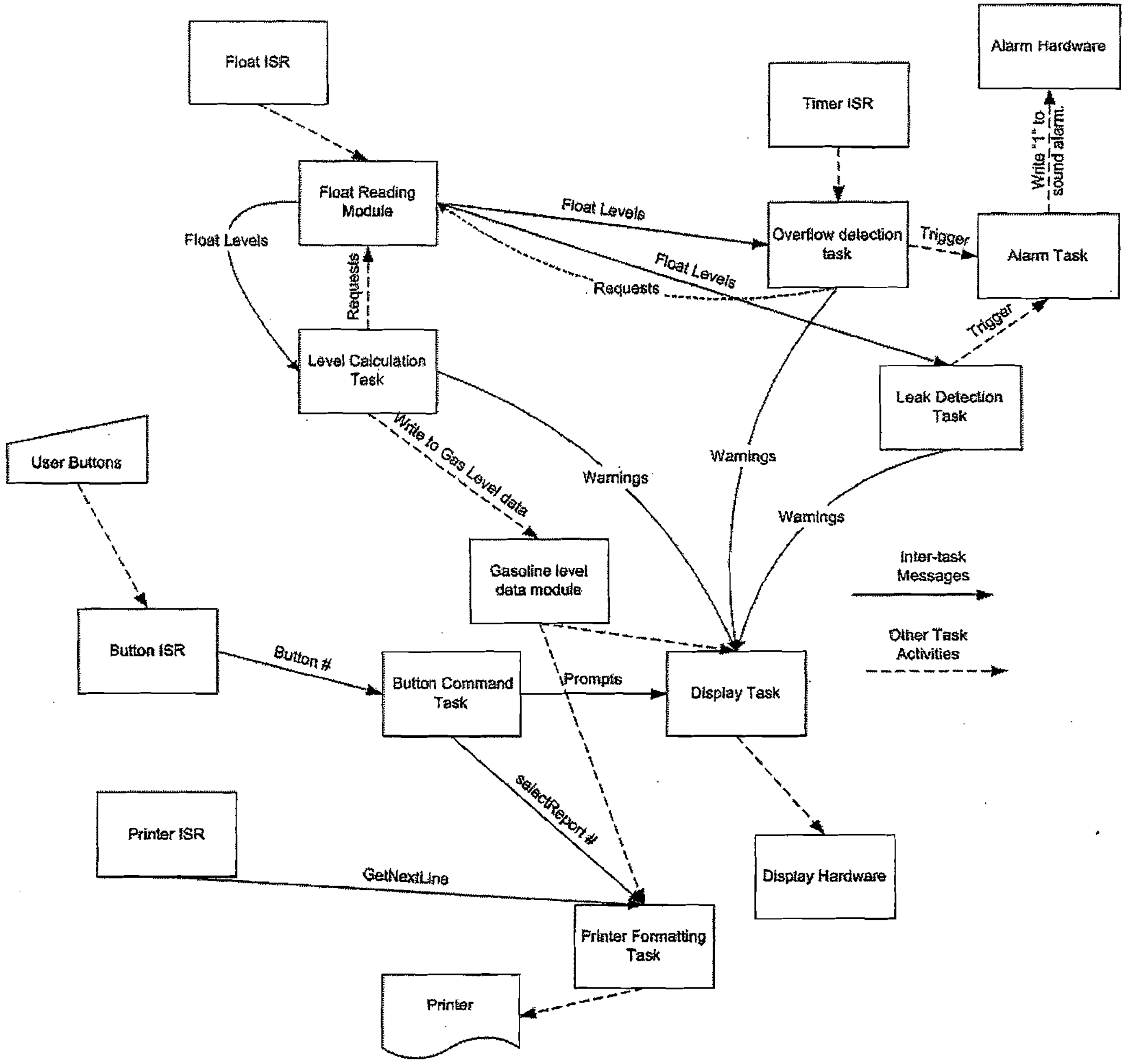
```
interrupt void TimerISR();
```

c. Write the `main` program, which will register the timer ISR, monitor the temperature, and control the pumps and impeller to fulfil the requirements as mentioned earlier. You may decompose the problem into sub-functions if you wish. Assume the availability of a library call to register interrupts, with the following prototype:

```
void RegInterrupts((void *) intISR(), unsigned IRQNum);
```

The first argument is a pointer to the ISR, the second is the IRQ number. (8 marks)

Appendix



End of Paper