EXAM
(Tentamen)

TDDI11
Embedded Software

2021-06-03 kl: 08-12

On-call (jour):
ahmed.rezine@liu.se

Admitted material:
- You can access your individual notes, books, and even search the internet.
- No contacts, whether physical or virtual, are allowed during the duration of the exam with any person, whether the person is related to the course or not, except for contacting the examiner via email for questions if any.
- Any **suspected breach** will be **systematically reported to the disciplinary board**.

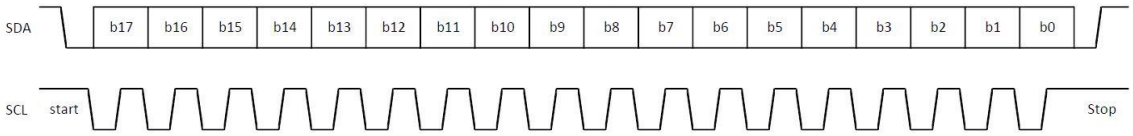
General instructions:
- **The questions will refer to your "D_1D_2" digits.** These are the last two digits of the "anonymous-ID" you are assigned during the exam. You find the "anonym-ID" on the Lisam page you have retrieved this pdf from (i.e., the page you got the exam questions from). For instance, if your "anonymous-ID" is 2709 then your D_1 is 0 and your D_2 is 9. If your "anonymous-ID" is 123 then your D_1 is 2 and your D_2 is 3. Ask the examiner if this is unclear.
- **The questions refer to a "values"** file (given in both pdf and xlsx formats for convenience). These can be fetched from the same Lisam page you got these questions from.
- The assignments are not ordered according to difficulty.
- You may answer in either English or Swedish.
- Do **not take pictures or draw**. We only accept pdf, text or docx/odt files obtained from a text editor or a word processor. You are free to use any text editor (examples include notepad, vim, gedit, emacs, etc) or other text-based office programs (Microsoft Word or Open/Libre Office or LaTeX). We expect you to generate and to submit a single file.
- Be **precise** in your statements.
- **Clearly motivate** all statements and reasoning.
- **Explain** calculations and solution procedures.
- If in doubt about the question, write down your interpretation and assumptions.
- Grading: U, 3, 4, 5. The **preliminary** grading thresholds for p points are:

$0 \leq p < 20$:	U
$20 \leq p \leq 30$:	3
$31 \leq p \leq 35$:	4
$36 \leq p \leq 40$:	5

Question 1. (17 points)

a. Assume the I2C bus protocol discussed in the course lectures. Assume 128 devices are connected to the serial bus. Suppose a Master wants the device with address **addr** to send it a byte, and that the device with address **addr** sends the Master the byte **byte** (the values of **addr** and **byte** associated to your D_1D_2-ID are given in binary in the "values" file). Suppose transmission occurs without problems and acknowledgments are sent. In addition, suppose acknowledgments are active low (i.e., devices write 0 to mean they acknowledge according to the protocol) and that 1 is used for read and 0 is used for write. Finally, assume when sending a byte **byte** or an address **addr**, that the bit most to the left (i.e., the most significant bit) is sent first.

Give the corresponding sequence of bits b_{17}, b_{16}, …, b_0 depicted in the following timing diagram (3pts)



b. Can a real time system be correct and still sometimes miss some deadlines? If yes, give an example and explain what kind of real time system it is. If not, argue why not. (2pts)

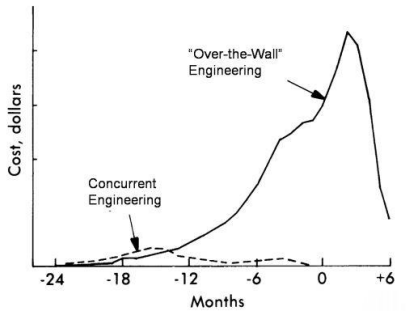
c. Answer the two questions:

c.1.Write a C function "`int foo(unsigned char ss, unsigned char mm, unsigned char hh)`" that encodes the seconds `ss`, minutes `mm` and hours `hh` in the 17 least significant bits "$b_{16}b_{15}…b_2b_1b_0$" of the 32-bits result: $b_{31}b_{30}…b_2b_1b_0$ (2pts):

- Seconds, passed to `foo` via the parameter `ss` of type `unsigned char` will be stored using the 6 least significant bits "$b_5…b_2b_1b_0$",
- Minutes, passed to `foo` via the parameter `mm` of type `unsigned char` will be stored using the following 6 least significant bits "$b_{11}…b_7b_6$",
- Hours, passed to `foo` via the parameter `hh` of type `unsigned char` will be stored using the 5 following least significant bits "$b_{16}…b_{13}b_{12}$",
- The remaining bits $b_{31}b_{30}…b_{19}b_{18}b_{17}$ should be 0.

c.2. What is the result of `foo(ss,mm,hh)` for the values of `ss`, `mm` and `hh` given in decimal in the "values" file and associated to your D_1D_2-ID (1pt)

d. Assume the figure to the right describes costs' evolutions for two similar embedded IT-projects (same technical complexity, labor costs, etc). The only difference is the adopted organizations of the work. Discuss what would explain the higher costs associated to concurrent engineering in the beginning of the projects, and why that can be worth it given the evolution of the curves. (3pts)



e. Consider the sequence of C instructions to the right. The values associated to your D_1D_2-ID for the pointer `pb` (in hexadecimal) after execution of line 3 and before the execution of line 5, and for the integers `k1` and `k2` (in decimal) are to be retrieved from the "values" file. Recall the value of a pointer is the address stored in it.

```
1  unsigned char b[64];
2
3  unsigned char* pb = b;
4
5  unsigned int* pi = (unsigned int*) pb;
6
7  pi+=k1;
8
9  pb = (unsigned char*) pi;
10
11 pb+=k2;
```

1. What is the value of `pi` (in hexadecimal) after execution of line 5 and before execution of line 7? (1pt)
2. What is the value of `pi` (in hexadecimal) after execution of line 7 and before execution of line 9? (2pts)
3. What is the value of `pb` (in hexadecimal) after execution of line 9 and before execution of line 11? (1pt)
4. What is the value of `pb` (in hexadecimal) after execution of line 11? (2pts)

Question 2. (6 points)

1. Explain two existing and different approaches for the CPU to know which instructions it should execute when it receives an interrupt. (2pts)

2. Polling is often said to waste CPU cycles because it repeatedly checks the status of a register. CPUs using interrupts can also "continuously check" the status of an interrupt pin. Explain why this checking is said to be more efficient than the checking performed in Polling. (2pts).

3. Direct Memory Access uses a co-processor that interrupts the CPU to gain access to the memory bus. Describe two scenarios: one where using a Direct Memory Access is better (performance wise) than using plain interrupts, and one where it is worst (performance wise) to use a Direct Memory Access. (2pts)

Question 3. (11 points)

Consider a task set with three periodic tasks: Task 1 with execution time C1 and period T1; Task 2 with execution time C2 and period T2; and Task 3 with execution time C3 and

period T3. Values associated to your D_1D_2 ID-number for the periods T1, T2 and T3 and the execution times C1, C2 and C3 are to be fetched from the "values" file. All three tasks are to run on the same processor using some scheduling algorithm. Some of the questions use a duration value called hyper-period. The hyper-period associated to your D_1D_2 ID-number can also be found in the "values" file.

1. Give the processor utilization ratio in case the tasks are scheduled. (1pt)
2. Which task would get the highest priority if Rate Monotonic Scheduling (RMS) is used? (1pt)
3. Can the tasks be scheduled using Preemptive RMS? Explain with a diagram using hyper-period time units. (3pts)
4. Can the tasks be scheduled using Preemptive Earliest Deadline First (EDF)? Explain with a diagram using hyper-period time units (3pts)
5. Suppose you are given a set of three tasks (not necessarily those associated to your D_1D_2-ID). Suppose the tasks have to be scheduled using the Preemptive RMS algorithm (this is a requirement that cannot be changed). However, your analysis shows that Preemptive RMS cannot schedule those tasks on your sequential controller. Discuss what options do you have to schedule the work with Preemptive RMS on a sequential controller and without missing deadlines. (3pts)

Question 5. (6 points)

As it is often the case, newly pushed stack elements get smaller addresses. Assume the calling conventions you used in the labs. A C-function `foo` has just been called (with `foo(x,y)`) using two 32-bits arguments `x` and `y`. The return address is stored on top of the stack in the 4 bytes `0x3fffffe8-0x3fffffeb`. (see the figure below).

The arguments `x` and `y` associated to your D_1D_2-ID can be found in the "values" file. The 4 bytes associated to `x` are listed in the "values" file where the most significant byte of `x` is `x:a`. It is followed, in decreasing order of significance, by `x:b`, `x:c` and the least significant byte `x:d`.

In a similar manner, byte `y:a` in the values file is the most significant byte of `y`. It is followed, in decreasing order of significance, by `y:b`, `y:c` and the least significant byte `y:d`.

Answer the following questions:

1. Recall `x` has 4 bytes, each occupying an address location in the stack. The same is true for `y`. What is the smallest address containing anyone of the bytes of `x` in the stack? What is the smallest address containing anyone of the bytes of `y` in the stack? (2pts)
2. Assume a big-endian system. Give the content of the memory locations `0x3fffffec-0x3ffffff3.` (2pts)
3. Assume a little-endian system. Give the content of the memory locations `0x3fffffec-0x3ffffff3.` (2pts)

```
1  0x3fffffe8 <-- esp
2  0x3fffffe9
3  0x3fffffea
4  0x3fffffeb
5  0x3fffffec
6  0x3fffffed
7  0x3fffffee
8  0x3fffffef
9  0x3ffffff0
10 0x3ffffff1
11 0x3ffffff2
12 0x3ffffff3
13 0x3ffffff4
14 0x3ffffff5
15 0x3ffffff6
16 0x3ffffff7
17 0x3ffffff8
18 0x3ffffff9
19 0x3ffffffa
20 0x3ffffffb
21 0x3ffffffc
22 0x3ffffffd
23 0x3ffffffe
24 0x3fffffff <-- stack bottom
```