EXAM

(Tentamen)

TDDI11

Embedded Software

2021-08-18 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:

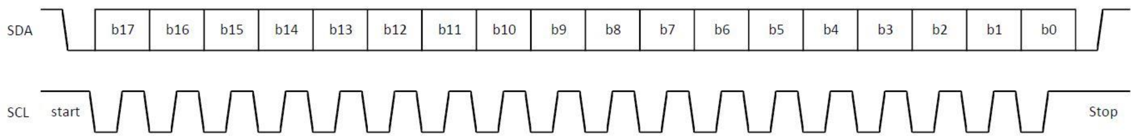
- **The questions will refer to your "D_1D_2" digits.** These are the last two digits of the "anonymous-ID" you are assigned during the exam. You find the "anonym-ID" on the Lisam page you have retrieved this pdf from (i.e., the page you got the exam questions from). For instance, if your "anonymous-ID" 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, and manages, to send a byte **byte** to the device with address **addr** (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 (reading and writing is from the perspective of the Master). 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 $b_{17}, b_{16}, \ldots, b_0$ depicted in the following timing diagram (3pts)



b. Explain the difference between a soft and a hard embedded real time system? (2pts)

c. Answer the two questions:

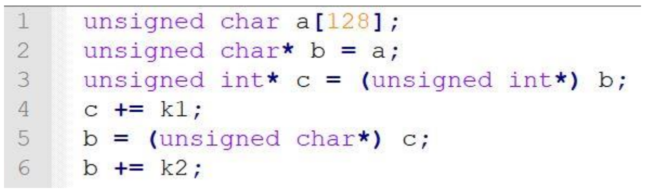
c.1.Write a C function "`int foo(unsigned int yy, unsigned int mm, unsigned int dd)`" that encodes the year `yy`, month `mm` and day `dd` in the 21 least significant bits "$b_{20}b_{19}\ldots b_2b_1b_0$" of the 32-bits result: $b_{31}b_{30}\ldots b_2b_1b_0$ (2pts):

- Day of the month is passed to `foo` via the parameter `dd` of type `unsigned int` and will be stored using the 5 least significant bits "$b_4 b_3b_2b_1b_0$",
- Month of the year is passed to `foo` via the parameter `mm` of type `unsigned int` will be stored using the following 4 least significant bits "$b_8b_7b_6b_5$",
- Year is passed to `foo` via the parameter `yy` of type `unsigned int` will be stored using the 12 following least significant bits "$b_{20}\ldots b_{10}b_9$",
- The remaining 11 bits "$b_{31}\ldots b_{22}b_{21}$" should be made 0.

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

d. Discuss advantages and disadvantages of cross functional teams in engineering projects. (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 2 and before the execution of line 3, 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 a[128];
2   unsigned char* b = a;
3   unsigned int* c = (unsigned int*) b;
4   c += k1;
5   b = (unsigned char*) c;
6   b += k2;
```

1. What is the value of c (in hexadecimal) after execution of line 3 and before execution of line 4? (1pt)
2. What is the value of c (in hexadecimal) after execution of line 4 and before execution of line 5? (2pts)
3. What is the value of `b` (in hexadecimal) after execution of line 5 and before execution of line 6? (1pt)
4. What is the value of `b` (in hexadecimal) after execution of line 6? (2pts)

Question 2. (6 points)

1. Explain the advantage of using a vectored interrupt mechanism instead of a fixed interrupt mechanism. (2pts)

2. Explain the advantage of using an interrupt address table compared to a vectored interrupt mechanism. (2pts)

3. In what situation is writing and extending code for Programmed I/O more cumbersome and difficult than writing and extending code for interrupt driven I/O? Explain. (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. Give an example of a set of three tasks (you can use/modify the tasks associated to your D_1D_2-ID) that are not schedulable (i.e., at least one of them misses a deadline) using the Preemptive RMS algorithm. Show that diminishing execution time can make them schedulable with Preemptive RMS. Explain at least two ways this could be achieved and discuss the tradeoffs. (3pts)

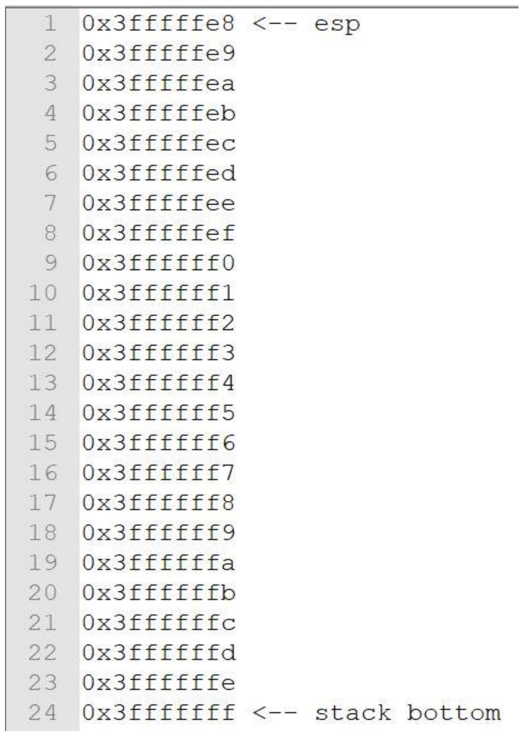
Question 5. (6 points)

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:



```
0x3fffffe8 <-- esp
0x3fffffe9
0x3fffffea
0x3fffffeb
0x3fffffec
0x3fffffed
0x3fffffee
0x3fffffef
0x3ffffff0
0x3ffffff1
0x3ffffff2
0x3ffffff3
0x3ffffff4
0x3ffffff5
0x3ffffff6
0x3ffffff7
0x3ffffff8
0x3ffffff9
0x3ffffffa
0x3ffffffb
0x3ffffffc
0x3ffffffd
0x3ffffffe
0x3fffffff <-- stack bottom
```

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)