Försättsblad till skriftlig tentamen vid Linköpings Universitet

Cover page for written exam at Linköping University

Datum för tentamen Date of exam	**2010-05-25**
Sal Room	**VALMAT**
Tid Time	**08:00-12:00**
Kurskod Course code	**TDDI11**
Provkod LADOK code	**TEN1**
Kursnamn/benämning Course name	**Programmering av inbyggda system** Embedded software
Institution Department	**IDA**
Antal uppgifter som ingår i tentamen Number of assignments	**6** 6 assignments for a total of 40 points
Antal sidor på tentamen (inkl. försättsbladet) Number of pages including cover	**4**
Jour/Kursansvarig Responsible/Examiner	**Klas Arvidsson** klas.arvidsson@liu.se
Telefon under skrivtid Phone during exam	**013 - 28 21 46**
Besöker salen ca kl. Time of exam visit	**Start + 1h**
Kuradministratör Course administrator	**Gunilla Mellheden** 013 - 28 22 97, 070 - 597 90 44, gunilla.mellheden@liu.se
Tillåtna hjälpmedel Allowed aids	**Ordlista och enkel miniräknare (+,-,*,/)** Dictionary and simple pocket calculator (+,-,*,/)
Övrigt Other information	*Preliminary graded: U < 50% < 3 < 67% < 4 < 84% < 5* *Grades may be raised or lowered based on overall impression.* ***Precise, explained and clearly motivated assumptions, statements and reasoning raise the impression and may triple the points. Solve at most one assignment per sheet.*** *Results available within 10 working days.* *Read all assignments and estimate how many points you can get before you start.*
Typ av papper Paper to use	**Rutigt, linjerat eller blankt** No preference
Antal anmälda Number of exams	**35**

1. (8p)

a) *Define an embedded system.*
What makes a system embedded? Explain each term you use. (4p)

b) When you design an embedded system certain technology can provide (technical or economical) benefits or drawbacks. List four of the metrics (technical or economical) commonly used to compare design- and IC-technology? No motivation required. (2p)

c) Given the task to design a system where the four metrics you gave in the previous question is very important, select one design technology (special-purpose, general-purpose, or application-specific) and one IC-technology (full-custom, PLD, or semi-custom) that provide a good compromise between them. Motivate. Motivate. Motivate. (2p)

2. (3p)

Describe the workings and organization of a foreground-background system. An explained picture is appreciated.

3. (3p)

N is an integer variable in the range 0-7. D is an one byte variable. Provide C-code (or comparable pseudo-code) to do each of the following tasks:

a) Show how to retrieve the value of the N:th bit in D. All bits must remain unchanged. (1p)

b) Set the N:th bit of D to logical one '1'. All other bits must remain unchanged. (1p)

c) Reset the N:th bit of D to logical zero '0'. All other bits must remain unchanged. (1p)

4. (6p)

A 4-bit CPU needs to multiply two 8-bit integers to produce a 16-bit result. But only 4-bit multiplication with 8-bit result is available, as well as 4-bit addition with or without carry.

a) Show mathematically or graphically (no code) how to perform the multiplication on the CPU. It should be clear which (4-bit) multiplications and additions that are done. (4p)

b) Addition may produce a carry (overflow). Adding the carry (+1) may in turn produce a new carry (overflow). But certain of the additions with one (+1) you do in a) will never overflow. Which and why? (2p)

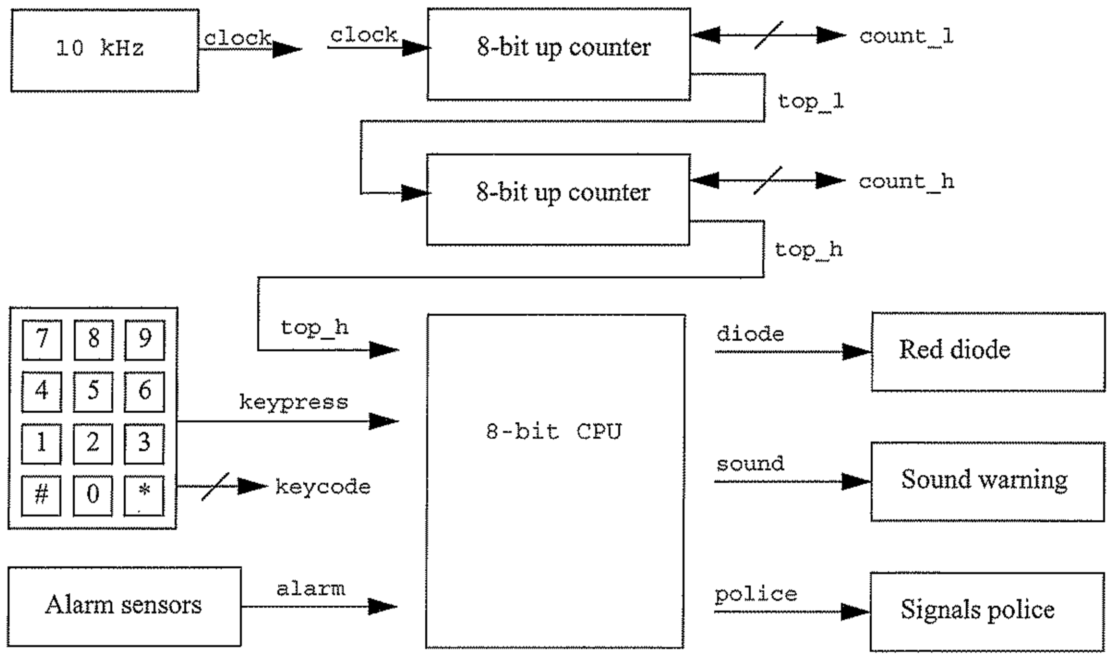
5. (16p)

The following text describe a scenario that will be used for several theory and code assignments. Italic words outline keywords you are expected to know extra well from the course.

A design team consisting of you, you, and you should implement a burglar alarm controller. The figure on next page describe the system devices connected to a 8-bit *general-purpose* processor with four interrupt pins. *Vectored interrupts* is used. Each of the device output signals `top_h`, `keypress` and `alarm` is connected to an interrupt pin. The last interrupt pin is left unconnected.

Each of the device registers `count_l`, `count_h`, and `keycode` is *memory mapped* to the system bus. They use addresses `COUNT_L`, `COUNT_H` and `KEYCODE` respectively. A small `ROM` memory is addressed from `0` to `ROM_END`, and a small `RAM` memory is addresses from `RAM_START` to `RAM_END`. The *8-bit counter (timer)* `clock` input is connected to a 10 kHz clock (100 micro-seconds per tick).

The system have three *write-only* outputs. They share a *port* named `DSP` on the system, where (when writing to the port) bit 0 control the diode, bit 4 control the sound and bit 5 the "police" signal. The remaining 5 bits is not defined and should be kept low (0).



The following specification is given:

- The alarm system is turned on and off by entering a 4 digit code followed by a star (*).
- After three consecutive failed attempts at the code the system should refuse further attempts on the code. Only a service technician can unlock it again (by resetting the software).
- Any partially entered codes should be reset when the keyboard have been inactive for 30 seconds. This is to avoid that partially entered codes mess up the next attempt to activate or inactivate the alarm system.
- The red diode should be alight when the alarm is active and off when the alarm is inactive.

The alarm signal goes high if any sensor on doors or windows is activated, it means someone likely entered the surveillance area. The system should now activate the sound warning, and if the correct code is not provided within 30 seconds the system should send a signal to the "police". If the correct code is entered within 30 seconds the system will be inactivated and the diode turned off. The system is turned on by entering the code again.

You plan to implement the burglar alarm controller with simple interrupt handlers that trigger the execution of a finite state machine. The FSM is only called when an handler change an input signal used by the FSM. The FSM will in turn activate the system outputs as needed.

If you believe you need to know details not specified in the assignment you may assume them as you like. But clearly state all assumptions you do.

a) Explain the meaning of a *memory map* and draw an example how it will look like with the given system. Use symbolic names when specific addresses or sizes are unknown. (1p)

b) Explain the meaning of a *interrupt service vector*. Draw an example of how you arrange it in the given system. Use symbolic names when specific addresses are unknown. (2p)

c) State the difference between *port-based* and *memory-mapped* I/O. What is the difference from assembly point of view? (1p)

d) Write the functions `timer_start`, `timer_stop` and *interrupt handler* for the two cascaded (time) counters. Use precise pseudo-code, C-code, or x86-like assembly as you like. Your code should call the function `main_fsm()` 30 seconds after the last call to `timer_start`, unless `timer_stop` is called before 30 seconds elapsed. You may use global variables. (5p)

e) Use bit-manipulation to implement functions a programmer can call in a simple way in order to *get* and *set* each of the three *write-only* output signals (on the DSP port). (2p)

f) Create a finite state machine (FSM) graph describing the main system functionality. You may assume the keyboard subsystem provide the two "signals" `triplefail` and `correctpin`. You can also assume existence of the output "signal" `timerstart` and the input `timeout` from the timer subsystem. Make sure to specify the value on *all* four output signals (`diode`, `sound`, `police`, `timer_start`) in each state, and consider all events on the four input signals (`triplefail`, `correctpin`, `timeout`, `alarm`) for state changes. Events you do not specify are assumed NOT to change state. It is thus enough to specify state-changing transitions. (5p)

6. (4p)

A programmer have three implementations of a max function. They are outlined below with names in bold text. Discuss how each implementation affect *correctness*, *performance*, and *code size*. State the technical reason why each solution is good or bad. Which of the given solutions is best to use? How can it be improved?

```
/* this structure will occupy the size of 129 integers */
struct big {
  int hash;       /* this is used for comparisions */
  int big[128];   /* a lot of stored data */
};

#define MACRO_MAX(a, b) (((a).hash > (b).hash) ? (a) : (b))

struct big* max(struct big* a, struct big* b) {
  if (a->hash > b->hash)
    return a;
  else
    return b;
}

inline struct big inline_max(struct big a, struct big b) {
  if (a.hash > b.hash)
    return a;
  else
    return b;
}

/* two ways to generate a random big variable */
struct big get_random() { ... };
struct big* alloc_random() { ... };

int main()
{
  /* the three implementations in use */

  /* how the macro would be used */
  struct big maxi = MACRO_MAX(get_random(), get_random());

  /* how the normal function would be used */
  struct big* maxi = max(alloc_random(), alloc_random());

  /* how the inline function would be used */
  struct big maxi = inline_max(get_random(), get_random());
}
```