Försättsblad till skriftlig tentamen vid Linköpings Universitet

Cover page for written exam at Linköping University

Datum för tentamen Date of exam	**2011-01-08**
Sal Room	**TER2**
Tid Time	**14:00-18:00**
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	**8** 8 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** klaar@ida.liu.se
Telefon under skrivtid Phone during exam	**0705 - 35 95 42** 013 - 28 21 46
Besöker salen ca kl. Time of exam visit	**Endast tillgänglig per telefon (mobil)** Only available by phone (mobile)
Kursadministratör 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	*Precise, explained and clearly motivated assumptions, statements and reasoning raise the impression and are required for highest score.* ***Solve at most one assignment per sheet.*** ***If in doubt, state clearly your interpretation of the question, your assumptions, and answer according to that.*** Preliminary graded: U < 50% < 3 < 67% < 4 < 84% < 5 Grades may be raised or lowered based on overall impression. Results available within 10 working days.
Typ av papper Paper to use	**Rutigt, linjerat eller blankt** No preference
Antal anmälda Number of exams	**3**

The following example is used in the next two questions.

A home appliance company produce washing machines. They have many different models combining different features as capacity, top- or front-load, spin-dry speed, water-use, energy-use, detergent-use, noise, detection of unbalanced wash during spin-dry step to avoid a machine that jumps and moves over the floor, and of course the available washing programs (different combinations and timing of pre-wash, main-wash, spin-dry [sv: centrifugering], temperature and rinsing [sv: sköljning]). Before 1990 all their machines had mechanic timing and program selection and execution. This involved a lot of special-purpose plastic and metallic mechanic details, buttons and dials keyed in various ways according to the specific machine and the programs it should provide. Selecting correct program among all dials and buttons was not always a straightforward task for the user. Around 1990 the company started incorporating microcontrollers in their machines.

1. (3p)

a) Explain the benefits from machine *design* and *production* point of view when incorporating microcontrollers. Consider the benefits over the entire line of models, not one individual machine. (1p)

b) Explain the benefits from user point of view. (1p)

c) Explain the benefits from maintenance/update/repair point of view. (1p)

2. (2p)

The new line of machines in the above example can be classified as embedded systems. Explain carefully why, in this example, we classify them as embedded systems. Which of the characteristics of embedded systems apply?

3. (1p)

State the assembly operation(s) needed to perform *port-based* I/O. Then state the assembly operation(s) needed to perform *memory-mapped* I/O. Is either of the assembly operations you state of use to anything else? You can assume x86 instructions, or if you do not know the exact assembly operation, you can just explain what the operation(s) do(es) instead.

4. (3p)

Describe precisely a situation and scenario when two executing threads require mutual exclusion by means of a lock (or a semaphore).

5. (3p)

Explain in detail what an interrupt service routine is. When is it called, who calls it, what does it do, what must it do, what happen when it returns? You may choose to exemplify a specific routine (your choice) as your explanation.

6. (5p)

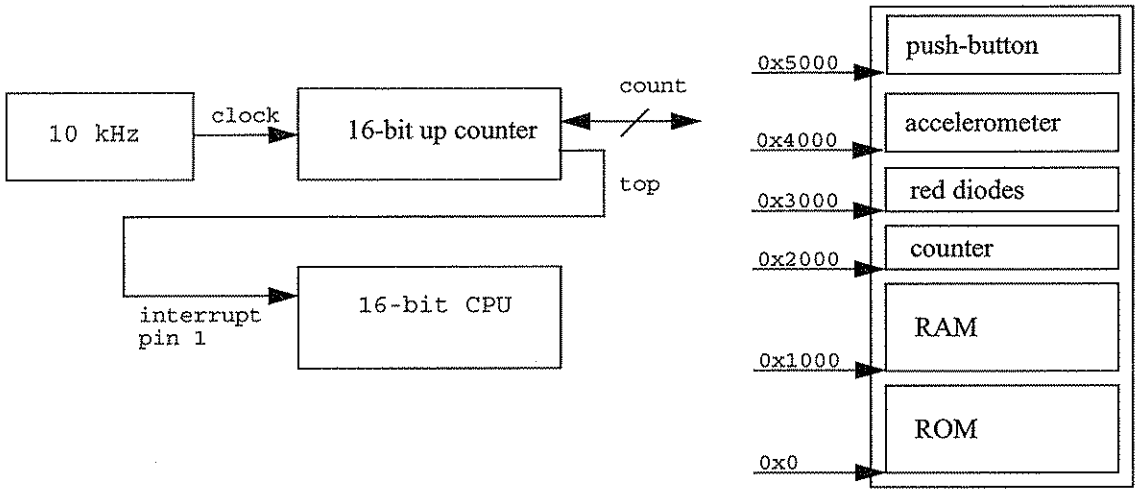
What benefits does a *general-purpose* controller implemented in FPGA (*programmable logic device*) provide over a *special-purpose* controller implemented as an *full-custom* ASIC (application specific integrated circuit)? (Consider technical benefits as well as time and money, and be specific: why is it better, where is any money and time saved?)

7. (20p)

A top-notch LED bicycle rear-light shall have the following features:

- 3 red LED diodes
- a battery pack
- one push-button (on / off)
- a motion sensor (accelerometer)
- automatically turn to blink mode after 10 seconds of immobility (if motion is detected during blink time it will go back to steady light)
- automatically turn off after 5 minutes of immobility (only a button-push can turn it back on)

A layout of the system and a memory map is shown below.



When the push-button is pressed a logic one is present at address 0x5000 (else logic zero).

By writing a logic 1 to address 0x4000 the accelerometer is requested to record any current movements and when done reset address 0x4000 to 0. The results are available as three 10-bit numbers (x, y and z) ***packed*** at addresses 0x4001 - 0x4004 (*first the 10 bits of x, then 10-bits of y, then 10 bits of z and last 2 unused bits*).

The red diodes is turned on by writing a logic one to the ***write-only*** address 0x3000, and turned off by writing 0. The 16-bit count register is available at address 0x2000.

a) Write an init function that initiate the counter to give next interrupt after 100ms, and request (only request) a reading from the accelerometer. (1p)

b) Implement a function that reads the accelerometer, set the variable named `motion` to true if any of x, y, z exceed 50 (values below 50 are considered noise), and request a new reading (you can assume the new reading is available before the function is called again). (4p)

c) Implement an interrupt handler that fires every 100ms and increment the global variable `seconds` every second. (4p)

d) Implement a function that looks at the global tri-state variable `light` (0 = off, 1 = blink and 2 = on) and turn the red diodes off, toggles them, or turn them on accordingly. (2p)

e) Draw a FSM (finite state machine) diagram that implements the functionality of the rear-light. Inputs will be the boolean flags `motion` and `push`, and the variable `seconds`. Output is the tri-state variable `light` (off/blink/on). Some states may reset the `seconds` variable. (6p)

f) Add code to put the pieces a) - e) together to a complete control program (you do not have to write code for the FSM, just assume it works according to your diagram). In blink-mode the diodes are turned on for 400ms and then off during 400ms and so on. (3p)

8. (3p)

An 8-bit processor have two registers for arithmetic operations, `areg1` and `areg2`. It also have the following assembly operations (the first operand is the destination):

```
mov [ADR] regX  ;; Move from register to memory
mov regX  [ADR] ;; Move from memory to register
add regX  regY  ;; Addition, regX = regX + regY
adc regX  regY  ;; Addition with carry
;; Example
DST EQU  0xA3F8   ;; Constant for destination base address
mov areg1 [DST+2] ;; Move third byte of DST to areg1
```

a) Base addresses A, B and C each store one 32-bit integer. Explain graphically how to perform the 32-bit addition C = A + B. (1p)

b) Base addresses A, B and C each store one 16-bit integer. Show assembly code to perform the 16-bit addition C = A + B. Assume that `areg1` and `areg2` are free, and that no other registers exist. (2p)