Verónica Gaspes
School of IDE

Embedded Systems Programming

Written Exam

April 30, 2011, from 09.00 to 13.00

- **Allowed tools:** An English dictionary.
- **Grading criteria:** You can get at most 20 points.

  To pass you need at least 50% of the points.

  For the highest grade you need more than 90% of the points.
- **Responsible:** Verónica Gaspes. Telephone number 7380.

- **Read carefully!** Some exercises might include explanations, hints and/or some code. What you have to do in each exercise is marked with the points that you can get for solving it (as **(X pts.)**).
- **Write clearly!**
- **Motivate your answers!**

Good Luck!

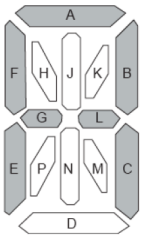

Figure 1: Digit segments in the LCD

Register Name	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
LCDDRx	K	–	–	A	K	-	-	A
LCDDRx+5	J	F	H	B	J	F	H	B
LCDDRx+10	L	E	G	C	L	E	G	C
LCDDRx+15	M	P	N	D	M	P	N	D

Figure 2: LCD data registers

1. In the first three laborations of this course we used a demonstration platform that includes an LCD. The LCD is controlled via the registers `LCDDR0` to `LCDDR19`, each of them one byte wide. Figure 1 shows what items can be turned on/off in each digit while Figure 2 shows what bits of the registers are used to control each of the items. In this exercise we are interested in the fact that four registers are used to control 2 positions.

   (3 pts.) Write a code fragment that shows the character A (the pattern shown with dark items in Figure 1) in the position controlled by the higher nibbles of the registers `LCDDR0`, `LCDDR5`, `LCDDR10` and `LCDDR15` and the character 5 (you have to decide what items have to be turned on) in the position controlled by the lower nibbles of the same registers.

2. Consider the following fragments we discussed in one of the lectures. The first two functions use busy waiting to detect and read input values from a sonar and a radio device. We do not show the code for the functions that implement ordinary algorithms (`control` and `decode`) and for generating output to the servo device.

```
int sonar_read(){
   while(SONAR_STATUS & READY == 0);
   return SONAR_DATA;
}

void radio_read(struct Packet *pkt){
  while(RADIO_STATUS & READY == 0);
  pkt->v1 = RADIO_DATA1;
  ...
  pkt->vn = RADIO_DATAn;
}

main(){
   struct Params params;
   struct Packet packet;
   int dist, signal;
   while(1){
     dist = sonar_read();
     control(dist, &signal, &params);
     servo_write(signal);

     radio_read(&packet);
     decode(&packet,&params);
   }
}
```

(3 pts.) Discuss some of the problems with this style of programming. Show some way of modifying the program so that one of the input sources does not shadow the other one. Are there some remaining problems?

3. In the course we discussed how to support concurrency with a kernel that can be used to spawn threads and that does automatic interleaving using *time slicing*: threads are placed in a queue and are given turns to use the processor for a given time slice each. Using this kernel a program can start several threads and we can think of a program as having *several mains* executing concurrently.

   (2 pts.) Organize the program from the previous exercise using the services of the kernel. Comment on any remaining issues that have to be addressed.

4. The following is the definition of a reactive object that can be used to produce a sequence of prime numbers.

```
#include "tinytimber.h"

typedef struct {
  Object super;
  int lastPrime;
} PrimeCalculator;

#define initPrimeCalculator() {initObject(),2}

int is_prime(int i){
  int n;
  if(i==0 || i==1) return 0;
  for(n=2; n<i; n++){
    if ((i%n)==0) return 0;
  }
  return 1;
}

int next(PrimeCalculator *self, int x) {
  int returnValue = self -> lastPrime++;
  while(!is_prime(self->lastPrime))self->lastPrime++;
  return returnValue;
}
```

   (a) **(2 pts.)** Show how to call the function `next` as an ordinary C function and and how to use it via `SYNC` and `ASYNC`. Explain the consequences of these different ways of using `next` in a TinyTimber program.

   (b) **(2 pts.)** Write a little TinyTimber application that uses a `PrimeCalculator` to print prime numbers on some device when a user presses a button. Provide the interfaces for the reactive objects that you need, but you do not need to provide implementations!

5. In laboration 4 you used a reactive object to implement a *blinker* that turns on and off a LED. The blinker can be started, stopped and the period can be set.

   (a) **(3 pts.)** Assuming that you have a class for reactive LEDs that can be turned on and off, implement a blinker. The blinker should not be hard-coupled to a LED, instead, it should be possible to instantiate blinkers for different LEDs.

   (b) **(1 pts.)** If your blinker is going to be used together with other reactive objects that require processor time (for example computing prime numbers!), what can you do in your implementation to make sure that the blinker keeps up with its frequency?

6. Say that three periodic tasks A, B and C have periods 80, 40, 20 (and deadline equal to the period).

   (a) **(1 pts.)** What does it mean that A has utilization 50%, B has utilization 25% and C 25%?

   (b) **(1 pts.)** Given these utilizations, what is the total utilization?

   (c) **(2 pts.)** Provide the timelines that result from scheduling according to rate monotonic (RM) and according to earliest deadline first (EDF).