Verónica Gaspes
School of IDE

Embedded Systems Programming

Written Exam

December 19th, 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!

1. In a 16-bit microprocessor with memory mapped I/O the control register `ADC_CONTROL_REG` for an ADC (Analog to Digital Converter) is organized as follows:

Bit	Name	Meaning
0	A/D start	Set to 1 to start a conversion.
6	Interrupt Enable/Disable	Set to 1 to enable interrupts.
7	Done	Set to 1 when conversion is complete.
8-13	Channel	The converter has 64 analog inputs, the particular one required is indicated by the value of the channel.
15	Error	Set to 1 by the converter if device malfunctions.

(2 pts.) Write a function

```
void startConversionAtChannel(int channelNr)
```

that sets the value of the control register `ADC_CONTROL_REG` to enable interrupts and start a conversion via the channel indicated by the argument.

2. Consider the embedded system illustrated in Figure 1 taken from the course book. In this simple embedded system T takes readings from a set of ther-

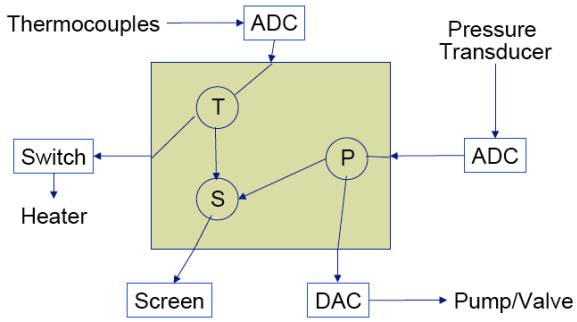

Figure 1: A simple embedded system.

mocouples (via an ADC) and makes appropriate changes to a heater (via a digitally controlled switch). P is doing a similar thing but for pressure. P and T communicate data to S which presents measurement data to an operator via a screen. The overall purpose of the system is to keep the temperature and pressure of some chemical process within defined limits. Someone has already written functions for some of the parts of the program:

```
int readTemperature(){
   while(! TEMP_READY );
   return TEMP_VALUE;
}

int readPressure(){
   while(! PRESSURE_READY );
   return PRESSURE_VALUE;
}
void writeToSwitch(int v){
   SWITCH_VAL = v;
}
void writeToPump(int v){
   PUMP_VAL = v;
}

void temperatureConvert(int temp, int* switchValue){
  // A function that calculates the value to write to
  // the switch given a temperature measurement.
}

void pressureConvert(int press, int* pumpValue){
  // A function that calculates the value to write to
  // the pump given a pressure measurement.
}
```

(4 pts.) Write a program for the complete embedded system assuming that you only have a C compiler: no language support for concurrency and no kernel that supports concurrency. You can print values on the screen using `printf`. Comment on the problems you addressed and what problems remain.

3. During the course you got to implement parts of Tinythreads, a kernel supporting concurrency. For the programs using the kernel, the important operation is `spawn` to start a thread. But, because now programs can have several threads sharing global variables there is need to support mutual exclusion. That is why the kernel provides a type for mutexes and the operations `lock` and `unlock`:

```
struct mutex_block {
    int locked;
    thread waitQ;
};
typedef struct mutex_block mutex;

#define MUTEX_INIT {0,0}
void lock(mutex *m);
void unlock(mutex *m);
```

(3 pts.) Define the operations `lock` and `unlock`.

Hints: You can use (without defining them!) the operations for manipulating queues:

```
void enqueue(thread p, thread *queue);
thread dequeue(thread *queue);
```

Recall that the thread that is executing is recorded in the global variable

```
thread current;
```

and that the threads that are ready to execute are kept in a global queue

```
thread readyQ;
```

4. Using Tinytimber you can organize programs with *reactive objects* while programming in C. As a programmer you have to follow some conventions and Tinytimber guarantees that the methods of a reactive object are executed strictly sequentially, thus protecting the local state of the object from critical section problems.

   (3 pts.) Program a class for reactive objects that can be used to *protect* (or encapsulate) a port. The port (a pointer to unsigned int) to be encapsulated can be provided on object initialization. Let the type introduced for the class be `Port`. Then, the methods that have to be provided are

```
// set the bits given by the argument mask
int set(Port *self, unsigned int mask)

// clear the bits given by the argument mask
int clear(Port *self, unsigned int mask)

// toggle the bit in position given by argument bitNr
int toggle(Port *self, int bitNr)
```

5. In laboration 3 you programmed Butterfly to produce sounds. Here are the interfaces of two reactive objects that could be part of the implementation:

Piezo:
A device driver for the piezo element.

```
typedef struct {
  Object super;
} Piezo;

#define CONFPIEZO DDRB|=(1<<PINB5);PORTB|=(1<<PINB5)
#define initPiezo() {initObject()}
int on(Piezo * self, int nothing);
int off(Piezo *self, int nothing);
```

Beeper:
Can be used to produce a sound with given frequency and duration.
The duration is given when calling `turnon`,
(just use an integer, the number of milliseconds)
The frequency can be set by providing half the period (the pulse width) to the method `setPW` (again just an integer)

```
typedef struct {
  Object super;
  Piezo * speaker;
  unsigned char running;
  unsigned char state;
  unsigned int pw; // half period
} Beeper;

#define initBeeper(speaker, pw) {initObject(),speaker,0,0,pw}
int oscillate(Beeper * self, int nothing);
int turnon(Beeper *self, int duration);
int turnoff(Beeper *self, int nothing);
int setPW(Beeper * self, int p);
```

(4 pts.) Program a reactive object that can be used as a ticking clock: it produces a short (100 ms) tone every second and a longer (300 ms) tone of another frequency every minute. The clock can be started and stopped. You can use the following abbreviations for the durations and pulse widths of the tones you need:

```
#define MINUTE_PW  18
#define SECOND_PW  35
#define MINUTE_DURATION  300
#define SECOND_DURATION  100
```

6. (a) **(1 pts.)** The documentation for Android recommends starting a (worker) thread when an Activity needs to do a time consuming task. Why?

   (b) **(1 pts.)** The documentation also explains that Services are used to do background tasks that do not need a user interface. When should you choose a worker thread and when a Service to do some time consuming task?

   (c) **(2 pts.)** How can an Activity start a Service and pass data to the Service? Give an example where an Activity starts a Service programmed in a class `AService` and passes a String to the Service. How can the Service access this data?