S. Feo-Arenis, D. Dietsch
M. Greitschus, Dr. B. Westphal

Sommersemester 2014

Formal Methods for C

`http://swt.informatik.uni-freiburg.de/ teaching/SS2014/FormalMethodsforC`

Exercise Sheet: Battery Monitoring Software

Contents

1 Battery Monitoring

Many battery-powered devices monitor the battery condition and indicate low power situations to the user such that a fresh battery can be inserted in time.
In this exercise, the software for a battery monitoring module needs to be developed. This software should be part of a larger firmware, which controls the overall device and calls the battery monitoring periodically. When called, the software is required to measure the battery voltage using a analog/digital (A/D) converter available on the device. If the measured value is below a given threshold, the software should switch on a light emitting diode (LED), and to switch it off if the voltage is above the threshold.
In addition, the software should react on signals from the device which indicate that the battery is removed or inserted.
As the overall device is battery powered, in particular the battery monitoring software must be energy efficient. Energy consumption of the battery monitoring task is dominated by the operation of the A/D converter. For the measurements, the A/D converter attaches a load to the battery, so each measurement is expensive in terms of energy. Furthermore, the A/D converter continuously consumes energy when it is ready for measurements.

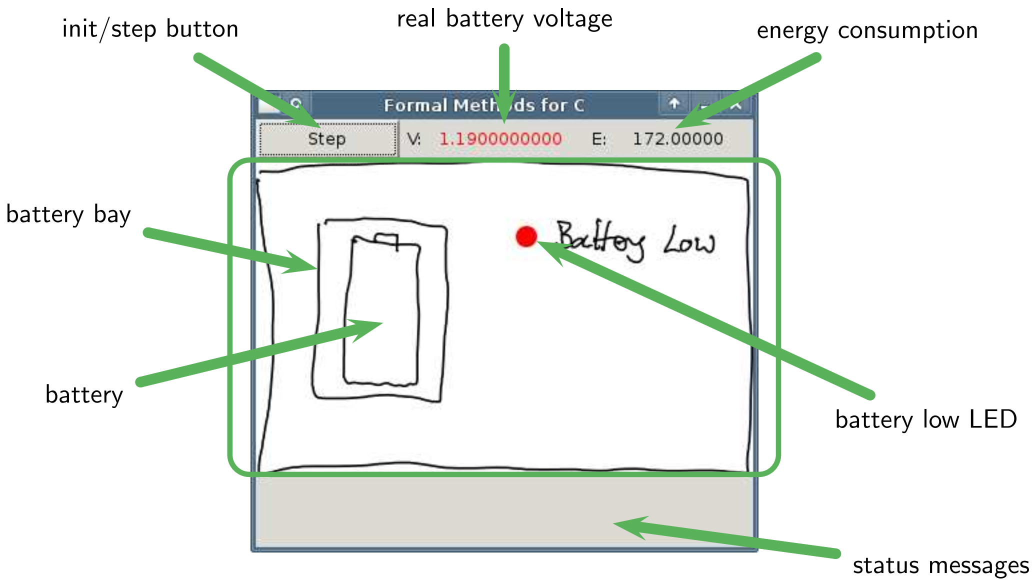

Figure 1: The prototype device.

2 The Prototype Platform

To test the battery monitoring software, a prototype device simulator is available (cf. Figure 1). The area with white background represents the panel of the final device. There is a LED which indicates whether the battery is low. The battery is placed in a battery bay and can freely be removed and inserted (in the simulator by clicking on the battery bay).
The gray areas above and below the white panel are for development purposes and specific to the prototype platform; they will not be present in the final device.
The button on the top left corner can be used to initialise and call the battery monitoring software, it emits either an INIT or a STEP event (see below). If there is no battery in the bay, the STEP function is disabled. The prototype device simulator also simulates battery drain. Battery drain is triggered by each button press, i.e. the battery does not change just by letting time pass. The condition of an inserted battery is chosen randomly. With a certain probability, a weak battery may be inserted.
The number in the middle of the top row (labelled with “V”) provides the *actual* battery voltage, which may me slightly different from the value reported by the A/D converter due to inaccuracies in the measurement. If the battery voltage is (strictly) below the threshold (1.21 V), the number is shown in red, otherwise in black.
The prototype device in particular allows to monitor the energy consumption of the battery monitoring software. The number at the right of the top row (labelled with “E”) shows the energy consumption of the last call to the battery monitoring software. The battery software is allocated at most 200 energy units per call. If the energy consumption is (strictly) above 200 energy units, the number is shown in red, and in black otherwise.
Furthermore, the prototype device simulator works as an observer for certain properties. First of all, it warns if the status of the LED does not match the condition of the battery. Secondly, it checks whether the battery monitoring software properly reported events consumed. Finally, it checks whether the interface to the LED is operated properly. If any of these checks fails, a corresponding message is displayed in the status message area below the panel.

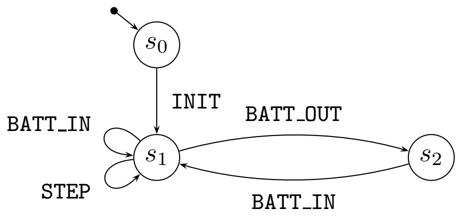

Figure 2: Behavioural interface to battery monitoring software.

3 Resources and Requirements Specification

You assume the role of a contractor which develops the battery monitoring software for the company which develops the overall device.

3.1 Resources

The company provides you with an archive containing the following files:

`ad.h` this header declares the interface to the A/D converter to be used by the battery monitoring software.

The A/D converter is calibrated to yield the integer value 20,000 if the battery voltage equals 1.0 V. The measurement is perfectly linear, i.e. a voltage value of 0.5 V is measured as 10,000 by the A/D converter (modulo noise, cf. `ad.h`).

`controller.h` this header declares the interface between the overall device and the battery monitoring software

`device.a` this library contains the overall device software including the A/D converter functionality and the prototype device simulator; for intellectual property reasons, the company does not provide you with the source code but only this binary library

`Makefile` if your battery monitoring software is called `controller.c`, you can directly use this makefile to compile your module and link it with the provided library in order to obtain an executable

3.2 Static Interface

The static interface between the overall device and the battery monitoring software consists of (cf. `controller.h`)

- `cmd_port`, an unsigned int variable with external linkage whose second bit controls the LED,
- the four events `INIT`, `STEP`, `BATT_IN`, and `BATT_OUT` that the overall device software may pass to the battery monitoring software
- `takeEvent` is to be implemented by you, this function is called by the overall device software with one of the four events as a parameter.

3.3 Behavioural Interface

The behavioural interface between the overall device and the battery monitoring software is specified by the company using the protocol state machine shown in Figure 2.
On power on, the battery monitoring software should be in a configuration corresponding to state s_0. The overall software issues an `INIT` event when the battery monitoring software is supposed to initialise itself and in particular switch the LED off.

After the `INIT` event, the overall software issues either a `BATT_IN` or a `BATT_OUT` event to indicate whether a battery is present or not, respectively.
After initialisation, the overall software periodically issues a `STEP` event. On `STEP`, the battery monitoring software should conduct a measurement using the A/D converter and set the LED accordingly.
Furthermore, after initialisation, each removal of the battery is indicated by a `BATT_OUT` event; whenever a battery is inserted, a `BATT_IN` event is issued. If there is no battery in the bay, the LED should be off. On a `BATT_IN` event, the battery monitoring software should also conduct a measurement and set the LED accordingly as users may insert weak batteries.
Handling one event should not consume more than 200 energy units. The overall device software observes energy consumption by a (hidden) interface to the A/D converter.
The battery monitoring software need not be prepared to consume `STEP` or `BATT_OUT` events when its configuration corresponds to state s_2 because the overall device software ensures that they are not issued in that state.

4 The Task

1. Formalise the requirements using your favourite formalism (state invariants, pre/post conditions, ...).
2. Implement a battery monitoring software which runs with the provided resources.

   Hint: which of the requirements can you annotate to the code using assertions from `assert.h`?
3. Test your implementation thoroughly.

   Hint: do you need to do "GUI testing", i.e. test via the GUI of the prototype device simulator or can you think of automated tests in the style of unit tests?
4. Try to verify your implementation against your requirements using your chosen verification tool.