KU LEUVEN
DEPARTMENT OF COMPUTER SCIENCE

Software for Real-Time and Embedded Systems Project

Hoobergs–Kleynen

Academic Year 2018 – 2019

H04L2a: Software for Real-Time and Embedded Systems

Jesse Hoobergs (r0588750)
Bo Kleynen (r0624034)

1. Problem statement

The goal of this project was to create a simplified version of an airbag system on an Arduino Leonardo using FreeRTOS. The system has to do the following:

- Log calibrated temperature data to the internal EEPROM of the micro-controller resembling the logging of vehicle data. This should be done every 500ms.
- Handle certain user input:
  1. Printing the last entry in the database.
  2. Going into a low power mode. In this mode the temperature logging is disabled.
  3. Printing all records in the database.
- React on a high input on one of the pins by setting a high signal on another pin simulating airbag deployment. This task has to respond withing 144ms to 166ms, with a target of 150ms.
- Ensure database consistency by using synchronisation primitives.

2. Solution

2.1 Overview

A timer is used to write temperature data every 500ms. Timer 1 is configured in CTC mode to execute it's interrupt service routine (ISR) when the counter reaches a certain value. The prescaler is set to the maximum value of 1024 and this means that the timer should interrupt when it reaches the value 7812. This value is calculated as 500ms * 16 000 000 Hz / 1024. Timer 0 can't be used because it's counter only has 10 bits [1], which can't represent the value 7812. Timer 1 has a 16 bit counter so it can be used.

A database class that manages the communication with the EEPROM is created. It provides a clean API to the rest of the program, hiding all the technicalities of synchronisation and task creation regarding the EEPROM from the rest of the program. To save storage space the temperatures are stored as their raw 10 bit reading. These values are converted to actual temperature values when a command to read them is issued. Besides these records the database also stores the amount of records written (*nRecords*) and the index of the last written record (*head*), both are also kept in memory. Limiting the amount of records to 256 has some benefits. This way the head can be stored as a byte without the need of additional bookkeeping or operations to reset it, as overflow will simply result in resetting to 0. Using a byte also results in only one write cycle each time head is written to EEPROM, what happens every time a record is stored.

Pin 3 is the input pin for the airbag deployment because it is the fastest to trigger it's ISR. The output pin is 12. Additionally a database reset and an extra wake-up pin are provided. The database reset is on pin 5, setting it to high resets the database head and nRecords. The system can be brought out of low power mode without triggering the airbag deployment by a rising edge on the wake-up pin, pin 2.

2.2 Real-Time constraint

During start up a task with highest priority is created, this way it will always execute when it becomes available. This task will immediately suspend itself. When the interrupt is triggered by a rising edge the ISR will resume it and yield to that task. This would however result in a response that is nearly instantaneous, so to meet the real time constraint the task will delay for 10 ticks (150ms) before setting the output to high. This delay is implemented as a non busy wait so other tasks can execute in the mean time. As is usual for a real airbag this task will only execute once, at the end the ISR is detached and the task is deleted.

2.3 Task Synchronisation

Database inconsistencies can occur when two or more processes (tasks) access the database concurrently and at least one of them is a write. There are several possibilities to accomplish this: a semaphore guarding the database, with multiple tasks trying to acquire it; a single task managing the database or two tasks, one managing the reads and one managing the writes with a mutex to synchronise them.

The last approach was chosen. Values to be written are sent to this task through a queue. A queue is an inter task communication primitive that can act as first in, first out. This way the records are written in the order in which they were measured.

The mutex ensures consistency between the database and the serial output when printing all records. Before starting to fetch records, the mutex has to be acquired and held until the last record is read. Similarly when writing, the mutex has to be acquired before the record is written and held until the head and the nRecords field are written. When printing only the last value, however, it isn't necessary to acquire the mutex, since the task responsible for writing to the EEPROM has a higher priority than the task to read from the EEPROM and simultaneous reading and writing is not possible.

2.4 Low power mode

The application goes to sleep in two situations: when there are no tasks to execute and when it is explicitly asked by sending a '2' through the serial communication. In the first case, the *Idle* sleep mode is used because waking up from this sleep mode is easy. This mode is activated every time the end of the loop method (the idle task) is reached. In the second case the *Power-save* sleep mode is used to consume as least power as possible. The Watchdog timer is disabled so it doesn't wake up the device. The analog comparator and the digital input buffers for ADC 13..4 and AC1..0 are also disabled to reduce power usage. The device wakes up when an interrupt happens on pin 2 (wake up) or on pin 3 (collision). On waking up, the Watchdog timer and timer 1 are re-enabled.

To further reduce power, timer 3 and 4 are always disabled because they are never used.

2.5 Testing results

In the testing setup the input pin is connected to one probe of an oscilloscope and the output pin to another probe of the oscilloscope, a switch is used as a trigger. The response time of the system was slightly below the targeted 150ms when in normal operation mode and slightly above it when in low power mode. In both case the response time was between the 144ms and 166ms boundaries.

To measure the power consumption of the system, the V_{in} and Gnd pins of the Arduino were connected to a variable DC power supply, with integrated current measuring. At 6.00V the system used 7mA of current which results in a power consumption of 42mW. The same setup was used to measure the current drawn while in normal operation mode. At 6.00V this resulted in 27mA, which corresponds to a power consumption of 162mW.

2.6 Possible improvements

Due to time pressure some ideas weren't tried out. Instead of using the *Idle* sleep mode, the *Standby* sleep mode could probably be used when there are no tasks to execute to reduce the power usage even more. It might also be a good idea to use the *ADC Noise Reduction Mode* to get better temperature readings.

Bibliography

[1] ATmega16U4/ATmega32U4 Datasheet, http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf.