University Polytechnica of Bucharest
Faculty of Automatic Control and Computer Science
2018

Microcontroller Subsystem Software Requirements

SW_uC_Reqs

DRAFT

Author:
TEODORESCU OCTAV-IULIAN

Table of Contents

I. Document purpose 3

II. The software of the uC subsystem 3

III. Software requirements 3

IV. References 5

V. Document History 5

VI. Other 5

I. Document purpose

This document provides a general overview of the software layer of the uC subsystem. Section II briefly describes what the SW is intended to do. Section III contains the detailed requirements that will be enforced: during later project phases these requirements will serve as guidelines, for example helping decision making during design-phase and setting certain standards during the implementation. Section IV contains relevant documentation. Sections V and VI are reserved for document management purposes.

II. The software of the uC subsystem

The uC subsystem is an intermediary layer between the PC and the other hardware components. This subsystem has been designed to:

- facilitate sensor data acquisition
  The program running on the uC subsystem will acquire data from the sensors during a training cycle or maintenance tests.

- facilitate control of peripheral devices
  The elements which provide visual feedback to the user will be automatically controlled from this program.

- (optional) contain routines to process acquired data

III. Software requirements

ID	Category	Description	Reasoning
swucr1	General	The C language shall be used.	This language is a natural choice for the project because it guartantees speed and memory performance on target system and provides quick low-level access. The multitude of open libraries for the chosen platform also facilitates development.
swucr2	Functionality	The program shall offer three operation modes: auto, tethered and maintenance. In auto mode the subsystem shall run its tasks independent of the PC. In tethered mode the subsystem	These modes have been designed in order to give the operator the possibility of using the system without a connected PC, and to separate the update/debug/processes from the normal operation modes. It is a clean solution to the problem of multifunctionality.

		shall be able to communicate with the PC. In maintenance mode the subsystem shall offer additional debugging, updating and calibration functionalities. It must communicate with the PC. This mode may be used as a tool for remote-upgrading of the firmware.	
swucr3	Functionality	In tethered and maintenance modes the program shall communicate with the PC via USB in order to receive commands, transfer data. In maintenance mode the program may download firmware from the PC.	USB communication (especially 3.0+) is more than enough to guarantee a speedy communication channel.
swucr4	Functionality	The program shall automatically control the peripheral devices. The user shall not have direct control of peripheral devices.	[At the moment of this document – ver. DRAFT – the only peripherals to be controlled are 220V lightbulbs and a MUX.]
swucr5	Functionality	In auto mode the program may store acquired data (up to 24kB) in the flash memory of the board.	Normally the data should constantly be processed and sent to the PC or processed and lost. However in auto mode, the user may want to store some of the data for later. The Arduino board does not offer a large memory space. For later prototypes an external memory may be added.
swucr6	Functionality/ Performance	In tethered mode the program shall acquire sensor data and send it to the PC. The data will not be stored. The data may be processed on the spot or transmitted in raw format to the PC, the results being processed by the PC. In maintenance mode data may be sent, received or stored.	This is a big question that requires testing to answer: which option is faster – processing the sensor data on-board or having it processed by the PC? If the serial communication is fast then it would be a better idea to have this solved on the PC-side.
swucr7	Design	The program may use protothreads in order to facilitate development.	Arduino does not support multithreading. Protothreads offer an alternative to the finite-state machine approach. [However tests need to be run to determine if this approach would significantly affect performance]
swucr8	Performance	The firmware should not be larger than 1.6kB.	Arduino SRAM is 2kB large. To guarantee speed performance the firmware will have to be optimized for size.
swucr9	Design	The program shall be split into	For example one such configuration may

		modules according to their functionality. The modules will be designed as to allow independent testing.	be: a module for serial communication with the PC, a peripheral control module, calculation module etc. The details will be specified in the design and are not to be enforced in this document, however the priority is having each module independent enough to allow separate testing and debugging.
swucr9	Performance	Algorithms should be optimized for speed.	Given the hw limitations, solutions should be found to ensure maximum efficiency of resources. However this is not mandatory. The mandatory requirements guarantee the desired performance; this would be an improvement.
swucr10	Functionality	The program shall perform hw and sw tests for consistency and fault-prevention.	To guarante maximum stability and robustness the program will have one or more of these features: functionality self-tests, boot-tests, memory tests, communication tests. This is critical in a system where peripherals may be damaged by user interaction and when a communication or memory fault could result in completely incorrect results.

IV. References

https://hackaday.com/2013/06/03/benchmarking-usb-transfer-speeds/
https://www.hackster.io/survey
https://code.google.com/archive/p/arduino-class/wikis/ProtoThreads.wiki

V. Document History

Date	Status/Version	Comments
12/9/2017	Awaiting Approval/DRAFT	

VI. Other

Abbreviations: HW – hardware, SW – software, uC - microcontroller