AVR Temperature Monitor

Paolo Lucchesi

April 13, 2020

Contents

1 Introduction

avrtmon (acronym for *AVR Temperature Monitor*) is a software/hardware device based on an Atmel ATmega2560 AVR board, which allows to measure the external temperature at regular, configurable intervals. The temperatures are stored in the board's internal EEPROM and can be retrieved by the user with an ad-hoc program, which is compatible with POSIX-compliant systems. Also, many other parameters, such as the pins to which the tactile buttons are linked, are configurable.

Both the AVR and PC programs have been developed as a set of self-contained, independent modules which can be reused out-of-the-box for other purposes.

From now on, we will refer to the AVR board as "the *tmon*", to the stuff related to the tmon as "the *AVR-side* stuff" and to the stuff related to the PC as "the *host-side* stuff".

1.1 How to use

The project uses *GNU Make* as the main build system. Also, the host-side program is shipped with a manual page[1]; once installed, you can read it with `man avrtmon`.

The tmon configuration structure is defined in a CSV file in the form:

```
field-name,type,default-value
```

The relative source files can be dynamically generated with the *config-gen* utility shipped in the repo.

You can refer to table 1 for practical use of implemented *make* commands and recipes.

1.2 Testing

The produced code (in particular the host-side one) had been extensively tested with an own-developed test framework. Ordinary tests are performed with AVR-side parameters and executed at host-side; some tests, however, are host-specific. Others are AVR-specific and must be performed compiling and flashing a mock firmware into the AVR board.

1.3 Technical and Architectural notes

The codebase is in common for host-side and AVR-side programs, and these ones share the same makefile. To perform host-related tasks with GNU Make, the environment variable *ARCH* shall be defined as *host* (e.g. you can do `export ARCH=host`), even if there are wrapper recipes in the makefile to handle the most common tasks without those hacky ways. By default, if *ARCH* is unset or different from *host*, AVR-side tasks will be performed[2].

A very strict policy have been followed for module dependencies across the entire codebase; indeed, very few modules are cross-dependent.

[1]Generating the man page requires *pandoc* installed
[2]For example, *avr-gcc* instead of *gcc* will be used

Table 1: GNU Make rules and recipes legend

Command	Description
`make`	Compile and link both host-side and AVR-side .elf executable
`make host`	Compile and link the host-side executable
`make avr`	Compile and link the AVR-side .elf executable
`make install`	Install the host-side executable under `/usr/bin`
`make flash`	Flash the AVR-side .hex executable into the AVR board
`make docs`	Generate the UNIX *man* page
`make install-docs`	Install the man page under `/usr/share/man`
`make config-gen`	Generate configuration from `resources/config/default.csv`
`make test`	Perform all the standard tests
`make test-*`	Perform a single standard test unit
`make avr-test-*`	Perform a single AVR-specific test unit
`make host-test-*`	Perform a single host-specific test unit

Function pointers have been widely used: for example, tactile buttons are configured to execute a user-defined callback, the entire AVR-side executable command system is based on them, as well as the host-side shell module.

2 AVR-side

2.1 Peripherals and Schematics

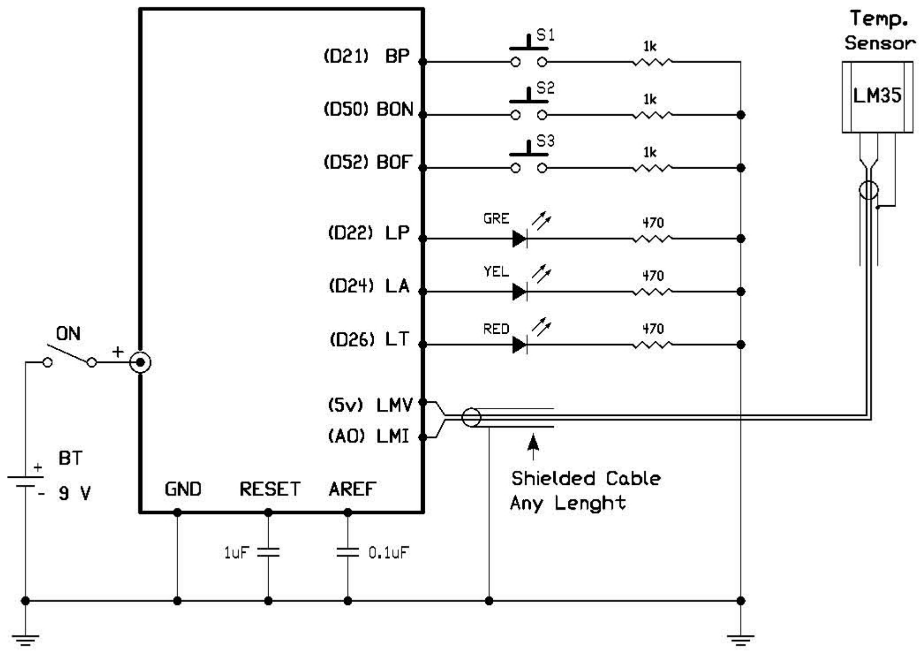
For this device, the following embedded and external peripherals have been used:

- 3 embedded timers:
  - One for packet communication Retransmission TimeOut
  - One to track temperature registering intervals
  - One for the buttons debouncer
- The USART hardlinked to the USB cable (i.e. *USART0*)
- The embedded EEPROM to store the configuration and the temperatures
- 3 external LEDs (each with a 470Ω resistor):
  - One to signal that the tmon is turned on (on pin *LON*)
  - One to signal that the tmon is registering temperatures (on pin *LT*)
  - One to signal that the AVR MCU is actually working actively (on pin *LAC*)
- 3 external tactile buttons (each with a $1k\Omega$ resistor):
  - One to start registering temperatures (on pin *BON*)
  - One to stop registering temperatures (on pin *BOF*)
  - One to turn the tmon ON and OFF (on pin *BP*)

- 2 external capacitors:
  - One to reduce the ADC noise ($0.1\mu F$, on pin *AREF*)
  - One to prevent the tmon from resetting every time the USB cable is plugged in ($1\mu F$, on pin *RESET*)

The bindings between the sensor schematics and the AVR board pins are dynamically configurable in software (see section 2.2.2)

Figure 1: Circuit schematics



2.1.1 Printed Circuit Board

A PCB containing all the required hardware had been realized; it had been designed to stick over the AVR board. To use that, the pin bindings must be the ones specified in table 2. The PCB schematics are shown in figure 2.

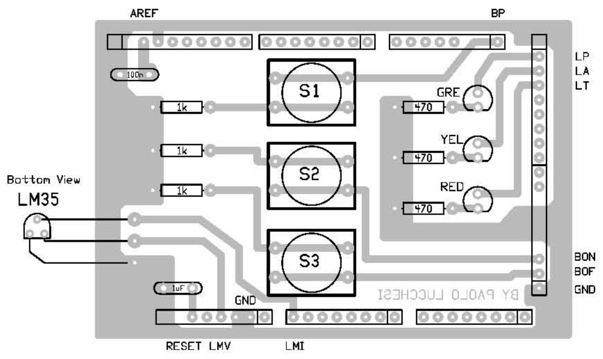


Figure 2: PCB schematics

PCB	AVR
BP	D21
BON	D50
BOF	D52
LON	D22
LAC	D24
LT	D26

Table 2: PCB pin bindings

2.2 Firmware

The tmon firmware is almost completely interrupt-based (including the USART module, for both reception and transmission). The general idea behind the architecture is that every event is notified via interrupts, and the program deals with those events later using *handler functions*.

2.2.1 Power saving

The program follows a strict power-saving policy; the AVR MCU is in idle sleep mode the most of the time, waking up only when an action is effectively to be performed, and the tmon can be even be put in power-down sleep mode on-demand. Moreover, any unneeded module (e.g. unused timers and USARTs, the TWI, the SPI, the JTAG and the Watchdog Timer) is deactivated.

2.2.2 Direct user interaction

The pins to be dedicated to tactile buttons are software-configurable: digital pins [50,53] and 21 can be used. Those pins must be specified in the CSV configuration as *Dxy* (e.g. *D50* for digital pin 50). Moreover, the buttons module is shipped with a software, non-blocking, interrupt and timer based debouncer with configurable debounce interval.

2.2.3 Temperature registering

A 16-bit timer is used to trigger an ADC convertion to sample a temperature; to raise the maximum interval much higher (the maximum interval of that timer would be around 4 seconds), two 16-bit unsigned integer variables are used (one stores the *resolution* of the timer expressed in milliseconds, the other the *interval* expressed in resolution units). This way, the maximum registration interval value raises to:

$$\frac{(2^{16}-1)^2}{15625} \approx 274869 \text{ seconds} \approx 72 \text{ hours}$$

Moreover, when an ADC convertion is performed, the AVR board enters in *ADC Noise Reduction* sleep mode, in order to increase the measurement quality.

The tmon can store and handle multiple databases; in fact, a new one is created every time temperature registering is stopped and then restarted. Different databases can have different registration intervals.

2.2.4 Interaction with host

The firmware allows to remotely execute arbitrary commands from the host-side (via the communication module). This system is designed to be very resilient to changes: a command is essentially represented by a function pointer and some metadata, so one can easily add, remove or modify commands, and even build commands that are self-contained in separated source files. Below a list of remotely executable commands is given:

config_get_field Get the value of a given configuration field

config_set_field Set the value of a given configuration field

start Start registering temperatures (like pressing start button)

stop Stop registering temperatures (like pressing stop button)

temperatures_set_resolution Set registering timer resolution until the tmon reboots

temperatures_set_interval Set registering timer interval until the tmon reboots

temperatures_download Download all the temperature databases stored in the EEPROM

temperatures_reset Delete all the registered temperatures

echo Send the received message back to the host (implemented essentially for debug)

To offer maximum responsiveness, the communication module is the only one which contains some sort of busy-wait facilities (indeed, if data is available on the serial port, the communication handler won't return until a packet is received, or until a reception is failed).

3 Host-side Program

An ad-hoc program can be used to communicate and retrieve temperatures from the tmon, as well as to alter its behaviour.

3.0.1 User interaction

For user interaction, a POSIX-inspired shell module had been written; among other things, it supports non-interactive scripts execution. The commands are basically identified by:

- a name
- a brief description, which can be displayed using the `help` command
- a function pointer, which is the command action itself

3.0.2 Serial module

The host-side program contains its own serial module, which makes heavy use of the *termios* UNIX interface. This module use a standalone thread to continuously attempt to read the serial port, storing the read data in a race-protected circular buffer.

3.0.3 Temperatures

The temperatures are stored in a singly linked list (which I have developed myself) containing multiple databases; each database, once filled, is conceptually considered to be *read-only* (i.e. it should be deleted but not modified), hence this behaviour is not enforced. Each database is identified by a session-unique, incremental ID.

4 Communication protocol

The tmon and the host use an ad-hoc, packet-based communication protocol to talk to each other. The sanity of each packet header is checked with a parity bit, and the sanity of each entire packet is checked with the CRC-8 algorithm.

4.1 Serial communication

The serial port is used in non-canonical mode; data is sent and received as-is, with no ASCII character interpreted in any special way. No hardware control flow is used, and basic software control flow is provided by the packet communication protocol. The baud rate used is 57600 baud/sec.

When a packet is sent, the sender must wait for an acknowledgement response from the receiver.

4.2 Packets

A packet is composed of a 2-bytes header (described in table 3) and a variable length data body. Each packet is shipped with a trailing CRC. The size of each packet is limited to 31 bytes to decrease the probability of well-known errors related to serial communication (e.g. data overrun).

Table 3: Composition of a packet header

Field	Width (bits)	Description
ID	4	Packet ID - Must be incremental
Type	4	Packet type
Size	7	Size in bytes of the whole packet, including header and CRC
Parity	1	Header even parity bit

Multiple packet types (see table 4) are used in the communication protocol. According to the type, some the constraints on the ID are the following:

- For *HND* packets, the ID must be 0
- For *ACK* and *ERR* packets, the ID must match the one of the previously received packet
- For every other packet type, the ID must be the same of the previously received packet, incremented by 1

4.2.1 Sanity check

The sanity of a packet header is checked via even parity bit; if a header is corrupted, the packet is discarded immediately. The sanity of a whole packet is checked via trailing CRC8 value; the algorithm is partly parameterized, so different polynomials can be used (the default CRC polynomial is *0x07*, and for that the division rounds are optimized with a lookup table). Moreover, the packet implementation is designed to easily substitute the CRC8 with a stronger integrity check algorithm (e.g. CRC16).

Table 4: Packet types

Type	C enum code	Description
HND	0x00	Handshake
ACK	0x01	Acknowledgement
ERR	0x02	Communication error
CMD	0x03	Execute command
CTR	0x04	Control sequence used in command communications
DAT	0x05	Data packet

4.2.2 Error Recovery Techniques

To recover from errors, some techniques are adopted:

- A *Retransmission TimeOut* (*RTO*) is used on both sides
- When a packet exchange fails, both sides flush their serial buffers and wait for the RTO to elapse
- When too many consecutive packet delivery failures happen, both sides reset the packet ID (i.e. ID of next packet will be 0)