Distributed Measurement of Temperatures for the HADES-MDC-Detector

Giacomo Ortona
Universitá di Torino, v. Pietro Giuria 1, 10125 Torino, Italy
ortona@studenti.ph.unito.it

Implementation of a new system for measure the temperature of the MDCs in the HADES experiment: the system is based on the 1-Wire© protocol provided by Dallas Semiconductors.

1 Introduction

The HADES (**H**igh–**A**cceptance–**Di**–**E**lectron–**S**pectrometer) experiment is measuring electron-positron pairs in order to investigate the properties of hadronic matter at high temperatures and densities. To measure direction, position and with the help of the HADES magnet, the momentum of particles, 4 MDCs (**M**ultiwire **D**rift **C**hambers), are used. When a particle passes through the gas of a MDC, it ionizes gas particles which drift, due to an electric field, to the wires passing through the chamber. In every chamber there are 6 layers of sense wires. From the fired wires it is possible to calculate the crossing point of the particle, but to do this very accurate it is needed to know the temperature of the MDC, since the drift speed of the ions is influenced by this parameter.

The core of the system to measure temperatures is a new multipurpose board (fig.1) as described in section **??**. The goal is to have an easy maintainable system, thus smart devices – Dallas DS1820 sensors (2.2) – are used to measure the temperature. They can be placed in several critical points, and provided a logical unit that can perform analog to digital conversion. Therefore only one bus for the data transmission is needed, and on that bus many sensors can be connected using the 1-Wire© protocol (2.3), instead of using a couple of cables for each sensors to read the analogical signal of the thermocouple.

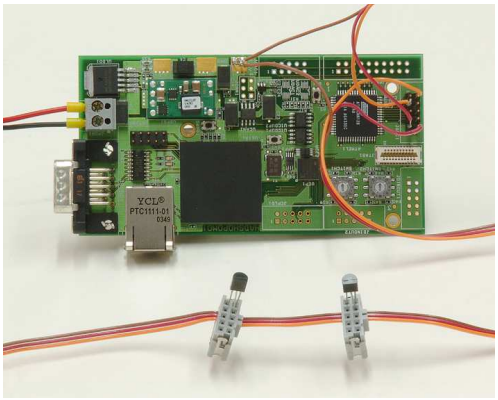

Fig. 1: *HADSHOPOMO1 board. The big black square in the center is the Linux processor, on the wire 2 devices, on the board next to the wire connection the microcontroller.*

2 Hardware

2.1 Board

The board built for the temperature control, called HADSHOPOMO1, is a multipurpose board, and so not all the features on it are used for this project.
On the board are placed a microcontroller (AT90CAN128), a Linux processor, an external connector for 100 MBit Ethernet and a serial port.
In the project is mainly used the 8–bits AVR microcontroller AT90CAN128, manufactured by ATMEL© , which has 128KB of programmable Flash memory. This microcontroller can do several actions, in the specific case is used to send commands to the devices placed on the 1-Wire bus(2.3) and to store the data received on the

same bus. This data are then sent to the Linux computer trough a special serial communication chip called USART (**U**niversal **S**ynchronous and **A**synchronous **s**erial **R**eceiver and **T**ransmitter) which is highly flexible and can perform at the same time the receiving and transmittance of data.

From the computer on the board the data are then forwarded to a server using a TCP protocol.

The board has to be supplied by an external voltage of 6 – 9V but the internal voltage provided to the sensor is 3.3V.

2.2 Temperature Sensors

The temperature sensors used in the project are DS1820 temperature sensors by Dallas. Each one of them has a unique 64-bits identity code which allows to address every single device. In this way it is possible to have on a single bus as many devices as the electronics can support. The devices contain the logic needed to perform automatic conversion from analog to digital temperature data when a specific instruction is received, and control the operations the devices can perform. Every device has a 8–bytes memory in which the values of the last temperature conversion are stored; the configuration register and the user defined limits of temperature, as one feature of the devices is the possibility to set one upper and one lower temperature limit. Out of those limits a special bit (alarm flag) is turned to 1.

The devices are working between –55°C and 125°C and have an accuracy of ±0.5°C between –10°C and 85°C. The resolution of the measure can be set from a minimum of ±0.5°C to a maximum of ±0.0625°C. However, the time needed to perform a conversion is strongly affected by the resolution – a high resolution conversion can last up to 750μs –.
The devices have 3 pins, one for the communication bus, one for the ground level and one for the power supply.

2.3 1-Wire© Protocol

The 1-Wire© protocol allows to use a single data line for all the communication between a master (here: the AT90CAN128 microcontroller) and the slaves (the DS1820 devices). In the idle state a resistor pulls up the line to 3.3V. The communication works on a very precise time–slots protocol; the master can issue write slots for writing 0 (line down) and 1 (line up), or reading slots. Any of this slots has to last at least 60 μs.

The communication starts with a reset pulse (line pulled down by the master for more than 480 μs) followed by a master command (e.g. to identify the devices the master is referring to) and they end with a command triggering a device operations, for example to start a conversion or to send the data to the master. The set of commands is defined by Dallas.

3 Software

3.1 Functionality

The software is divided in 3 different parts, each one running on different machines. The main part is running on the microcontroller and performs the temperature reading, data conversion into a string and transmission of these informations to the Linux processor on the board. On this Linux processor is running the second part of the code. It acts like a bridge, sending the data received via serial transmission from the microcontroller to the server via Ethernet and vice versa. The third piece of software is running on the server and simply receives the data sent by the Linux processor.

To make use of the functionality, the board will be integrated into the HADES slow control system, built on basis of EPICS [11] (**E**xperimental **P**hysics and **I**ndustrial **C**ontrol **S**ystem). Through it is possible to control these temperature sensors, for example turning them on or off; setting parameters like thresholds and adjusting sensibility.

The code written in C cannot be compiled directly on the microcontroller, so a cross compiler is needed: this special compiler creates executable code for a platform which is not the one on which the compiler runs. Once the code is compiled the binary output of the compiler can be transferred to the microcontroller's flash memory. Atmel© provides a special device to perform this operation.

The last step in the development process is the debugging of the code. For this one uses special functions which have to be implemented in addition as well as external programs like a software debugger.

After the code is tested it is stored in a version management system (e.g. CVS –**C**oncurrent **V**ersion **S**ystem). So it is possible to trace the code evolution while the project goes on.

3.2 Code

The code consists of 3 files:

Tempsens2.c implements the code running on the microcontroller;

serialtrx.c is a program running on the Linux processor;

sockettcp.c contains the functions executed on the server server.

3.2.1 tempsens.c

The most important software part of the project is the one running on the microcontroller.

First, this code initializes the 1–Wire© buses and the USART component (2.1) of the AT90CAN128 to allow communication with the embedded Linux PC on the board.

After a successful initialization, the MCU (**M**icro**C**ontroller **U**nit) program waits for a function call by the Linux processor – represented by a character. Such function can be:

- temperature detection;
- search for alarm conditions;
- setup of temperature limits;

To perform the temperature measurement, first of all the master must identify every device on the bus by storing its ID number. This is done by calling a special function, provided by the manufacturer of the sensors. The function works asking all the devices to send their ID number one bit a time. Every time it reads a bit, the function will check if there are any errors and if there is still more then one device active on the bus. If not, the master will select all the devices that putted the same bit, idle all the other, and store the information that there are still devices not recognized at this point. This is done for each one of the 64 bits of the ID.
Since this function identifies just one device, it is necessary to loop over the defined maximum number of devices on the bus, giving as argument the bit position where there still was more then one device active.

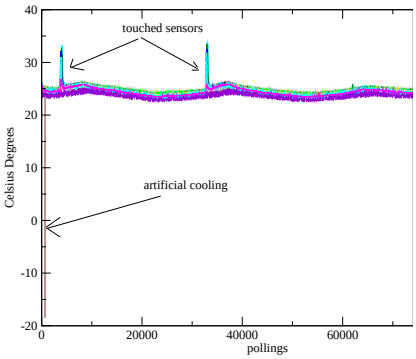

Fig. 2: *27 devices, running for 2 days data acquisition. At the beginning one of the sensors is cooled down to about –20° C. On x axis is shown the progressive number of polling.*

After that, the master will trigger a simultaneous temperature conversion on all devices and wait until they have finished. Then it will read out the results of the conversion from each sensor – one by one. This operation is done by sending a special command followed by an ID number of the devices being read out. As an effect of this command, all devices which do not match the ID will turn into an idle state until a reset pulse is send via the bus.
Each time the temperature is read, the value and the ID number of the respective sensor is stored in a string, which is then send character by character through the USART interface to the Linux processor.
Looking up every time the device's ID – instead of reading it just once during initialization – allows the program to react flexibly on changes in the number of devices on the bus.

Until now the program has been tested with 37 devices, placed on two different buses. Figure 2 and 3 show temperature data collected over 2 days.

The search for an alarm condition works exactly in the same way; the only difference is that just the devices which measured during the last conversion a temperature higher or lower than the limits are read out. These devices have set special bit turned to 1, which is used to set A string will be sent with the IDs and temperatures of the devices with alarm or with a message if there no alarm flags on to the Linux processor.

To change the temperature limits, the user has to provide as hexadecimals numbers those limits and the ID number of the device on which these new limits have to be set. All this infor-

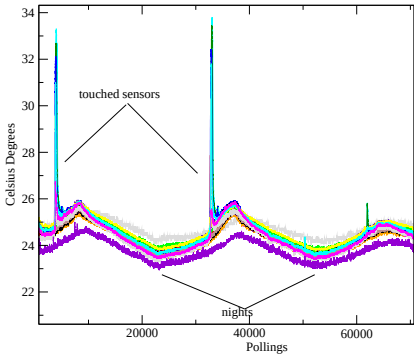

Fig. 3: *27 devices 2 days running for 2 days data acquisition, particular without the cooled sensor. Touch–heated devices are clearly visible. On x axis is shown the progressive number of polling.*

mation are checked and if there are no problems, the master will address the wanted device, idle all the others, and send on the bus a command that allows it to write 3 new bytes of the register memory, the high temperature limit, the low one and the register configuration, which is overwritten every time with the default values. A message with the result of the operation will then be sent to the Linux processor.

3.2.2 serialtrx.c

Serialtrx.c is a program that run on the Linux processor installed on the board.
As the program has to work with both, sockets and serial transmissions, at the startup these interfaces are initialized. Necessary parameters like the number of the serial port and the IP address of the server have to be provided on the command line when launching the program.
Then a command selecting the operation which will be performed by the devices (3.2.1) is received and forwarded to the microcontroller. If the command is the "set limit command" the program will also forward the temperature limits and the requested ID. At this point the program will wait for 5 seconds for any data arriving through the serial port from the MCU; if no data arrives a "no data" message is sent to the server. All these steps are repeated in an endless loop.

3.2.3 sockettcp.c

Sockettcp.c opens a socket on the server (the TCP-IP port needed for the communication has to be specified on the command line) and sends operation command to the client – the Linux processor on the board. If the command used to set the temperature limits is send, then the user is asked for the necessary values. In the next step the program will print on the screen a single string of data received from the client and send again an operation command.
To terminate the program, the termination signal (CTRL–C) is captured and the socket connection is closed before the execution of the program is stopped.

Acknowledgments

I want to expecially thanks my tutor, Michael Traxler for the constant help he gave to me and for all the time he spent on my project.
I also want to thanks Simon Lang and Peter Zumbruch for the support, information and teaching they gave to me.

References

[1] Dallas Semiconductor, DS1820 High-Precision 1–Wire© Digital Thermometer Manual

[2] Dallas Semiconductor, **Overview of 1–Wire© Technology and Its Use**, 2002

[3] Atmel© , **AT90CAN128 DataSheet**, 2005

[4] Atmel© , **8–bit AVR© Instruction set**, 2002

[5] Rich Neswold, **A GNU Development Environment for the AVR Microcontroller**, 2002

[6] avr-libc reference manual 1.4.4

[7] The Grace Team, **Grace User's Guide (for Grace-5.1.20)**, 2006

[8] http://www.cplusplus.com/

[9] http://www.coding-zone.co.uk/cpp/articles/140101networkprogrammingj.shtml

[10] http://gnosis.cx/publish/programming/sockets.html

[11] www.aps.anl.gov/epics/