



# IMPLEMENTING TEMPERATURE SUPERVISION FOR THE ALICE COMMON READOUT UNIT (CRU) CARD USING THE ONBOARD MICROCONTROLLER

*SUMMER STUDENT REPORT*

RUBÉN PÉREZ BERNABEU
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

SUPERVISOR
JOZSEF IMREK

CO-SUPERVISOR
FILIPPO COSTA

# 1. INTRODUCTION

## 1.1. PROJECT DESCRIPTION

The Common Readout Unit (CRU) is a sophisticated custom PCI Express FPGA card developed by "Centre Physique des Particules de Marseille" in collaboration of LHCb and ALICE. The ALICE trigger and read-out upgrade is based on 400 CRU cards that will be hosted by the First Level Processor node (FLP) server computers of the ALICE Online System. The CRUs will interface with all the upgrading ALICE subdetectors, the ALICE Trigger and Timing System (TTS), and the ALICE Online System (DAQ). These PCI Express add-in cards will accommodate an enormous amount of data traffic that will result in high power consumption and dissipation with the risk of overload or thermal damage. Implementing multiple ways of monitoring, controlling, and safeguarding of this complex FPGA card is very important to safe and reliable operation of the whole ALICE detector control, data read-out and trigger distribution system.

While the main effort has been focused on the development of the FPGA firmware that implements all the communication needs mentioned above, there are several independent design tasks identified to ensure the safe operation of the CRU card under all possible conditions. One such task is to implement a robust local (on-board) temperature monitoring and safeguarding subsystem that can autonomously prevent the thermal damage of the card even if the remote HW monitoring and controlling functions (integrated in DCS) failed for any reason. This safety function shall be implemented independently of the user developed FPGA firmware. It will be based around an Atmel ATmega128 microcontroller and it will autonomously handle on board temperature and voltage sensors in order to have direct control over the power supply subsystem of the card in critical situations. It will also communicate to the IPIMI subsystem of the host computer and log the monitored data into databases.

## 1.2. ALICE OVERVIEW

ALICE is one of the four big experiments at the CERN Large Hadron Collider (LHC). It is optimized to study heavy-ion (Pb-Pb nuclei) collisions at a centre of mass energy of 2.76 TeV per nucleon pair. The resulting temperature and energy density are expected to be high enough to produce quark–gluon plasma, a state of matter wherein quarks and gluons are freed. Similar conditions are believed to have existed a fraction of the second after the Big Bang before quarks and gluons bound together to form hadrons and heavier particles.

ALICE is focusing on the physics of strongly interacting matter at extreme energy densities. The existence of the quark–gluon plasma and its properties are key issues in quantum chromodynamics for understanding color confinement and chiral symmetry restoration. Recreating this primordial form of matter and understanding how it evolves is expected to shed light on questions about how matter is organized, the mechanism that confines quarks and gluons and the nature of strong interactions and how they result in generating the bulk of the mass of ordinary matter. [1]

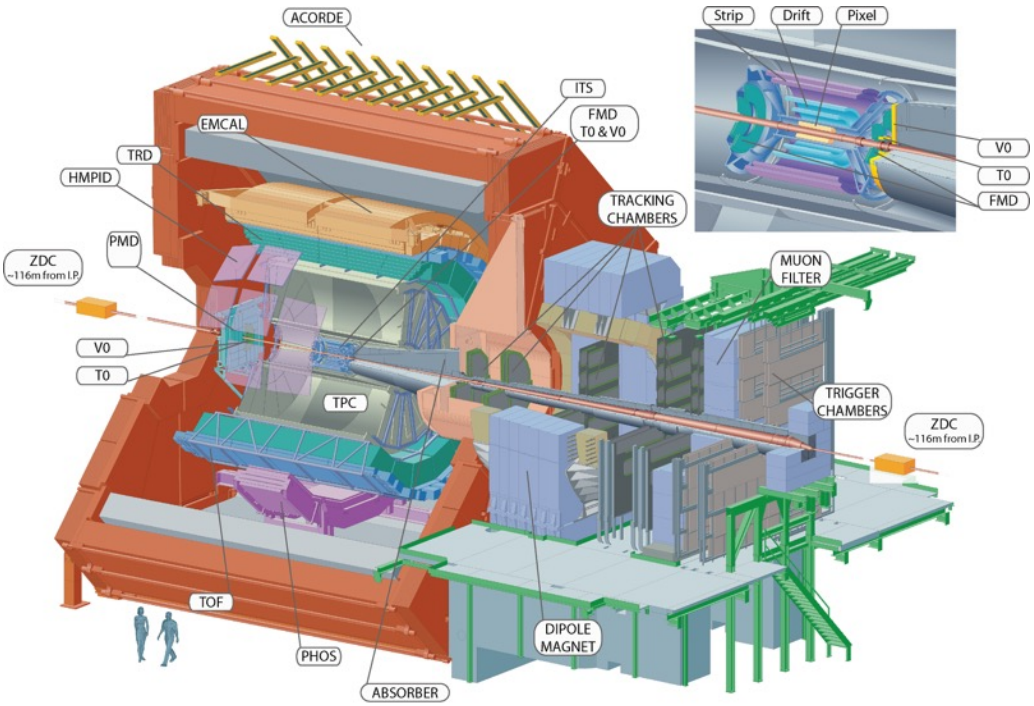


*Figure 1. ALICE Detector Diagram[1]*

## 1.3. UPGRADE OF THE ALICE READ-OUT AND TRIGGER SYSTEM FOR Run3

To cope with the increasing luminosity of the CERN Large Hadron Collider, ALICE experiment is planning a major upgrade of the detectors during the Long Shutdown-2 (LS2) period, which is at present foreseen to start in mid-2018. The increased luminosity will significantly enhance the physics reach of the experiment. The luminosities for Pb-Pb collisions will increase from $1x10^{27}cm^{-2}s^{-1}$ corresponding to collision rates of 50KHz. The upgrade strategy is based on collection greater than 10 $nb^{-1}$ of Pb-Pb ion collisions. Allice will also collect 6 $pb^{-1}$ of p-p collisions and 50 $nb^{-1}$ of p-Pb collisions, both at a leveled collision rate of 200KHz. Due to this increased interaction rate there will be manifold increase in the traffic of experimental data, the need for increase the readout rate.

The maximum readout rate of the present ALICE detector is 500Hz of Pb-Pb events but the overall goal is to readout 50kHz Pb-Pb collisions and 200 kHz p-p and p-Pb collisions. Consequently, the detectors and electronics need to be upgraded to handle higher rates. To tackle this requirement, a new approach based on Common Read-Out Unit (CRU) is being developed [2].

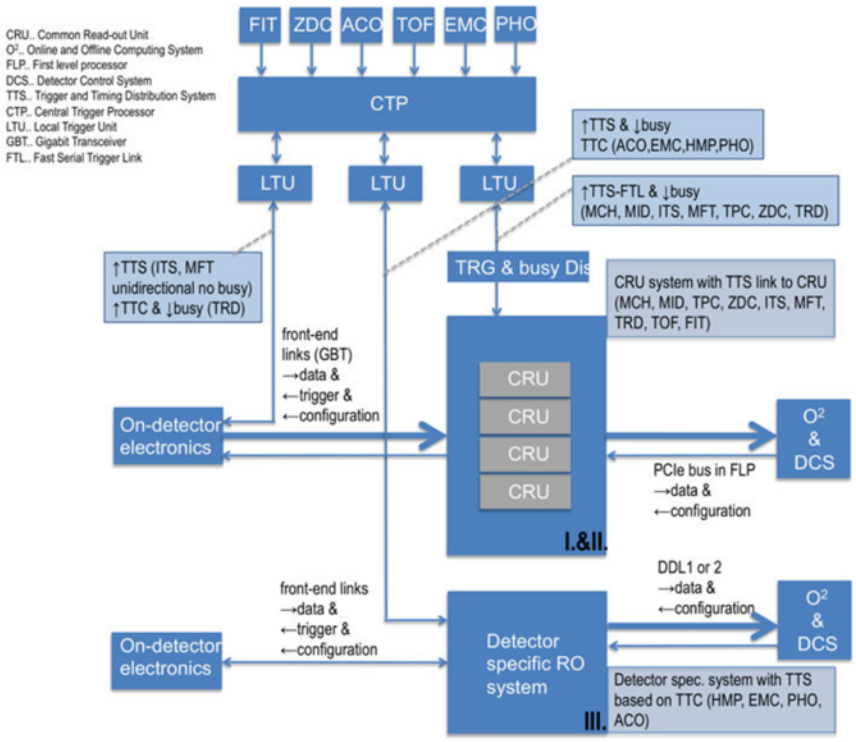


*Figure 2. ALICE upgrade architecture for LS2 [2].*

Last figure shows the ALICE upgrade architecture for LS2. The detectors can be grouped into three groups based on CRU usage:

1. The CRU delivers the LHC clock, the trigger information and participates in the data taking.
2. The CRU only participates in the data taking
3. The detector in this group will not use the CRU in any way.

# 2. FIRST STEPS IN ATMEGA128

## 2.1. DISPLAYING SOME NUMBERS

In order to become familiar with Atmel ATMEGA128 microcontroller we have started programming a custom board developed by the Department of Experimental Physics of University of Debrecen which contains that microcontroller:

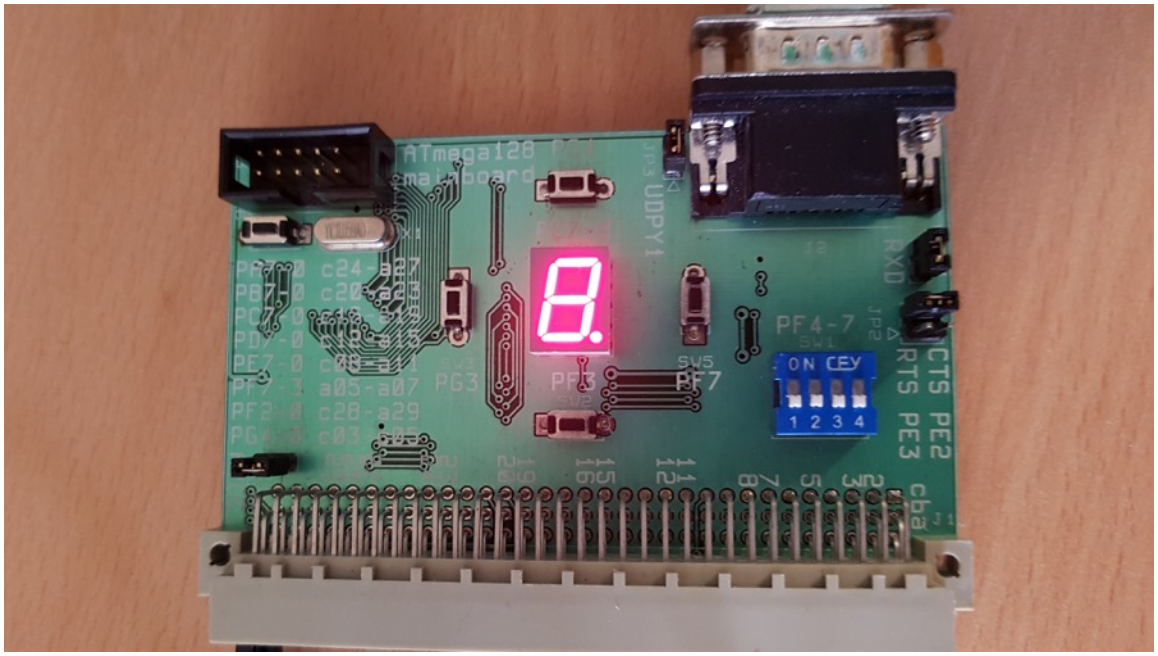


*Figure 3. ATMEGA128 mainboard*

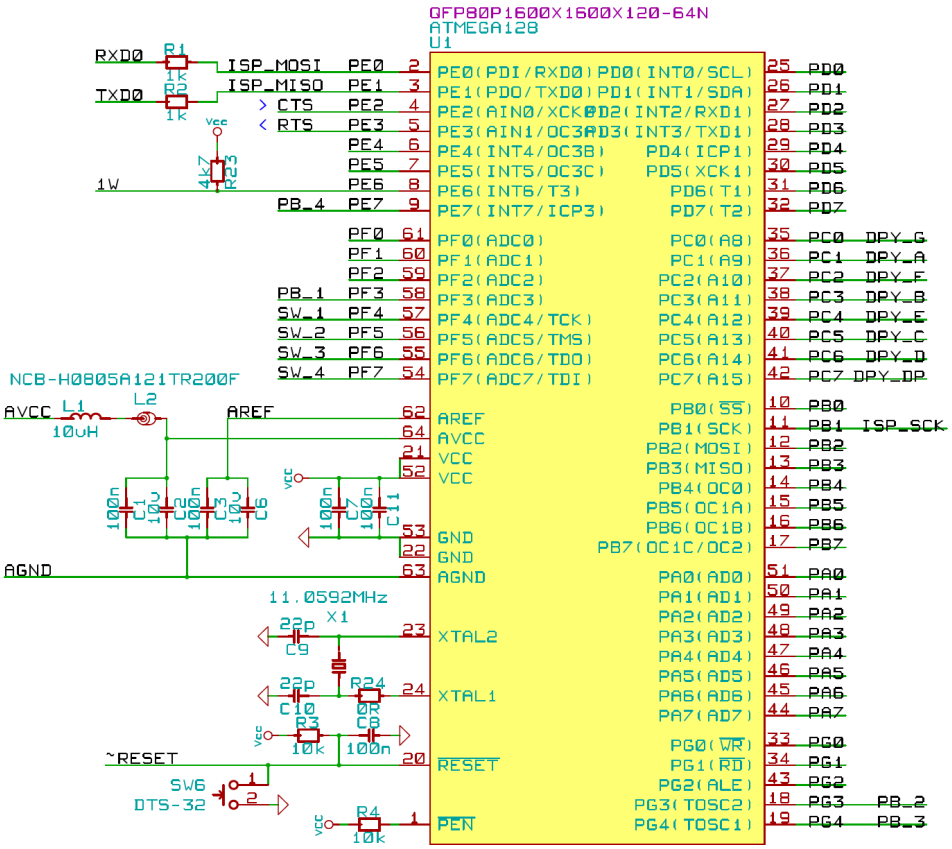


*Figure 4. ATMEGA128 Schematic [4]*

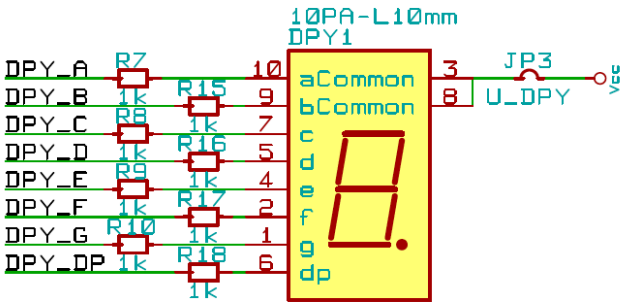


*Figure 5. Seven-Segment LED display Schematic [4]*

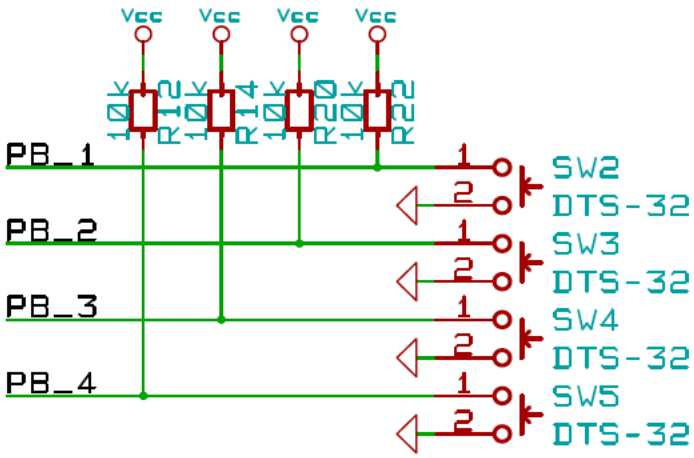


*Figure 6. Buttons included in ATMEGA128 custom board [4]*

In this part of the project, we have used a custom AVR Doper designed by University of Debrecen as well. It can be connected via Serial connection to the ATMEGA128 mainboard:

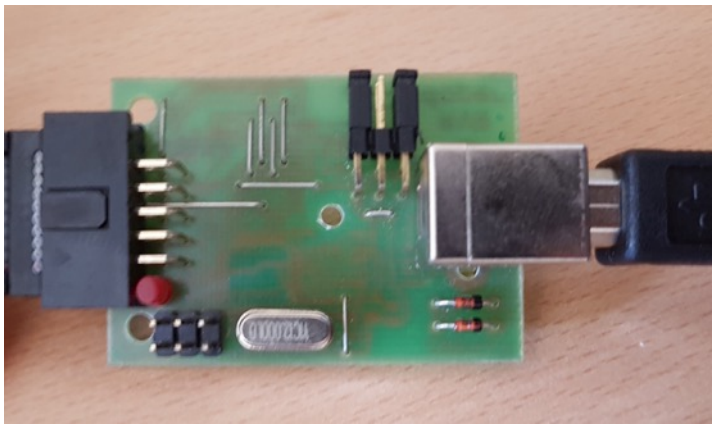

*Figure 7. AVR Doper custom programmer*

Our first program was very basic, we used the 7-segment display and 4 buttons. When we pressed the button SW2 we got a 1 on the display. If we pressed SW3 we got a 9, if we pressed SW4 we obtained a 3, and finally, if we pressed SW5 we displayed a 4. In order to implement this functionality, first of all we had to include io.h and delay.h libraries. Then we had to implement how to display numbers from 0 to 9. We have elaborated a table that sums up how to show each number with 8 bits:

| | 7-SEGMENT PATTERN | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | DP | D | C | E | B | F | A | G |
| Zero | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| One | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| Two | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| Three | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| Four | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| Five | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| Six | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| Seven | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| Eight | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Nine | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

*Figure 8. Seven-segment display bits*

For example, number 3 was ob10010100 which means that LEDs "d,c,b,a and g" are ON (LOW signal) and LEDs "DP, e and f" are off (HIGH signal). We proceeded in the same way for the other numbers.

Then, we had to define our LED ports by using DDRC=0xFF and define each button. For instance, SW2 was defined as DDRG = (1<<PG3); PORTG=0X08 because PG3 was connected to PB_2 (see the schematic) and if we check the iom128.h (*www.avrfreaks.net/sites/default/files/iom128.h*) we can observe that PG3 is assigned as 0x08.

Finally, we proposed if-else structures in order to meet our goal, when we pressed the button SW2 we got a 1 on the display. If we pressed SW3 we got a 9, if we pressed SW4 we obtained a 3, and finally, if we pressed SW5 we displayed a 4.

Link to the code: https://github.com/rupeber/starting

## 2.2. COUNT UP AND COUNT DOWN

In this case, we have gone a little bit deeper. Indeed, this code could be considered an enhanced version of the last program. It was able to count up by pressing the button SW5 and was also able to count down by pressing the button SW3. Moreover, if we pressed button SW2 at any time we reset the count.

This program was made of three main files: main.c, printNumber.c and waitDelay.c. With the main code we did the functionality that we have described before using if-else structures. PrintNumber.c was an auxiliary function that allowed us to display any number on the 7-segment following the table shown in Figure 1. In addition, we developed another function called waitDelay.c which produces a delay of 200ms if we call it.

The novelty of this program is that we used a Makefile. This file describes how to create a. hex file from .o files of our waitDelay.c, printNumber.c and main.c codes. Using a terminal, if we go to our working directory and we type "make", we will obtain main.o, printNumber.o, waitDelay.o, main.elf and main.hex. The file that has to be loaded in our board is main.hex. We also can add a new functionality in our Makefile called "program" followed by the instruction "sudo avrdude –c –stk500v2 –p m128 –P avrdoper –U flash:w:main.hex:a", this means that if we type "make program" we will write the code in our board directly and we will not need to write the previous instruction anymore. Furthermore, we have added another target called "clean" which will delete all .o files as well as main.elf and main.hex.

Link to the code: https://github.com/rupeber/atmega128/tree/d36dce35

# 3. TESTING TCN75A SENSOR IN OUR ATMEGA128 BOARD

## 3.1. TCN75A SPECIFICATIONS

The TCN75A is a digital temperature sensor capable of reading temperatures from -40°C to +125°C. Temperature data is measured from an integrated temperature sensor and converted to digital word with a user selectable 9 to 12-bit Sigma Delta Analog to Digital Converter. Small physical size, low installed cost and ease of use make the TCN75A an ideal choice for implementing sophisticated temperature system management schemes in a variety of applications [3].

FEATURES

| |
|---|
| 2-wire I2C compatible interface |
| User selectable 9 to 12 Bit resolution |
| ±3ºC Accuracy from 25ºC to 125ºC |
| Low operating current: 500 µA |
| Shutdown mode: 2 µA |
| Power saving one-shot temperature conversion measurement |
| Multi-drop capability |
| Space-saving MSOP-8 and SOIC-8 packages |
| AEC-Q100 qualified grade 1 |

*Figure 9. TCN75A sensor features [3].*

## 3.2. TCN75A WITH ATMEGA128: CODE DESCRIPTION

After a lot of previous versions, we developed a genuine program that was able to read the temperature from a TCN75A sensor via i2C hardware interface and send it again via UART to the PC. This program had two modes of operation, manual and automatic. If we pressed SW5 we were able to read the value manually each time we pressed it. However, if we wanted to start the automatic mode and read a temperature value every 0.5s we only needed to push SW3. The automatic mode was stopped if we pressed and maintained SW2. By pressing SW4 button we were able to change the seven-segment display mode, the first mode showed the last digit of the temperature in degrees and if this temperature was above 30 degrees the dot was also be turned on. The second mode turned on LEDs depending on the temperature level so that if temperature was higher, more LEDs were on.

In this program we have used some codes apart from the main one, they are i2c.c, printNumber.c, printNumberDot.c, tcn75.c, waitDelay.c and uart.c.

- I2c.c: Allow us to control the i2c hardware interface by using easy commands like i2c_send or i2c_read.
- tcn75.c: This code is able to control the TCN75 temperature sensor implementing some functions like tcn75_init or tcn75_read_temperature. We need to define here the slave address, checking the TCN75 datasheet we can conclude that (A6,A5,A4,A3) are factory-set to <1001> and for A2, A1, A0 we will choose <111>.
- Uart.c: it is used to send information via the UART connection. In this case, we send the temperature in Celsius degrees to the terminal of the computer.
- printNumber.c: it is a function that has been used before, it is able to display any number at the seven-segment display in case that temperature is below 30 degrees.
- printNumberDot.c: it is a variation of the previous function which will display any number with a dot in case that temperature is above 30 degrees.

Link to the code: https://github.com/rupeber/atmega128/tree/9bea89afc

## 3.3. TCN75A WITH ATMEGA128: TEST

First of all, we needed to assemble the TCN75A sensor in a small protoboard. As it is indicated in the sensor datasheet, we needed to use pull-up resistors on SCL and SDA pins.

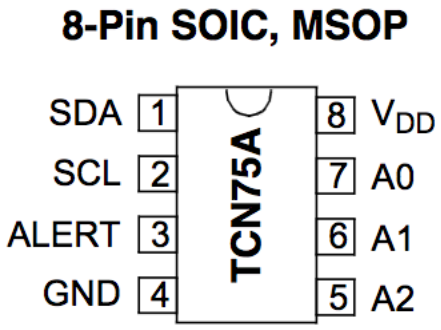


*Figure 10. TCN75A diagram[3]*

We also decided to connect A0, A1 and A2 to VCC by using small wires that were located under the red board. So, finally, our temperature sensor looked like this:

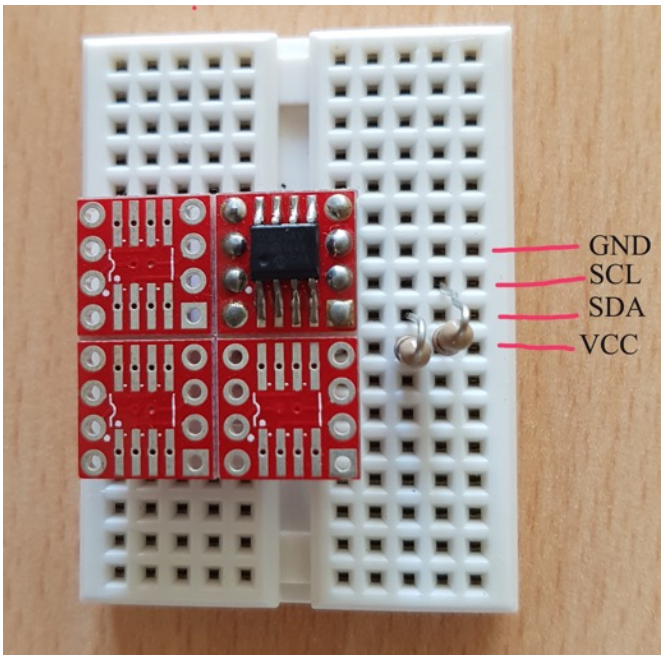


*Figure 11. TCN75A mounted on our protoboard with pull-up resistors*

At that time, we needed to connect properly our sensor to ATMEGA128 board. This board has access to all of his pins via an external connector called DIN41612:

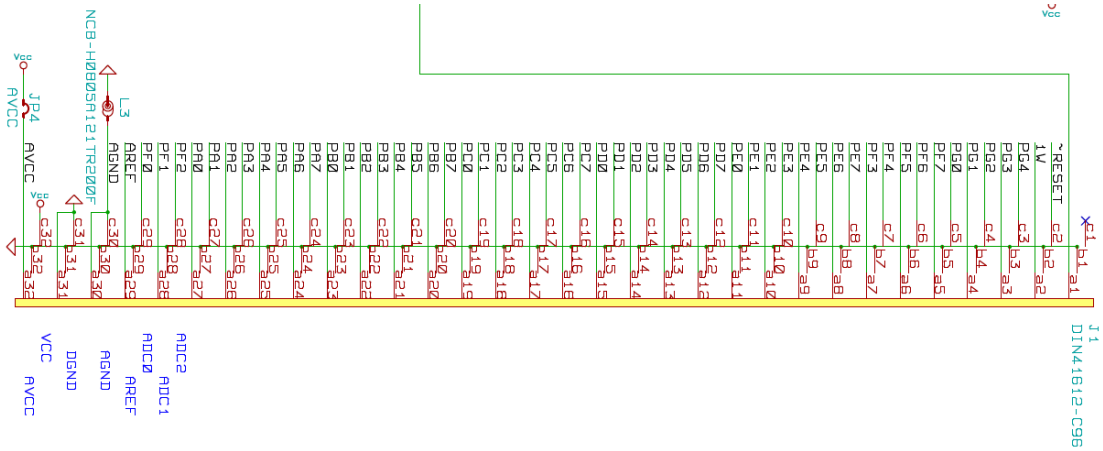


*Figure 12. DIN41612 connector schematic [4]*

Comparing Figure 8 and Figure 1, we concluded that SCL (PDO) and SCA(PD1) pins from ATMEGA 128 were connected to a15 and C15 respectively. Consequently, we connected there our sensor. Moreover, VCC and GND signals were taken from pins a32 and b32.

In order to connect the UART interface to our PC we developed a homemade connection as follows:

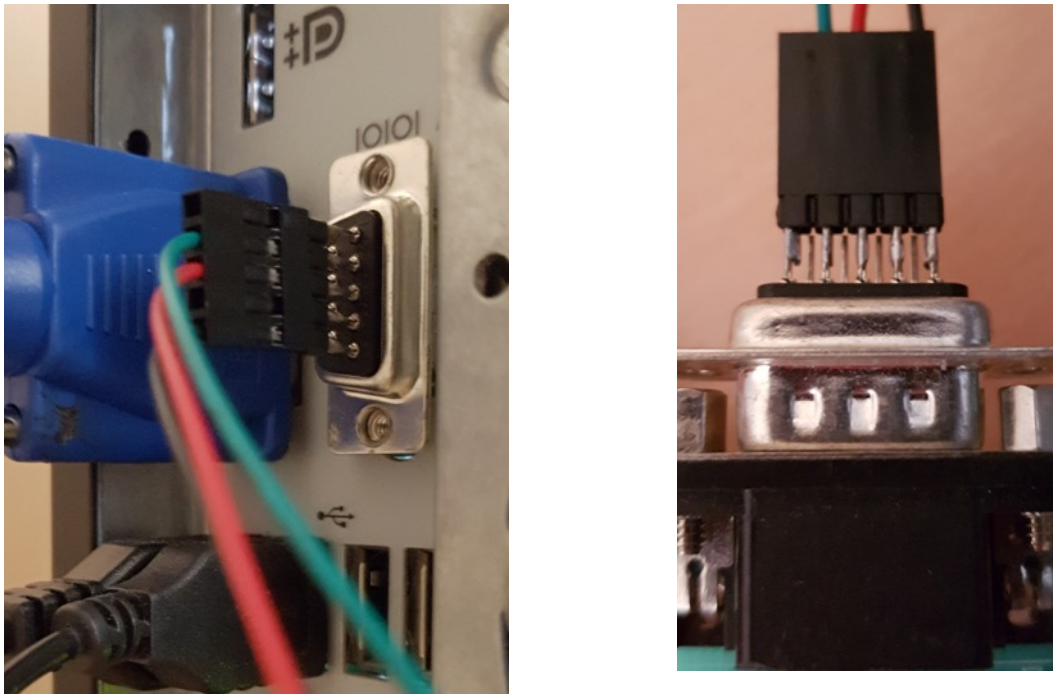

Figure 13. UART connection to PC (left); UART connection to board (right)

Once we had all connections ready, we were able to connect the AVR Doper programmer to the computer. Before loading the program into the board we opened a terminal and typed “minicom –s”. We needed to select ttySo as our default device which corresponds to the UART connection. When minicom was running we went

to our project directory and executed “make program” in order to write our code on ATMega128 board.

At that moment, the temperature in the office was 27ºC and, as we can observe, the seven-segment display was showing a 7.

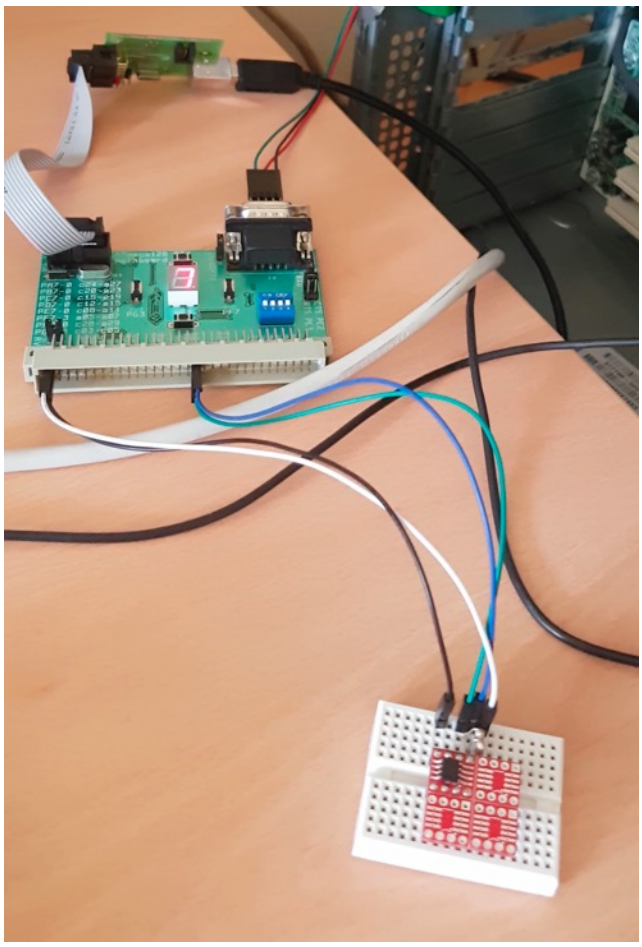

*Figure 14. First test of TCN75A*

Latter, we increased the temperature of the office artificially in order to check if our sensor was working properly and the seven-segment display showed a dot after the number indicating that temperature was above 30 degrees.

*Figure 15. Testing higher temperatures*

Additionally, we checked the temperature looking at the computer terminal.

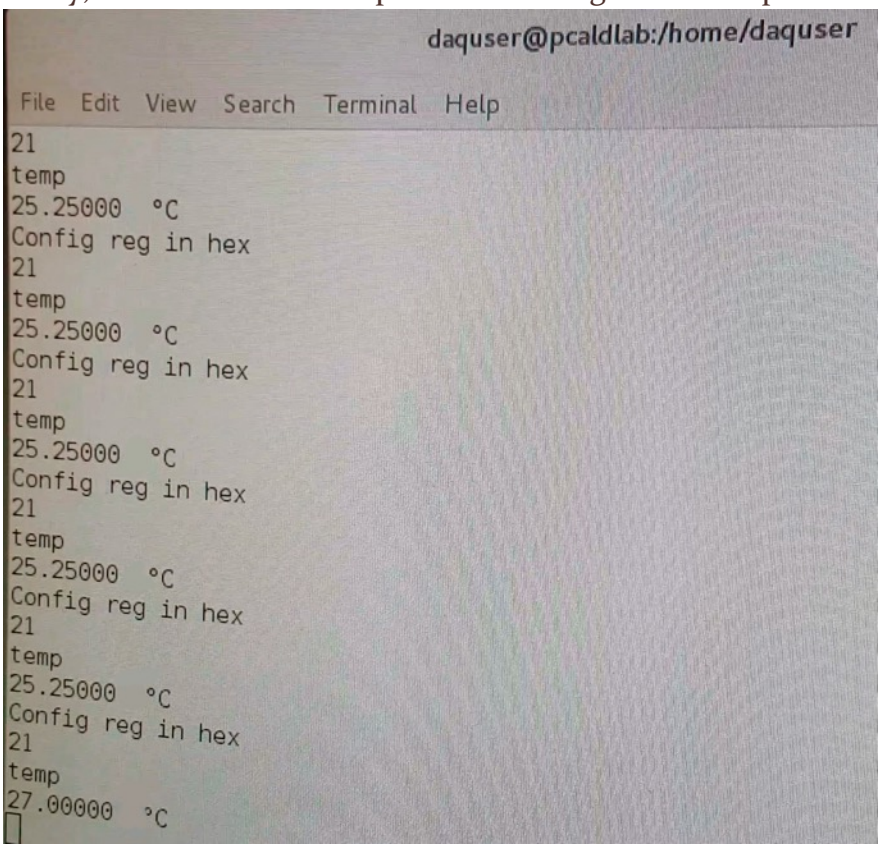


*Figure 16. Temperature acquisition via UART*

# 4. GOING FURTHER: I2C SOFTWARE IMPLEMENTATION

At this point, we had to take into account that SCL and SCA hardware implemented pins of ATMEGA128 in the CRU board were going to be used for PCIe interface and not to access the onboard temperature sensors. Consequently, we were not able to use the previous i2c code because it had been developed specifically for hardware implementations.

However, we found a good software implementation written by Peter Fleury [5] which was be very useful for us. Using that implementation, we did a program that worked in a software implemented i2C and displayed a 0 if the temperature was below 30 degrees and a 1 if temperature was above that limit. The temperature value was also sent via UART connection. In this program we used some codes apart from the main one, they are i2cmaster.S and uart.c.

- I2cmaster.S is an assembly code which contains basic routines for communicating with I2C slave devices. We need to adapt the SCL and SDA port and pin definitions to the ones we want.
- Uart.c it is the same code that we have described before, and it is used to send information via the UART connection. In this case, we send the temperature in Celsius degrees to the terminal of the computer.

Following the same steps provided in the last section we programmed our ATMega128 board and we observed that we were obtaining the same result in this software implementation.

Link to the code: https://github.com/rupeber/i2cTcn75/tree/f7d0a44bc.

# 5. JTAG PROGRAMMING USING ATMEL ICE

CRU has been programmed using JTAG interface, this is the reason why we needed to change our AVR Doper custom programmer for Atmel ICE.


*Figure 17. Atmel ICE Programmer [6]*

Studying the datasheet of Atmel ICE Programmer, we were able to extract these two diagrams:

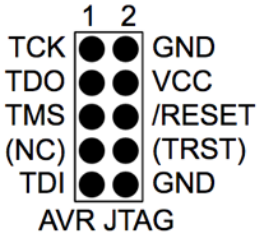


*Figure 18. JTAG Header 10-pin [6]*

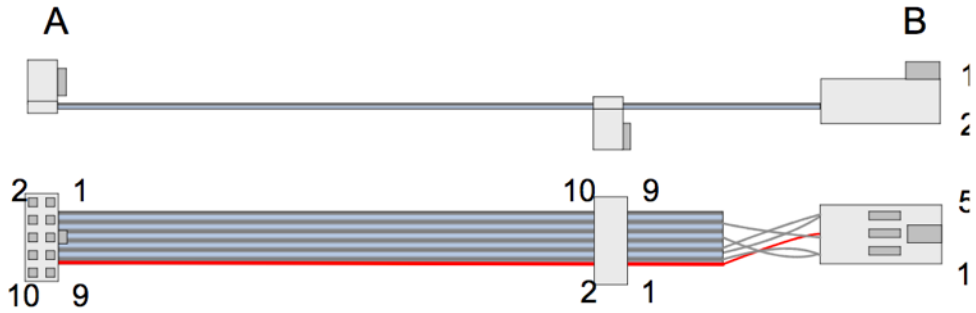


*Figure 19. Debug wire [6]*

Combining last two figures we have concluded that the end wire connection looks like this:

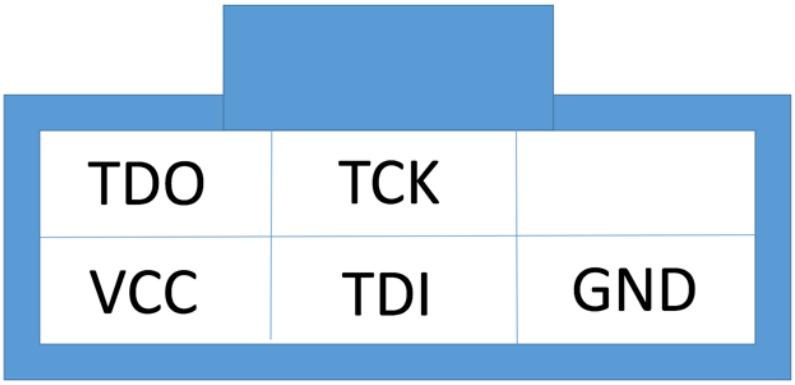


*Figure 20. End connection of Debug wire*

We also needed the TMS signal which was available on the JTAG Header 10-pin but not on the 6-pin header. We soldered a wire to an extra pin header in order to obtain it because it was necessary in a JTAG connection.

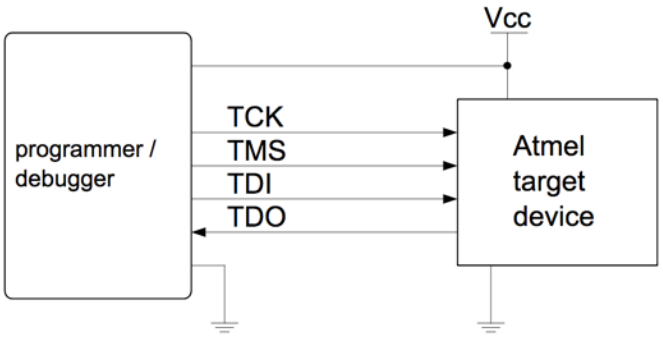


*Figure 21. JTAG Interface basics [6]*

# 6. LAST STAGE: CRU

The CRU has been programmed with an i2C software implementation because, as we have mentioned, SCL and SCA pins are being used for other purposes. Our goal has become to communicate the Atmega128 integrated in the CRU with the temperature sensor ltc2990, which has been possible thanks to the i2C interface.

## 6.1. Ltc2990 sensor specifications

The LTC 2990 is used to monitor system temperatures voltages and currents. Through the I2C serial interface the device can be configured to measure many combinations of internal temperature, remote temperature, remote voltage, remote current and internal Vcc. The internal 10ppm/ºC reference minimizes the number of supporting components and area required. Selectable address and configurable functionality give the LTC2990 flexibility to be incorporated in various systems needing temperature, voltage and current data. The LTC2990 fits well in systems needing sub-millivolt voltage resolution, 1% current measurement and 1ºC temperature accuracy or any combination of the three. [7]

FEATURES

| |
|---|
| ±0.5ºC Accuracy, 0.06ºC Resolution |
| ±1ºC Internal Temperature Sensor |
| 14-Bit ADC Measures Voltage/Current |
| 3V to 5.5V Supply Operating Voltage |
| Four Selectable Addresses |
| Internal 10ppm/ºC Voltage Reference |
| 10-Lead MSOP Package |

*Figure 22. LTC2990 sensor features [7]*

## 6.2. Programming the CRU: Code description

We have used the program that we developed in point 4 modifying some aspects in the main code called test_i2cmaster.c. First of all, the device address in this case have been different, as we can find in the LTC2990 datasheet (A6,A5,A4,A3,A2) are factory-set to <10011> and for A1, A0 we chose <10> which means A1 connected to VCC and A0 connected to GND. Observing the schematic of the atmega128 included in the CRU designed by Centre de Physique des Particules de Marseille we have deduced that pins PB6 and PB5 are connected to a green led and a red led respectively, so we only needed to define them as outputs pins.

We have included also two auxiliary functions that let us turn on and turn off the red and green LEDs, they are called respectively show6 and show7. Furthermore, we have added another function that is able to display through the red LED some light patterns. These patterns depend on the scale of temperature that we are measuring, so that, LEDs blink faster if temperature is higher.

The main part of the test_i2cmaster.c has implemented perfectly the temperature reading protocol specified in the LTC2990 datasheet.

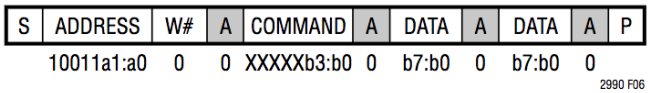


**Figure 6. LTC2990 Serial Bus Repeated Write Byte Protocol**

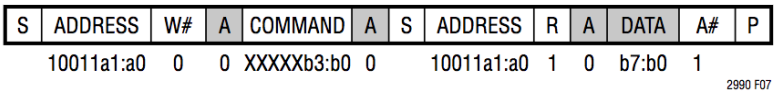


**Figure 7. LTC2990 Serial Bus Read Byte Protocol**

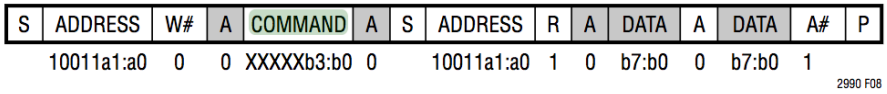


**Figure 8. LTC2990 Serial Bus Repeated Read Byte Protocol**

*Figure 23. Serial Bus Protocols of LTC2990[7]*

I2c start condition is sent along with the slave address and green led is turned on as a flag. Then, the "Trigger and Conversion" command 02h is written and a stop condition is sent. At this moment, the Serial Bus Repeated Read Byte Protocol can begin: a start condition is sent with the address followed by a write action in order to send the command 04 and indicate that the internal temperature is going to be read. The master sends a repeated start condition but this time with address + 1, because we want to do a reading operation. Finally, the slave send the 8 most significant bits followed by the 8 least significant bits, which are stored in two different variables. Bit 7, 6 and 5 do not contain any temperature information as we can see in the next table:

**Table 8. Temperature Measurement MSB Data Register Format**

| BIT 7 | BIT 6 | BIT 5 | BIT 4 | BIT 3 | BIT 2 | BIT 1 | BIT 0 |
|---|---|---|---|---|---|---|---|
| DV* | SS** | SO† | D12 | D11 | D10 | D9 | D8 |

*DATA_VALID is set when a new result is written into the register. DATA_VALID is cleared when this register is addressed (read) by the $I^2C$ interface.

**Sensor Short is high if the voltage measured on V1 is too low during temperature measurements. This signal is always low for $T_{INT}$ measurements.

†Sensor Open is high if the voltage measured on V1 is excessive during temperature measurements. This signal is always low for $T_{INT}$ measurements.

*Figure 24. MSB pattern[7]*

So that, we have created a variable that stores the 5 remaining bits of MSB and the 8 bits of LSB. Indeed, this value is the temperature in Celsius and depending on this number LEDs blink faster.

Link to the code: https://github.com/rupeber/i2cTcn75/tree/ab97d5d2f8c

## 6.3. Programming the CRU: Hands-on.

In order to communicate with the CRU together with Atmel ICE programmer, we have used Atmel Studio 7 in our PC. Studio 7 is the integrated development platform (IDP) for developing and debugging all AVR® and SAM microcontroller applications. The Atmel Studio 7 IDP gives us a seamless and easy-to-use environment to write, build and debug your applications written in C/C++ or assembly code.

Before programming and testing our code in the CRU, we have backed up the original flash contents of the onboard microcontroller.

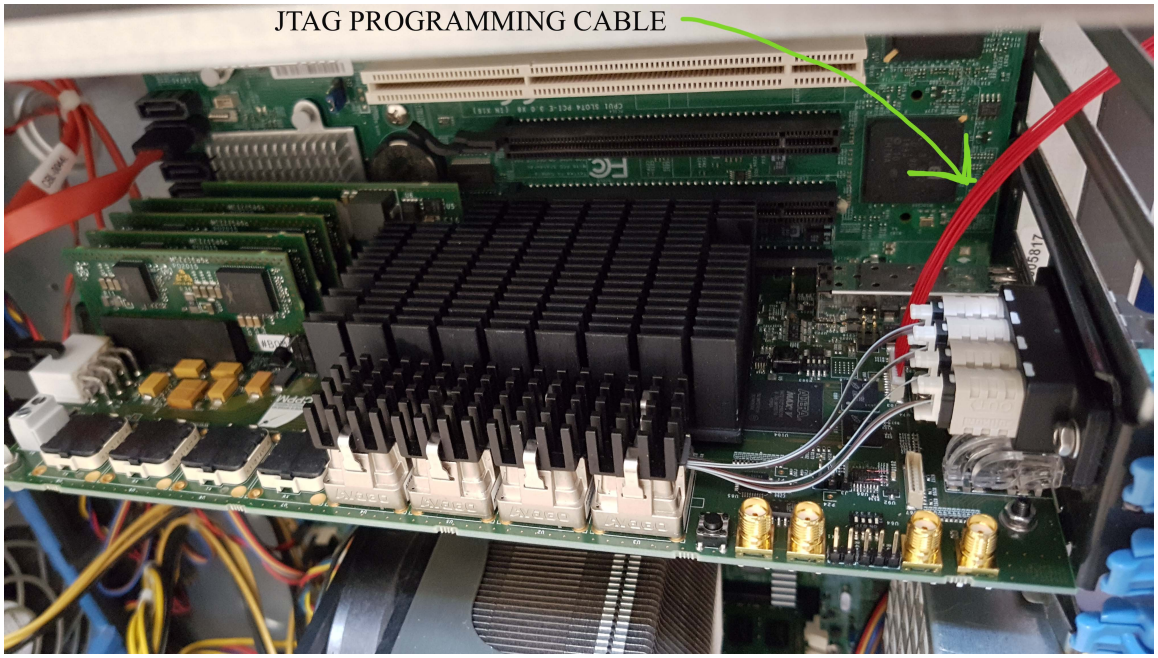


*Figure 25. Connecting JTAG cable to program the CRU.*

Atmel-ICE (J41800076613) - Device Programming

Tool: Atmel-ICE | Device: ATmega128 | Interface: JTAG | Apply | Device signature: 0x1E9702 | Read | Target Voltage: 3,3 V | Read

- Interface settings
- Tool information
- Device information
- Oscillator calibration
- Memories
- Fuses
- Lock bits
- Production file

**Detected device**

| | |
|---|---|
| Device names | ATmega128A, ATmega128 |
| Device signature | 0x1E9702 |
| JTAG id | 0xB970203F |
| Revision | L |

**Datasheet information**

| | ATmega128L-8AU | ATmega128L-8MU | ATmega128-16AU | ATmega128-16MU | ATmega128L-8AN | ATmega128L-8MN | ATmega128-16AN | ATmega128-16MN |
|---|---|---|---|---|---|---|---|---|
| CPU | AVR8 | | | | | | | |
| Flash size | 128 KB | | | | | | | |
| EEPROM size | 4 KB | | | | | | | |
| SRAM size | 4 KB | | | | | | | |
| VCC range | 2,7 - 5,5 V | 2,7 - 5,5 V | 4,5 - 5,5 V | 4,5 - 5,5 V | 3 - 5,5 V | 3 - 5,5 V | 4,5 - 5,5 V | 4,5 - 5,5 V |
| Maximum operating speed | 8 MHz | 8 MHz | 16 MHz | 16 MHz | 8 MHz | 8 MHz | 16 MHz | 16 MHz |

Copy to clipboard

External Links:

Datasheet (Summary)

Device Page

*Figure 26. Detecting CRU Atmega128 based.*

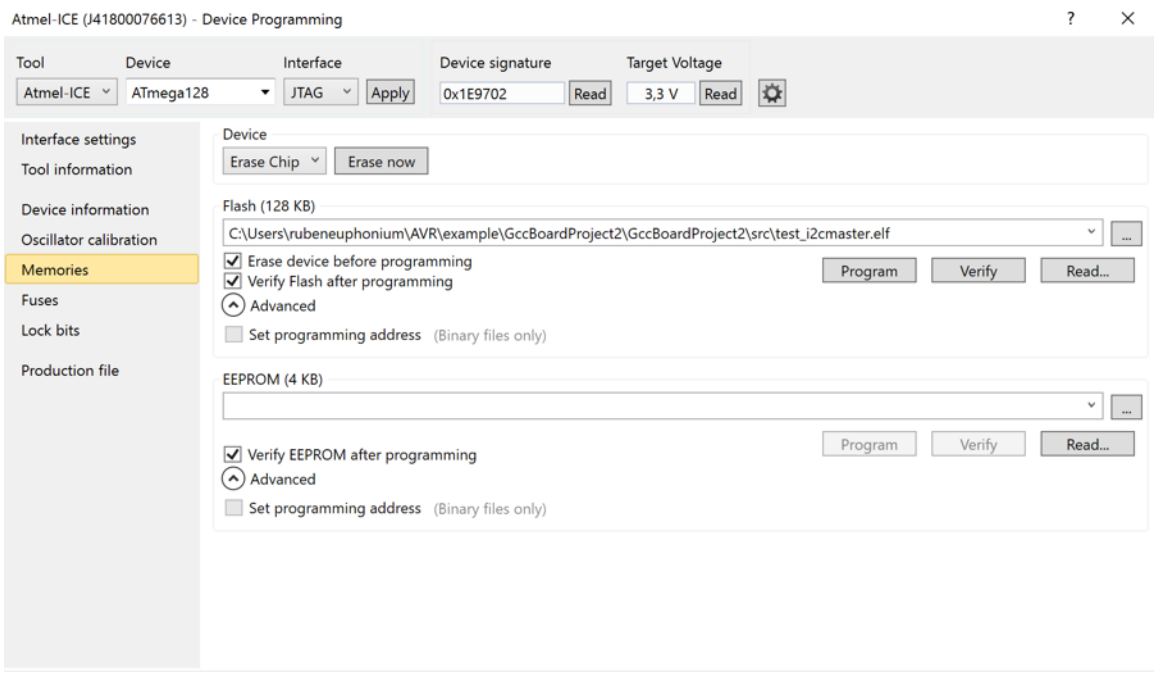


*Figure 27. Reading flash and EEPROM memories*

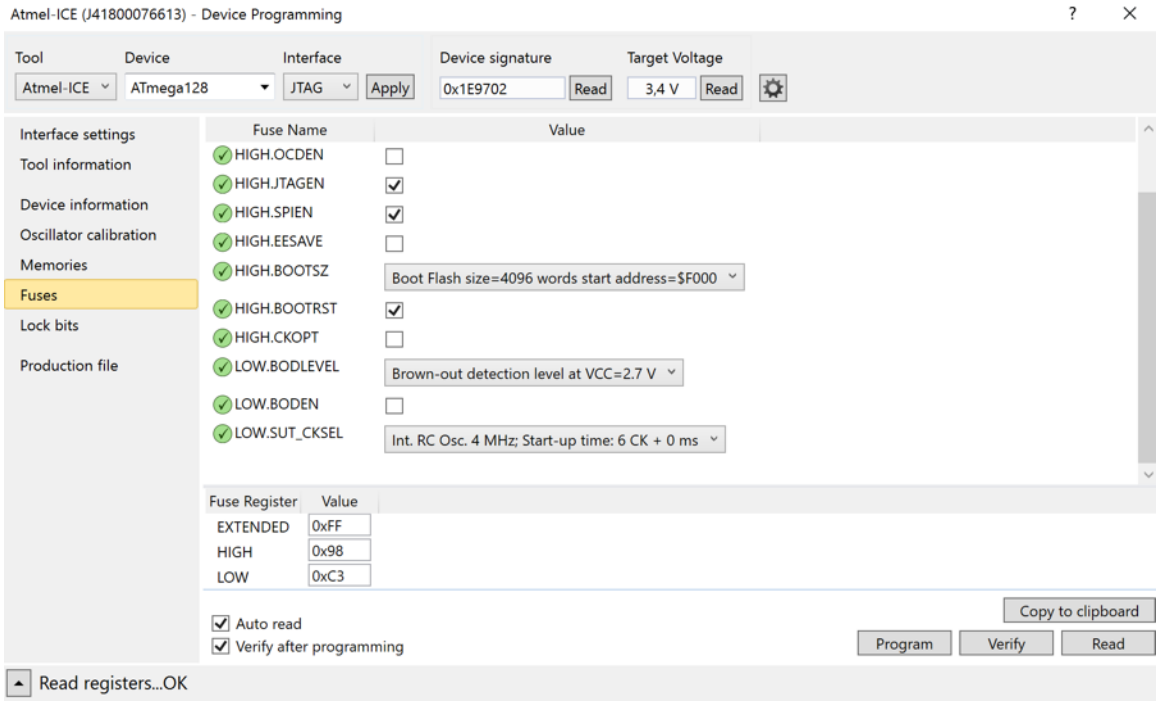


*Figure 28. Reading fuses*

Then, we have built the code and we have started a debugging setting a breakpoint at the beginning of the main part.

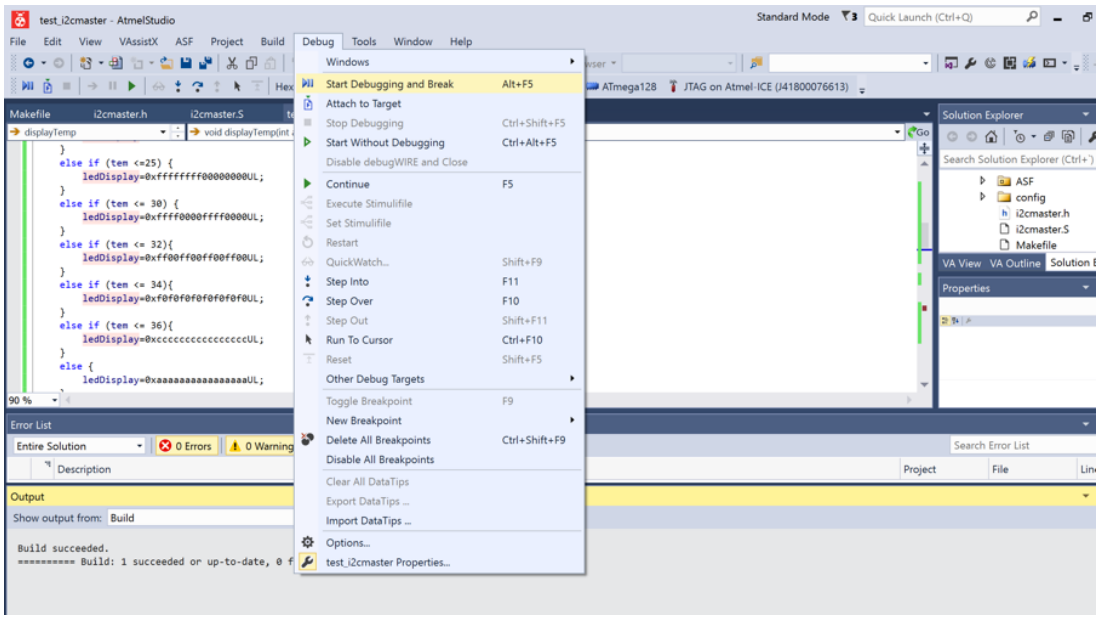


*Figure 29. Starting debugging.*

In the first iteration, we have seen that ack value is 0, which means that the slave has been detected correctly. In addition, the temperature is being measured correctly and it indicates 36 degrees.

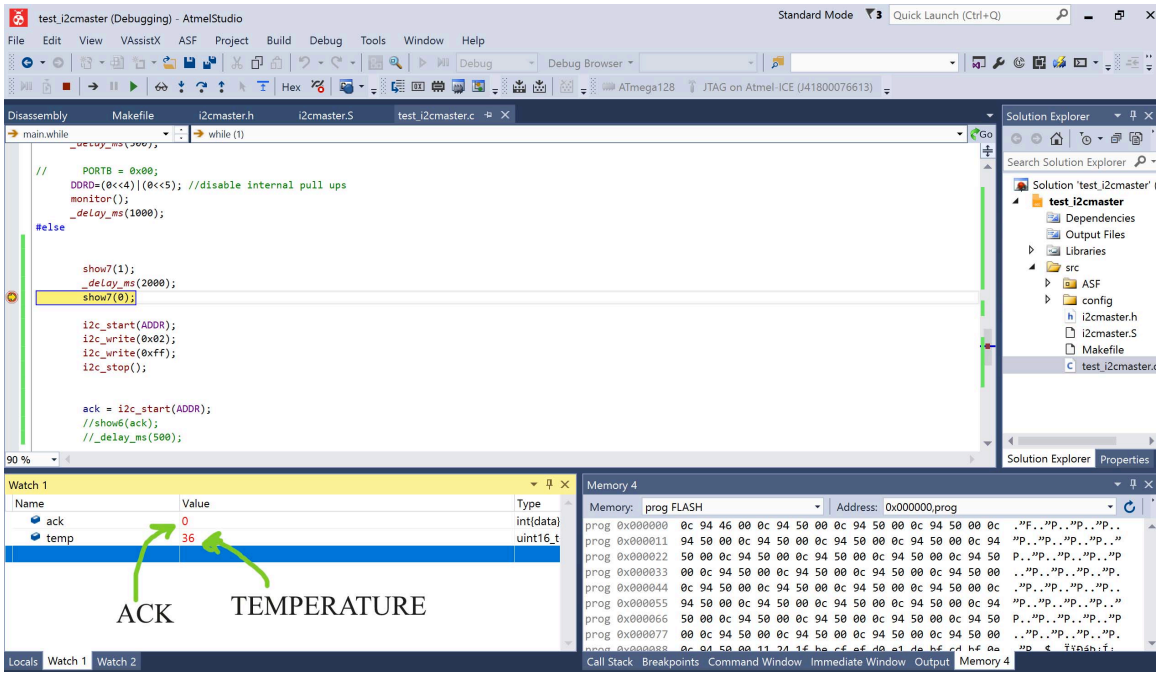


*Figure 30. First temperature measurement.*

With an external refrigeration system, we have observed after 10 seconds that the temperature has decreased 1 ºC. Moreover, thanks to the LEDs patterns we have checked that we are acquiring the temperature data correctly through a i2C interface software implemented and after eight weeks of work we have met our goal.

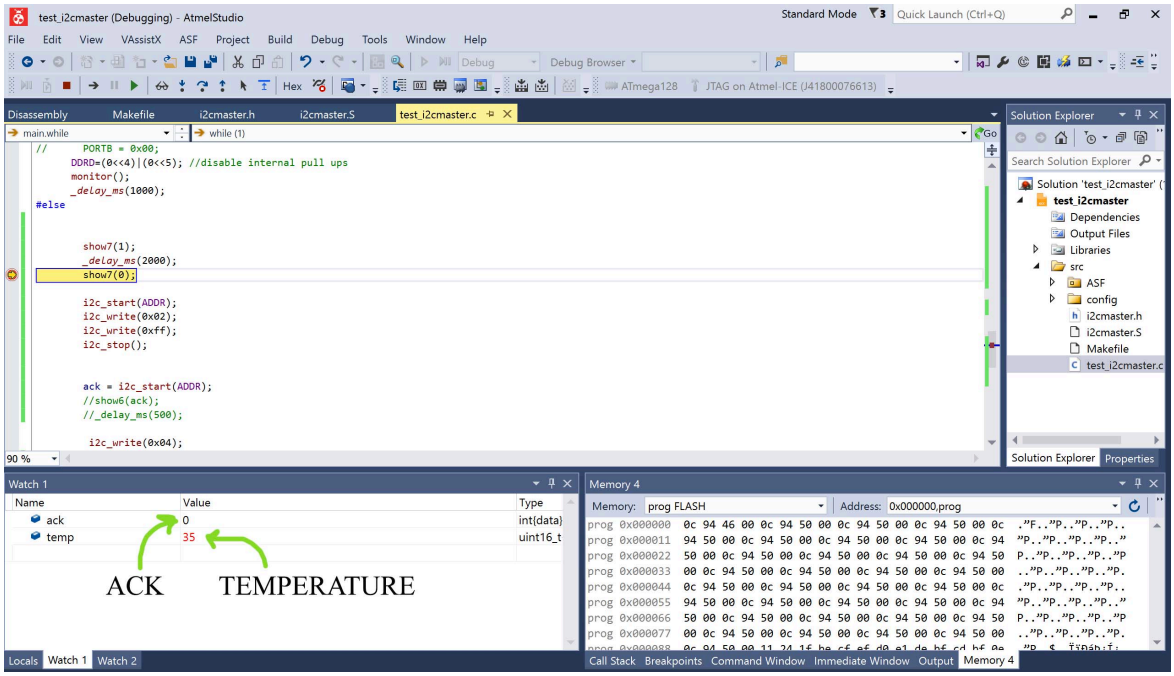


*Figure 31. Another temperature measurement.*

# 7. ACKNOWLEDGEMENTS

First of all, I would like to express my special thanks of gratitude to all the people that has been organizing the Summer Student Programme 2017. They gave me the golden opportunity to know deeply the world of particle physics. I consider that the skills and knowledge that I have gained throughout my project at CERN are a very valuable component in my future career development. My best thanks are for my supervisors Jozsef Imrek and Filippo Costa for their generous support, coaching and companionship during my two months stay at ALICE department. I would also like to acknowledge all the friends from all over the world that I have had the opportunity to meet. Definitely, CERN Summer Student Programme has probably made this summer the best of my life and I hope I have more opportunities to work at CERN again.

# 8. REFERENCES


[1] https://en.wikipedia.org/wiki/ALICE_experiment

[2] CRU User Requirements v0.8. ALICE Internal Note. Prepared by: E.David, T.Kiss, F.Costa

[3] TCN75A temperature sensor datasheet: http://ww1.microchip.com/downloads/en/DeviceDoc/21935D.pdf

[4] University of Debrecen: Experimental Physics Department

[5] I2C Software Implementation by Peter Fleury: http://homepage.hispeed.ch/peterfleury/avr-software.html

[6] Atmel ICE programmer User Guide: http://www.atmel.com/Images/Atmel-42330-Atmel-ICE_UserGuide.pdf

[7] LTC2990 temperature sensor datasheet: http://cds.linear.com/docs/en/datasheet/2990fe.pdf

[8] Atmel128 datasheet: http://www.atmel.com/images/doc2467.pdf