PCIe40 temperature protection system

Angel Romero
C-149360-1

Summer Student Report

LHCb
European Organization for Nuclear Research

Supervision:

Paolo Durante

4th August 2017

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Description of the project

PCIe40 is a high-throughput data-acquisition card based on PCI Express that is currently under development for the next upgrade of the LHCb experiment readout system. As part of this development, SMBus is intended to be used as a lightweight, out-of-band protocol to monitor the health of each data acquisition board. Starting from a simple prototype, the student will work on enabling SMBus communication between a COTS linux host and various on-board sensors, on top of existing linux facilities.

1.2 Hardware

A functional, basic scheme of the hardware in which this project is based (PCIe40 board) is shown in figure 1.1.

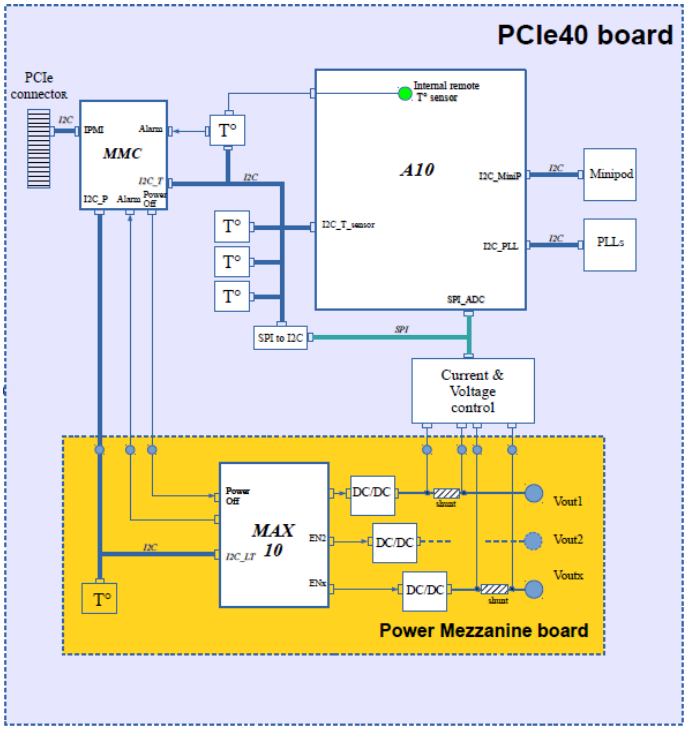


Figure 1.1: Functional scheme of the PCIe40 board.

This project will be mostly focused in the development of the communication between the microcontroller (MMC) and the different parts of the board (sensors, MAX10, SMBus).

1.3 Objectives

The a priori objectives of this project are listed below. These objectives will vary as we discover which things are feasible and which ones are not.

- Development of the communication between the embedded microcontroller and the sensors present in the board, via I2C.
- Search of the possibilities of communication of the host Linux system with the microcontroller via SMBus.
- Search of the ways of controlling the fans present in the host Linux system via SMBus from the microcontroller.
- Finally, development of a protection system that keeps the temperature of the FPGA between some security limits, and if not possible, it shuts it down.

Chapter 2

AVR: programming and debugging

In this chapter we will go through the steps that have to be followed in order to set up the entire toolchain necessary to start programming and debugging the microcontroller present in the PCIe40. An overview and explanation of the source code will also be done here.

2.1 Setting up the toolchain

The tool that will be used to program and debug the microcontroller is Atmel-ICE, connected to the microcontroller through JTAG interface.

2.1.1 Connection of Atmel-ICE to JTAG

The first step to program the AVR microcontroller is to connect it to the Atmel-ICE debugger. Figure 3.2 shows how to connect the wires coming from the debugger to the JTAG connector in the board.
Notice that:

- We have to identify the pin number 1 in the board by looking for a small dot printed in the PCB.
- The labels in the Atmel-ICE connector go from 0 to 9. 0 means 10 in all the pictures shown in figure 3.2.

Taking into account these details, we just have to connect the proper labeled wires from Atmel-ICE connector to the proper ones in JTAG connector.

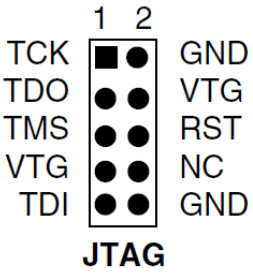
2.1.2 Windows

To prepare the toolchain in a Windows system, we need to:

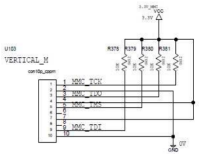
- Install the latest version of Atmel Studio: http://www.atmel.com/tools/atmelstudio.aspx
- Connect Atmel-ICE tool via USB to the computer in which Atmel Studio has been installed.
- When debugging the project in Atmel Studio, select as tool Atmel-ICE, as device atmega128, and as debug interface, JTAG. Now a debug session can be started.

Name	AVR port pin	SAM port pin	Description
TCK	1	4	Test Clock (clock signal from the Atmel-ICE into the target device).
TMS	5	2	Test Mode Select (control signal from the Atmel-ICE into the target device).
TDI	9	8	Test Data In (data transmitted from the Atmel-ICE into the target device).
TDO	3	6	Test Data Out (data transmitted from the target device into the Atmel-ICE).
nTRST	8	-	Test Reset (optional, only on some AVR devices). Used to reset the JTAG TAP controller.
nSRST	6	10	Reset (optional) Used to reset the target device. Connecting this pin is recommended since it allows the Atmel-ICE to hold the target device in a reset state, which can be essential to debugging in certain scenarios.
VTG	4	1	Target voltage reference. The Atmel-ICE samples the target voltage on this pin in order to power the level converters correctly. The Atmel-ICE draws less than 3mA from this pin in debugWIRE mode and less than 1mA in other modes.
GND	2, 10	3, 5, 9	Ground. All must be connected to ensure that the Atmel-ICE and the target device share the same ground reference.

(a) Atmel-ICE pinout for AVR and SAM devices



(b) Standard JTAG connector



(c) JTAG connector in PCIe40 board

Figure 2.1: JTAG connections

2.1.3 Linux

As the Atmel-ICE is an Atmel propietary tool, and Atmel has no support for Linux systems yet, the steps to set up the toolchain in Linux are a bit more complicated.

Compiling and programming

- First, we need to install the crosscompiler and the linker:
  sudo yum install uisp avr-binutils avr-gcc avr-libc
- We also need to install the uploader, avrdude. For this tool to be compatible with Atmel-ICE, version 6.3 or higher has to be installed, which can be downloaded from here: . While installing it, we have to make sure that when running *./configure*, the following libraries are present in the system: libusb, libusb1, libelf.

  To install them, the easiest way is to write the following command:

  pkcon search name libusb

  Then choose the lastest version available (in my case, for libusb1, for example is, libusb1-1.0.9-0.5.rc1.el6.x86_64), and install it using yum install. After that, install also the -devel version of it. We have to do the same for every library that is missing.

  In the end, when we run ./configure, the output of it has to be something similar to:

Configuration summary:
- - - - - - - - - - - - - -
DO HAVE libelf
DO HAVE libusb
DO HAVE libusb_1_0
.
.
.

- Now, if everything was correct, we should be able to run the Makefile present in our AVR repository and successfully run *make* and *make program*.

Debugging

Debugging in Linux with the non-linux-compatible tool Atmel-ICE has been shown to be quite difficult. During the course of this project, it has not been possible to configure the necessary tools to get it to work. However, for future development what has been tried will be listed here.

The main idea is to use OpenOCD or AVaRICE to communicate with the debugging tool (Atmel-ICE), and then use avr-gdb to debug.

- AVaRICE: an installation of the last version (2.13) has been successfully done, after manually installing all the necessary dependencies, but just to find that the software does not find any USB devices.

  After doing some research, this article (http://luniks.net/avr-debug.jsp) makes clear that Atmel-ICE is not compatible with AVaRICE, and I quote:

  At the time of writing (beg. of 2016), the old and discontinued JTAGICEmkII seems to be the best supported option on Linux, so I got myself a used one of those. The current Atmel-ICE Basic doesn't seem to work at all with AVaRICE because it uses a completely different protocol

- OpenOCD: the version of OpenOCD that we have been trying out is the latest: *openocd-0.10.0*. It can be downloaded from the official website. In this case, to properly get support for a CMIS-DAP Compliant Debugger, such as the Atmel-ICE, we need to install the next dependencies:

  hidapi-0.8.0-0.3.d17db57.fc26.x86_64
  hidapi-devel-0.8.0-0.3.d17db57.fc26.x86_64.

  This is the only version that has been found to work in my case. It can be found in rpm.pbone.net. Probably it is not the only one that works.

After installing this libraries, when we run *./configure*, CMSIS-DAP Compliant Debugger option should be set to *yes*.

Once installed, we can write the following command to start the connection with the debugger:

sudo openocd -f interface/cmsis-dap.cfg -c init

The output that we get when running this command is the one shown in figure 2.2. We have been trying to solve the error that we get, without success. That is why right now the debugging is done in Atmel Studio.

```
Open On-Chip Debugger 0.10.0
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
atmega328p
Info : CMSIS-DAP: SWD  Supported
Info : CMSIS-DAP: JTAG Supported
Info : CMSIS-DAP: Interface Initialised (JTAG)
Info : CMSIS-DAP: FW Version = 01.00.0021
Info : SWCLK/TCK = 1 SWDIO/TMS = 1 TDI = 1 TDO = 1 nTRST = 0 nRESET = 1
Error: CMSIS-DAP command CMD_DAP_SWJ_CLOCK failed.
```

Figure 2.2: Trial of debugging in Linux with Atmel-ICE.

2.2 Source Code

2.2.1 Main structure

The source code for the microprocessor can be found here: https://gitlab.cern.ch/aromeroa/aromeroa_w
For scalability reasons, the code has been developed in a state machine fashion, where we can easily choose to what state we should go at each moment, and most importantly, we can properly choose the state transition matrix, i.e., we can choose what function will be executed when we go from any state to any other state. This being so, the most important file in the source code is state_machine.[c|h]

In figure 2.3 there is an example of state transition matrix in the source code.

```
42 // -------------------- State transition matrix --------------------
43
44 // State-transition matrix:
45 //                                to:
46 //                     INITIAL     WORKING      IDLE
47 //      INITIAL
48 //from: WORKING
49 //      IDLE
50
51 typedef void transition_func_t( instance_data_t *data );
52
53 transition_func_t * const transition_table[ NUM_STATES ][ NUM_STATES ] = {
54     { NULL,              do_to_working,                  do_to_idle },
55     { NULL,              NULL,                           do_to_idle },
56     { NULL,              do_to_working,                  NULL       }
57 };
```

Figure 2.3: Example of state transition matrix

2.2.2 I2C interfaces

From direct design requirements, two different I2C interfaces are required: a master and a slave. The master will read all the corresponding sensors connected to it, and the slave will take care of the communication with the linux server, via SMBus PCIe. Since the atmega128 only has one I2C hardware module, one of the interfaces (master) has been done via bit-banging.

The source files that contain the I2C drivers are I2CMaster.[c|h] and I2CSlave.[c|h].

2.2.3 Temperature sensors

As each temperature sensor in the board has its own registers, configurations and temperature formats, some small drivers for LTC2990 and MAX1619 have been developed. These drivers are very basic, and only contain methods to read the local and remote temperatures of each sensor. There is still a lot of information and possible configurations in the datasheet that has not been written in the code. The reader can find further information in the comments of the source code.

Chapter 3

I2C connection: From PC to AVR and from AVR to sensors

3.1 From PC to AVR: upstream

For the communication between the microcontroller and the Linux PC via PCIe SMBus, we have had no success so far. However, we have discovered new limitations in the host servers that will be very important at the time of deciding which system to use. The different approaches that have been taken throughout the development of the project are:

- In the ASMB8-iKVM server, we have tried to find the I2C bus that is visible from user space (i2cdetect) and that is connected to the SMBus of the PCIe connector. To do that, what has been done is, using the Arria 10 FPGA that is in the PCIe40 board, and that is connected to these SMBus wires, we have pulled down to GND the SDA pin of SMBus, and then do i2cdetect in every available i2c port in the linux user space.

  Although there has been no success, we have found that the server ASMB8-iKVM is not capable of controlling the internal SMBus in the motherboard from user space.

- We have also found that the i2c modules that we can control from user space are connected to some I2C multiplexers (PCA9544A). This I2C multiplexers allow the expansion of the number of addresses in the corresponding I2C bus. However, none of these sub-addresses are connected either to the PCIe SMBus.

- When trying to control the fans that are in the motherboard of the server, we found that these fans can only be possibly controlled by the BMC (Board Management Controller), which in most of the cases is not open source. We have tried to use some IPMI commands to change their speed from linux user space, but we could only enable or disable the sensors that measure the speed. No control of any of the fans in the motherboard has been achieved so far.

This limitations raise concerns about whether the current server should be used for the next upgrade of the LHCb detector or if, on the other hand, a new host server has to be selected.

3.2 From AVR to sensors: downstream

Using the I2C master interface, and due to the lack of documentation in this part of the system, we have had to discover every sensor that is present in the board downstream, i.e., from the microcontroller to the inside of the PCIe40 (see figure 3.1).

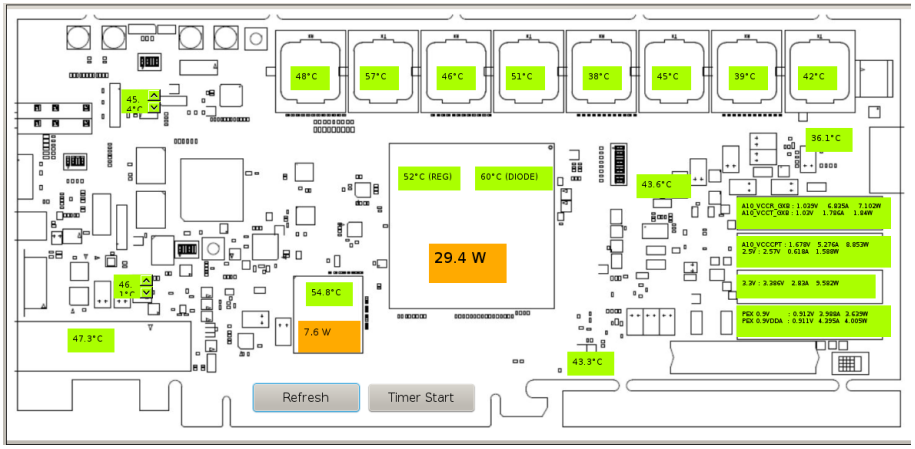


Figure 3.1: Diagram of the location of the sensors in the PCIe40 board.

To do that, a special discovery software has been developed that goes through every I2C address and reads the ACK given by that certain address. In table 3.2 we can see which ones are available and its description.

Discovered address (HEX)	Device
0x30	MAX1619
0x52	Not sure what it is[1]
0x98	LTC2990_1
0x9A	LTC2990_2[1]
0x9C	LTC2990_3
0x9E	LTC2990_4

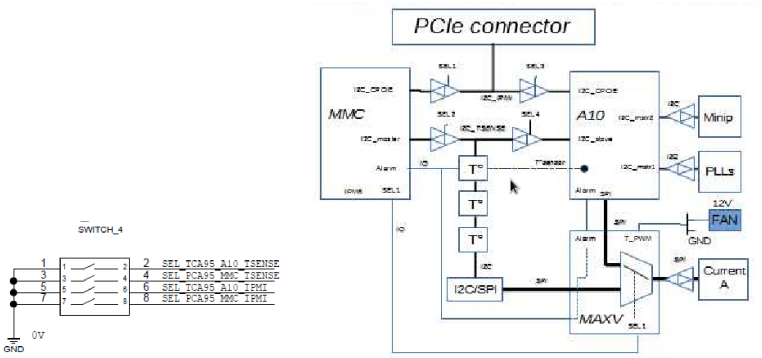
Table 3.1: Discovered sensors.

[1] Probably due to the positions of the 4-enable switch (see figure **??**), we only get ACK from these devices, but we are not able to communicate with them so far. Need to do more research on this switch. We assume 0x9A is a LTC2990 since its address exactly corresponds to it, but 0x52 can be both a converter I2C/SPI (SC18IS602) or another sensor (LM95235).

In order to be able to read the sensor MAX1619 we had to select the next combination in the switch: SW1 → ON | SW2→OFF | SW3→ON | SW4→ON.

It is important to say that from every LTC2990 we can read 3 temperatures: 2 remote and 1 local temperature. From the MAX1619 we can read 2, remote and local temperatures.

As it can be seen, we have a way more sensors in the diagram in figure 3.1 than in the discovery table. This is due to the fact that we do not know if the device in address 0x52 is actually a converter I2C/SPI, in which case we could have many different sensors



(a) Schematic switch enable (b) Diagram switch enable

Figure 3.2: I2C Master Selection Switch

connected to it. Because of this, we do not know which sensor from the discovered ones is related to which sensor in the diagram, in general. The only one that we know without uncertainty is the MAX1619 in address 0x30, whose remote temperature reading corresponds with the sensor that is in the middle of the FPGA in the diagram, the one labeled with (DIODE).

Chapter 4

Conclusion

As seen in the previous chapter, not all the objectives could be met mainly due to the limitations of the host server. We will now go through all the a priori objectives and see which ones could be met.

- The communication between the microcontroller and all the sensors, downstream, has been sucessfully developed.
- The communication from the host Linux system to the microncontroller via SMBus has not been possible. However, during the search we have found different details that will in the very next future help to choose a host PC that supports all the requirements.
- In the same way, the host server does not allow the control of the mother board fan speeds from user space. This control is reserved to the Board Management Controller, which is not open source.
- Therefore, since we cannot throtle the fans, the temperature control could not be done with the microcontroller. However, a simpler protection mechanism that shuts down the FPGA in case of high temperature could be perfectly developed.