CERN Summer Student Programme 2025

30 July 2025
acvelez@estud.usfq.edu.ec

Development of an I2C(PMBus) - USB interface for ALICE Trigger Board (ATB) monitoring

Ana Cristina Vélez Pinto

Supervisor:
F. Pazdic, R. Lietava - *University of Birmingham*

Keywords: A Large Ion Colider Experiment (ALICE), ALICE Trigger Board (ATB), Central Trigger Processor (CTP), PMBus, I2C.

Abstract

The ALICE Trigger Boards (ATB) used in Run 3 contain two Power Management (PM) from where the outputs are read by a GPIO2 Texas Instrument (TI) device. The output is accessed by the Command Line Interface (CLI), sent into the DIM server and deployed to the SCADA WinCC dashboard. While operational, the system is slow and customization is not feasible due to limited access to the CLI.

This summer student project goal was to create an I2C(PMbus) to USB interface that can be used with a range of I2C adapters. The interface is written on Python and implemented with the FTDI4232H chip as the adapter connected in placement of the TI device. The resulting interface is capable of reading outputs from the ATB and displays them on the DIM server as well as from the terminal according to the request from the user. The values in the DIM can later be deployed to the WinCC dashboard as it is done with the current TI device.

The resulting interface can easily be modified for any other I2C adapter as well as other chips using PMBus protocol to send data.

Contents

1 Introduction

1.1 CTP - Central Trigger Processor

The Central Trigger Processor (CTP) is a system that decides which data to keep based on a criteria. During RUn 3 most of the detectors in ALICE experiment have been operating in continuous readout mode. Data is, therefore, taken continuously delimited by Heartbeats (HB), signals sent by the CTP [1]. As of today, ALICE has 1 CTP and 14 Local Trigger Unit (LTU) boards which share the same hardware and differ on the firmware of the FPGA boards located under the fan in the ALICE Trigger Board (ATB) shown in the figure below.

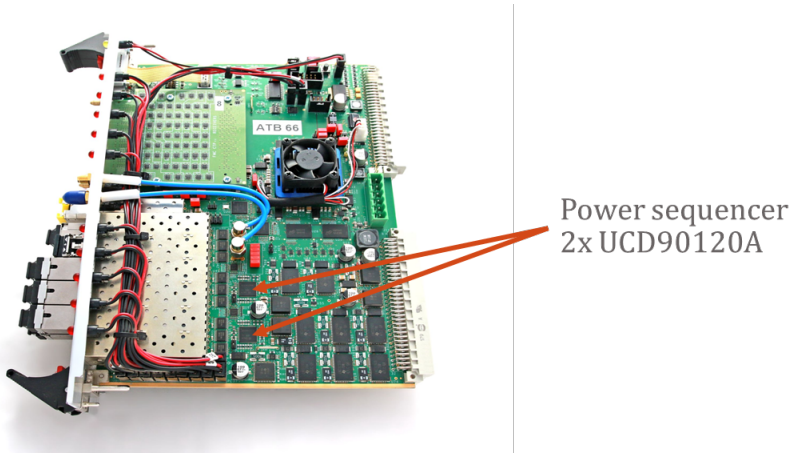

Figure 1: ALICE Trigger Board horizontal view [1]

LTU boards are connected to the CTP. The two chips marked by the arrow are the Power Management Chips (PM) from which the output values are read [2]. The addresses to the PM chips are determined using physical resistors and the Power Management Bus (PMBus) located at the side of the board, is connected to the GPIO2 Texas Instrument USB adapter.

1.2 The current Solution

At hardware level, the TI adapter is connected in a Daisy chain to multiple ATB via PMBus and via USB to the Anywhere USB Ethernet to USB interface module [2]. The Anywhere USB module is connected using Ethernet to the server.

The TI adapter reads the values of current, voltage and temperature from the PM chips of the ATB. The output is accessed through the Command Line Interface (CLI) interface by Texas Instruments with the name *FusionParamReader*. Data is then pipped to a custom C++ program which parses the output and publishes data to DIM. Data is then deployed to the SCADA WinCC dashboard from where it can be accessed by users.

The current solutions presents issue with the speed at which data can be taken from the PM chips with newer versions of the TI adapter being slower. It also eliminates the possibility of customization as the CLI is owned by Texas Instruments Incorporated and therefore, not open access. This results in the impossibility to use different adapters for the monitoring of the ATB.

1.3 I2C (PMBus) standard

Inter Integrated Circuit Communication (I2C) is a protocol that allows communication between devices or sensors [3]. I2C involves using two wires to send and receive data, namely the clock (SCL) and data (SDA) lines, from the controller device (master) to one or more peripheral devices (slaves) with a unique 8 bit address [4]. PMBus is a protocol based on I2C that allows for error correction within the communication by which the slave communicates to the master using SMBALERT# [5]. The basic format for I2C communication is as follows.

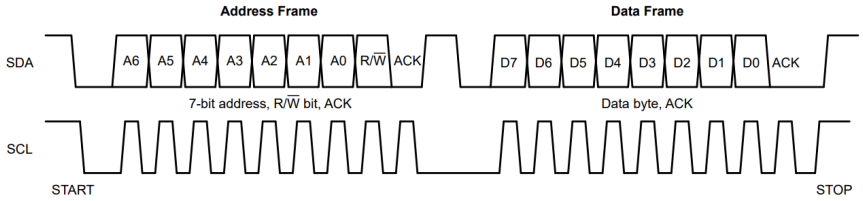

Figure 2: I2C address and data frames [4]

PMBus specifications offer a series of commands that are known [6]. These commands can read from, write to or exchange data with the slave and represent a standard that can also be used to decode the status of the device. Documentation of the protocol can, therefore, be used to create an I2C to USB interface that can be used for a range of I2C adapters and work at higher speeds than the current solution.

2 Initial testing

The first steps towards accomplishing the objectives of the project consisted in the throughout comprehension of the TI device operation, chip capability testing and a series of tests of the python library for the adapter.

2.1 Device analysis using logic analyzer

In order to have an insight into the hex commands being written and read from the TI device, a logic analyzer was connected to the SCL and SDA lines of the PMBus connected to the TI adapter. Commands were then sent from the terminal of the computer and the digital signals decoded inside the logic analyzer using the I2C protocol.

The commands sent and received by the device during the reading of current, voltage, temperature, address scanning and part ID requests were studied. The resulting hexadecimal values were compared to open access PMBus documentation to determine first if the standard was being followed. Once the later was proved, the documentation and obtained commands were compared to understand the setup process of the PMBus before the outputs can be read from the ATB. The setup process was seen to include the reading of the part ID, revision ID and page number of the address. Other commands were observed to determine memory address writing and pin routing, however, documentation advised not to manipulate these

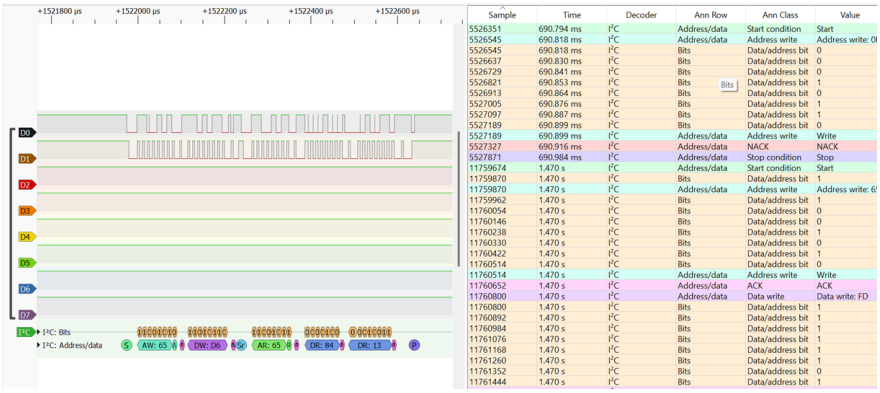

Figure 3: Digital signals received in the logic analyzer and decoded using I2C standard when IOUT is requested from the PM chip.

commands if not strictly necessary as any wrong command sent can potentially brick the device.

Once the PMBus standard and the TI adapter inner-works are understood and documented, the testing of the new adapter could start.

2.2 Chip testing with Arduino

To test the capabilities of the FTDI4232H Mini Module chip [7] used as an adapter for this project, the chip's I2C ports were connected to the I2C ports of an Arduino UNO R4 Wifi board [8]. A SparkFun level shifter [9] was used during initial testing to shift from 3.3V used in the FTDI4232H chip to the 5V of the Arduino board. The level shifter was removed in later stages as it was seen not to be necessary for the data transmission and, instead, generated interference in the I2C communication between devices.

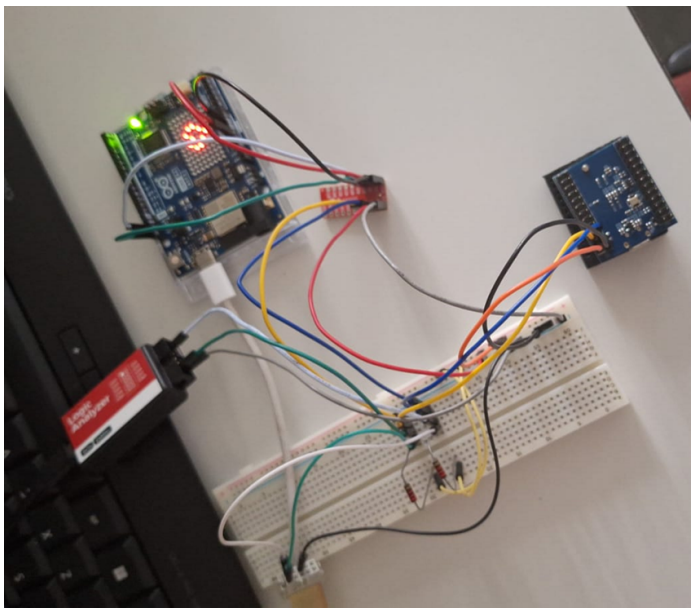

Figure 4: Hardware connection of the FTDI4232H chip to the Arduino UNO 44 Wifi board.

Using the Arduino IDE, the Arduino board was configured as the slave and the FTDI3242H chip as the master using the Pyftdi library. Blocks of data in hexadecimal were written and read from the Arduino board using the Wire.h library's functions of OnRequest() and On-Recieve() [10]. For the initial testing, the address was assigned to the Arduino board and read from the Python script using the exchange() function form Pyftdi [11].

3 Hardware Connection

After the successful runs from the test, the Arduino board was disconnected and the FTDI4232H chip was connected to the PM chips on the ATB with the SCL line connected to pin 9, SDA line to pin 10 and ground in pin 6. $2.2k\Omega$ resistors were used for the pull up. The chip itself is connected to the computer using a USB cable.

4 Python Interface

The resulting interface follows the flux diagram shown in the figure below. The interface consists of three libraries that exchange data with each other. The Adapter Wrapper library contains all connections to the hardware from the Python; the PMBus library contains classes for the setup of the adapter, the data exchange with the adapter and the decoding of the data received; the DIM server library contains the class that generates the services for different outputs and publishes it into the DIM and finally, the Param Reader library prints and manages the handlers for requests coming from the computer.

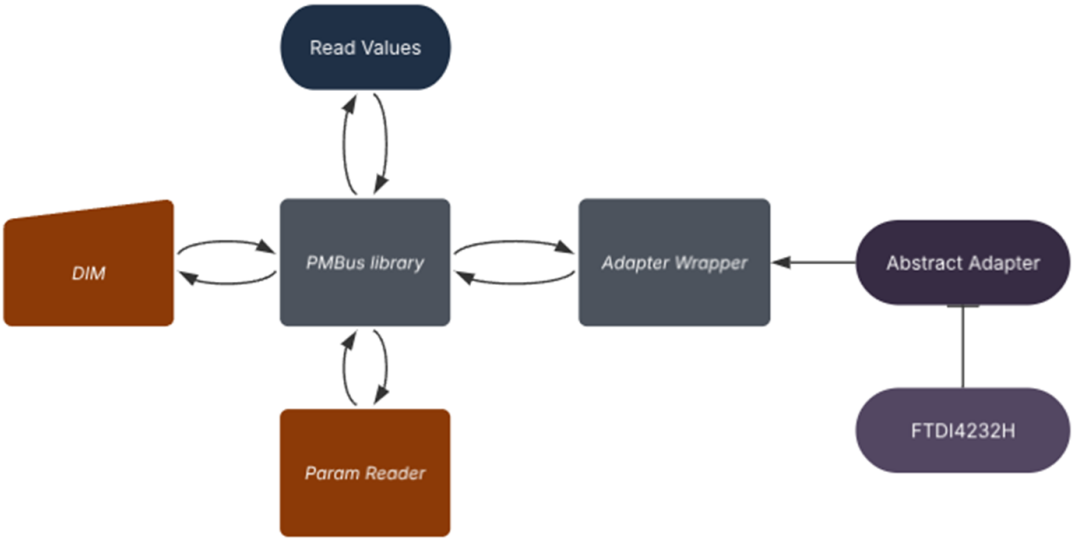

Figure 5: Flux diagram of the I2C(PMBus) - USB interface.

Specifics about each library and the classes contained are developed in details in the next sections.

4.1 Adapter Wrapper library

The Adapter Wrapper library contains and abstract class with the read, write, configuration and port setup functions that would be necessary to any adapter used for the I2C communication. An specific class for the FTDI4232H chip was created with the detailed functions using the Pyftdi library [11].

The adapter is configured to function at $10kHz$ frequency which is 10 times below the PMBus standard and the expected working frequency for the chip. Limitations in this aspect were caused by the clock stretching in the chip. Clock stretching means data is taken in inconsistently at higher frequency rates and, therefore, operation above the $10kHz$ was not possible with this adapter.

The abstract class allows to solve issues of this type as the adapter can easily be switched for a different one that allows higher frequency data exchange.

4.2 PMBus library

The PMBus library consists of four separate classes that target a different aspect of the PMBus decoding and data exchange. The Setup class is ran every time an output is requested from the PM chip; this class requests the part ID, revision and page number form the address. The first two values requested assure the address is real and responding as it should, while the page number informs the Read Values function of the amount of rails it needs to read through.

The Read Values function contains the functions for output reading. Each functions sends the correct code based on PMBus documentation, sorts through the pages and returns the decoded value. Values are decoded in the Format function both for outputs and Status words.

The values relevant for the purpose of the project and the codes are displayed on the table below.

Command Code	**Value**
0x8B	VOUT
0x20	VOUT_MODE
0x8C	IOUT
0x8D	TEMPERATURE_1
0x8E	TEMPERATURE_2
0x8F	TEMPERATURE_3

Table 1: PMBus commands for output values [12]

Voltage values are read using *READ_VOUT* command and decoded using the linear 16 standard

$$V = M \times 2^X \tag{1}$$

;where M is the mantissa value taken from the byte word received using the 0x8B code and X the exponent value read from the adapter using *VOUT_MODE* (0x20) which changes with every rail in the address [12].

Other telemetry outputs such as temperature and current are decoded using lienar11 which utilizes little envy format. The mantissa in this case consists of 11 bits form the word while the exponent constitutes the remaining 5 bits. The output value then is

$$R = M \times 2^X \tag{2}$$

where M is the mantissa, X the exponent and R is the requested value [12].

The Status word is decode based on the PMBus documentation. Flags are raised depending on the bits pulled to high within the word. If all bits are pulled high, the status words returns an *-No faults detected*. Faults on the status byte are raised according to the following table

Bit Mask	**Status Flag**
0x80	OFF
0x40	VOUT_OV_FAULT
0x20	VOUT_UV_FAULT
0x10	IOUT_OC_FAULT
0x08	INPUT_FAULT
0x04	MFR_SPECIFIC
0x02	FAN_FAULT
0x01	TEMPERATURE_FAULT

Table 2: Bitmask decoding of PMBus status byte [12]

Decoding of the Status word is as seen on the following table

Bit Mask (Hex)	**Status Flag**
0x8000	BUSY
0x4000	OFF
0x2000	VOUT_OV_FAULT
0x1000	IOUT_OC_FAULT
0x0800	VIN_UV_FAULT
0x0400	TEMPERATURE
0x0200	CML
0x0100	NONE_OF_ABOVE
0x0080	FAULT
0x0040	INPUT
0x0020	MFR_SPECIFIC
0x0010	PGOOD_NEG
0x0008	PGOOD_POS
0x0004	TON_MAX_FAULT
0x0002	TOFF_MAX_WARN

Table 3: Bitmask decoding of PMBus status word [12]

The output values are sent as hexadecimal and float values to the Param Reader function

or to the DIM server; while the status of the device is sent as flags that can be read from the terminal using the Param Reader function.

4.3 Param Reader library

The main purpose of this library is to display the decode values in a readable manner and to facilitate requests to the adapter. This class, therefore, contains the handlers used from the terminal which are programmed to mirror what was seen from the FusionParamReader function of Texas Instruments. The class contains handlers for specific address reading and address scanning, as well as the valid commands that can be requested and the command to be used from the terminal.

The class loops through the valid addresses once a valid command is received. Inside the loop the port is set again and a function is called on to read the command. The functions calls on the Setup class before using the Read Values class to obtains the hexadecimal and decoded values requested. The results or flags are passed to a printing function.

```
[LAB][altrgdev@acsl-ctpapp pmbus-monitoring]$ python -m ParamReader.main --scan MFR_ID READ_VOUT
CmdIDWithPhase  PageHex SAAStatus       Decoded DecodedNumeric  EncodedHex
UCD90120A       101             MFR_ID          0xFF    ACK             MFR_ID  0x0C4D46525F5245564953494F4E91
UCD90120A       102             MFR_ID          0xFF    ACK             MFR_ID  0x0C4D46525F5245564953494F4ECD
UCD90120A       101     VOUT    0x00    ACK     0.948 V 0.947998        0x3CAC
UCD90120A       101     VOUT    0x01    ACK     0.999 V 0.999268        0x3FF4
UCD90120A       101     VOUT    0x02    ACK     0.947 V 0.947266        0x3CA0
UCD90120A       101     VOUT    0x03    ACK     1.201 V 1.201172        0x4CE0
UCD90120A       101     VOUT    0x04    ACK     1.788 V 1.787842        0x3936
UCD90120A       101     VOUT    0x05    ACK     4.863 V 4.863281        0x1374
UCD90120A       101     VOUT    0x06    ACK     2.503 V 2.50293 0x5018
UCD90120A       101     VOUT    0x07    ACK     0.0 V   0.0     0x0000
UCD90120A       102     VOUT    0x00    ACK     1.802 V 1.801758        0x39A8
UCD90120A       102     VOUT    0x01    ACK     1.794 V 1.793945        0x3968
UCD90120A       102     VOUT    0x02    ACK     1.195 V 1.19458 0x4C74
UCD90120A       102     VOUT    0x03    ACK     1.795 V 1.795166        0x3972
UCD90120A       102     VOUT    0x04    ACK     3.306 V 3.306396        0x34E7
UCD90120A       102     VOUT    0x05    ACK     4.926 V 4.925781        0x13B4
UCD90120A       102     VOUT    0x06    ACK     0.0 V   0.0     0x0000
```

Figure 6: Terminal output from the Param Reader library using the developed interface and the FTDI4232H chip.

In the example above, the –scan handler was used to read both addressed in the ATB and retrieve the revision (MFR_ID) and the output voltage (READ_VOUT). The results are displayed with the part ID, address in decimal, value, the page number, ACK or NACK from the device, the value in float and the value in hexadecimal. The class was built with testing purposes in mind as the telemetry outputs read are published into the DIM for their use.

4.4 DIM server library

The DIM server uses the Pydim library to create services for voltage, temperature, current, address and part ID for the ATB [13]. The code is made for easy change of device address or adapters. Within the class, the device and port are configured and setup once before creating the services and again every time one of the services is called on by the main function.

Voltage and current service functions loop through the pages in every address and create a different service for each page or rail. The name of the services follow a t.value.device.rail

format in which the value corresponds to voltage, current or temperature, the device depends on the number of addresses being read and the rail corresponds to the current page.

The service is updates every 10 seconds in the current interface. This could potentially be reduced, however and due to the clock stretching limitations of the FTDI4232H chip, shorter update times can create inconsistent responses and unstable connection to the DIM.

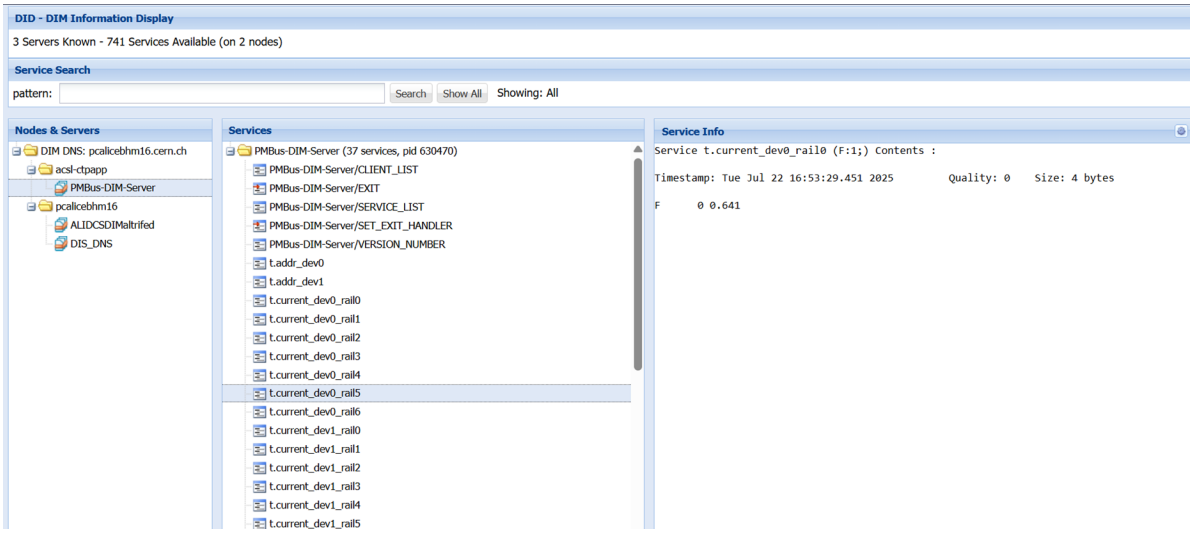

Figure 7: DIM webpage containing the published services form the developed interface using FTDI4232H chip.

The image above displays the service for the current value in page 5 of the device 0, in this case, the 101 decimal address form the ATB used for the testing. The service contains current values for both devices and the addresses; scrolling down the page would reveal voltage and temperature values.There services can then be deployed to the SCADA WinCC dashboard.

5 Conclusion

An I2C(PMBus) to USB interface was successfully developed using the FTDI4232H Mini Module adapter to monitor the PM chips on the ATB. The interface is capable of reading and decoding voltage, current and temperature values in the same way the TI adapter does and publish them as services in the DIM. The values can then be deployed to the SCADA WincCC dashboard as needed for their use. Additionally, the interface can be easily modified to work for other PMBus devices and to work with a different adapter form the one used during the project. The result is a program that can be customized and can operate faster than the current solution given that the adapter used can operate at high frequency rates.

6 Acknowledgments

I would like to thank my project supervisors Filip Pazdic and Roman Lietava for their guidance and support during my stay. In the past two months they have both made me a part of their team and taught me all they could with great patience and kindness, fostering my curiosity and creativity. I would also like to thank the CERN Summer Student Programme for offering me this incredible opportunity.

References

[1] F. Pazdic, 'Central trigger processor,' PowerPoint presentation, unpublished.

[2] F. Pazdic, *Dcs project for ctp run3*, https://twiki.cern.ch/twiki/pub/ALICE/CTP-DCS/DCS_project_for_CTP_Run3.pdf, 2022.

[3] Arduinodocs, *Inter-integrated circuit (i2c) protocol*, https://docs.arduino.cc/learn/communication/wire/.

[4] T. I. Inc., *A basic guide to i2c*, 2022.

[5] I. System Management Interface Forum, *System management bus (smbus) specification v3.3.1*, https://pmbusprod.wpenginepowered.com/wp-content/uploads/2024/10/SMBus_3_3_1_20241020.pdf, 2014.

[6] T. I. Inc., *Usb interface adapter evaluation module user's guide*, https://www.ti.com/lit/ug/sllu349/sllu349.pdf, 2022.

[7] F. T. D. I. Ltd, *Ft4232h mini module datasheet, v1.8*, https://ftdichip.com/wp-content/uploads/2020/07/DS_FT4232H_Mini_Module.pdf, 2012.

[8] Arduinodocs, *Arduino uno r4 wifi qwiic connector*, https://docs.arduino.cc/tutorials/uno-r4-wifi/qwiic/, 2025.

[9] T. I. Inc., *Txs0108e 8-bit bi-directional, level-shifting, voltage translator for open-drain and push-pull applications*, https://cdn.sparkfun.com/assets/1/8/1/e/2/txs0108e.pdf, 2020.

[10] Arduinodocs, *Wire*, https://docs.arduino.cc/language-reference/en/functions/communication/wire/, 2025.

[11] E. Blot, *Pyftdi*, https://eblot.github.io/pyftdi/, 2024.

[12] T. I. Inc., *Ucd90xxx sequencer and system health controller pmbus command reference*, https://www.ti.com/lit/ug/slvu352g/slvu352g.pdf, 2019.

[13] C. L. O. Group, *Pydim documentation*, https://lhcbdoc.web.cern.ch/lhcbdoc/pydim/guide/index.html, 2009.