Visual Data Acquisition System (VDAS)

Eric D'Andria Leal

July 2, 2024

Abstract– This project entails the design and development of an open hardware visual data acquisition (DAQ) system, divided into three phases: hardware, firmware, and software. The hardware phase involves selecting electronic components, the firmware phase focuses on creating a digital circuit for data transmission, and the software phase centers on developing a user-friendly GUI for data display. The final DAQ device, which connects via USB, features 4 ADCs (2 for current and 2 for voltage measurement), 2 DACs, and 16 digital pins (8 input, 8 output).

Keywords– DAC, ADC, FPGA, Data Acquisition

Acronyms and abbreviations–

Acronym	Meaning
DAC	Digital to Analog Converter
ADC	Analog to Digital Converter
GUI	Graphical User Interface
PCB	Printed Circuit Board
DAQ	Data Acquisition
DAS	Data Acquisition System
FPGA	Field Programmable Gate Array
AWG	Arbitrary Waveform Generation
MSPS	Mega Samples Per Second
FSM	Finite State Machine
AST	Abstract Syntax Tree

1 Introduction

When working with electronic circuits, having an efficient data acquisition (DAQ) system is crucial. DAQ systems enable complex data analysis and help in diagnosing issues within circuits. Traditional debugging tools can be cumbersome and often require multiple devices. The goal of this project is to streamline these tools into a single, user-friendly system, specifically designed for low-frequency circuits (less than 1 MHz).

1.1 Project Scope and Objectives

This project aims to develop a comprehensive visual data acquisition system. The core components of this system include a data acquisition device capable of interfacing with a computer, and a desktop graphical user interface (GUI) application. This application will facilitate the transmission and retrieval of data between the DAQ device and the computer. By integrating these components, users will be able to acquire data and control electronic circuits more effectively.

1.2 Advantages of the Unified DAQ System

The proposed unified DAQ system offers several benefits:

- Simplification of Tools: By consolidating multiple debugging and data analysis tools into a single device, the system reduces the complexity and learning curve for users.
- Enhanced Usability: The desktop GUI application provides a visual and intuitive interface, making it easier for users to interact with the DAQ device and analyze data.
- Improved Debugging: The system's ability to perform complex data analysis aids in quickly identifying and resolving issues within electronic circuits.

• Contact e-mail: 1566456@uab.cat
• Menció realitzada: Enginyeria de Computadors
• Treball tutoritzat per: Dolores Isabel Rexachs Del Rosario (CAOS)
• Curs 2023/24

1.3 Potential Applications

While the primary focus of this project is on low-frequency circuits, the versatility of the DAQ system allows for broader applications. For instance, it can be adapted for general-purpose data acquisition tasks in various fields of electronics. In the following sections, we will explore additional use cases and potential enhancements for the DAQ system, demonstrating its flexibility and wide-ranging utility.

By creating a visual data acquisition system that combines hardware and software components into a cohesive package, this project aims to make data acquisition and circuit control more accessible and efficient for a wide range of users.

Also, this project is open hardware, anyone can implement it using the source code provided. Making it accessible and cheap to assemble.

2 State of the Art

Data acquisition systems tailored for electronic circuits play a pivotal role in various fields such as electronics testing, sensor interfacing, and control applications. Recent developments emphasize the need for efficient, flexible, and cost-effective solutions to meet the demands of modern electronics research and industry.

2.1 Integrated Circuit-based Data Acquisition

Integrated circuit (IC) solutions offer compact and integrated data acquisition capabilities suitable for electronic circuits. Analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) integrated into microcontrollers or dedicated ICs provide essential functionalities for signal acquisition and generation [5].

2.2 High-Frequency Data Acquisition

For some applications, the bandwidth offered by standard DACs and ADCs is insufficient to capture high-frequency signals accurately. This is particularly relevant in fields such as quantum computing, where qubit frequencies can reach up to 8 GHz. To address these challenges, systems often use mixers for upconversion and downconversion, allowing higher frequency signals to be processed effectively by the data acquisition system [1].

These high-frequency data acquisition instruments are typically equipped with Python libraries and APIs for seamless interfacing and control. This facilitates integration into experimental setups and allows researchers to automate data collection and processing tasks efficiently [2].

2.3 Sensor Interface and Conditioning

Data acquisition systems for electronic circuits often require precise sensor interfacing and conditioning circuits. Integrated solutions that provide programmable gain amplifiers, filters, and signal conditioning circuits simplify the interface between sensors and data acquisition hardware [4].

2.4 Real-Time Processing and Control

Real-time processing and control capabilities are becoming increasingly important in data acquisition systems for electronic circuits. FPGA-based solutions offer parallel processing capabilities and customizable logic, allowing for real-time signal processing and control tasks.

2.5 Conclusion

Data acquisition for electronic circuits continues to evolve to meet the demands of modern electronics research, testing, and industry. Integrated circuit solutions, high-speed acquisition, high-frequency data handling, sensor interface and conditioning, real-time processing, and wireless integration are key areas of development shaping the future of data acquisition systems.

3 Objectives

The main objective of this project is to create a visual data acquisition system. This system should be able to, at least, implement one of the following use cases.

3.1 Power analyzer

For this use case we will need to measure the current and voltage consumed by the circuit. By using the DAC, we can specify the voltage that we want to be applied to the analog output pin. We can also measure the current that is being put to that same pin. With the combination of both data, we can simply calculate the power consumed as:

$$P = I * V$$

3.2 Controlling a circuit

This is the most typical use case, as it controls the operation of a circuit, with the GUI of the VDAS. For this we can use all the inputs and outputs of the system (digital, DAC, ADC, filtered...).

3.3 Inductance meter

Measuring inductance values is often useful when we are dealing with custom inductors, electromagnets, transformers, or other kinds of inductive devices. These inductors (especially if they are small) are not detected or measured with bad precision by general purpose cheap transistor testers that you can find on the internet. The inductance can be calculated from the frequency response of the circuit, when connected to a capacitor of known value in parallel and applying a pulse signal to that tank circuit.

3.4 Circuit verification

When an electronic circuit needs to be manufactured repeatedly it is a clever idea to automate the tests to ensure its correct operation. We will define some tests for a circuit with tolerances to ensure easily that a circuit is working as expected.

3.5 Logic analyzer

This use case consists of configuring the VDAS to sniff and display messages of a digital communication protocol. We can use I2C because it has only two wires and is simple.

4 Methodology

For this project I will use a slightly modified version of the V-Model methodology, as it makes the verification and the integration easier, and because I am working alone there is no need to use any agile methodology. In figure 1 we can see the methodology used.

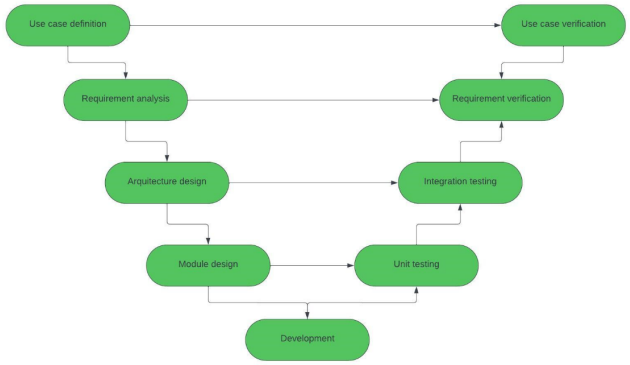

Fig. 1: Methodology diagram

Each one of the phases explained:

- **Requirement analysis**: In this phase the requirements are analyzed and written down, with tests to verify them.
- **Architecture design**: This is the first approach to define the functional blocks of the application and their interaction.
- **Module design**: This is a more in detail definition of the modules that compose the previous blocks. This also includes simulations.
- **Development**: Here the real development of the project starts.
- **Unit testing**: Once the modules are implemented, they must be tested one by one.
- **Integration testing**: The individual modules are connected between them, and it ensures the correct interaction between them.
- **Requirement verification**: Finally, the requirement tests are executed to ensure the correct behavior of the system.
- **Use case verification**: Implement some use case and verify that works as expected.

5 Requirements

In this section, I'll specify all the requirements and tests needed for this project. At the table 1 can be found the functional requirements.

5.1 Functional

The functional requirements can be found at the table 1.

Requirement ID	Description	Associated test
REQ01	The system shall be capable of generating an arbitrary voltage	TEST01, TEST03
REQ02	The system shall be capable of reading analog voltage	TEST02
REQ03	The system shall be capable of reading and generating TTL voltage	TEST01, TEST02
REQ04	The system shall be capable of measuring the output current	TEST04
REQ05	The system shall have information about the time passed since start of measurement	TEST03
REQ06	The system shall allow choosing the voltage references for analog readings	TEST01
REQ07	The system shall have implemented a calibration algorithm	TEST01, TEST02, TEST03
REQ08	The system shall have capability to do AWG from a file	TEST05
REQ09	The system shall export data to files	TEST03
REQ10	The system shall have trigger conditions	TEST03

Tab. 1: Functional requirements

5.2 Non functional

At the table 2 can be found the non functional requirements of the system.

Requirement ID	Description
NFREQ01	The system shall be modular to allow modifications
NFREQ02	The system shall use precision shunt resistors to measure currents

Tab. 2: Non functional requirements

5.3 Test plan

For the test plan, some tests have been created, to verify the behaviour of the system. These tests can be found at table 3.

Test ID	Description
TEST01	Measurement of the output voltage with precision multimeter
TEST02	Generation of voltage with precision source
TEST03	Recording voltage of a sine generated with oscillator
TEST04	Load the system with a precision load
TEST05	Output measurement with oscilloscope

Tab. 3: Test plan

6 Architecture design

A logic architecture of the system can be found in the figure 2. In this architecture we find the following components:

- **Circuit**: This can be any circuit with input or output signals.
- **DAS**: The Data Acquisition System, the interface between the circuit and the computer program.
- **GUI**: The Graphical User Interface that enables the user to interact and see the information of the circuit, is composed by the following parts
  - **Program**: The program that orchestrates the communication between the computer and the VDAS. Also sends the program to the DAS.
  - **Visual**: This is the interface that uses the user to interact to the circuit and see the information.
  - **Design**: The interface that allows the user to program the application and design the graphical display.
- **User**: The end user of the VDAS.

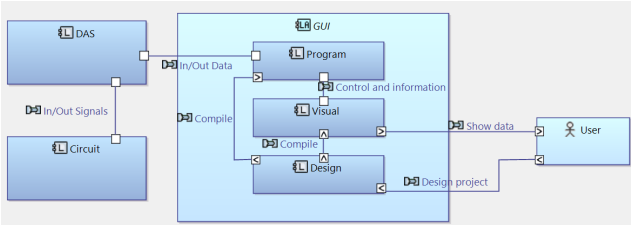

Fig. 2: Logic architecture of the system

7 Module Design

7.1 DAS

This is the first approximation for the design of the data acquisition system. We can see that the core of the system is the FPGA. This has an internal memory where stores the program that is sent via USB from the computer. In the figure 3, there are 2 input ADCs, 2 DACs, and a shunt resistor for each DAC, connected to a current sensing ADC. Finally, we have 8 digital inputs and 8 digital outputs.

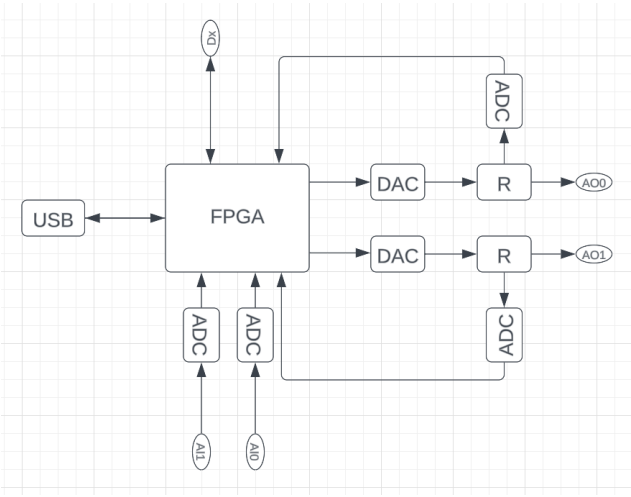

Fig. 3: DAS block diagram

7.2 GUI

For the GUI PySide6 will be used, and the development is separated in three distinct phases. These phases are:

- Graphical part: This comprises the development of the main window that allows the user to write and define the setup.
- Language definition: This is the custom language that the user will use to define the setup.
- Setup generation: This is the last part, it comprises the compilation of the language and generation of the visual setup to interface with the DAS. This will generate a grid with the visual components.

8 Development

8.1 Hardware

In this section, we outline the design of the hardware component of the project, focusing on the selection of components and the creation of detailed connection diagrams. Additionally, electronic simulations will be conducted to verify circuit behavior, ensuring robustness and functionality.

8.1.1 Core Component: FPGA Selection

The cornerstone of our data acquisition (DAQ) system is the Field-Programmable Gate Array (FPGA). Given the wide array of FPGAs available, we have chosen the EP4CE6E22C8N. This selection is informed by our prior experience with this model, facilitated by an existing development kit. The EP4CE6E22C8N offers a maximum operating speed of 200 MHz and more than 100 input/output (I/O) pins, which adequately meets our project requirements. Its integration with other components is straightforward, thanks to the familiarity and tools we possess.

8.1.2 Interface Circuits: ADCs and DACs

The interface circuits, including Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs), are critical for converting analog signals to digital data and vice versa. The selection criteria for these components focus on sampling frequency and resolution. Higher values for both parameters enhance performance but also increase cost. Additionally, the FPGA's 200 MHz speed imposes a bandwidth constraint on these components.

For the ADCs, we categorize them into two types: input ADCs and current-sensing ADCs, each serving distinct roles within the system.

8.1.3 Current-Sensing ADC

We have selected the ISOSD61 for current sensing. This component is tailored for current measurement applications, featuring a two-wire interface (clock and data output) and delta-sigma modulation, which necessitates additional filtering within the FPGA. The resolution and speed of the ISOSD61 depend on the filter's decimation factor, allowing for flexibility in performance tuning.

8.1.4 Input ADC

The ADC10040 is chosen for input signal conversion. It provides a 10-bit resolution and a throughput of 40 mega samples per second (MSPS), which is sufficient for most general applications. This ADC strikes a balance between performance and cost, making it a suitable choice for our needs.

8.1.5 DAC

For digital-to-analog conversion, the AD5445YRUZ is selected. This DAC offers a 12-bit resolution and a throughput of 20 MSPS. Its current-based output allows for flexible reference voltage configuration, enhancing the adaptability of our system.

8.1.6 Packaging Considerations

When selecting IC packages, ease of soldering is a key consideration. We avoid packages with pins underneath the chip surface, such as Ball Grid Array (BGA), due to the

difficulty in soldering without specialized equipment. Instead, we focus on packages like MSOP (Mini Small Outline Package), which are easier to handle and solder.

8.1.7 USB Connectivity

For USB connectivity, we will utilize the serial port on the FPGA development board, coupled with an RS232 to USB cable. This setup facilitates straightforward communication between the FPGA and a host computer, ensuring reliable data transfer.

8.1.8 Summary of Component Characteristics

The selected components and their characteristics are summarized in Table 4. This table provides a clear overview of the key specifications for each component, aiding in the overall design and integration process.

Purpose	Chip ID	# bits	Throughput (MSPS)
Current ADC	ISOSD61	Filter dependent	Filter dependent
Voltage ADC	ADC10040	10	40
DAC	AD5445YRUZ	12	20

Tab. 4: ICs CHARACTERISTICS

By carefully selecting and integrating these components, we ensure that the hardware design meets the project's performance requirements and is aligned with our technical capabilities and resources.

8.2 Software

8.2.1 VLang

Previous to begin writing the software part we have to define the language. We have three types of sentences, definitions, initializations and links.

- **Definitions**: Are used to define constants to use the rest of the program, and they have the following syntax: "$define < name >< value >$", for example "define voltage 5".
- **Initilizations**: Are used to initialize either hardware components or visual components, the syntax is: "$< type >< name > (< arguments >)$", for example "Plot p(label='Example plot')". The accepted types are explained in the table 5.
- **Links**: Link the data from input components to output components. The syntax is: "$< expression > - >< component >$", like "$cadc0 * voltage - 3 - > DAC0$".

Type	Description
ADCX	Activate an ADC, change X by the ADC number (0 or 1)
DACX	Activate a DAC, change X by the DAC number (0 or 1)
CurrentADCX	Activate a current ADC, change X by the ADC number (0 or 1)
DIX	Activate a digital input, change X by the digital input number (0 to 7)
DOX	Activate a digital output, change X by the digital output number (0 to 7)
Plot	A plot to display data
NumDisplay	Numerical display
NumInput	Numerical input
BoolDisplay	Boolean display
BoolInput	Boolean input

Tab. 5: COMPONENT TYPES

8.2.2 VASM

The VASM is an assembly language, created for the FPGA control unit. This language is compressed into a byte stream that is sent to the FPGA via serial protocol. With this instructions, we can activate and control the hardware components connected to the FPGA. At the moment, it features the instructions stated at table 6.

Instruction	Arguments	Comments
Activate	component	Activates the input stated in component argument
SetDigital	component, value	Sets the digital output to the stated value
SetAnalog	component, value	Sets the analog output to the sated value

Tab. 6: VASM INSTRUCTIONS

8.2.3 Language

For the software development let's begin with the language definition. The compiler for this language uses OCaml and Menhir [7]. Menhir is LR(1) parser generator, this means that takes a semantic and lexical description of the language and generates OCaml code that can interpret the text and return the AST representation. Then we can work with this AST to generate the "ui" file that will be fed to the GUI.

The syntax of the language is very simple. In the figure 4 can be seen the code of the AST.ml file, which creates the "ins_t" type, that is the return type of the parser.

First, we have the "comp_t" type, that represents a component, can be either only the component name, or the component and attribute. Then, we have the "bop_t", used for representing the binary operations. Then the "expr_t", that can be, a binary operation, a component, or a constant. Finally, the "ins_t" type that holds and instruction, and can be a initialization, with the type, name and attribute info, a definition, with name and value, or a link, with a expression and a component.

```
type comp_t = (string, (string * string)) Either.t

type bop_t = Add | Sub | Mul | Div | Pow

type expr_t = Op of bop_t * expr_t * expr_t |
              Comp of comp_t |
              Const of float

type ins_t = Init of (string * string * ((string * string) list)) | (* type, name, arguments *)
             Def of (string * float) | (* name, value *)
             Link of (expr_t * comp_t)
```

Fig. 4: Text contents from AST.ml file

The OCaml code generates two binaries, one for the language server, that outputs the syntax errors for the IDE output, and the compiler itself, that generates the FPGA initial instructions and a UI file for the Python code.

8.2.4 GUI

Then using Python and PySide6 library we can create a simple IDE, with an input to edit the code, and an output to see the syntax errors. The language server will be running every time the text is modified, and the compiler will be executed when the compile button is clicked. Once the program is compiled, another python script will load the UI file generated.

To interface with the machine, a FSM is used. This FSM waits in the "start" state until it receives a 0 byte, then it

moves to the next state, which is ”type”. In this state classifies the input data into one of the 7 possibles devices using the incoming byte that the FPGA sends with the identifier. Lastly, it goes to the ”data” state, where waits for the necessary bytes, depending on the bits of the device, and returns the data and the type.

The interface is shown in figure 5. In the upper part is where the code is written, the lower part is for the compiler output. Under the ”File” menu, the options to create, load, save and compile the file can be found.

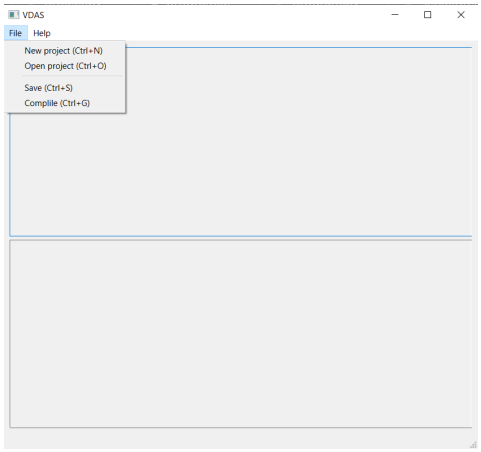

Fig. 5: GUI display

8.3 Firmware

This part is composed by the code of the FPGA needed to do the data transfer to the computer. Before writing the actual code we need to make sure that everything is well defined and verified.

Let’s begin with a block diagram of the internal circuit that we want to design at the FPGA.

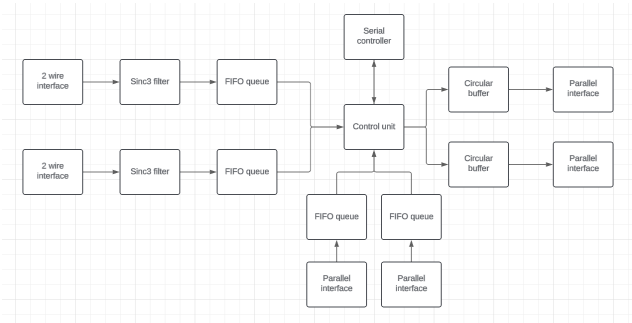

Fig. 6: Firmware block diagram

First, to in order to communicate the FPGA with the computer, we need a serial controller[8]. Then, this connects with the control unit that orchestrates the data flow. After the control unit, the path is divided between the input (ADCs) and the output (DACs). The input path goes to separated FIFO queues, where the data coming from the ADC controllers will be stored and will wait to be sent to the PC. Then we have the output path, that is connected to circular buffers. This is to have AWG capabilities on the DACs. On the current ADC path we also need a decimator digital Sinc3 fiter[6].

The controller interfaces are selected based on the interfaces of the selected ICs in the hardware section.

For the controller, the FSM that summarizes the behavior can be seen in the figure 7. The controller initializes and goes to the receive state. In this state if the receive buffer size is not 0, it reads the buffer and executes the instruction, if it’s 0, goes to the send state. In the send state, if there is data in the FIFO queues it sends the data, if not it returns to the receive state. For sending the data it first sends one byte of zeros, then a byte containing an identifier to know where the data comes from, and, depending on which data is sending, one or two bytes with the data.

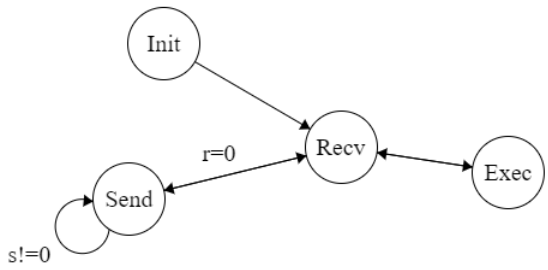

Fig. 7: Controller FSM

Once written, the firmware written in Verilog generates the digital circuit shown in figure 8. Here we can see clearly all the stages that the data passes through.

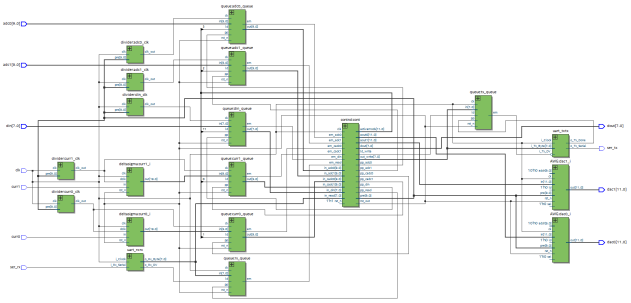

Fig. 8: Circuit generated by Quartus Prime synthesizer

9 Results

Due to time constraints, the implementation of the circuit was not possible, so the tests of the full system can not be performed. Even this, each module in the firmware is tested by its own testbench.

To test the compilation of the code, I wrote a simple VDAS script that can be seen in the figure 9, and produces the output shown on figure 10.

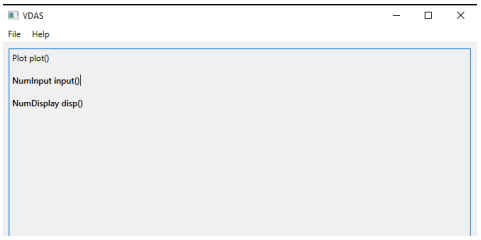

Fig. 9: Test script

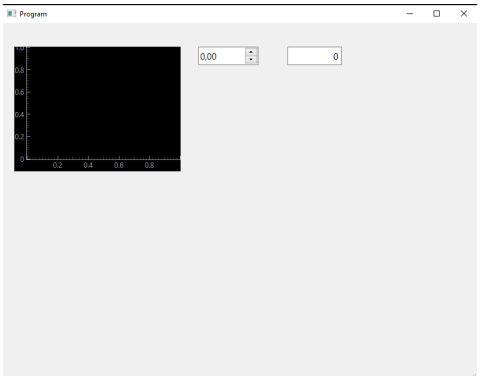

Fig. 10: Output of the program on figure 9

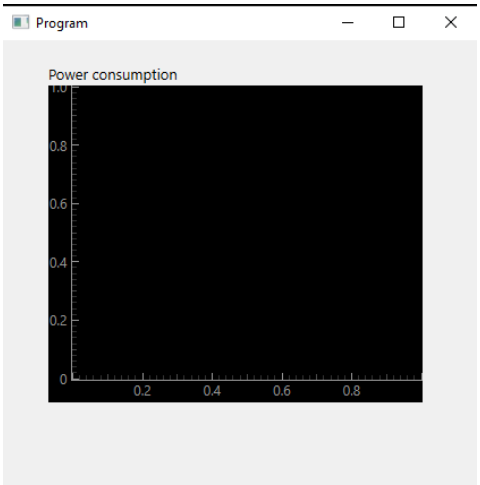

Fig. 11: Power analyser program

As it is right now, the implementation for the power analyser produces the output on figure 11 would be this one:

```
define voltage 5

Plot power(label="Power consumption")

DAC0 source()

CurrentADC0 current()

voltage -> source

current * voltage -> power
```

10 Conclusions

10.1 Summary of Work

In this project, we successfully implemented the full stack of a data acquisition system, encompassing data visualization and user interaction, and seamlessly integrating hardware with software. We designed the circuit, developed firmware for the FPGA, created a program to facilitate communication between the computer and the FPGA, and built a compiler for a custom language.

While this project serves primarily as a proof of concept rather than a finished product, it demonstrates significant potential to evolve into a valuable tool for working with electronic circuits.

10.2 Further improvements

One of the key decisions was the adoption of a serial protocol for data transmission between the FPGA and the computer, instead of implementing a USB controller. This choice was driven by time constraints, making the implementation simpler but resulting in a lower data transmission rate. The hardware of the DAQ system is capable of much higher time resolution if the USB protocol is properly implemented. Future iterations should consider this to fully leverage the system's capabilities.

The GUI was developed using two programming languages, with Python chosen for its ease of implementation, saving significant development time. However, this approach complicated integration between the programming language and the GUI components. Writing the entire GUI in a single language, such as OCaml, would lead to more natural and straightforward integration, improving both the development process and the user experience.

In terms of the programming language used for the project, introducing variables would avoid the repetition of expressions, enhancing code maintainability. Additionally, adding more arguments for components would provide greater design flexibility, allowing for more customized and robust system configurations.

The choice of the Sigma-Delta ADC, while not poor, was not optimal either. It requires a filtering stage within the FPGA, consuming more digital resources. Although it supports a clock rate of up to 25 MHz, the necessary decimation ratio for achieving decent precision significantly reduces the ADC's effective throughput. Future designs should consider alternative ADCs that demand less filtering and offer higher effective throughput to optimize FPGA resource utilization.

The project's time constraints posed a significant challenge, influencing the decisions to adopt simpler protocols and languages at the expense of performance and integration complexity. The reliance on the Sigma-Delta ADC required additional digital resources for filtering, impacting overall system efficiency. Additionally, using multiple programming languages for the GUI introduced integration complexities, which could be mitigated in future designs by standardizing on a single language.

Looking ahead, implementing the USB protocol for data transmission should be a priority to significantly improve data rates and time resolution capabilities. Developing the entire GUI in a single language, such as OCaml, would streamline integration and enhance the user experience. Furthermore, introducing variables and additional component arguments in the programming language would improve flexibility and reduce redundancy. Exploring and integrating alternative ADCs that require less filtering and provide higher effective throughput will optimize resource utilization and enhance system performance.

Overall, this project successfully addressed key aspects of DAQ system design, striking a balance between ease of implementation and performance requirements. While certain trade-offs were necessary due to time constraints, the foundational design demonstrates significant potential for

future enhancements. By addressing the identified challenges and focusing on the proposed improvements, the DAQ system can achieve higher efficiency, better integration, and enhanced overall performance.

10.3 Final Remarks

Overall, the project successfully met its goals of designing a reliable, efficient, and cost-effective DAQ system. The chosen components and design approach provide a solid foundation for various applications, demonstrating the potential impact of this system in enhancing data acquisition and analysis processes. Future work will focus on refining the design and exploring new functionalities to further improve the system's performance and applicability.

REFERENCES

[1] *Measurement and control of a superconducting quantum processor with a fully-integrated radio-frequency system on a chip*

[2] *PyVISA: Control your instruments with Python*

[3] D. Bucci, *Analog Electronics for Measuring Systems*, Wiley.

[4] Maxim Integrated. (n.d.). *Use High-Performance Simultaneous-Sampling ADCs for Sensor Signal Conditioning in Industrial Multichannel Data Acquisition Systems (DASs)*

[5] W. Takao, *Introduction to analog-to-digital converters principles and circuit implementation*, Japan: River Publishers

[6] M. Oljaca, T. Hendrick, Combining the ADS1202 with an FPGA Digital Filter for Current Measurement in Motor Control Applications

[7] P. François, R. Yann, *Menhir Reference Manual (version 20230608)*, France

[8] NandLand, UART, Serial Port, RS-232 Interface

APPENDIX

Due to the size of the code it can not be included here. All the code and schematics can be found in the GitHub repository.