

Shorten development time of Functional Testing (FCT) in electronic manufacturing with smart instruments

Niklas Gustafson



LUND
UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6151
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

© 2021 by Niklas Gustafson. All rights reserved.
Printed in Sweden by Tryckeriet i E-huset
Lund 2021

Abstract

The development of electrical circuit boards is increasing and the time spent in the development stage can be quite long. The thesis has tried to shorten the time spent developing new electrical circuit boards, by first finding a common denominator for efficiency losses in the development process and then implementing an example solution. The problem identified was the lack of attention towards production testing early enough in the development process. The testing equipment often comes as an after thought which delays the start of production. Since the product can not be mass produced before quality assurance can be done using testing equipment, this will lead to unnecessary time spent with products in development. Another problem addressed in the thesis is the expenses of acquiring a testing system. Today's standard for testing systems is both big and expensive and a better solution would be beneficial.

To solve these problem the thesis purposes a solution using a micro controller connected to a small analog to digital converter. This solution is a lot cheaper compared to today's standard and offer additional benefits described further in the thesis. The drawbacks of loss of performance compared to the more expensive equipment is also discussed.

To test said solution, three example implementations of the same smart instrument was done and evaluated. The implementation was then evaluated both on the possible time saved from using a smarter instrument compared to the same instrument without the smart API implementation and the performance of each of the three possible implementations.

The example implementation resulted in shorter development times and easier interaction with the instrument. It also fulfilled most of the requirements set on a smart instrument in the beginning of the thesis. Performance wise the implemented example instrument was good enough for most production testing situations and is therefore able to replace the more expensive versions for some applications.

Acknowledgements

Foremost, I would like to express my sincere gratitude to Mikrodust and everyone at the office for all the help, access to equipment and guidance I have got during the thesis. To be able to be at the office and get support when things were not going smoothly was much appreciated. An extra thanks to Mats Iderup, Tobias Petersen and Linus Hellman for all their efforts in helping me achieve a successful thesis.

I would also like to thank my handler and examiner, Johan Eker and Giacomo Como for guidance in the directions of the thesis and support during the writing of the thesis.

Contents

1. Introduction	9
2. Background	11
2.1 Need for testing equipment	11
2.2 Product Development Cycle	11
2.3 Functional testing platforms	12
2.4 Need for smarter testing equipment	13
2.5 Smarter automatic test platform	14
2.6 Smart instrument	16
2.7 Setup of a test sequence	17
3. Method	19
3.1 Aim of the thesis	19
3.2 Implementation of smart instrument	19
3.3 Evaluation of the implemented system	20
4. Example Implementation of a Smart Instrument	21
4.1 Problem description	21
4.2 Project objective	24
4.3 Nyquist Sampling Theorem	25
4.4 Coding for micro-controllers and embedded systems	25
4.5 Micropython	26
4.6 REPL	27
4.7 STM32	27
4.8 Peripheral Interface	29
4.9 Direct Memory Access	31
4.10 Implementation Timeline	31
4.11 Evaluating the API	33
5. Result	37
5.1 Functionality of the instrument	37
5.2 Sampling capabilities using the API implementation	38
5.3 The API implication on development times	39

6. Discussion	40
6.1 Hardware limitations in the example instrument	40
6.2 Optimization of implementation code	40
6.3 Implications on development times	41
6.4 Performance of the example instrument	41
6.5 Use of Micropython for commercial use	42
6.6 Further work	42
7. Conclusion	44
Bibliography	45

1

Introduction

Electronic devices are getting more and more common in every aspect of our lives. The digitalization of products which formerly did not use any electronics is growing at a rapid rate and there are no signs of the transformation slowing down anytime soon. One example of this is the growth in sales of smart devices such as refrigerators, coffee makers and other household appliances. These products are using electronics, such as micro-controllers, to add smart features and enable appliances to connect to the internet. Two types of workloads have increased in response to this phenomena, construction of the electronics boards located inside the devices and the software development of said devices. By increasing the number of manufactured circuits boards in the world, the demand for testing equipment is consistently high. Today the testing equipment used is expensive, big and more difficult than necessary to work with. If there was a smarter way to handle the increasing demand of testing systems and make them cheaper, smaller and easier to work with, it would benefit both the customers in lower prices but primarily the developing companies.

The thesis includes investigations of product development cycles and how a smarter and more efficient testing system would have an impact on the development times of new testing fixtures. The focus was mostly directed towards shortening the setup times of test sequences and enable a smarter integration of automatic testing systems. A smart device is often regarded as a device being able to connect remotely, send and receive data, and make tasks more easily manageable. The thesis will then investigate the possibility to use multiple smart instruments connected together and therefore creating a smart system of devices. When investigating the possibility of constructing a smarter testing system an implementation of a smart instrument focused on the problem area was explored.

The implementation was focused on the programming of embedded controllers, specifically for the purpose of analog-to-digital conversion and to attain a lower setup time for each programmed micro-controller. This will include both getting high-level code to run as efficiently as possible and interacting with low level programming languages for better performance on the chip. Better performance was in

the thesis defined as less clock cycles or time to perform the same task. By using a high-level programming language for writing board task specific code that utilizes modules of a lower-level programming language to keep the performance high, the thesis evaluated the possibility to speed up the configuration process of setting up a analog-to-digital converter connected to a micro-controller.

The thesis was conducted in collaborations with Mikrosdust AB located in the norther part of Lund in Sweden. Mikrodust is a small company consisting of around 15 employees and is currently expanding. The company was started in 2011 with the aim to design and sell radio modules that could be used in a multiple of small remote low-powered devices. Since then its business has evolved into including project based consultant work and customized test fixtures for commercial circuit board testing. The interest for the future of testing instruments and smarter testing platform gives both parties an interest in collaborating to achieve a smarter testing instrument and a more efficient way of developing and setting up smart testing platforms. By having the guidance of a company currently active in the business a more realistic view on how the functional testing platform scene operates. This also helps the thesis to focus on the correct bottlenecks of the current standard and what could potentially be improved up on.

2

Background

2.1 Need for testing equipment

The world is using more electronics every year and the production is steadily rising. When circuit boards are being produced, the units need to be tested to ensure their functionality. One of the ways used to test circuit boards, are to manually measure on pre-specified test points on the boards. The manual testing process can be time consuming and costly. A person measuring a lot of test points on a board, will take a long time and the equipment used can be expensive. Both multi-meters and oscilloscopes are commonly used and manually operated during small batch production or prototyping. It introduces a lot of problems to manually test circuit boards when the production volume increases. For example, a human occasionally makes errors due to lack of attention to the task and will also take significantly longer time if there are multiple measuring points on the board. As a direct consequence of the increased number of manual hours, the salary expenses for the company will increase linearly. Therefore, gains are to be made by automating the testing of commercially produced boards that will be produced in large quantities. To analyze the development of a testing fixtures impact on the total product development times, the current industry standard will be introduced.

2.2 Product Development Cycle

During the development of a new product there are multiple steps below. Each product development cycle is unique and what will be described here is a generalization of the development of an electrical circuit board for an unspecified use case.

2.3 Functional testing platforms

Hardware

There are already a solution to the problem of manual testing described above. Using a clam-shell like testing rig, which attaches testing needles onto the circuit boards testing points. The attached instruments are measuring and evaluating each test point using a test sequence. The testing instruments are arranged in 19 inch racks under the testing fixture (see figure 2.1). This is seen as today's manufacturing standard and manual testing is kept to prototyping or really small batches were setting up a functional test system would either be too expensive or time-consuming.



Figure 2.1 Displaying a test fixture with instruments inside a 19 inch rack.

As can be seen from the figure, the rack takes up a lot of space and is not particularly portable to move around. Under the test fixture all the testing equipment used for this specific case, can be observed. The equipment used is for example an oscilloscope or similar instruments, which is attached to a computer that handles all the testing sequences. The equipment is both expensive and takes up a lot of space, but have the benefit of being precise and able to handle high speed measurements. These testing rigs are considered the standard for circuit board production testing today. Offering companies producing electrical products a more effective way of testing the produced boards compared to doing so manually.

Software

To connect all the testing equipment and measuring instruments of a traditional Functional Testing platforms, a program called TestStand can be used, which is one of the most common automated testing platforms. TestStand is one of the most used software for configuring an automated testing system and is provided by the company National Instruments. The program is only compatible with computers run-

ning windows as the operating system, and therefore testers running mac and Linux are left out. Since software developers prefer to use different operating systems for developing purposes, this is a drawback of TestStand. According to Statista the preference was, 2020, distributed as 60% Windows, 44% macOS and 50% Linux [n.a., 2021a]. Noting that the percentages add up to more than 100% reflects that a lot of software developers using multiple operating systems for development. by combining macOS and Linux more software developers prefer using another system than windows. Solving this issue was not necessarily a requirement of the thesis but was considered an added bonus.

2.4 Need for smarter testing equipment

When developing new electronic devices, the product cycle can be divided into four stages: Research and development, ascent, maturity and decline. Research and development are the first part of the technology life cycle path and it is during this phase all the prototyping and product development is done. Ascent is the phase where the product is starting to gain market shares and is introduced to the customers. Maturity consists of the period where the product is well known and has a high and stable demand. And lastly the decline phase when the product starts to get irrelevant on the market and therefore loses shares of the market space.

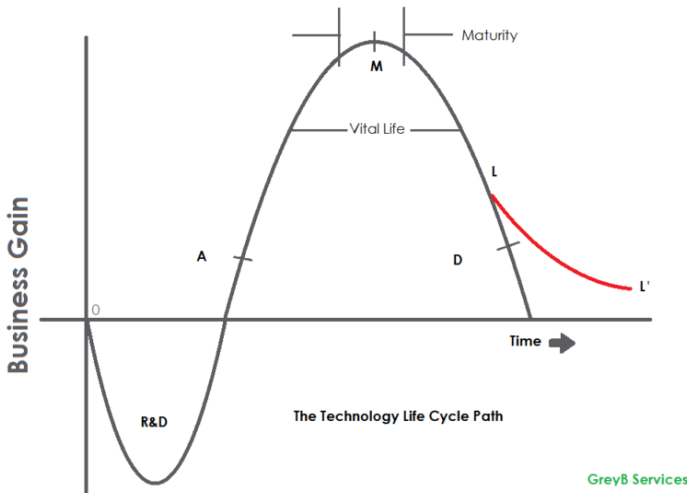


Figure 2.2 Technology Life Cycle. Source: [Sahni, 2021]

During all the discussed stages of the technology life cycle, except the research and development phase, the product is producing an income for the company (see figure 2.2). The R&D phase is not only losing the company money but actually

costing them money, therefore, to be able to reduce the amount of time spent in this stage would be highly beneficial. There are many things to focus on when trying to shorten the R&D period of a products life cycle and the thesis mainly focused on the last part of the process, namely testing.

Testing of a newly developed product is one of the key things which often is forgotten during the development of the product. While development time of the testing system is not as crucial if started at an appropriate time, it still affects the available resources for product development. When the testing system is forgotten and must be developed after the product is ready for production, every day spent on constructing a reliable testing system cost money in terms of being stuck in the R&D phase. If a more efficient way of developing test systems and a more modular design was achieved, this phase could be shortened and therefore increase the profit margins on products needing testing.

Another downside to the current system that would have to be solved with a smart instrument, would be the possibility to easily configure and control the testing equipment from a remote location. This would enable the programmers for the testing sequence to be on a different site and not have to travel to the testing facility. This would not only save the travel times to and from different offices but also enable a central unit responsible to setup a testing sequence remotely instead of having multiple units spread across all areas needing testing equipment. It should be noted that the current standard of using TestStand supports remote control of the system. However, the process of doing firmware updates remote in the current system is seen as complex and time consuming and therefore is an area where a smarter approach could be implemented.

2.5 Smarter automatic test platform

To make the testing process more efficient, an automatic testing system could, as described above, be used. By using a FCT device, see figure 2.1, to attach the connection for the oscilloscope and multi-meters, manual working hours can be reduced. Today the standard is to use a full-scale oscilloscope in more advanced testing processes. In a lot of testing situations, the equipment used are over-powered for the precision needed by the testing sequence. This introduces a couple of drawbacks by having bigger and more expensive equipment than necessary. By introducing smaller instruments, all the instruments can be mounted inside the testing fixture instead of having them mounted under the fixture in a rack. This would significantly reduce the space needed for a testing fixture and make it easier to move to another location. By only using smart instruments inside the test fixture it would also enable remote update of firmware updates and reconfiguration of the testing sequence. An example use case would be a company with multiple factories spread around the country, updating all their testing fixtures at once using a web interface. This would

save considerable time compared to sending out an employee to update each rig manually.

An example of a smaller design of an empty Functional Testing Platform can be seen in figure 2.3 and 2.4.

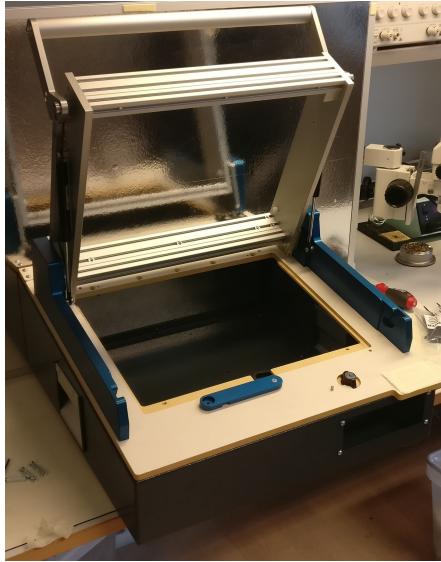


Figure 2.3 Showing a smaller test-platform design open.

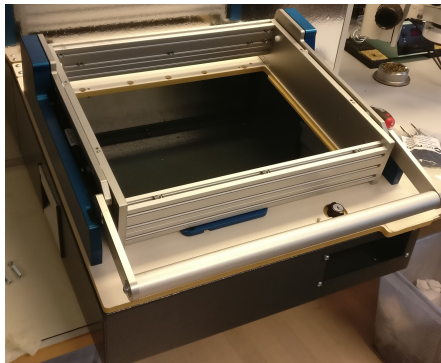


Figure 2.4 Showing a smaller test-platform design closed.

It consists of a clam-shell box which the circuit board to be tested is inserted

into. When the board is placed in the correct position inside the testing platform, the lid closes to attach the testing needles to the boards testing points. After the lid is closed, the testing sequence starts and transfers its results to an attached computer. Depending on the testing results for the circuit board, either a pass or fail can be given. Then the tested board can be removed from the testing box and be swapped for a new board to be tested. The workflow is summarized in the picture below (see figure 2.5).

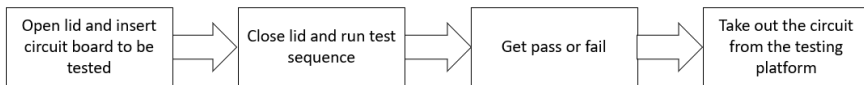


Figure 2.5 Workflow for testing a circuit board using Mikrodusts' testplatform

When designing a more automatic testing system many design and component aspects has to be considered. The system must fulfill all the requirements for the testing purpose and still be cheap enough to be a compelling option. Since a traditional oscilloscope is quite expensive, to find another way to measure with adequate precision and speed would be beneficial. This is where the smart instruments described above could be used as a substitute. Using the oscilloscope as an example of a testing system which could be subsidized for a smaller and smarter alternative, the use of a Micro controller unit and an analog-to-digital convert will be described below.

2.6 Smart instrument

In an attempt to save space and cut down on cost, an micro controller unit and small analog to digital converter could potentially be used instead of an oscilloscope. This combination is significantly cheaper and smaller compared to a traditional oscilloscope and is easier to include in an embedded system.

This approach has some drawback and limitations, not present using an oscilloscope. A small micro controller will have limitations in both processing power and available memory. To achieve optimum performance from a micro controller, it is important to minimize the effect of these constraints. However, since the functional testing platform is used to verify the production quality and not the products itself, the extra performance present in the oscilloscope is in most cases unused.

2.7 Setup of a test sequence

With a rising demand on testing equipment test fixture companies need to speed up their setup time for new testing equipment. In an attempt to make the setup a more efficient process the workflow of the setup will be described and evaluated. The thesis focused on speeding up one part of the workflow to try and make it more efficient.

Workflow of setup

The setup of a new testing box for a client can be split into the hardware and the software setups. First the hardware will be setup and afterwards the software. These are not done fully in a serial fashion, but done parallel and the described way is just a generalisation.

Hardware During the setup of the hardware for the testing box a multiple of variables has to be taken into consideration. First and foremost, the box has to be large enough to fit the circuit to be tested. The size is not only determined by the size of the circuit board, but also needs to take how much testing equipment is needed to fit inside the test box. If the required testing equipment does not fit inside the box the testing requirements would not be met. When an appropriate box is chosen all the necessary testing equipment is fitted inside the box and needles are positioned to fit the testing points on the circuit board.

Software When all the hardware requirements are fulfilled and decided, it is time to set up the software to run the test fixture. It starts by deciding what tests should be done on each circuit and determine the error margin each test result can have. After the specifications are decided, the software development can start. The software consists of three parts, the configuration of all the testing instruments, the testing sequence and the evaluation of the results. The first two parts have to be done on some kind of controller inside the test fixture and the evaluation part can be done either inside the test fixture or on an external computer that is attached to the testing system.

When using the program TestStand for configuring and setting up a test sequence, all the configuration is done on a windows computer connected to the instruments. The configuration of the instruments is sent from the attached computer to the instrument which then modifies its parameters.

Even though a smart instrument could use a micro controller unit as described above, this would not implicate a smart system. By using C-code to setup the testing sequence the micro controller would act as a normal but smaller than standard measuring instrument. If all the code for the micro-controller inside the test fixture is written in C-code, it has to be compiled and flashed to the chip after each modification of the code.

A block diagram of the workflow can be seen in figure 2.6.

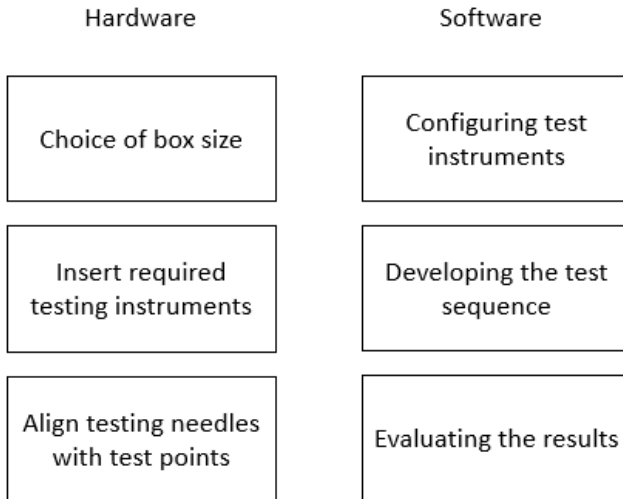


Figure 2.6 Workflow for the development of the test fixture. Starting with hardware and then software.

To speed up this process the idea of using a more high level programming language to keep software development times as low as possible, was purposed in the thesis as a solution. It also has to consider possible performance losses of using a more high level programming language and weight them against the time saved in development times.

3

Method

3.1 Aim of the thesis

The aim of the thesis was to shorten the time spent in the development stage of the Technology Life Cycle and in result, increase the efficiency of product development of electrical boards. To achieve these goals, a inefficient step in the current development cycle had to be identified and improved. Development of new electrical products can be severely different from company to company and it can therefore be hard to point out one deficiency which exists in all development cycles. However, since nearly all circuit boards developed require an efficient way of ensuring the production quality, the time spent setting up the testing equipment was chosen. There are, as described in the background chapter, multiple steps when setting up a functional testing platform for a new circuit board. The thesis has been mostly focused on the software side of the implementation of new testing sequences, but some benefits and drawbacks of change in hardware is also discussed. A couple of goals for the increased efficiency was set of making it easier for the developer to code, faster deployment times to the system and enable faster deployment of modified testing sequences. These are the goals the implemented system, described in the next chapter, tried to solve.

In regard to the identified deficiency of the current development cycle, the aim of the thesis was extended to evaluate in which capacity the use of smart instruments would improve the way to create automated testing equipment. The use of smart instruments to create testing fixtures will be discussed both in a time saving perspective and the possible reduction in complexity involved remotely controlling the test system.

3.2 Implementation of smart instrument

To test and validate the smart instrument idea, there was an implementation of a smart instrument. This implementation was conducted in collaboration with Mikro-

dust AB located at Ideon. The instrument to be replaced by a smarter implementation was an oscilloscope. Its functionality was implemented using a Micro Controller Unit (STM32) connected to an analog-to-digital(AD7606) converter communicating using a serial interface. The analog-to-digital converter was measuring the input signal and transfers the sampled value to the microcontroller. To achieve the goal of making the setup of a testing sequence less time consuming, an API used to control the interaction between the analog-to-digital converter and the micro controller was developed. The API enables the combination of the ADC and the MCU to act in a more smart way compared to coding directly using C. The base of the API was Micropython (see [n.a., 2021d]) and the API was implemented in three different variations.

Due to the length of the needed implementation background and method, the reader are referred to chapter 4 "*Example Implementation of a Smart Instrument*", for a full implementation description and method. In chapter 4 all implementation related information will be presented.

3.3 Evaluation of the implemented system

Finding an evaluation method for comparing the industry standard of big and expensive rack-based testing systems with a smarter and smaller solution is hard. By just looking at specifications of smaller analog-to-digital convert it is quickly realized that the sampling rate and precision of a small ADC compared to a full-sized oscilloscopes is much lower. However, this does not necessary means the MCU implementation is worse, if the performance requirements are satisfied with both solutions. This makes the comparison between the oscilloscope and MCU implementation, in terms of performance, quite uninteresting. Instead, the implementation was done in three variations, where all three would qualify as a smart instrument implementation. These three were then be compared against each other, both in a performance and practical point of view. For performance measure the maximum sampling frequency was used. Since all three implementations used the same analog to digital converter, the precision of the instrument was the same and therefore irrelevant to use as an evaluation metric. The time a test sequence takes to code is difficult to determine, since it varies depending on the coders experience in the used language, but also from case to case basis depending on the complexity of the project. To get an estimate of how much time the implemented solution would save, the setup time was measured from finished code until it was deployed and running on our test system.

4

Example Implementation of a Smart Instrument

In this chapter a possible implementation of a smart instrument will be done. Three different implementation methods will be included and evaluated against each other in regard to usability as a smart instrument and ability to replace the current solution. First part of the chapter will inform the reader about needed information to understand the implementation and the latter part will include the actual implementation idea. The results of the implementation and evaluation of the implementations will be conducted in the results chapter.

4.1 Problem description

Mikrodust are constructing and selling test-fixtures, used for electrical boards production testing. Since every board being tested are different, the testing sequence has to be programmed from scratch for each testing platform. By using C for coding the test sequence, the setup times are quite long compared to using a higher level programming language. The idea is to use different optimization methods, to maintain the highest possible performance using micropython instead of regular C code. By enabling the use of a Python-like language, the setup times could be drastically lowered and therefore also make the production time shorter. Another problem addressed is the possibility to update the test sequence without having to re-flash the micro controller with new software, for every change made to the sequence.

Hardware

Pyboard V1.1 The Pyboard is a development board sold by the creators of micropython. It provides a good base for this project, since it is one of the development boards for micropython. The pyboard consists of a STM32F4 microcontroller which is the same STM32 family that mikrodust's own board is using. It also has easy access to most of the GPIO (General Pin Output Input) of the STM32 controller, with

female connectors for dupont prototype cables. The board also includes a female usb (universal serial bus) micro connector and four light emitting diodes to make the testing of written code easier. It has 1024 kilobyte flash ROM and 192 kilobyte RAM available to the user and a micro SD card slot in case more storage is needed. The microcontroller comes preflashed with micropython and is therefore an easy way of getting used to coding in micropython before compiling a custom version. [n.a., 2021e]

AD7606-6 AD7606 is an analog to digital converter using 16 bits of precision with a maximum measurement speed of 200 kSPS. The reference voltage can be set to two internal voltage references of either 5 or 10 volts and is then able to measure ± 5 or ± 10 volts. The chip includes different ways of extracting the measured data using either an SPI connection (Serial peripheral interface) or using a parallel bus of either 8- or 16-bits at a time. The control of the AD7606-6 is done using different input pins on the AD7606-6 chip. These input pins has to be pulsed low or high in correct order to start an conversion and to be able to clock out the data that is being measured.

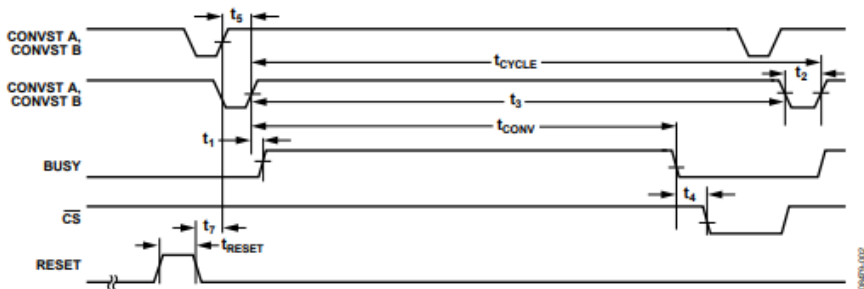


Figure 4.1 Timing diagram used to start a sampling of data. Source: [8-/6-/4-Channel DAS with 16-Bit, Bipolar Input, Simultaneous Sampling ADC 2020]

As seen in figure 4.1, there are five pins in the timing diagram provided by the datasheet. Four out of this, all except BUSY, has to be controlled by the microcontroller and toggled with the correct timing to work as expected. Also the setup parameters such as reference voltage, oversampling rate or data order is set using the input pins. [8-/6-/4-Channel DAS with 16-Bit, Bipolar Input, Simultaneous Sampling ADC 2020]

Setup

The setup used was built around the Pyboard and it was connected to all other components. The Pyboard was connected to the AS7606-6 using a SPI connection for data extraction and GPIO pins to set the required control pins. The Pyboard is also connected to an external computer using an usb cable. The usb connection is

used for flashing new firmware onto the STM32F4 chip and to communicate with the pyboard over a COM port. The port was used to enable the use of a REPL prompt, sending measured values and other communication with the external computer.

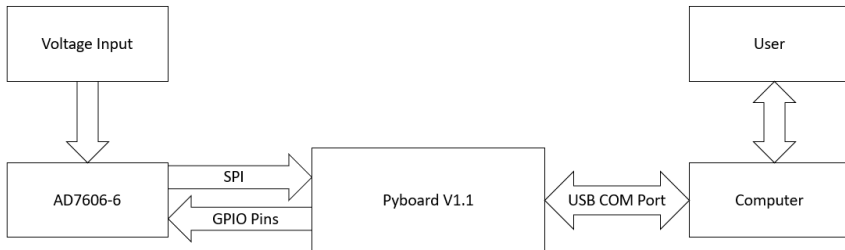


Figure 4.2 A block diagram showing the hardware setup communication.

All the connections necessary for the analog to digital converter to communicate with the STM32 micro controller was made using project wires (see figure 4.3). Also present in the figure are the potentiometer used to change the input voltage. This voltage was the one being measured to test the functionality of the instrument. Also used for creating a input voltage was a function generator, which could make a variable input to make the test case more close to how the instrument is supposed to be used.

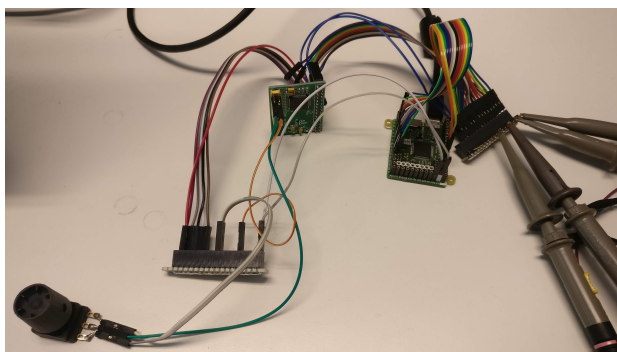


Figure 4.3 A picture showing the hardware setup used.

4.2 Project objective

The final objective of the project was to create an efficient and optimized API that works with the described system. This would enable the use of micropython as a programming language. By using micropython the API can work with dynamically imported code and it includes a lot of functionality that will be useful for the programmer.

Optimization

The code used to create the API should be implemented in a way that it is as optimized as possible. Since the API has to be able to be used from within micropython, the optimization options are as described in the Theory section. The optimization used in the project and later also compared are the viper emitter and c-module. The reasoning for not using the native emitter is that its optimization is the same as the viper emitter but with less optimization and attention of variable types. The baseline used was un-optimized micropython code and should therefore indicate the possible advantages of using the different methods for optimizing the APIs performance and achieve higher sample rates.

API Requirements

To be able to use the analog to digital converter in an efficient way as possible, a couple of functionality requirements on API is described below.

- First and foremost there needs to be an easy way to start up the communication and configuration of the analog to digital converter and establish a connecting with the micro controller. This includes configuring the SPI bus but also setting respective pins to either a high or low idle state to make sure the analog-to-digital converter is set up in the preferred way.
- After the initial setup there has to be a way to configure the sampling to a desired rate, number of channels used and precision of the returned values. Therefore a configuration function of some sort should be implemented to take care of sampling configuration.
- Preferable there should be two types of measuring implemented for testing purposes. One where the user ask for the current value and one sample is taken and returned. The other type of sampling functionality needed is the continuous sampling which was configured using the configuration function. This function should start a automatic sampling session which stores sampled values in some sort of buffer.
- The buffers content should then be able to be retrieved through another function. This function should have a number of different options for which data that is being returned to the user.

Apart from the functionality of the API there is also requirements for the efficiency of the code. The API should be easy to use and understand to make an improvement in the time needed to setup a new testing sequence. It should also not be unnecessarily heavy for the micro controller to run. The codes performance will be determined by analysing maximal sampling speed in the automatic sampling case and time from requested single sample to the sampling is retrieved. This will give a fair representation on the codes efficiency in a realistic use case.

Impact on development times

The use of micropython have several benefits which will be discussed in the result and discussion part of the report. One of the benefits that will be analysed further in this report is the impact using micropython instead of C-code have on the development times of a project. It will also include a explanation of the different workflows developing in C-code compared to micropython and how this will effect development times. Also, the impact of using a more high-level language (micropython) compared to using a low-level language (C-code) will be discussed briefly.

4.3 Nyquist Sampling Theorem

To be able to reconstruct the continuous signal it has to be measured at a adequate sampling rate. According to Shannon [Shannon, 1949], the signal has to be sampled at a minimum frequency of two times the highest occurring frequency. This sample frequency is often referred to as the Nyquist frequency. By sampling at strictly higher than twice the highest signal frequency, it is possible to perfectly reconstruct the sampled signal. If the signal is sampled at exactly the Nyquist frequency the sampled signal will have an accurate frequency, but the magnitude of the sampled signal will be undetermined.

If sampled at a frequency under the nyquist frequency, an aliased signal will occur in the frequency analysis.

Aliasing occurs when a signal is sampled below the Nyquist frequency. When a signal is sampled below the Nyquist frequency, the frequencies above will be folded over the Nyquist frequency and mirrored into the first Nyquist section. This will result in inaccurate measurements and therefore it is important to make sure the measurement frequency is high enough for the sampled signal.

4.4 Coding for micro-controllers and embedded systems

Coding programs for micro-controllers and embedded systems can be done using a multiple of programming languages. The most common language used in micro controller programming is C [Nahas and Maaita, 2012].

4.5 Micropython

Micropython is an open-source project to port the functionality of Python 3.x to different micro-controller boards and embedded systems equipped with a supported cpu [GitHub - micropython/micropython: MicroPython - a lean and efficient Python implementation for microcontrollers and constrained systems n.d.]. Since Micropython is an open-source project, anyone can download and compile Micropython making it possible to alter and modify the source code to suit specific needs of the user. Micropython is written almost exclusively in C99 [n.a., 2021d] and is therefore portable to a lot of micro-controllers. Since it is an open-source project and the developers are open to adapt new ports, the number of supported devices rise all the time. If the required micro-controller is not supported the individual is free to make their own port and share with other users of the same controller.

Optimizing code

By using the Python API instead of C to code for the micro-controller, the number of required code lines are in most cases reduced. The reduction of needed lines of code often results in easier and faster programming. The drawback of coding using an high level coding language is using more clock cycles to achieve the same result, compared to the same program written in a lower level language. This is due to the high-level code needs to be converted to low-level instructions for the hardware to understand. Another reason Micropython is slower than C-code is the fact that C code is compiled while micropython, as normal Python, compiles at runtime. The microcontroller then has to translate the code to machine instructions, in real-time, instead of doing it before runtime. To counter this drawback of Micropython a couple of optimization solutions is purposed to the programmer [n.a., 2021c]. The solutions consists of code improvement, such as avoiding use of certain storage types, but also includes optimizers that precompile Micropython code. These precompilers will be introduced below.

Native code emitter and Viper code emitter

The micropython library includes two types of code optimizer. Both options have a lot in common and mostly comes down to how specific the code to be optimized has to be. The native code emitter accepts almost any type of micropython code and is therefore preferable if the code to be optimized is already written. In contrast, the Viper emitter need a more specific micropython code to work. The programmer has to be more precise in telling the emitter the type (uint, bool, string etc) and size of each variable. By having this additional information the Viper emitter is able to optimize the program even further compared to the Native emitter. There is a couple more restrictions in using the viper code emitter, that does not occur in the the native emitter. For example the Viper emitter can not use more than four arguments input to a function and it can not have default values [n.a., 2021c].

Both emitters are precompiling the micropython code to machine instructions, instead of using bytecode, and is therefore able to speed up the code execution. According to micropythons website the speed up of the native emitter compared to non-optimized code is two times and the viper emitter is even faster [n.a., 2021c].

There are cases where even optimised micropython code is not fast enough or simply does not provide the required functionality. The next step would then be to utilize a C-module, which is described below.

C-modules

Micropython is written using the C99 language and then compiled into an *.elf file to be flashed onto the micro-controller. Since the code is available to everyone to modify and recompile as they like, the micropython community suggest writing an external C-module when the functionality and speed of optimized micropython code is not sufficient. The module will then be compiled together with and included into the compiled *.elf file. This inclusion will result in a modified version of micropython. The microcontroller then has to be reflashed with the new modified version of micropython to be able to use the module.

MPY files

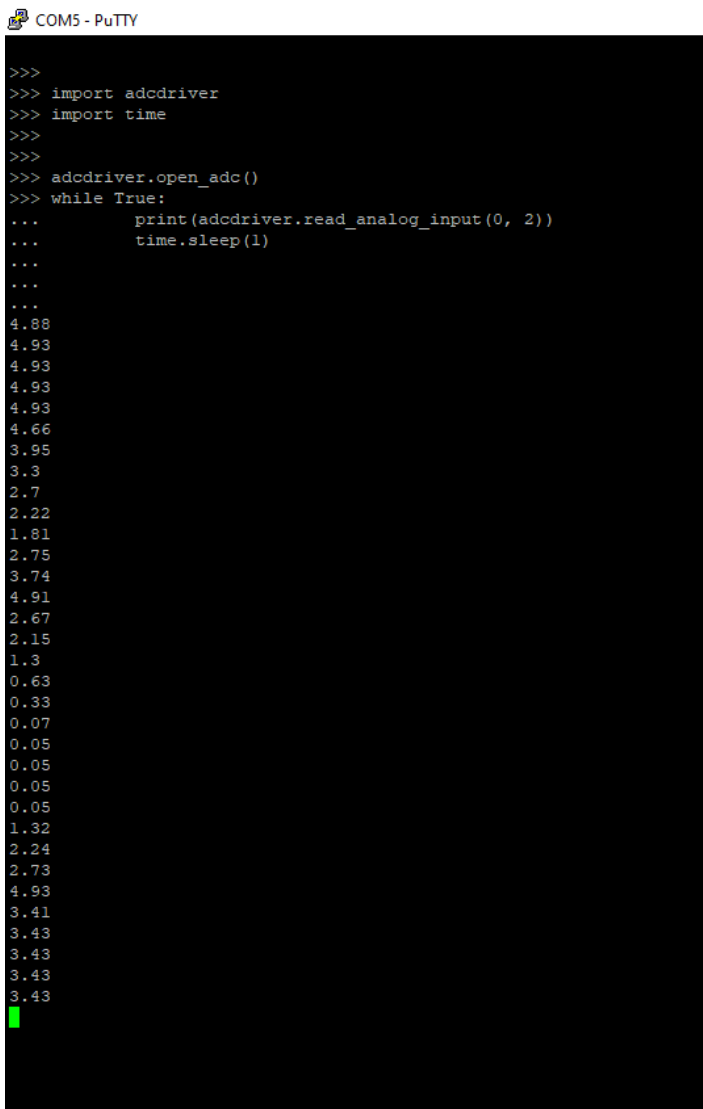
Micropython also has the possibility to use precompiled C code in external files. By using an included *.mpy file generator, the coder can code a C program and compile it to a *.mpy file. This *.mpy file can then be imported at run time and therefore the microcontrollers firmware does not have to be reflashed [n.a., 2021f].

4.6 REPL

REPL stands for Read, Evaluate, Print and Loop. Repl is a command prompt for Python that are able to read input, evaluate the commands, print the results and then loop all over. This is a great way to interact with Python to test new ideas or print out results from code [*What is a REPL?* N.d.] The idea is to test out python functions to see how to use them in your program or to print out results from your code to debug and find errors. An example of a REPL terminal executing commands and printing out returns is shown in figure 4.4.

4.7 STM32

The STM32 microcontroller is a chip developed and produced by ST Microelectronics and could be used in a multiple ways. This is the target micro controller for all implementations done during the thesis. The reason the ST32 is chosen is because it is a commonly used chip in micro controller boards. Below, programs and libraries important to the use of the STM32 microcontroller family will be explained.



```
COM5 - PuTTY
>>>
>>> import adcdriver
>>> import time
>>>
>>>
>>> adcdriver.open_adc()
>>> while True:
...     print(adcdriver.read_analog_input(0, 2))
...     time.sleep(1)
...
...
4.88
4.93
4.93
4.93
4.93
4.66
3.95
3.3
2.7
2.22
1.81
2.75
3.74
4.91
2.67
2.15
1.3
0.63
0.33
0.07
0.05
0.05
0.05
0.05
1.32
2.24
2.73
4.93
3.41
3.43
3.43
3.43
3.43
```

Figure 4.4 REPL terminal executing commands and printing out returns

STM32CubeMX

The STM32cubeMX is a graphical setup helper for the STM32 family of micro controllers. It can be used to generate a code skeleton with correct pin layout and setup parameters for the chosen STM32 microcontroller. [*STM32 configuration and*

initialization C code generation 2020]

STM32Programmer

The STM32 programmer is used to interact with the micro controller chip. It can be used to manipulate specific registers or erase all data in the chip. The programmer is also used to flash the STM32 chip with new firmware. [*STM32CubeProgrammer all-in-one software tool 2019]*

Hardware Abstraction Layer(HAL)

To make it easier and more straightforward for developers to use the STM32 family, ST Microelectronics has created the Hardware Abstraction Layer (HAL) API. The HAL API features the similar API across all STM32 micro controllers and it is therefore easy to move code from one STM32 controller to another. The HAL library features support for all available peripherals on the STM32 boards. [*Description of STM32F4 HAL and low-layer drivers 2020]*

Low-layer API (LL)

The Low-Layer API is also provided by ST Microelectronics and supports more lightweight and closer to hardware API. The LL has less functionality and is using register based operations which requires more understanding of the targeted micro controller and therefore offers less portability between different microcontrollers inside the STM32 family. The upside compared to the HAL library is better possible optimization. [*Description of STM32F4 HAL and low-layer drivers 2020]*

4.8 Peripheral Interface

To get the micro controller to communicate with peripheral devices such as an analog-to-digital converter, a communication interface has to be chosen. There are many ways to communicate between devices and the two relevant and here described are a parallel and serial approach.

Serial Peripheral Interface

Serial Peripheral Interface, SPI for short, is a synchronous communication interface. Depending on the use case the interface needs anywhere from two to four wires connecting the two components that needs to communicate. The four connections used in spi is shown in figure 4.5. [Dhaker, 2018]

Chip Select The chip select line is used when there are multiple peripherals on the same SPI bus. By changing it from high to low or low to high (depending on chip select polarity) the SPI master can choose which SPI slave it is talking to. [Dhaker, 2018]

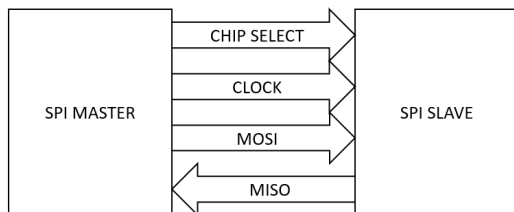


Figure 4.5 Pin description for SPI.

Clock The clock is provided by the SPI master and is used to synchronously clock data in and out of the SPI slave. The clock has two variables that needs to be taken into account, clock phase and polarity. The clock phase describes on which edge the data is sent namely on the falling or rising edge. The polarity selects the idle state for when the SPI clock is not in use. [Dhaker, 2018]

Master Out Slave In (MOSI) The MOSI connection is used to send the data from the master node to the slave. This is used by cycling the clock and when either falling edge or rising edge (depending on clock phase) occurs, the value of the MOSI pin is read by the slave chip. [Dhaker, 2018]

Master In Slave Out (MISO) The Miso pin works in the same manner as the MOSI pin but in reverse. It sends data from the SPI slave to the SPI master on either the rising or the falling edge of the master’s SPI clock. [Dhaker, 2018]

Parallel Interface

A parallel interface can be implemented using the same idea as SPI which is described above. Instead of using a singular wire connection for MISO and MOSI, a multiple of connections would be used to be able to send more data at each clock cycle. This could for example be in size of 8 or 16 connections to send one or two bytes at each clock pulse.

Configuration of peripheral pins

Since peripheral devices has different purposes and communication needs it is not always necessary to include all of the connections in figure 4.5. For example in a measuring device there might not be a need for the master to be able to send data to the slave and therefore the MOSI pin could be excluded. The same idea could be used in the case of controlling a led strip where the led strip might not have to be able to send data to the controller and the MISO is not needed.

Another pin that is sometimes not necessary is the chip select pin. If there is only one device connected to the SPI bus there is no need for the master to tell the system which slave it wants to talk to and can be set to static high or low depending on polarity of the slaves chip select pin.

4.9 Direct Memory Access

Direct Memory Access, DMA, is used for a multiple of reasons such as unloading the workload of moving data from the Central processing unit (CPU) and to be able to transfer data even when the CPU is in a low power mode. The idea is to have an external controller that are able to access to systems main memory and handle reads and writes to specific parts of the memory. [n.a., 2021b] While it saves clock cycles from being used to move data it also ensures that data flow is not interrupted by interrupts that affects the running code. This enhances performance of the process especially in a low-performance system where every clock cycle has to be used efficiently to ensure maximum performance.

4.10 Implementation Timeline

Beginning of the project

The project began with different hardware compared to the one used in the final version of the API implementations. The idea was to develop a API that would work on Mikrodusts own custom board featuring a lot of similar components as the final hardware. After a lot of testing and trying to get the custom card working, the project change hardware to a pyboard and external analog-to-digital converter. The switch to a pyboard was decided after the attempts to get a stable SPI line working failed.

After switching to the described hardware, consisting of a Pyboard v1.1 and an ad7606-6 analog-to-digital converter, the SPI worked as expected. Clean noise free SPI signals was verified using an oscilloscope and a loop-back test. Then the development of the API could begin after attaching everything together using project cables.

Setting up communication between hardware components

The first objective, after the verification of the SPI signals functionality, was to set up the measuring and communication between the Pyboard and the analog-to-digital-converter. The main task was to make sure the timings on all control pins on ad7606-6 was pulsed low and high in correct order and with proper timings, see figure 4.1. The setup of the timings went fairly smooth and the difficult part was getting all polarities correctly setup and get all the pins toggling in the correct order.

Compiling micropython firmware and flashing the pyboard

The Pyboard V1.1 comes pre-programmed with micropython firmware. Therefore, it was a great way to get used to micropython. However, there are a multiple of reasons to compile and re-flash the board. Since micropython is still in beta stage of its development cycle, it frequently gets new updates and added functionality. To

get the latest software can be beneficial but has its drawbacks of potential changing APIs. Therefore to be able to flash your preferred version is out most important for commercial use. Another reason for compiling your own version of micropython is that it enables import of external C-modules (described more in-depth below). External modules not only enables the coder to create their own modules but also including module written by others into their micropython firmware. Micropython is written with being lightweight in mind and has therefore excluded functionality the creators does not feel is being used by the general user. By including external C-code the user can then change what is included in their version of the micropython firmware. This creates a firmware that is lightweight but still includes the preferred functionality of the coder. This also works the other way around where users can disable functionality to make their version smaller in size.

During the project the STM32Programmer software was used to flash the board with *.elf files. The files being flashed was produced using a C-compiler in a linux system using makefiles and the make command.

Setting up the API

Setting up the API to work as expected was done three times: one with micropython, one with micropython optimized with the viper emitter and one using an external c-module.

Micropython The regular micropython implementation was the first one to be implemented. Most of the API implementation was pretty straight forward, but timers interrupt handling created some problems. This was due to how micropython handles memory allocation during timer interrupts. The first version of used a variable containing the most reason sampled value which was updated at each timer interrupt. During the interrupt we also activated a flag telling the main program it had a new value to add to the array containing samples. This works great if the only thing the microcontroller is doing is sampling values. In this instance, however, the microcontroller is suppose to not have to take care of saving samples in the main loop. Since interrupts in micropython does not allow arrays to be modified or new memory allocated, a bytearray was created and then used as a circular buffer to store sampled values. After this change the API implementation was able to run in the background (still interrupting the running code) and not needing any extra coding in the main loop except setup commands.

Micropython with Viper Emitter A substantial amount of the micropython code in the non-optimized version, could be re-purposed in this version of the API implementation. To use the viper emitter to optimize the relevant part of the code, namely the sampling and memory handling, more board specific libraries had to be used. Due to how the emitter work, the code has to contain more information of which type and sizes the variables contains. This resulted in using library with non-existing documentation and almost all information had to be retrieved from the source code.

The stm library is only shown as an example in the inline assembler and viper emitter examples. This lack of documentation resulted in some own testing and trying to find other projects using the library. After figuring out how all the needed imports and registers worked, the API was successfully implemented.

External C-module The external C-module had to be implemented from scratch since it is coded in C instead of micropython. This solves some of the complications faced when implementing the micropython versions, but also introduces some other problems. Since the C code is the language normally used to code for the stm32 micro controller family, there is a lot of available documentation provided by ST Microelectronics. One example is the HAL library described in the theory section. Setting up the correct timing for toggling of control pins was quite non-problematic. The issues began when trying to use the SPI or clocks on the board. Since both the C-module and micropython wants to decide how clocks and peripheral interfaces should be implemented, there was some code collisions and unexpected errors. This resulted in the need to modify micropython's source files to allow for the C-module to fully control the timer clocks and SPI interface.

4.11 Evaluating the API

When the API was implemented using all of the three described methods it was time to test the APIs against each other. In this section the APIs will mainly be compared using measured numbers and concrete results, while more in-depth discussion and use-case based comparison will be left for the discussion part of the report.

API Description

The API consist of ten different function calls available to the user of the API. A description and name of each function call will be described below.

- **turn_on_adc()**

The `turn_on_adc` function is used as a first step to use the adc for sampling purposes. The function has the purpose of setting the correct idle values of all the necessary pins and configuring the correct pins for SPI use. All the required clocks for spi and GPIO are also enabled during this step.

- **turn_off_adc()**

The `turn_off_adc` is used when the analog to digital converter should no longer be active. It makes sure the pin that have had its idle state set to a specific value are being reset to its default value. It also uninitialize the timer 4 clock which is required so the timer stops counting and also stop triggering the timer overflow interrupt.

- **set_sampling_parameters(channels, measurement_frequency, decimals)**

In `set_sampling_parameters` most of the configuration of the continuous sampling parameters. There are three parameters to be configured, namely: `channels`, `measurement_frequency` and `decimals`. The `channels` parameter requires a list of the channels to be sampled. Since there are 8 channels available to the target hardware, the channel numbers available are between 0 and 7. The `measurement_frequency` are set as an integer and is specified in hertz. Depending on the implementation method chosen, the maximum attainable frequency varies. This will be presented and discussed in the result section of the report. Lastly the preferred number of decimals has to be chosen, by setting the `decimals` using an integer. The specified number of decimals will be used when returning the result of the sampling with one of the latter described functions.

- **set_size_of_buffer(buffer_size)**

To be able to have variable buffer size the function `set_size_of_buffer` was implemented. The implementation of this function also varies with which of the three methods that was used. The `buffer_size` are specified as an integer and is on a per channel basis. For example a number of samples of five would result in a buffer of five samples per channel.

- **start_sampling()**

To start the continuous sampling of specified channels the `start_sampling` function has to be called. The main functionality of the function is to start timer 4 on the micro controller unit and specify its overflow interrupt function. By doing so the overflow interrupts will be triggered at the specified intervals and a sampled will be saved.

- **stop_sampling()**

To stop the sampling from running by disabling the timer four from running. The function also disables the the timer interrupt associated with the timer and therefore prevents the sampling function from triggering if another application starts the timer.

- **get_buffer(only_new_data)**

To retrieve sampled values from the buffer specified in function above two function can be used. The `get_buffer` returns the full sampling buffer unless the `only_new_data` is set to true. When the attribute `only_new_data` is set to true only the not already read data will be returned. Since the buffer used are using a circular approach the full buffer will be returned, even if `only_new_data` is set to true, when all the old values has been overwritten.

- **get_past_nbr_samples(nbr_samples)**

To return a specific number of samples the function `get_past_nbr_samples` can be used. The function returns the number of samples specified by the `n_samples` as an integer. In the same fashion as the `set_size_of_buffer`, the number of samples are specified for each channel. For example by using the function by setting `n_samples` to five, five values from each chosen channel will be returned.

- **`get_analog_value(channel, decimals)`**

The API up until this point has mainly described ways to set up, start and return a continuous sampling of analog values. The `get_analog_value` function provides the user of the functionality of sample once on selected channels and directly returning the result. The attribute `channel` is specified as a list of integers ranging from zero to seven to specified which of the eight channels to sample from. The `decimals` set the preferred number of decimals to be returned on the sampled value.

- **`get_difference(channel1, channel2, decimals)`**

The function `get_difference` shares a lot of similarities with the `get_analog_value` function. It samples only once and the accuracy displayed are set using the `decimals` function in the same way as before. Where it differs from `get_analog_value` is in the number of channels to be chosen and the return. The channels have to be exactly two and is specified using the `channel1` and `channel2` variables using integers between zero and seven to choose two of the eight channels. The returned value are then calculated as `channel1` minus `channel2`.

Example of usage

To further give the reader an understanding on how to use the API can be used an example micropython script where the API sets up an imaginary test sequence will be described. It will outline all the functionality present in the API and give an example on how it could be used.

```
import adcdriver
import time

channels = [0, 3, 5]
measurement_freq = 1000
decimals = 2
buffer_size = 100

#opens the adc
adcdriver.turn_on_adc()

#return the sampled value of channel three
#(indexing from zero) with two decimals
one_sample = adcdriver.get_analog_value(2, decimals)

#return the sampled value of channel one minus channel two with two decimals
```

Chapter 4. Example Implementation of a Smart Instrument

```
diff_channel_sample = adcdriver.get_difference(0, 1, decimals)

#configure the adc, sets buffer size and starts the sampling
adcdriver.set_sampling_parameters(channels, measurement_freq, decimals)
adcdriver.set_size_of_buffer(buffer_size)
adcdriver.start_sampling()

time.sleep(10)

# Stops the sampling after 10 seconds of sampling
adcdriver.stop_sampling()

# Returns all the data currently in the buffer since only_new is set to false.
list_of_data_1 = adcdriver.get_buffer(False)

# Returns the 10 last samples of all channels
list_of_data_2 = adcdriver.get_past_nbr_samples(10)

#closes the adc
adcdriver.turn_off_adc()
```

Evaluating the implication on development times

To evaluate the time being saved using the implemented API a baseline has to be decided. Since the alternative approach to using the API would be to code everything in C-code, compile it and flash the MCU, this will be used as a baseline for the time saved per code testing. Further, a higher-level programming language will result in a lower time spent programming [Parthasarathy, 2009], but this will not be analysed further since that would require a more in depth data collection and is therefore regarded as outside the scope for this thesis.

Therefore the time saved using the API, will be measured from the coding job is done and until it is running on the Micro Controller. By doing this an estimation on the time consumption of the two can be evaluated and compared against each other.

5

Result

The full API description can be found in chapter 4, "*Example Implementation of a Smart Instrument*" and only the functionality and performance of the three implementations will be presented here.

5.1 Functionality of the instrument

The instrument works as expected and is able to sample the correct voltages provided by the voltage source. The API that was implemented had the required functionality in all three implementation cases and was able to fulfill the instruments requirements stated in the method. The user interacts with the API only using micropython which enables dynamic imports of new code, dynamic change of existing code and sending commands to the instrument using a REPL terminal. This implementation also enables the user to control the board from remote using a web-based REPL terminal. The web-based REPL terminal requires a internet connected microcontroller, which the pyboard is not, and therefore this possibility was not explored further. Different level of performance, in terms of sampling rate, was achieved and will be presented below. All the results will be discussed further in the discussion part of the thesis.

An implementation which was made easier with the use of the API was implemented and was able to produce the same experience of graphical representation of the measured data. Here the API was used to set up the sampling and extracting of the samples and then sent over a communication port to the host system. An example of a signal measured using this approach and displayed on a host computer using a python script can be seen in figure 5.1.

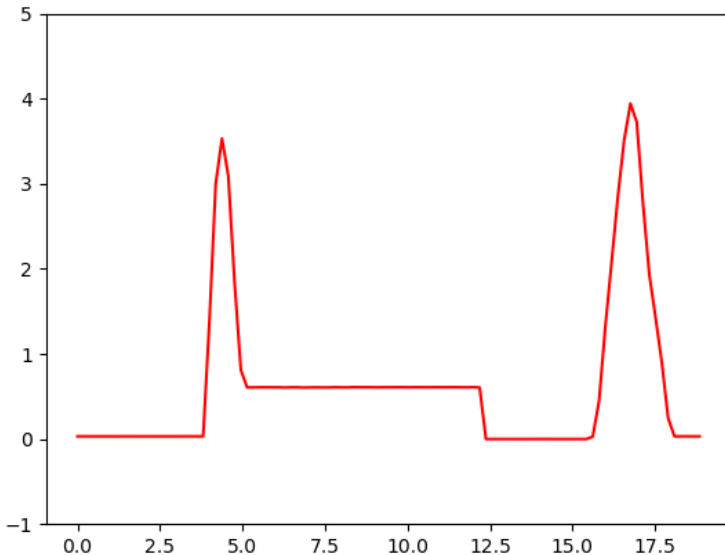


Figure 5.1 A measured analog signal transferred to the host computer.

5.2 Sampling capabilities using the API implementation

To compare the different implementations of the API for the example instrument, the times and maximum sample frequencies were measured (see table 5.1). The times were measured using an oscilloscope and was measured from start of one sample to the start of the next one. As can be seen in the table, the C-module was by far the fastest measuring the whole sampling period, but the SPI bus was only relying on the SPI master clock frequency and was therefore the same on all three. The SPI bus time is measured from the start to the end of the SPI clock pulse.

	Single measurement time (us)	SPI bus (us)	Theoretical max frequency (Hz)	Stable tested max frequency (Hz)
C-module	60	54	16666	16500
Viper-emitter	1230	54	813	800
Micropython	2220	54	450	450

Table 5.1 Measurements of the three implementations.

5.3 The API implication on development times

The times required to compile, flash and run C-code on the Micro Controller was measured using an external manually operated timer on a phone.

	C-code only (mm:ss)	Using the API (mm:ss)
Compile the code	1:20	not needed
Put MCU in DFU mode and flash new firmware	0:31	not needed
Start up communication program again	0:22	not needed
Run the code	0:07	0:07
Total time	2:21	0:07

Table 5.2 Times spent on running C-code compared to using the API

As can be seen in table 5.2 there are a couple of steps which can be excluded when using the API to implement a test sequence instead of using C-code. All the steps that was timed was done on a low-powered laptop and should be seen as a possible time saving measurement and not as absolute times.

6

Discussion

6.1 Hardware limitations in the example instrument

The example implementation of a smart system used a Pyboard and an external analog to digital converter which was connected using project wires. This introduces a couple of possible problems which may or may not be present in the final implementation of a similar instrument. The problem using long project wires for high speed data transfers is the increased capacitance and resistance in the circuit. When trying to increase the SPI frequency to decrease the transfer speed, the clock signal started to deform from its square-wave normal. This is due to the pyboard not being able to drive the clock signal fast enough and therefore could be a potential problem if driven at a faster rate. When connecting the two components on the same circuit board, the clock signal will have less resistance and could potentially be driven at a faster rate and therefore decrease the transfer time needed.

6.2 Optimization of implementation code

Choice of optimization

Using C or Micropython to implement the API Implementing the API using micropython code is a fast and easier way to implement the API. There are however some possible drawback with this approach. The micropython library is written using C code and therefore the in an optimal case the micropython code will only be as fast as C code. In most cases the C code implementation will be faster and more efficient than the micropython implementation.

The viper emitter compared to the native emitter The project was implemented using the Viper emitter and an external c-module. The reasoning behind the choice of optimizations was based on advantages and disadvantages of each method. The native code emitter was ruled out because it did not provide any benefits over the viper emitter for our use case. The native emitter uses the same way of optimization but requires less for example definitions of variable types which leads to an less

effective optimization of the code. The perk of using the native code emitter instead of the viper code emitter is that it offers more freedom of how the coder can write their code. But since the projects code was fairly easy to implement inside viper emitter, there was no need to utilize the extended freedom in the native emitter.

C-module compared to MPY files C-module and Mpy files are both written in C code and the compiled so they have similar if not identical potential to speed up the code during runtime. The difference of MPY files being imported from outside the micropython firmware does however introduce some potential problems. Since the original copy of micropython are trying to control a lot of the registers in the microcontroller, a potential problem with using *.mpy files is that the compiler will not pick up on certain conflicts in the code which would have been caught if the module was compiled together with micropython.

C-module compared to an external file C-modules differ from the rest of the coding solutions for implementing the API in the sense that it from the coders point of view looks like a native function. The native emitter, Viper emitter and *.MPY files, are all located on the accessible memory of the micro controller and can therefore be accessed by a third-party user. This could be preferable in an development stage of the API since it gives the coder more insight in how the API works. When the API is well documented and consider to be in its final state, it would be better from the end users point of view if the API would behave like any other micropython function and have its files hidden from user. This is more important if Mikrodust decides to let costumers code the test sequences themselves instead of having a mikrodust employee code the sequence for them.

6.3 Implications on development times

In the result an example of a program being run on a pyboard using both only C-code and micropython code. It was apparent that the testing of code written in micropython instead of C was much faster saving a two minutes and 14 seconds per test run of code. This could greatly affect the times developers spend testing code during a development process. Also the ability to dynamically import code make the process easier to do remotely as a full flash of new firmware isn't necessary when updating the test sequences. The ability to remotely control the instrument would mean less time spent traveling back and forth between the development site and production facilities and therefore not only save time in the programming step but also on the overall maintaining of the system.

6.4 Performance of the example instrument

The performance of the implemented instrument was not as good as a full oscilloscope. A maximum sample frequency of 16 500 Hz is much lower that most full

sized oscilloscopes. This is not necessarily a problem though since most measurements needed in production testing does not require such high sampling rate. In addition to this there is a possibility to achieve a sampling rate of 200 000 Hz (maximum for the analog to digital converter) if switching from serial to parallel communication between the micro controller and the analog to digital converter. Even though this is a lot faster than the current setup, it is still nowhere near the oscilloscope standard. In regard to this it is important to know the requirements of the system when implementing testing equipment and then choose suitable instruments that fulfill the requirements. Also by changing the hardware to a more capable micro controller and analog to digital converter the sampling rate could be increased further. This is a balancing act that has to be made during the development of the testing system of choosing cheaper equipment that still full-fills the testing requirements.

6.5 Use of Micropython for commercial use

Micropython offers a lot of potential for fast development of software to embedded systems and micro controllers. Since it shares a lot of similarities with regular python, many companies already have the expertise needed. While using micropython for prototyping or simpler finished products is a valid options. The drawbacks of micropython for commercial use are more prominent when implementing more complex programs or need to use more of the non-standard part of the micropython library. The documentation for these use cases are severely lacking and sometime non existing. When trying to find documentation on certain board specific libraries or more advanced, the forums explained to either try to understand the source code, find someone else who already implemented something using the preferred library or sometimes just saying it exists but have never been documented anywhere. The developers are open with the project being in a BETA stage of the development process. This also implies that the micropython API could change over time and therefore old code could start to be non-functioning. This could introduce unwanted problems for a company wanting to use micropython for commercial use. However, since the micropython firmware is flashed onto the micro controller chip, the company can choose to use an earlier version of micropython for their boards to maintain consistent APIs and functionality, with the drawback of missing out on new features and bug fixes.

6.6 Further work

Smart testing systems

The implemented smart instrument is only one small part of building a smart testing system. To be able to call the system smart a more complete "solution would be

needed. By implementing a multiple of these instruments and connecting them in a smart way to be able to interact with them over internet or similar would be one of the key components to easier manage several systems from a remote location. By having the code being dynamically imported onto all testing equipment no physical interaction with the device to set it into firmware update mode would be required. Instead all the testing scripts could be transferred over internet onto the instruments internal memory and then being run using commands.

This could even enable a drag and drop functionality where the tester does not have to interact with any code and could setup the whole testing process using a graphical interface.

Implementation of example instrument

All the different approaches has their pros and cons. In the case of the different versions of micropython implementations there is not to much further work to be done. With the C-module there is some future work to be done. To make sure the micro controller is being used to its full potential the DMA should be implemented and used by the API. This would enable the Micro Controllers processor to do additional work while the SPI transfers the sampled data to the memory of the MCU. Since the SPI signal is 54 us of 60 us of the sampling period in the c-module case 90% of the sampling time the processor is used to only transfer data which could be handles by the integrated DMA controller. This would increase the possibility of including more smart features without hindering the maximum sampling frequency.

To further increase the potential maximum sampling frequency a parallel interface could be used instead of the Serial Peripheral Interface. This would enable the clock to stay at the same frequency as before but by increasing the number of sent bits per clock cycle from one to for example sixteen the transfer speed would increase with a multiple of sixteen. Since as stated above the SPI transfer is 90% of the sampling period using the C-module implementation, by increasing the transfer speed by a factor of sixteen the overall sampling frequency could potentially be increased by over fourteen times. This would give us a theoretical sampling rate of 231 KSPS (kilo samples per second) and therefore be able to run the analog to digital converter at full speed without the SPI being a bottleneck.

Also when the c-module is transferred over to the final hardware system as much as possible should be converted from using the HAL library to the LL library to optimize the code even further, this was not desirable during the project since it was not done on the final hardware. Since the LL system is specific to each STM32 chip and the pyboard and the final target system does not share the exact same chip, this was not implemented.

7

Conclusion

The product development cycle has been analyzed and one problem area in the end of the R&D phase was targeted. The thesis was able to solve part of the problem, by shortening the time spent researching and developing a testing platform for new products. Also the ease of use was discussed and improved compared to the baseline. Combined this results in less time spent waiting for the test fixture to be developed.

The implemented solution was an example of a smarter testing instrument, which could be used to construct a smarter testing system. The possibility to use the implemented instrument or similar instruments has been discussed and both the pros and cons of using said system has been analysed. The purposed system has shown to have a positive impact on both the size of the overall system, but also the development times of setting up a testing system for new products. The same level of performance as the instrument to be replaced by the example implementation was not achieved, but was to be expected. In most scenarios the loss of performance has little to no impact of the usability of the instrument, since the measured signals in production quality testing does not require higher performance than the implemented system can deliver.

Most of the goals with the thesis was reached and even tough not everything desired in a smart testing instrument was implemented, the potential of said system was shown and demonstrated.

Bibliography

- 8-/6-/4-Channel DAS with 16-Bit, Bipolar Input, Simultaneous Sampling ADC* (2020). Tech. rep. <https://www.analog.com/media/en/technical-documentation/data-sheets/ad76067606-67606-4.pdf>.
- Description of STM32F4 HAL and low-layer drivers* (2020). Tech. rep. https://www.st.com/resource/en/user_manual/dm00105879-description-of-stm32f4-hal-and-ll-drivers-stmicroelectronics.pdf.
- Dhaker, P. (2018). “Introduction to spi interface”. URL: <https://www.analog.com/media/en/analog-dialogue/volume-52/number-3/introduction-to-spi-interface.pdf>.
- GitHub - micropython/micropython: MicroPython - a lean and efficient Python implementation for microcontrollers and constrained systems* (n.d.). URL: <https://github.com/micropython/micropython>.
- n.a. (2021a). *operating systems for software development worldwide 2020 | statista*. URL: <https://www.statista.com/statistics/869211/worldwide-software-development-operating-system/> (visited on 2021-05-18).
- n.a. (2021b). *Direct memory access | microchip technology*. URL: <https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/8-bit-mcus/peripherals/core-independent-peripherals/direct-memory-access> (visited on 2021-05-04).
- n.a. (2021c). *Maximising micropython speed — micropython 1.15 documentation*. URL: http://docs.micropython.org/en/latest/reference/speed_python.html (visited on 2021-05-04).
- n.a. (2021d). *Micropython - python for microcontrollers*. URL: <http://micropython.org/> (visited on 2021-05-04).
- n.a. (2021e). *Micropython store*. URL: <https://store.micropython.org/product/PYBv1.1H> (visited on 2021-05-04).

Bibliography

- n.a. (2021f). *Native machine code in .mpy files — micropython 1.15 documentation*. URL: <http://docs.micropython.org/en/latest/develop/natmod.html> (visited on 2021-05-04).
- Nahas, M. and A. Maaïta (2012). “Choosing appropriate programming language to implement software for real-time resource-constrained embedded systems”. DOI: 10.5772/38167. URL: <http://dx.doi.org/10.5772/38167>.
- Parthasarathy, B. (2009). “International encyclopedia of human geography”. DOI: 10.1016/B978-008044910-4.00180-2. URL: <https://doi.org/10.1016/B978-008044910-4.00180-2>.
- Sahni, S. (2021). *How technology life cycle can give first-mover advantage? - greyb*. URL: <https://www.greyb.com/technology-shifts-can-give-first-mover-advantage/> (visited on 2021-05-18).
- Shannon, C. (1949). “Communication in the presence of noise”. *Proceedings of the IRE* 37:1, pp. 10–21. DOI: 10.1109/jrproc.1949.232969. URL: <https://doi.org/10.1109/jrproc.1949.232969>.
- STM32 configuration and initialization C code generation* (2020). Tech. rep. https://www.st.com/resource/en/data_brief/stm32cubemx.pdf.
- STM32CubeProgrammer all-in-one software tool* (2019). Tech. rep. https://www.st.com/resource/en/data_brief/stm32cubeprog.pdf.
- What is a REPL?* (N.d.). URL: <https://codewith.mu/en/tutorials/1.1/repl>.

Lund University Department of Automatic Control Box 118 SE-221 00 Lund Sweden		<i>Document name</i> MASTER'S THESIS	
		<i>Date of issue</i> September 2021	
		<i>Document Number</i> TFRT-6151	
<i>Author(s)</i> Niklas Gustafson		<i>Supervisor</i> Mats Iderup, Mikro dust AB, Sweden Tobias Petersen, Mikro dust AB, Sweden Linus Hellman, Mikro dust AB, Sweden Johan Eker, Dept. of Automatic Control, Lund University, Sweden Giacomo Como, Dept. of Automatic Control, Lund University, Sweden (examiner)	
<i>Title and subtitle</i> Shorten development time of Functional Testing (FCT) in electronic manufacturing with smart instruments			
<i>Abstract</i> <p>The development of electrical circuit boards is increasing and the time spent in the development stage can be quite long. The thesis has tried to shorten the time spent developing new electrical circuit boards, by first finding a common denominator for efficiency losses in the development process and then implementing an example solution. The problem identified was the lack of attention towards production testing early enough in the development process. The testing equipment often comes as an after thought which delays the start of production. Since the product can not be mass produced before quality assurance can be done using testing equipment, this will lead to unnecessary time spent with products in development. Another problem addressed in the thesis is the expenses of acquiring a testing system. Today's standard for testing systems is both big and expensive and a better solution would be beneficial.</p> <p>To solve these problem the thesis purposes a solution using a micro controller connected to a small analog to digital converter. This solution is a lot cheaper compared to today's standard and offer additional benefits described further in the thesis. The drawbacks of loss of performance compared to the more expensive equipment is also discussed.</p> <p>To test said solution, three example implementations of the same smart instrument was done and evaluated. The implementation was then evaluated both on the possible time saved from using a smarter instrument compared to the same instrument without the smart API implementation and the performance of each of the three possible implementations.</p> <p>The example implementation resulted in shorter development times and easier interaction with the instrument. It also fulfilled most of the requirements set on a smart instrument in the beginning of the thesis. Performance wise the implemented example instrument was good enough for most production testing situations and is therefore able to replace the more expensive versions for some applications.</p>			
<i>Keywords</i>			
<i>Classification system and/or index terms (if any)</i>			
<i>Supplementary bibliographical information</i>			
<i>ISSN and key title</i> 0280-5316			<i>ISBN</i>
<i>Language</i> English	<i>Number of pages</i> 1-46	<i>Recipient's notes</i>	
<i>Security classification</i>			