

Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of measurement

Atomic Clock Backup Power System

David Šibrava

Supervisor: Doc. Ing. Jaroslav Roztočil, CSc.

Field of study: Kybernetika a robotika

Subfield: Cybernetics and robotics

August 2019

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Šibrava** Jméno: **David** Osobní číslo: **434931**
Fakulta/ústav: **Fakulta elektrotechnická**
Zadávací katedra/ústav: **Katedra měření**
Studijní program: **Kybernetika a robotika**
Studijní obor: **Kybernetika a robotika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Systém zálohovaného napájení atomových hodin

Název diplomové práce anglicky:

Atomic Clock Backup Power System

Pokyny pro vypracování:

1. Navrhněte a realizujte „inteligentní“ systém zálohovaného napájení atomových hodin umístěných v Laboratoři přesného času a frekvence FEL.
2. Navrhněte a realizujte tester pro zjišťování stavu Pb akumulátorů s kapacitou v řádu desítek Ah použitých pro zálohování. Tester použijte pro kontinuální monitoring stavu zálohovacích akumulátorů.
3. Vypracujte a implementujte metodiku pro zajištění optimálního provozu použitých akumulátorů s ohledem na dosažení vysoké spolehlivosti zálohovacího systému a maximální životnosti použitých akumulátorů.
4. Systém by měl umožňovat lokální i vzdálený přístup s možnostmi řízení, diagnostiky systému, přenosu dat a hlášení kritických a havarijních stavů (např. nepřijatelného poklesu napětí na akumulátorech) pomocí e-mailu nebo SMS.

Seznam doporučené literatury:

- [1] Chang W.: The State of Charge Estimating Methods for Battery A Review. Hindawi Publishing Corporation, Volume 2013, Article ID 953792.
[2] Piller S., Perrin M., Jossen A.: Methods for state-of-charge determination and their applications [online]. Journal of Power Sources, Volume 96, Issue 1, 1.6.2001, p. 13-120.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Jaroslav Roztočil, CSc., katedra měření FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **11.02.2019**

Termín odevzdání diplomové práce: **07.01.2020**

Platnost zadání diplomové práce:

do konce letního semestru 2020/2021

doc. Ing. Jaroslav Roztočil, CSc.
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Ing. Pavel Ripka, CSc.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

Datum převzetí zadání

Podpis studenta

Acknowledgements

Děkuji panu Doc. Ing. J. Roztočilovi, CSc. dále Doc. Ing. Janu Fischerovi, CSc. za podněty při tvorbě diplomové práce. Dále také děkuji své rodině za podporu. V neposlední řadě mé velké díky patří Petru Brožovi, Kateřině "Kačeně Stračeně" Stránské a Eduardu Bakalářovi za podporu a pomoc nejen při tvorbě diplomové práce, ale úplně ve všem.

Declaration

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 19. August 2019

Abstract

This work focuses on the design of a system for power backup of atomic clock and for the continuous monitoring of the current state of the batteries used in such a setup. The device built as a part of this work also enables the detection of power outages. System states can also be reported over the GSM network, both automatically and on-demand.

Keywords: battery, accumulator, testing, system for backup power, atomic clock

Supervisor: Doc. Ing. Jaroslav Roztočil, CSc.

Abstrakt

Tato práce se zabývá návrhem systému pro zálohované napájení atomových hodin a pro monitorování stavu použitých akumulátorů. Zařízení zkonstruované jako součást práce rovněž umožňuje detekci výpadků napájení. Stav systému dokáže zařízení hlásit pomocí sítě GSM, jak automaticky, tak na vyžádání.

Klíčová slova: baterie, akumulátor, testování, systém pro zálohování napájení, atomové hodiny

Překlad názvu: Systém zálohovaného napájení atomových hodin

Contents

Part I	
Assignment analysis	
Part II	
Uninterruptable power source	
Part III	
System for reporting the battery state and power outages	
1 Introduction	10
2 Hardware	12
2.1 Microcontroller	12
2.2 GSM Module	14
2.3 Physical user interface	14
2.4 Li-pol battery and relevant management circuitry	15
2.5 Power supply	15
2.6 Power outage detection	16
2.7 Power supply filtering	17
2.8 Simple analog frontend	17
2.9 Lead-acid batteries	18
2.10 Internal connections	19
3 Operation	20
3.1 Information displayed on the LCD screen	21
3.2 SMS messages	21
3.3 Alarm	25
3.4 Functionality of the physical button	26
3.5 Data logging	26
3.6 PC communication	26
3.6.1 Target phone number configuration	27
3.6.2 Extracting generic information	28
3.6.3 Data logging	28
3.6.4 Automatic timeout	30
3.7 PC application	30
3.7.1 Port settings	30
3.7.2 Phone number	31
3.7.3 Sending generic information to the PC	32

3.7.4 Displaying chart and event history	32	11.0.2 Operation principles	56
4 Precision and accuracy of the measurement of lead-acid batteries	33	11.0.3 Object oriented design basic concept	58
5 Third party libraries used	36	11.0.4 Multiple users and synchronization	58
6 Further development	37	11.0.5 Working with the library...	59
Part IV Previous attempts		12 Other attempts	61
7 Adjustable load built around a single MOSFET	43	Part V Recommendations regarding the battery maintenance	
8 Adjustable load based on a group of individually controlled mosfets	45	Part VI Conclusion	
9 Resistor-based load	48	Appendices	
9.1 Generic description	48	A Unfinished previous diploma thesis	72
9.2 Problems of the complicated solution	50	B Used abbreviations	73
10 Resistor-based load - prototype stage	54	C Bibliography	75
11 Resistor-based load - old firmware	55		
11.0.1 GUI library	55		


Figures

1.1 Physical form of the system for battery management	11	7.1 Adjustable load utilizing a single transistor	44
2.1 Block schematic of the system for battery management	13	8.1 Load module schematic	46
2.2 Maximum frequency vs. voltage for the atmega328p microcontroller	13	8.2 Multiple modules of opamp-driven mosfets on a large heatsink	47
2.3 SIM800L module source: nettigo.eu	14	9.1 Block schematic of the resistor-based load	50
2.4 Clamping diodes	16	9.2 Layout of the hardware components of the resistor-based load	51
2.5 Schematic of the device	19	10.1 Prototype for the resistor-based load	54
3.1 Information displayed on the LCD	22	11.1 GUI built using my library	56
3.2 GUI of the Python application	31	12.1 Number of cycles vs. depth of discharge (source: https://pvc-drom.pveducation.org/BATTERY/charlead.htm)	65
3.3 Charts with events	32	12.2 Battery tester	69
4.1 Zener diode temperature coefficient Source: http://users.tpg.com.au/ldbutler/ZenerDiodes.htm	35		
6.1 Back side of the PCB	38		
6.2 Front side of the PCB	38		
6.3 Enclosure design	39		



Part I

Assignment analysis



The assignment states that I should build a system, which provides backup power to a 5071A cesium primary frequency standard in case of power outages, while simultaneously being able to report the current state of the batteries remotely.

It is crucial to ensure that the frequency standard is running at all times, 24/7, as it is an official member of the group of national standards of Czech Metrology Institute. In case of a power outage, a backup power source must be used. It's also necessary to inform the user about the fact that the mains power is no longer available, which in turn allows them to take corrective measures.

As the system is used for online measurement and it tests batteries, state of which should be kept as good as possible at all times, the requirements imposed on it differ from the requirements on a system intended for offline testing, such as the one built by ing. L. Dastych, as a part of his diploma thesis, also written under the guidance of Doc. Ing. J. Roztočil, CSc. [Das] Namely, such a system should impose no additional stress on the batteries, should not discharge them, and so on. For this reason, I have concluded that it is satisfactory to rely on the measurements of open-circuit voltage solely. For lead-acid batteries, this measurement serves well for the estimation of the state of charge. It is the most common method to determine the state of charge. This method is affected when the battery is loaded - its voltage drops, and even once the load is released, it takes hours for it to return to the original value. [How] Since the batteries in the system will not be under load for the majority of time, this is acceptable.

For the reasons listed above, I have decided to build a device, which allows the frequency standard to be backed up by 2 lead-acid batteries while displaying their state of charge on screen and allowing the readings to be accessed over the GSM network, while also reporting critical states (such as a low state of charge of the batteries or power outages).

The problem of ensuring the frequency standard is powered in case of power outages, and the problem of reporting the battery status can be decoupled. This is reflected in the structure of this thesis. The first part discusses some of the considerations when designing the uninterruptable power source. The second part covers the design of the system for reporting the battery state and power outages; the third part then talks about my previous attempts to solve the problem and about the various prototypes. The next, fourth part speaks about recommendations related to the battery maintenance and the final part summarizes my findings and results.



Part II

Uninterruptable power source

The task of designing an uninterruptable power supply for the frequency standard is comparatively trivial to the task of designing a system that reports power outages and the current state of the batteries.

The Microsemi 5071A frequency standard is normally powered from mains (230V) and offers a multitude of options when it comes to providing an uninterruptable power supply for it. Firstly, it contains an internal lead-acid battery, which can allegedly keep it powered for up to 45 minutes in case of a power outage. This being said, this figure describes an ideal case, since it assumes a perfect state of the battery used. Even then, this duration is not adequate, since the outage event may last up to several hours. Luckily, there are ways to extend this period. The standard allows an external DC power source to be connected to it. The voltage of this power source should be between 22 and 42 volts. [Mic]

Naturally, an alternative would be to use a commercial-grade UPS device, connecting it between the 230V power input of the standard and a wall socket. However, such an approach would introduce inefficiencies. Those are mainly related to the downconversion of the 230V to the voltage used by the batteries in the UPS device, and the upconversion necessary to power the standard during the power outage. There are some other issues related to the commercial-grade UPS devices, such as additional introduced costs, unknown internal circuitry, and more. [Das] Last but not least, any attempt to monitor the state of a battery, mounted inside a commercial UPS would necessitate some form of hardware hacking of it - this might imply getting dangerously close to high voltage potentials, and overall I just see no reason to take this path.

When we concentrate on the other option, that is supplying backup power to the standard over the DC input, we see that we can use 2 12V nominal lead-acid batteries connected in series, to obtain one larger battery with a nominal voltage of 24V.

The correspondence between the open-circuit voltage of a lead-acid battery and its SoC is very close to linear, and the shape is dependant on the specific type of battery. I believe that flooded batteries are good for this application since they are much less expensive than other options. [dee19].

For flooded batteries, such a serial combination will have an open-circuit voltage of around 25.3V when fully charged and about 23.7V when fully discharged.[SL]

Multiple 24V batteries can be connected in parallel to increase the overall capacity. There is only one problem - if the voltages of the individual batteries differ, once the batteries are connected in parallel, large currents start flowing until the voltages reach equilibrium. This is very undesirable, as such currents generate excessive heat, make it impossible to use protective fuses (since those would always burn under the circumstances described), and more. Luckily, a simple solution is to put a diode in series with every battery. When a load is connected to the output terminals, current flows into it only from the battery, which has the highest voltage.

A drawback of this approach is that there is a certain amount of voltage drop across the diode. However, this drop is only around 0.6V for silicon-based diodes and is comparatively small to the nominal voltage of the two lead-acid batteries in series. Furthermore, the minimum required input voltage of the frequency standard is already about 1V lower than the full-discharge voltage of the 24 battery.

Assuming two couples of 50 Ah, 12V lead-acid batteries are used, we get 100 Ah of capacity. Given the power consumption of the frequency standard is around 50W [Mic], assuming the average voltage of the batteries to be 24V, for the average current we get

$$I_{avg} = \frac{P_{avg}}{U_{avg}} = \frac{50}{24} = 2.08A \quad (1)$$

From there, we can easily calculate the time for which the batteries will be able to provide backup power as

$$t = \frac{C_{bat}}{I_{avg}} = \frac{100Ah}{2.08A} = 48.07h \quad (2)$$

That is, the batteries should be capable of providing backup power for around two days.



Part III

System for reporting the battery state and power outages



Chapter 1

Introduction

As mentioned above, the problem of providing a backup power source for the atomic clock was decoupled from the problem of reporting the current battery state. What remains is to solve the second problem, which is done by introducing the system described henceforth.

The physical form of the device is depicted in image 1.1.

The device measures the open-circuit voltage of two lead-acid batteries, displaying it on the LCD screen. It then takes this voltage and uses it to estimate the SoC of the battery. It also monitors the state of mains it's connected to - In case of a power outage, it uses a built-in Lithium-Ion battery to remain operational for up to 5 days. In case one of the voltages drops to a critical level, it sends an SMS message to the user. A message is also sent as soon as a power outage occurs. Such critical states are also reported audibly by the use of a siren. The user can optionally also send an SMS message to request the report of current status.

Additionally, the device logs the voltages and different events periodically. A purpose-built computer application can be used to extract the logs, as well as to change different settings.



Figure 1.1: Physical form of the system for battery management



Chapter 2

Hardware

The basic block schematic of the system is in image 2.1

In the next sections, I will describe the block schematic in detail.



2.1 Microcontroller

I used Arduino Nano as the main processor board. I am familiar with the Arduino platform, and this particular board was very inexpensive at around 50 CZK. I did not require any special features from the processor board (such as Wi-Fi); The board features a built-in voltage reference and a 10-bit ADC. It has a very small footprint and can be put in low-power sleep modes. It can be powered from voltages as low as 2.7 volts when the main clock frequency is set to 8 MHz. (See image 2.2.) [ATM, p. 260]

Since the system is powered from a Li-Ion battery, voltage of which is nominally 4.2V, it would be out of specs to run it on 16 Mhz. Despite being designed to run at 16 MHz, the board can be configured to run at 8 MHz just by software modification. [Low] This being said, while running atmega328p based boards on 16 MHz at 3.3V is out of specs, it is possible. [Coo] I have confirmed this myself, concluding that the lower voltages are probably only a problem for EEPROM or FLASH writes, where they might cause memory corruption. [ATM, p. 19] I still do not recommend running any device outside

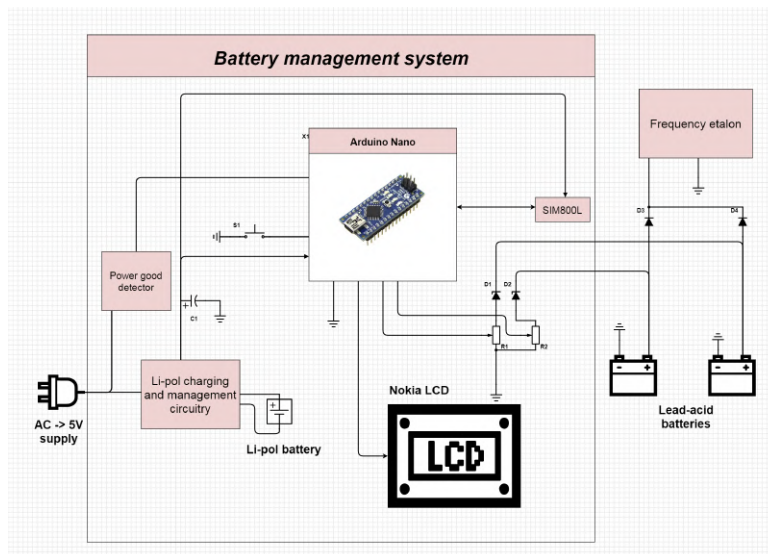


Figure 2.1: Block schematic of the system for battery management

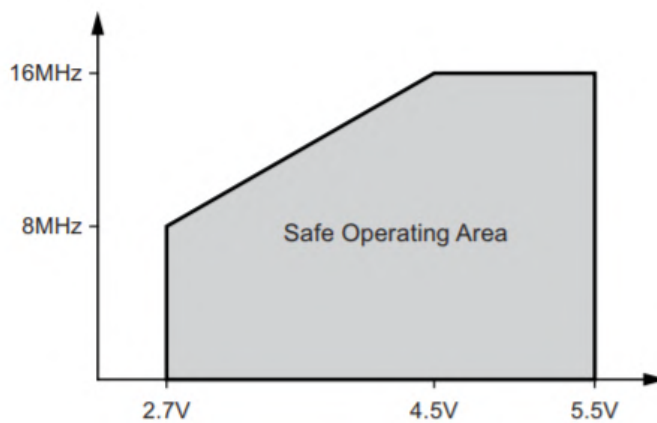


Figure 2.2: Maximum frequency vs. voltage for the atmega328p microcontroller

its specifications. For this reason, I have modified the bootloader of the Arduino to run at 8 Mhz, using the steps described in [Low].

Should the whole device be recreated, a PCB design I have created can serve to make the process of creating additional copies as easy as possible. This design incorporates a 3.3V version of Arduino pro-mini instead, but otherwise is virtually identical. The different processor board removes the step of bootloader modification, mentioned above, simplifying the build process. The software is compatible, and no changes to it are required.

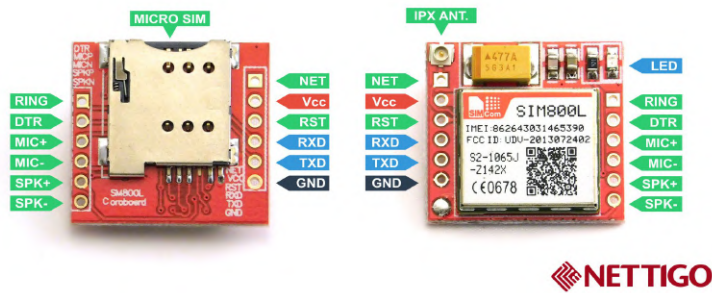


Figure 2.3: SIM800L module
source: nettigo.eu

2.2 GSM Module

The SIM800L GSM module is used as a means of communicating with the outer world, reporting the state of the batteries, and more. I have used a board with the useful pins broken out and am communicating with it over TTL-compatible UART. This board can be seen in image 2.3.

The communication has a form of AT commands. [SIMa] The device operates from 3.4 to 4.2 volts and is designed to run off Li-ion batteries. [Simc, p. 22] I am using it to send (and respond to) SMS messages over the GSM network. The module I used accepts micro-size SIM cards and also has an onboard antenna. An external antenna can be connected using an onboard connector, if necessary. The chip can enter various sleep modes. The current consumption in those modes is around 1mA with occasional peaks. [Simb]

2.3 Physical user interface

I am using an LCD module, featuring a monochromatic display, which is of the same type as the one used in Nokia 5110 mobile phones. The display itself is mounted on a printed circuit board, which provides easy access to the display pins and also adds blue backlight. The current draw of this display is very low - around 0.4mA with disabled backlight and 1-10 mA with the backlight enabled. The communication protocol used by the display is SPI. The resolution of the LCD is 48x84 pixels; this is enough to display 14x6 characters, using a typical 5 x 7 font. [LCD11]

A normally open, push-button switch is used as the only input user interface element. When pushed, it triggers an interrupt; the actions that follow are described in chapter “Operation.”

I am using an SFM-27 siren to alert the user in case of some critical state of the system. This siren can operate on voltages from 3-27V and only draws a few milliamperes when active. It oscillates automatically, meaning the sound can be heard as soon as the siren gets connected to DC. The low current draw also means the siren can be powered directly from the pin of the Arduino. The volume should exceed 90dB, according to the reseller. [BUZ] I have found that the sound is deafening at the upper range of the allowed voltage levels, while it is very noticeable, yet not deafening at the voltage levels around 4V that I use it on.

■ 2.4 Li-pol battery and relevant management circuitry

A 2000mAh li-pol battery is used to keep the monitoring system operational in the event of a power outage.

I am using a li-pol charger module with an under-voltage protection, based on the TP4056 integrated circuit. It operates at 5V and charges the connected battery up to 4.2V with currents up to 1A. When the battery gets discharged by the external load down to around 2.5V, it is forcibly disconnected to prevent it from being damaged. [TP4]

■ 2.5 Power supply

A standard 5V USB phone adaptor is used. The device is plugged into the USB socket of the USB adaptor; it is recommended to use a USB supply capable of providing at least 2 A, since the li-pol charging circuitry can draw up to 1A.

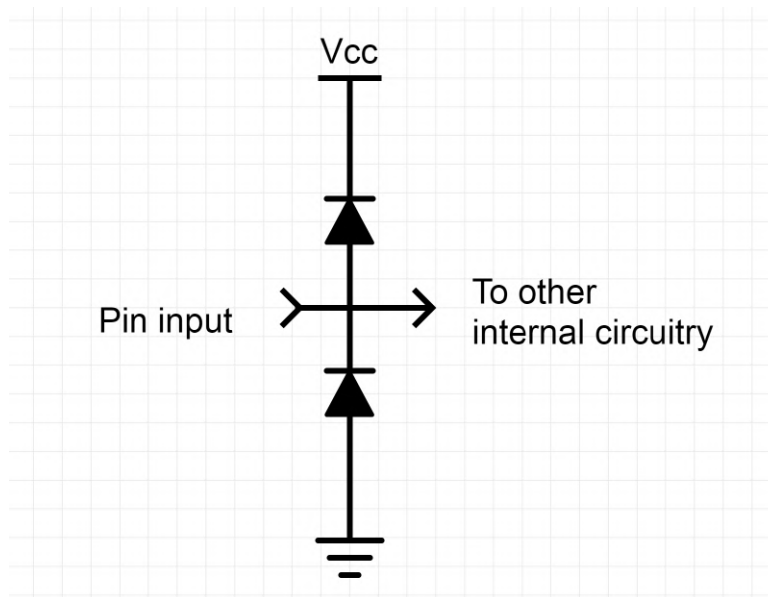


Figure 2.4: Clamping diodes

2.6 Power outage detection

The power good detector is just a single 10K resistor connected between one of the analog inputs of the Arduino and the 5V output of the charger. The atmega328p processor used by the Arduino uses ESD-protection diodes on each pin. [ATM, p. 58] These diodes make sure that the voltages on the pin never significantly exceed the supply voltage of the chip (as the upper bound) and zero volts (as the lower bound). See figure 2.4 for reference of the internal circuitry.

Since the Arduino is powered from the li-pol battery, the voltage of which is well below the 5V introduced by the charger, the two voltages would be “shorted” across the protection diode, if it weren’t for the resistor. In the case of a power outage, the phone adaptor input voltage drops to zero, and eventually (once the output capacitor inside the adaptor discharges), the output voltage drops to zero as well. This drop is detected by reading the analog input of the Arduino.

2.7 Power supply filtering

The SIM800L module can, in edge cases during transmission bursts, draw peak currents up to 2A. [Simc] When the power supply is unable to cover those current peaks, the voltage on the power input of the GSM module drops, rendering the module unable to operate properly, which often causes it to refuse to connect to the GSM network. [GSM] I have encountered this problem during my testing. Therefore, I am using a large 2200uF electrolytic, low ESR capacitor to make sure the current draw peaks are covered, and the voltage never drops below the critical threshold.

2.8 Simple analog frontend

The internal reference of the ATmega328 is used for the measurement of lead-acid batteries. Since the nominal voltage of the batteries is around 24V and it is only possible to directly measure voltages of magnitudes lower than the internal reference voltage, which is about 1.1V, a downconversion is needed.

Furthermore, the minimum voltage that we desire to measure is not zero. The 24V lead-acid batteries should never be discharged below 20V since such an event would drastically shorten their lifespan.[Bat] When the voltage of the battery drops below 20V, it is no longer capable of providing power to the frequency standard nonetheless. Therefore, it only makes sense to measure voltages in the range of around 20 to 30V.

Analog to digital converters only have a finite resolution. If we were to use it to measure the whole range from 0 to 30V, and our useful range is 20-30V, two-thirds of the ADC range would never be used. For those reasons, I have decided to use Zener diodes and potentiometers to map the battery voltages to useful ranges.

Zener diodes D1 and D2, both with breakdown voltages equal to 18V are connected in series with R_1 and R_2 trimmer potentiometers. The potentiometer keeps the current flowing through the Zener diode at a defined low level; this means that the voltage drop across the diode is almost constant with the changing voltage of the lead-acid batteries.

For the voltage across the trimmer we get:

$$U_t = U_{bat} - U_d \quad (2.1)$$

where U_t is the voltage across the trimmer terminals (the two that aren't the wiper terminal), U_d is the reverse breakdown voltage of the Zener diode, and U_{bat} is the voltage of the lead-acid battery.

Effectively, we subtracted the breakdown voltage of the zener diode from the battery voltage. As a result, we can now utilize a larger portion of the range of our ADC. The Zener diode has a voltage drop of around 18V. Then for battery voltages from 18 to 30V, the voltage across the trimmer ranges from 0 to 12V, respectively.

Now that the offset has been taken care of, it is necessary to use a voltage divider to convert it to range from 0 to 1.2V. The internal voltage reference of the Arduino, used for the measurements can be off by almost 10%. [ATM, p. 262] For this reason, it is useful for the voltage divider to be adjustable, allowing for calibration. Therefore, ten-turn, 10K trimmer potentiometers were used.

■ 2.9 Lead-acid batteries

Two lead-acid batteries are measured at the same time. Each battery depicted is a series combination of two 12V batteries. No assumptions about their capacity are made, but the algorithm for computing the state of charge from the open-circuit voltage was tuned to produce the best results with flooded batteries.

The batteries are each connected in series with diodes D3, D4, and these series circuits are then joined in parallel, as depicted in image 2.1 ; those prevent currents from flowing from one battery to another in case one has a slightly higher voltage, but still allow the batteries to power the frequency standard.



Chapter 3

Operation

The system periodically checks for power outages and measures the voltage of the batteries, displaying the relevant information on the LCD screen. It also calculates the state of charge from the voltages measured using linear interpolation, assuming that 23.7V corresponds to 0% and 25.3 corresponds to 100%. It is capable of automatically sending text messages - either on request or when a certain event occurs. A high volume siren is used as secondary means of alerting the user about a potential problem. A single button can be used to silence the siren and also serves a couple of additional purposes, which will be mentioned down below. The system only takes the measurements sporadically, since it makes no sense to do it very often, as the voltages are expected to change very gradually anyway. It sleeps most of the time; both the Arduino and the SIM800L module are kept in sleep mode most of the time.

In case of a power outage, the system is running off of its internal battery, the period between the measurements (and thus the period for which the system sleeps) is increased to around 30 seconds (from the normal 8 seconds) to preserve energy.

Additionally, whenever the user is supposed to run maintenance of the batteries, they are sent an SMS message, instructing them to recharge them.

The voltage of the lead-acid batteries, as well as the state of the internal battery, are logged periodically and stored into internal memory. This is also true for various events, such as the triggering of an alarm, presses of the button, and more. The logs can be extracted by connecting the device to a

PC and running a special application I have written.

■ 3.1 Information displayed on the LCD screen

Given the low current draw of the LCD screen used, it is kept on at all times. The following information is displayed:

- AC status - whether there is a power outage or the mains power is present
- Battery 1 voltage and estimated state of charge
- Battery 2 voltage and estimated state of charge
- Number of messages left that the system is able to send (see section “SMS messages” for more info)
- Credit left on the card, in CZK

Image 3.1 depicts the system displaying the relevant information.

When the user presses the button, the LCD backlight turns on for around 5 seconds. Each time the system wakes up, the backlight flashes briefly to indicate correct operation of the system.

■ 3.2 SMS messages

The system is capable of sending SMS messages whenever a power outage occurs and also when the power is restored. Additionally, it sends messages automatically as the batteries discharge (either by self-discharge or as a result of the fact they are being used to power the standard). It is also possible to request a report by sending an SMS message to the phone number assigned to the SIM card inside the device. The system is indifferent to the actual content of such a message. An empty message works as well. The text of the message sent by the device is almost always identical to the text displayed on the screen.



Figure 3.1: Information displayed on the LCD

The maximum number of messages sent per one day is limited to prevent the user from getting “spammed” and also to save money (since sending one SMS usually costs some small amount). When the maximum limit is reached, the system sends a special message, notifying the user about this fact. Sending any message to the device has the side effect of resetting the SMS counter, allowing further messages to be transceived. The reasoning behind this is that when the user wants updated information, they send an SMS, which also reallows further updates. The SMS counter also resets to zero automatically every 24 hours.

Deciding about the conditions under which should the state of charge be reported over an SMS message is not an easy task. On the one hand, the reports shouldn’t be sent too often, since each message costs some small amount of money to be sent and also to prevent the user from getting spammed. On the other hand, they should be sent often enough to provide the user with up-to-date information about the actual battery state.

It makes sense for the state of charge to be reported each time it changes by a fixed amount. As the SoC of the two batteries can differ, it seems like the best thing to do is to keep track of the states of the two batteries independently.

This being said, the circuit the batteries are connected in tends to eventually balance their voltages, since only the battery with higher voltage is getting discharged. In case the two batteries share the same states of charge and such states decrease simultaneously, for each battery, the same report would be sent at the same time instant. Such pointless duplication would not be very useful.

It is also important to consider that a charger might get connected to the battery, and its state of charge will raise back up and to take this fact into account.

For the reasons given above, a complex set of rules for reporting the SoC values has been developed. I would hereby like to describe it in further detail.

The state of the batteries is reported when the SoC of them drops below a certain threshold. Initially, when the system is first turned on, the thresholds are calculated as

$$SoC_{thres_1} = SoC_1 - x \quad (3.1)$$

$$SoC_{thres_2} = SoC_2 - x \quad (3.2)$$

Where SoC_n is the state of charge of the battery, expressed in percent, SoC_{thres_n} is the new threshold, and x is 10 when the respective SoC is below 50% and 20 otherwise.

When one of the SoC values falls under their respective threshold, an SMS message is sent; then, the equations 3.2 are applied again to recalculate the thresholds. The thresholds are also recalculated when the SoC raises 10% above its respective threshold.

The following example demonstrates the behavior of this mechanism.

- the system is first turned on, battery 1 is at 99%; battery 2 is at 100%.

This sets the thresholds to 79% and 80%, respectively.

- both batteries discharge to 80%. Since the battery 2 SoC is now equal below its threshold(80%), The system sends an SMS message, and the new thresholds are both set to 60%.
- the battery 1 discharges to 79% (hitting its initial threshold), but this is not reported, since the new thresholds have been set in the previous step
- the batteries discharge to 60%, the thresholds are set to 40%, which is reported
- the batteries discharge to 40%, the thresholds are set to 30% since the SoC is below 50%; this is reported
- the batteries discharge to 30%, which is, again, reported using SMS; the thresholds for both batteries are now 20%
- the battery 1 is connected to a charger; the voltage starts raising
- the voltage of battery 1 raises above 40% (that is, 10% above the threshold). The new threshold is set to $40 - 10 = 30\%$ (which has no effect in this particular case, since the new value is identical to the previous one); this is not reported, but the user can manually prompt the system to obtain the battery status
- the voltage of battery 1 raises above 50% (that is, 10% above the previous threshold). The new threshold is set to $50 - 20 = 30\%$ (since $x = 20$ now that $SoC > 50$. (which has no effect in this particular case, since the new value is identical to the previous one). This is not reported, but the user can manually prompt the system to obtain the battery status.
- the voltage of battery 1 raises above 60% (that is, 10% above the previous threshold). The new threshold is set to $60 - 20 = 40\%$. This is not reported, but the user can manually prompt the system to obtain the battery status.
- the voltage of battery 1 raises above 70% (that is, 10% above the previous threshold). The new threshold is set to $70 - 20 = 50\%$. This is not reported, but the user can manually prompt the system to obtain the battery status.
- repeating the process outlined above, the voltage of battery 1 rises to 100%. The new threshold for battery 1 is 80%, while the threshold for battery 2 is 20%. This is not reported, but the user can manually prompt the system to obtain the battery status.
- charger is disconnected
- battery 1 discharges below 80%, this is reported.
- (...)

As we can see, this mechanism implements hysteresis to cope with different events to report the battery voltages when relevant.

■ 3.3 Alarm

An alarm functionality was implemented to alert the user about problems when they are nearby. The alarm has three states: Triggered, armed, and disarmed.

When in the “triggered” state, the system emits loud, high pitch beeps, alerting the user about a potential problem. It can transition from the “armed” state to the “triggered” state when a problem is detected, that is, under one of the following circumstances:

- Battery 1 state of charge drops below 25%
- Battery 2 state of charge drops below 25%
- Power outage was detected

When the alarm is in the “triggered” state, the user can push a physical button on the device to transition it into the “disarmed” state. In this state, all sounds are turned off for user convenience. This could be useful when the user is already working on solving the problem and finds the loud alarm annoying. The alarm transitions back into the “armed” state from the “disarmed” or “triggered” states one once all the issues have been resolved, that is, when and only when all of the following conditions hold:

- Battery 1 state of charge is above 30%
- Battery 2 state of charge is above 30%
- AC power is present

The reason why the battery SoC thresholds are different for the alarm to transition from “armed” to “triggered” and for it to switch from “disarmed or triggered” to “armed” is that a certain amount of hysteresis is used to prevent the alarm to erratically jumping between states when the battery state of charge level floats right around the 25%.

■ 3.4 Functionality of the physical button

Pressing the button has several effects, listed below:

- the system wakes up from sleep, taking measurements
- the LCD backlight is activated for about 5 seconds
- the SMS counter (see “SMS messages”) is reset
- If the alarm state is “triggered,” it is changed to “disarmed” (see “Alarm”)

■ 3.5 Data logging

The system is capable of logging the state of charge of the lead-acid batteries, the internal battery, and of logging various events, such as power outages. The 1kB internal EEPROM memory of the Arduino Nano processor board is used to store the data. For more information, refer to section “PC Communication,” subsection “Data logging”.

■ 3.6 PC communication

The device can optionally be connected to a personal computer over a USB, which internally uses a USB to serial converter; then, the logs can be extracted, and measurements can be taken on the fly. I have written an app in Python, which can optionally be used to connect to the device. The API I wrote is open, enabling the potential creation of third-party applications.

The device can be connected to a PC over the USB interface; it makes use of a virtual COM port, meaning it behaves as if it was connected over a standard serial port, despite the fact a USB interface is used. The baud rate the device communicates on is 9600 bits per second.

In the following text, I will use “\r” to denote the carriage return ASCII character (decimal 13), and “\n” to denote the line feed ASCII character

(decimal 10). Additionally, I will use “\4” to denote the end of transmission ASCII character (decimal 4).

The device operates in two different modes. When the PC is connected, it’s ready to accept instructions; when connected to a pc, no measurements are taken automatically, the device doesn’t send any messages, and it only “does what it’s told to do”. The reason is that many actions carried out as a part of the regular operation (i.e. sleeping, measuring the batteries, etc.) are blocking and would introduce large delays, should the device also respond to commands over serial. In summary, there are two modes the device can run in; regular mode (where it cannot respond to commands) and PC mode (where it responds to commands).

When the serial port is open, the device automatically resets. Upon reset, it sends the string “INIT\r\n” on the serial port, announcing its presence and then waits for 30 seconds. If it receives characters over the serial port during this 30s window, it automatically enters the PC mode. Otherwise, upon timeout, it enters the normal mode. The device also automatically enters normal mode when no characters have been received for over 30 seconds when in PC mode. What follows is that it is imperative to periodically send queries to the device to make sure it doesn’t stop listening. A special command “PING” can be used for this purpose.

All the commands this device recognizes are written in capital letters in ASCII and are terminated by “\r\n”. Most commands take no arguments. Whenever the command takes arguments, it takes the form of “COMMAND|ARGUMENT1|ARGUMENT2\r\n”, that is, the “|” character is used as a separator. The reply to most commands is written in ASCII, ended by the “\4” character. This allows the reply to span multiple lines, which wouldn’t be possible, were the reply terminated with the “\n” character.

What follows is the list of commands, their parameters, and explanations of their functionality.

■ 3.6.1 Target phone number configuration

In case the user decides to change their phone number, it is important that they have the option to ensure the status reports will be sent to the new one instead of the old one. To enable the editing of the stored target number, the SETPHONENUM command can be used. This command takes one parameter,

that is, it must be issued in the form “SETPHONENUM|123456789\r\n”. The device then stores this phone number into the memory of the SIM card and will from there on use it as a target number, to which it will send SMS messages. If the number is stored successfully, the text “OK\r\n\4” is sent back. Upon failure, no reply is sent.

Sometimes, the user might want to know which phone number was previously set as the target number, for example, to ensure the setting is correct. The GETPHONENUM command was implemented precisely for this reason. This command takes no parameters. It echoes the target number (as saved by SETPHONENUM) over the serial port, in plain ASCII, terminating with the “\4” character.

■ 3.6.2 Extracting generic information

To extract generic information regarding the state of the lead-acid batteries, of the internal battery and more, the “GETINFO” command can be used. This command takes no arguments and echoes a textual representation of the status. This text is terminated by the “\4” character, contains newline characters, and is identical to the text regularly displayed on the LCD screen in the regular mode. That is, it contains all the information as described in section “Information displayed on the LCD screen” above.

■ 3.6.3 Data logging

In order to download logs from the system, the SHOWLOGS command can be used. This command takes no arguments. It requests all the logs stored on the device to be sent to the PC. This includes logs of events (such as power outages) and periodic measurements for the individual batteries. The reply is NOT in ASCII format, but instead, it is a stream of bytes. Since no terminating character can be used (by the nature of the data), the end of reply can be determined by the number of characters received. The reply is always 999 bytes long. It contains 111 log entries. Each entry consists of 9 bytes; there are 2 kinds of entries. The first kind is a regular log, noting the voltage of the two lead-acid batteries and the state of the internal battery. The second one is an event log. For a regular log, the meaning of the individual 9 bytes in each entry is as follows:

- bytes 1-4: Time (unsigned long). The timestamp of the measurement; the encoding of this timestamp is similar to UNIX time, with one exception: it already takes timezones and daylight savings into account. That is, when converted under the assumption it's a Unix time, the result is directly the local time at which the measurement was taken, and no other adjustments are required.
- bytes 5-6: Voltage of battery 1, in millivolts (unsigned int).
- bytes 7-8: Voltage of battery 2, in millivolts (unsigned int).
- byte 9: State of the internal battery, in percent (unsigned char).

All the data types are stored in little-endian format, that is, the most significant byte is last (!). This is related to the underlying architecture of the processor used.

For an event log, the meaning of the bytes is different.

- bytes 1-4: Time (unsigned long). Same as before.
- bytes 5-6: Log type identifier (unsigned int). Since the range of the voltages that can be measured starts at around 18000 mV, it is possible to assign special meaning to numbers below this threshold, indicating the log is, in fact, an event log, as opposed to a regular log, while also specifying its type.
- bytes 7-9: (reserved)

The type of the log is an unsigned integer, having one of the following values:

- 0: AC Disconnected - event which fires once the device detects that it no longer is connected to mains
- 1: AC Reconnected - the device was connected back to mains
- 2: Alarm triggered - something triggered an alarm
- 3: Alarm resolved - all problems that were initially causing the alarm to run were resolved, the alarm was automatically deactivated.
- 4: Button pressed - user pressed the button

- 5: SMS received - user requested the current status over an SMS
- 6: Charger connected - connection of a charger to the lead-acid batteries was detected

■ 3.6.4 Automatic timeout

Whenever the system receives the “PING” command, which takes no arguments, it replies with “PONG \r \n \4”. This command is mainly used to prevent the system from entering the normal mode, and to assure that it is running and responsive. The system automatically enters “normal mode,” as described above, when it hasn’t received any data over the serial port for around 30 seconds.

■ 3.7 PC application

I have written a simple application in Python, which utilizes the communication API, as described above. The screenshot of this application can be seen in image 3.2.

The user interface is very straightforward and is written using the AppJar Python GUI Library. It allows the user to view the current status of the device, as well as display the historical data in charts and set the phone number.

■ 3.7.1 Port settings

The user can select the correct port in the “PORT” list box and then hit “Connect” to connect to the device on the indicated port. Once the connection is established, the “connect” button is disabled, and the “disconnect” button is enabled, as well as the rest of the user interface. When establishing the connection, the PC waits for the device to send the “INIT \r \n \4” ASCII string; If this string is not sent for 50 seconds, the PC gives up. Once the connection has been established, the PC starts sending the “PING” command (see above) periodically every second. This prevents the device from entering the “normal” mode and also makes sure the device is present. When no reply

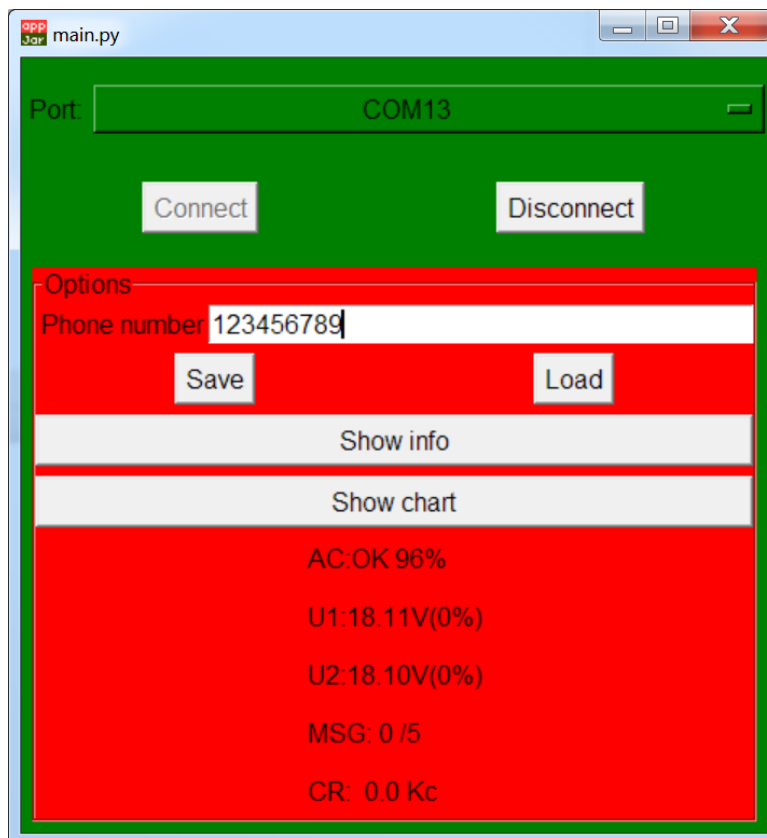


Figure 3.2: GUI of the Python application

is detected, the app detects this, reverting to the state where no devices are connected, disabling the GUI elements.

■ 3.7.2 Phone number

Two buttons and a text field are used to save the phone number to the SIM card inserted in the device. This number is used to determine where should the text messages be sent. To view the current phone number, the user presses the “load” button; upon such action, the text field is filled up with the phone number previously stored. “SETPHONENUM” and “GETPHONENUM” commands are used for this functionality.

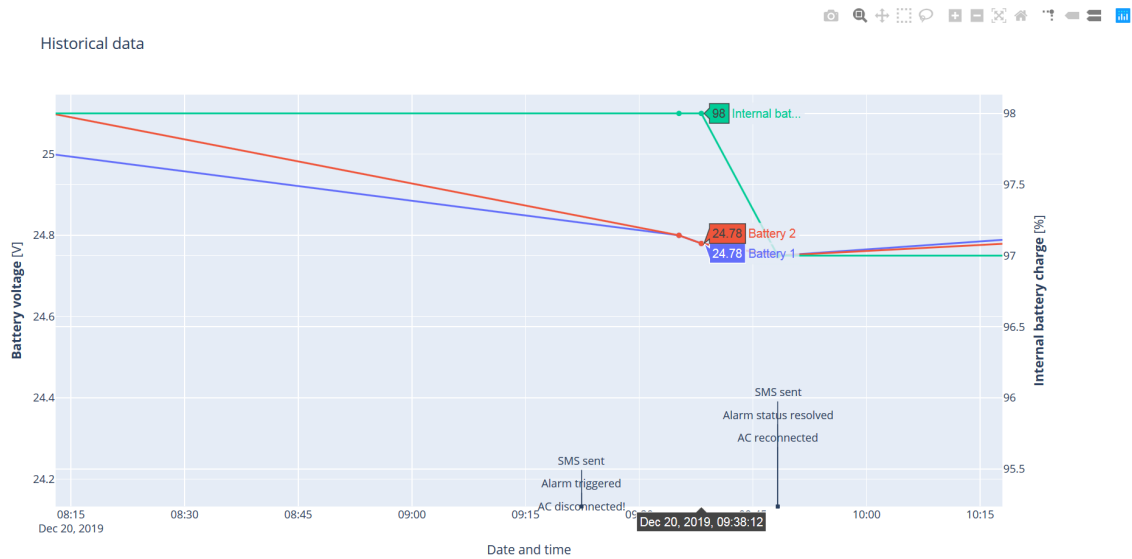


Figure 3.3: Charts with events

3.7.3 Sending generic information to the PC

The “Show info” button updates the information displayed on the bottom of the screen; this information is identical to the text displayed on the screen when the device is in the normal mode. The “GETINFO” command is used internally.

3.7.4 Displaying chart and event history

The “Show chart” button opens a browser window, displaying a chart with events plotted on it. An example of such a chart can be seen in image 3.3. Internally, the “SHOWLOGS” command is used.

Chapter 4

Precision and accuracy of the measurement of lead-acid batteries

The internal ADC of the ATMEGA328P and its internal 1.1V reference are used to take the measurements. One thousand consecutive measurements are taken, and the arithmetical average is calculated.

Since it is a 10-bit successive-approximation ADC, it should be theoretically capable of differentiating between $2^{10} = 1024$ different values (words). [ATM] Empirically, however, I have found that measurements of voltages close to the “knee” voltage of the diode are inaccurate, as the non-linearity of the diode becomes a significant problem. It would be theoretically possible to use a look-up table and perform corrections, but I don’t believe this path is worth the extra effort.

Since the lower range of the ADC isn’t utilized, the total useful range is reduced, specifically to 840 words. The voltage range spans from 18.4 to 30V, yielding a difference of $30 - 18.4 = 11.6V$. Therefore, for the smallest voltage difference (step) the ADC can measure we get $U_{step} = \frac{11.6}{840} = 0.0138V$.

The actual accuracy is impaired by temperature shifts, which mostly affect the Zener diode. I have also suspected the voltage of the internal reference of the ATMEGA328P to be a function of the input voltage initially, but later I have found out that the effect of this is negligible. In image 4.1 we can see that for our diode, the temperature coefficient is expected to be relatively large (about 0.08 % per degree C). 0.08% of the 18V Zener voltage is equal to 0.0144V. This means that running the device at temperatures 10C above

the temperature it was calibrated on will result in the measurement being “too high” by $10 \cdot 0.0144 = 0.144V$. During my measurements, the maximum errors from “correct value,” as determined by my Brymen 867s multimeter, were $-0.04V$ and $+0.05V$. Using our assumptions, we can easily conclude that this corresponds to a temperature difference of 6.26 C . This is entirely plausible and serves as a good sanity check.

Since the device was calibrated at 25 degrees, for temperatures ranging from $25 - 10 = 15$ $25 + 10 = 35$ degrees, the maximum error will be $\pm 0.144V$. This translates to about 0.8% maximum error from the measured value. Unfortunately, since the lower bound of the range is not zero, we should also consider the magnitude of this error with respect to the actual range. As mentioned, the difference between the lower and upper bounds of the measurement is $11.6V$; From this, we can calculate the maximum error with respect to the range to be 1.24% . When it comes to the estimation of the state of charge, the calculated error seems relatively large. This is caused by the fact that there is only a small difference between the open-circuit voltage of a fully charged and fully discharged battery is only about $1.6V$. The available measurement range was chosen to accumulate all valid voltages that a lead-acid battery can have, but the typical voltages span an even smaller range. The worst-case error was estimated to be 9% under the circumstances listed above. While this may seem inadequate, the SoC estimation from the open-circuit voltage is rather inaccurate, as the function binding the two is affected by the battery manufacturer and type, the charge/discharge history of the battery, the parasitic load at the time of the measurement, temperature and more. Furthermore, the relationship between the state of charge and the actual remaining capacity is also complicated, one of the reasons being the fact that the remaining capacity heavily depends on the discharge rate.

However, I believe that the way of estimating SoC, which I have used, is a good compromise between simplicity, accuracy, and the degree to which the measurement alters the battery state.

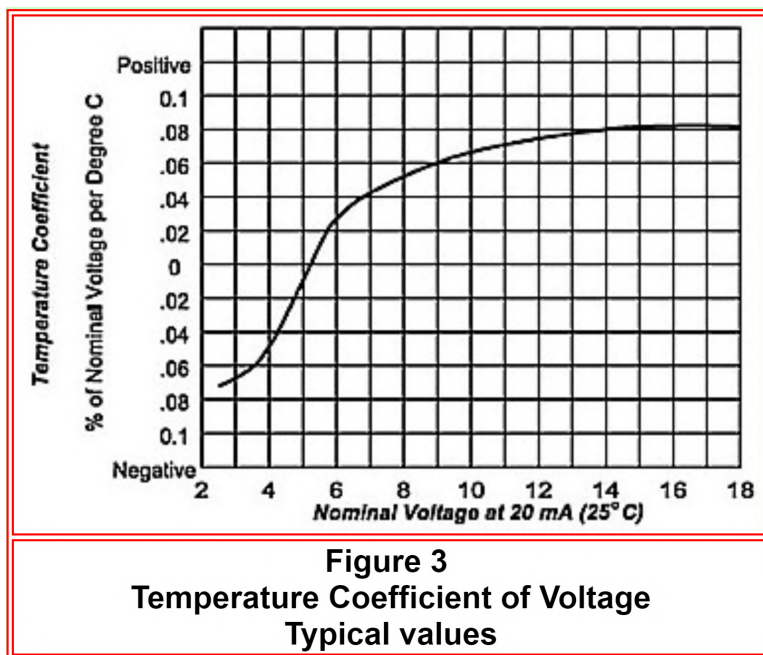


Figure 4.1: Zener diode temperature coefficient

Source: <http://users.tpg.com.au/ldbutler/ZenerDiodes.htm>

Chapter 5

Third party libraries used

Apart from libraries delivered as a part of the installation of the Arduino IDE, I have used some third party libraries for this project. For the communication with the SIM800L module, I have used the Arduino SIM8900L library by Christian Steib, available from <https://github.com/cristiansteib/Sim800l> . I have done modifications to this library, including, but not limited to:

- Implementation of support for sleep modes
- Reimplementation of time synchronization over the GSM network
- Implementation for functions related to accessing the on-sim storage memory

For the communication with the LCD screen, I have used the library by Juerd Wallboer, as available from <https://github.com/Juerd/Nokia5110> .

The Python application relies on the Plotly chart-plotting library available from <https://plot.ly/python/>. For communication with the serial port, the Py-Serial library is used. I downloaded it from <https://pyserial.readthedocs.io/en/latest/shortintro.html>. For running background tasks periodically, the Timeloop library is used. I downloaded this library from <https://pypi.org/project/timeloop/> . The background tasks are used to keep the communication channel between the PC and the system active.



Chapter 6

Further developement

Should the need to create additional copies of this device arise, I have designed a printed circuit board, which should make this process simple. Additionally, I have also decided to design a 3D-printable enclosure, which further simplifies the process, as it is no longer necessary to modify an existing plastic box. All the necessary source files are present on the CD, which accompanies this thesis.

I have also slightly modified the circuit, using Arduino Pro Mini, which has a slightly lower power consumption. The PCB layout can be seen on images 6.1 and 6.2. The enclosure design is depicted on image 6.3

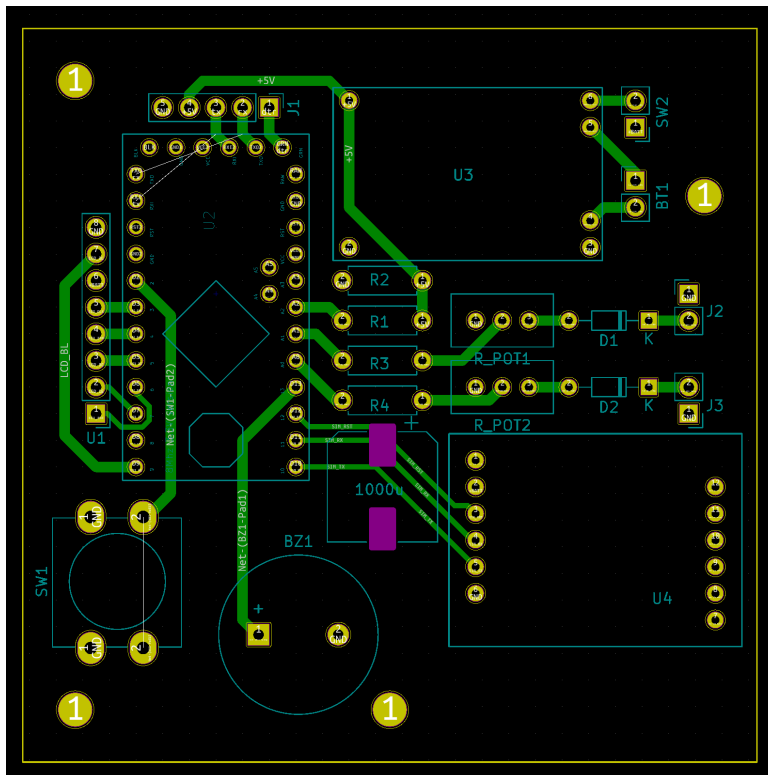


Figure 6.1: Back side of the PCB

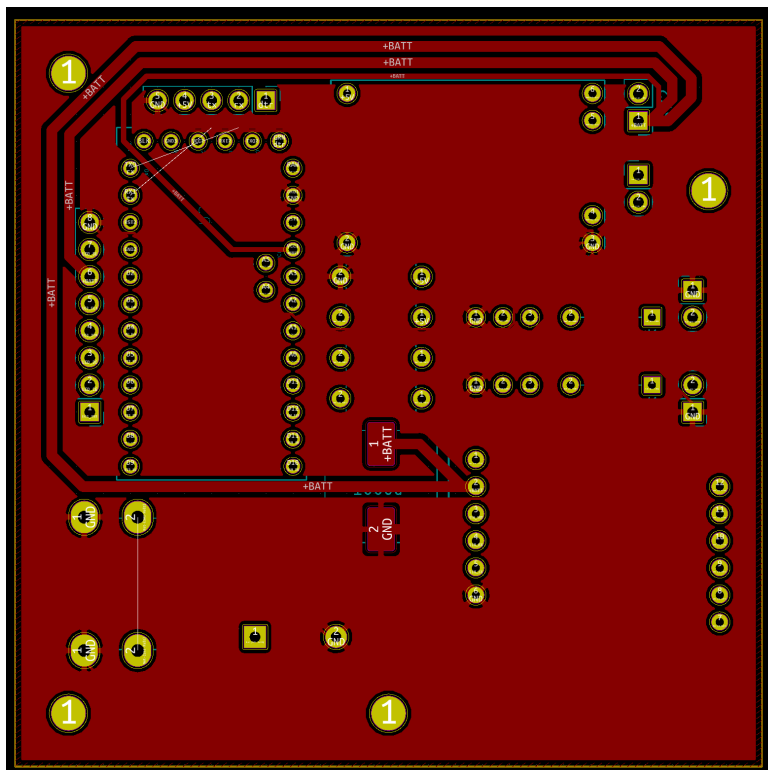


Figure 6.2: Front side of the PCB

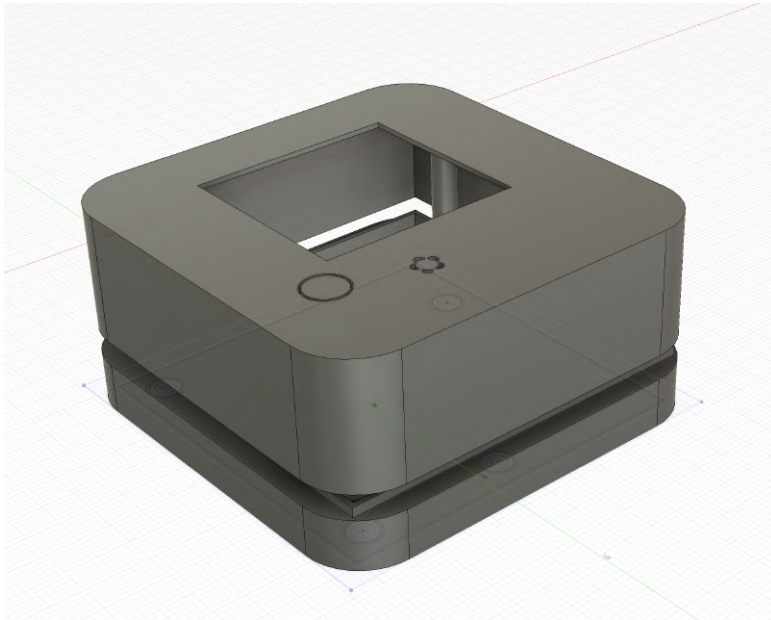


Figure 6.3: Enclosure design



Part IV

Previous attempts

The solution described in the previous part is only one of my attempts to solve the problem. It is the sixth device, including prototypes and reworks. In this part, I would like to describe my previous attempts. This listing may not be exhaustive, as I do vaguely recall having made some other prototypes, however, I am unable to collect enough meaningful information to describe them.

When I originally started working on this assignment, I have read the work by L. Dastych, which was recommended to me by doc. Roztočil and I have assumed that the best course of action will be for me to build a programmable load, controlled by a microcontroller. A programmable load is a device capable of sourcing power from a supply, typically used for testing of batteries, power supplies, and other laboratory work. Another motivation behind creating such a device for me was that I have access to copious amounts of heat sinks virtually for free. Following this line of reasoning, I have built several devices, centered around this premise. I have, however, underestimated the importance of discussion with Doc. Ing. Roztočil quite significantly and have later figured out, that his actual needs aren't necessarily met with such a device, despite the fact that formally, such a work might be up to the written requirements.

For this reason, I have decided to revisit my work, redo everything, and build a simple device, which would meet the exact requirements, as I learned them.

What follows is a brief description of my previous related works. I went from a single MOSFET-based solutions over solutions that included multitudes of MOSFETs, each complete with its driver, through concepts controlled over web-based user interface I built my own library for.

Chapter 7

Adjustable load built around a single MOSFET

In my first attempt, I decided to create a constant-current load built around a single transistor. This load can be seen in image 7.1.

The device is capable of acting as a constant current, constant voltage, constant resistance, or a constant power sink. On the hardware level, an operational amplifier is used in such a feedback configuration to make the device act as a constant-current sink, which uses voltage as a reference. This voltage is then adjusted by a (slower) software-based feedback loop to allow for other modes.

In order to enter one of the modes, the user first dials the required value, using the star on the numeric keyboard as a comma. Each of the buttons A, B, C, D is associated with one of the modes and pressing it enters this mode, using the value provided. Pressing the hash button turns the load on or off.

The OLED display shows both the setpoint and the actual value.

The processor board used is an ESP32 Dev module; I have originally planned to use a Blynk cloud service for an additional WiFi-based user interface, but I have discontinued the project before having done that. The built-in ADC and DAC of the ESP32 are also used. I am using a 50N50 n-channel MOSFET transistor mounted on a PC heatsink to dissipate the heat. This transistor is driven from an LT1006 high-precision op-amp. I have chosen

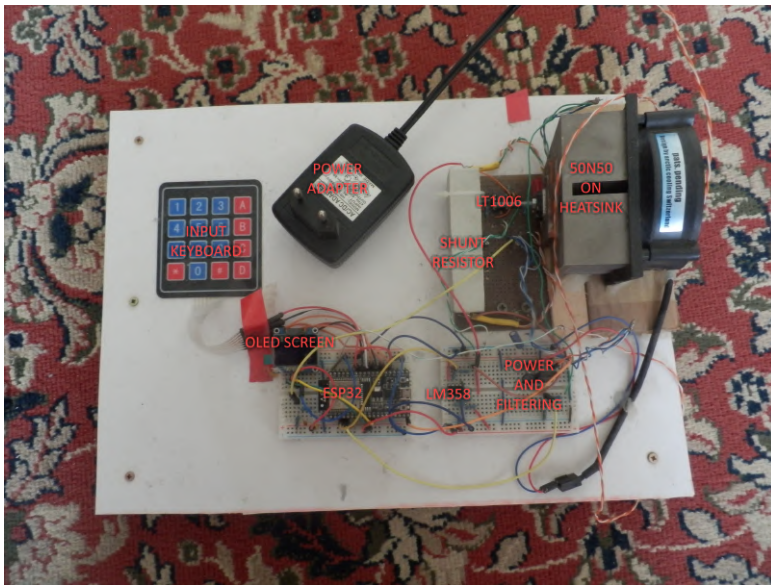


Figure 7.1: Adjustable load utilizing a single transistor

the transistor for its large power dissipation capabilities [Fai] and the opamp for its low offset voltages and currents and relatively high current driving capabilities. [Lin] I have used a 40W ceramic wire-wound 0.1 ohm resistor as a shunt resistor. Its parasitic inductance is probably less than ideal, but the large heat dissipation capabilities lower the maximum temperature drifts (the resistor is cold to touch even under heavy load) and therefore improve the temperature stability of the measurements. The maximum current was limited to 3A. The heatsink was mildly warm to touch under this current.

The voltage across the shunt resistor is then amplified by a half of the LM358 dual operational amplifier; the second half of this amplifier is then utilized to add a small offset to this voltage. The reason behind adding this offset is that the internal ADC of the ESP32 used is unable to measure voltages lower than some 100mV. [ESPa]

This device worked, but the performance was hindered by inadequate grounding and the fact it was mostly built on breadboards. As a result, the noise got coupled in the measurements. I wanted to rebuild the project on PCBs, but in the process of redesigning, I went over several prototypes (information about which has since been lost), and this development eventually led to the following prototype.

Chapter 8

Adjustable load based on a group of individually controlled mosfets

For this prototype, I have decided to go for multiple MOSFETs, each controlled by its own operational amplifier. The reason why I wanted to control the MOSFETs separately is that the gate voltage thresholds are quite different from part to part; when a common voltage would be applied to a group of different transistors, each would open by a different amount, which would result in unbalanced power and heat distribution between the transistors [PH18] The system consisted of multiple boards, each made according to the schematic seen on 8.1. The resulting assembly on the heatsink can be seen in image 8.2. They were controlled by the AD5662 16 bit DAC.

The boards were all wired in parallel, meaning all their respective nets were joined together (Uref to Uref, ULoad to ULoad, GND to GND, VCC to VCC).

The IRFP250M is used as the main power device; a constant current flowing through it is maintained by the use of negative feedback. A small 10mOhm SMD, thick-foil resistor is used as a current shunt. The voltage measured across it is then measured by a current-shunt monitor INA180A (U2). This device is a special kind of operational amplifier with the amplification already set to a fixed value by the manufacturer, intended specifically for measuring small voltages across shunt resistors. The measured voltage is then sent to MCP6062 (U1), which in turn drives the MOSFET. Resistors R2 and R4 were experimentally found to improve the stability of the setup. Capacitor C1 in the feedback also makes the configuration much more stable. Its value was found empirically. Each board is controlled from a reference voltage

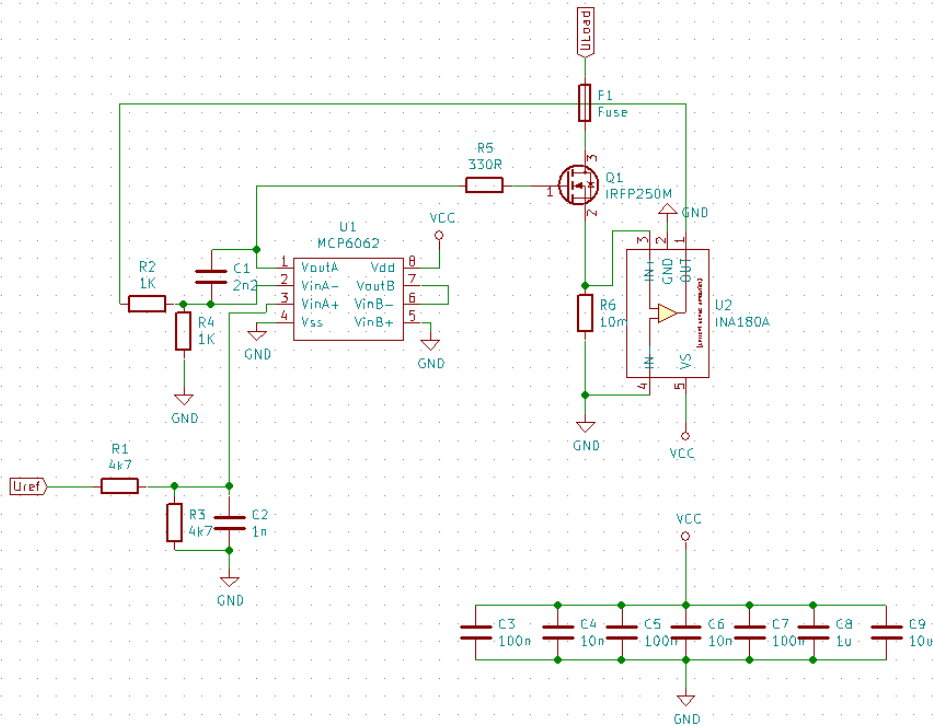


Figure 8.1: Load module schematic

Uref, which is then adjusted by a voltage divider (R1, R3) and filtered by the aforementioned resistors in conjunction with C2. C3 to C9 are various decoupling and filtering capacitors; F1 is a glass 7A fuse, which would burn in case the MOSFET fails short, preventing further damage. There was a flaw in this design. The bottom part of the shunt resistor was connected directly to the ground. This ruled out the possibility to use a global low-side current sensing common for all the modules, which would, in turn, enable the chance to use secondary "outer" control loops, as described in part "Single-transistor-based solution". Using a high-side current shunt monitor would still be possible.

I was originally planning to respin this board, fixing the ground issue mentioned above, and for the time being, I decided to temporarily use some large 100W resistors I had lying around. However, I found the results when using them quite satisfactory and for this reason have discontinued the idea with multiple transistors, transitioning smoothly to the next prototype.

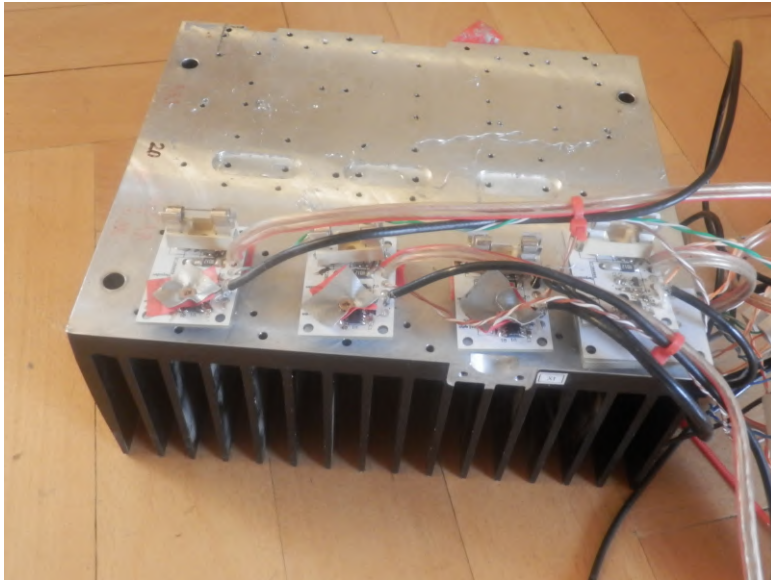


Figure 8.2: Multiple modules of opamp-driven mosfets on a large heatsink

Chapter 9

Resistor-based load

This prototype was by far the most complex and complicated one. I originally intended it to be the final product. For a multitude of reasons, I have decided against it and instead developed a much simpler solution. One of the main factors was that I learned that I need to extend my studies as I forgot to take a compulsory subject; Since I felt like this solution is overcomplicated, doesn't fill the actual requirements well and new problems surrounding it kept appearing, I remade it once again.

9.1 Generic description

The system I built was designed to provide backup power to the frequency standard in case of a power outage and also to test the batteries used for the backup. Those two functionalities are virtually independent and even in the case of failure of the majority of the components, the backup function is preserved. The system can be configured for the testing subsystem to either be functional even in case of a power outage or only when mains power is present to save battery.

It can run three different types of tests: an open-circuit voltage test, a so-called Fast test and a Discharge test. The open-circuit voltage test determines the state of charge of the batteries connected. It is almost immediate. The Fast test additionally determines the internal resistance of the battery, which has a relationship to the health condition of the battery. It takes about a

minute to complete. The Discharge test can be used to measure the capacity of the battery. It can take tens of minutes up to several hours to complete, depending on the settings. The tests can be run on-demand by the user at any time or can be scheduled to run automatically at given times. Each test has a set of user-adjustable failure criteria; that is, after a test is completed, the system can assess whether or not the test was successful. In case the test fails, further tests are prohibited. This ensures the battery that failed the test doesn't undergo further stress from the additional testing, and the other battery doesn't get discharged as it is needed for the backup to be operational. To re-allow tests, user intervention is necessary. A bright LED is used to signal the "test-failed, user action requested" state. When a test fails, the system also attempts to inform the user about this fact by means of sending them an email. Optionally, an email can be sent even when the test is passed.

The historical test results are stored in persistent memory and can be recalled on user request. The user has two ways of interacting with the system. One is by connecting to it via an optically-insulated serial port, and the other is by using remote access in the form of telnet. The user experience is virtually identical in both cases. The interaction has means of issuing a textual command (together with some parameters) to the system and obtaining a response. A VPN (or port forwarding, which, however, might have security ramifications) can be used to control the system from anywhere in the world. For Internet connectivity, WiFi is used. The WiFi can optionally be disabled.

12 high power resistors are used for the tests that necessitate the battery to be loaded; They can all be connected to the battery at the same time, or only half can be connected for 2 different load options.

The system contains a main processor which handles the user interface, test planning and other high-level matters and a coprocessor, which controls the low-level and safety-critical functionality, such as undervoltage protection and also serves as a watchdog for the main CPU. A high-resolution external ADC is used to measure the currents and voltages during the tests. A real-time-clock circuit, backed up by a 3V coin cell, is used to keep the accurate time even in the case of device reset. When connected to a WiFi, the Internet connection is also used periodically to adjust the time based on information received from an NTP server.

The block schematic can be seen in image 9.1.

The physical arrangement of the individual parts of the block schematic

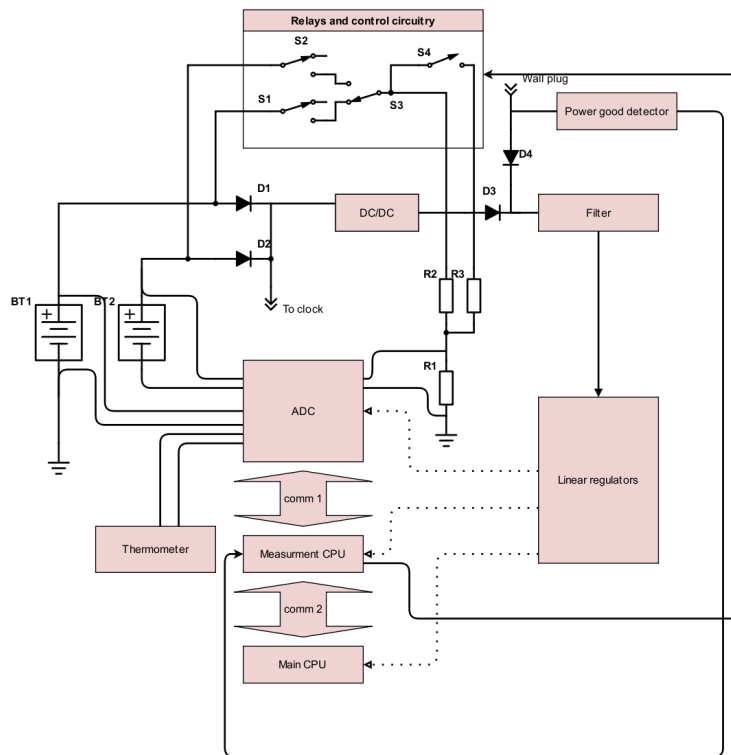


Figure 9.1: Block schematic of the resistor-based load

can be seen in image 9.2.

Despite the fact I have spent a lot of time on these solutions, new problems always kept appearing, and I did not really like it for other reasons described below; because of this, I have decided to rebuild the whole system again, finalizing with the system described in the previous part.

9.2 Problems of the complicated solution

In this chapter, I would like to describe some of the design flaws of the complicated solution.

I have significantly reduced the complexity of the system multiple times. The first time, I have discarded the constant-current load and have replaced it with a group of resistors. The second time, I have removed the web-browser-accessible user interface and have replaced it with a simple telnet/serial based

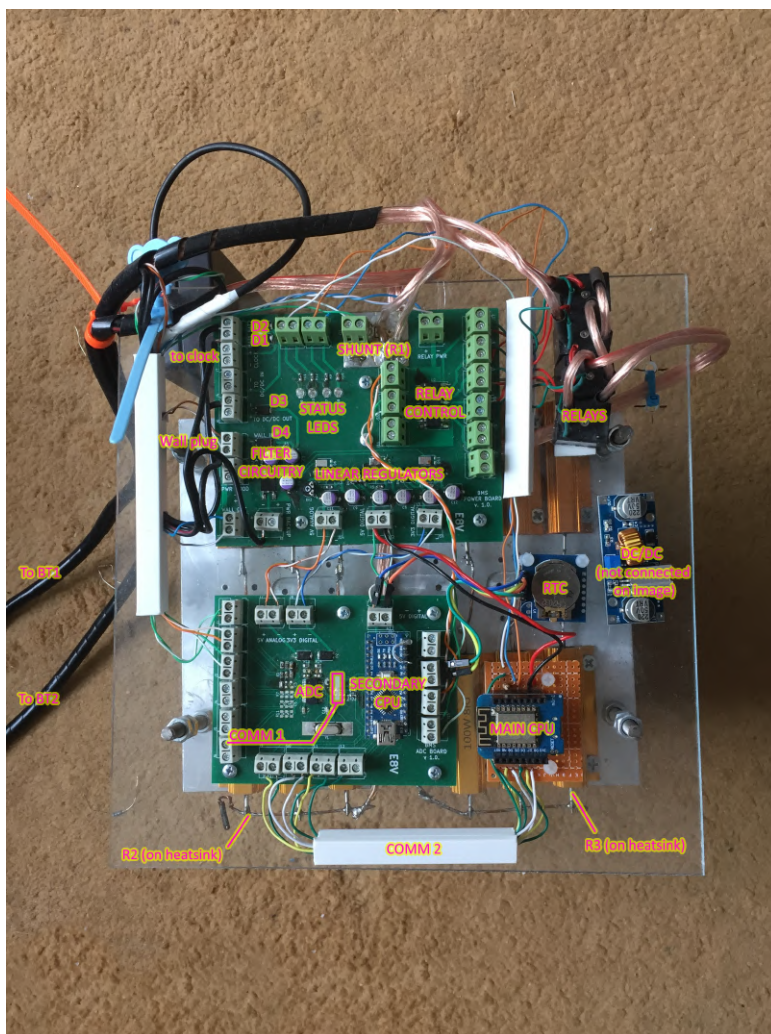


Figure 9.2: Layout of the hardware components of the resistor-based load

one. Despite this fact, it is still overly complicated for what it needs to achieve. The system uses two processors, one ADC, 4 relays, one RTC, and has 3 dedicated power supplies. This is absurd. This complexity serves no real purpose and only increases the possibility of failure. The software might also be buggy, and despite my best efforts to keep the user interface as simple as possible (and to include a useful help subsystem), it is difficult to interact with.

Another problem is related to the overall “bulkiess” of the device. The footprint of my system is roughly 30x30xcm, and the height is about 20 cm. The weight is in the order of several kilograms. Due to this size, it cannot easily be contained inside, say, a plastic box. Therefore, it is very prone to dust collection; furthermore, the dust cannot easily be removed because of the cables going everywhere.

I also believe that the “directly accessible” user interface was far too simplistic. The current status cannot easily be determined just by looking at the device. The only information we have is whether the device is operational and whether it has passed the last test - by looking at the status LEDs. Any further information can only be obtained by connecting to the device over WiFi or over USB and typing commands in. This means a computer is required even for resetting the device back to its operational state once a test has failed. This is stupid. It would be much, much better to include an LCD screen, showing the current test results and a “reenable tests” physical button.

One fact that I find rather unfortunate is that the device is dependant on infrastructure, which might be affected by power outages. The main purpose of the device is to monitor whether the batteries are in a working state. This being said, it depends on the presence of a WiFi access point. In order for this device to be capable of reporting outages in real-time, the WiFi access point would need to be operational in case of a power outage. This might imply that it would need to have its own uninterrupted power supply - yet another complication!

The setup of the system is overly complicated. The system needs to know an access point SSID, WiFi password, SMTP settings including the server address and port, email user name, email password, and target address. If any of those changes, the system functionality is impaired, and the values need to be adjusted. Additionally, in order for the system to be accessible from “the outside world,” a VPN or port-forwarding needs to be setup. The AP needs to be set up in such a way, so it always assigns the same IP to the device. In case the AP needs to be replaced, the new AP needs to be set up again.

I also believe that it would be rather difficult to produce additional copies of this design. The creation of additional similar devices is not straightforward, should the necessity to do so arise. It relies on relatively special components and takes hours to solder together. The parts are sourced from many different sources; The heatsink used was originally used on a TV broadcast tower, and there may be less than a dozen of its kind in the world. It also uses a custom-cut and drilled plexiglass board. The system is also relatively expensive (thousands of CZK).

The capacity of the batteries under test might actually be negatively affected by this system, which is yet another downside. Some of the tests used lower the immediately usable capacity of the battery, reducing the backup capabilities of the system. Furthermore, in case of a power outage, the device starts being powered from the batteries, draining them even more. This

is probably not very good. Despite the fact that I haven't really realized this when I was building the device, in hindsight, I think Dastychs solution should ideally be used to check the batteries periodically, while the online tests should ideally not drain the batteries very much.

Other concerns are related to the heat produced by the device. In scenarios where a test has been running for an extended period of time, the heatsink tends to get almost (but not quite) too hot to touch. Despite the fact that the dissipated power per resistor never exceeds 50W and the resistors are rated at 100W and utilize a large heat-sink, the lack of datasheet for the resistors prevents me from running reliability and safety-related calculations. Comparable resistors from Vishay are capable of dissipating the 50W even without a heatsink. [Vis] and I have also tested the system throughout without any problems. This being said, I am still not a big fan of running a system that can get up to temperatures exceeding 70C unattended.

The biggest problem I see, however, is that the device was created back when I did not have a good understanding of the actual requirements. I was introduced to the work by Lukáš Dastych by doc. Roztočil and I have mistakenly assumed that I am supposed to rebuild the system he has created, using custom hardware and software. However, this was not really the case, as I have figured out after additional discussions with Doc. Roztočil . In fact, the main goal is to ensure that the frequency standard always runs, even in the cause of a power outage and that the state of the backup power sources can be monitored. The system I previously built solves this problem only partially and not in an elegant manner. Some of the reasoning behind my other decisions also seems absurd in hindsight - for example, I initially decided to build a constant current load simply because I have free access to about 800kg (sic.) of aluminium heatsinks, and I wanted to make some use of them.

Chapter 10

Resistor-based load - prototype stage

Before having designed the PCBs for the resistor-based solution, I have experimented with it on breadboards. The hardware is similar, but the performance was much worse due to inadequate grounding due to the use of breadboards. The maximum current sourced was also lower (due to a different configuration of the resistors used), as there was no heatsink in place. This prototype can be seen in image 10.1.



Figure 10.1: Prototype for the resistor-based load

Chapter 11

Resistor-based load - old firmware

I have originally intended for the device to be controlled from a web-based user interface. Because none of the libraries already existing for this task were suitable, I have decided to create my own.

11.0.1 GUI library

There are a few libraries intended to enable a simple development of a web-based graphical user interface. These allow the user to connect to the processor (ESP32 or ESP8266) (which either acts as an access point or as a device on a local network, in which case it has an IP assigned to it) and then use a standard web browser to gain control.

EasyUI was developed by a user who goes by the name of ayushsharma82. [Eas] However, this library only contains a toggle button and label elements and is labeled as deprecated. ESP-dash by the same author is an extension and continuation of EasyUI. It even features limited chart support. [ESPb] However, it is still a bit restricted for my needs, and its design shows primary orientation on simple sensors, as it has dedicated elements for displaying humidity and temperature.

ESPUI developed by s00500 is a rewrite of EasyUI and is the most full-featured ESP GUI library I found. [ESPC] It contains switches, pushbuttons, labels, sliders, and more. It allegedly has upcoming chart support, but to

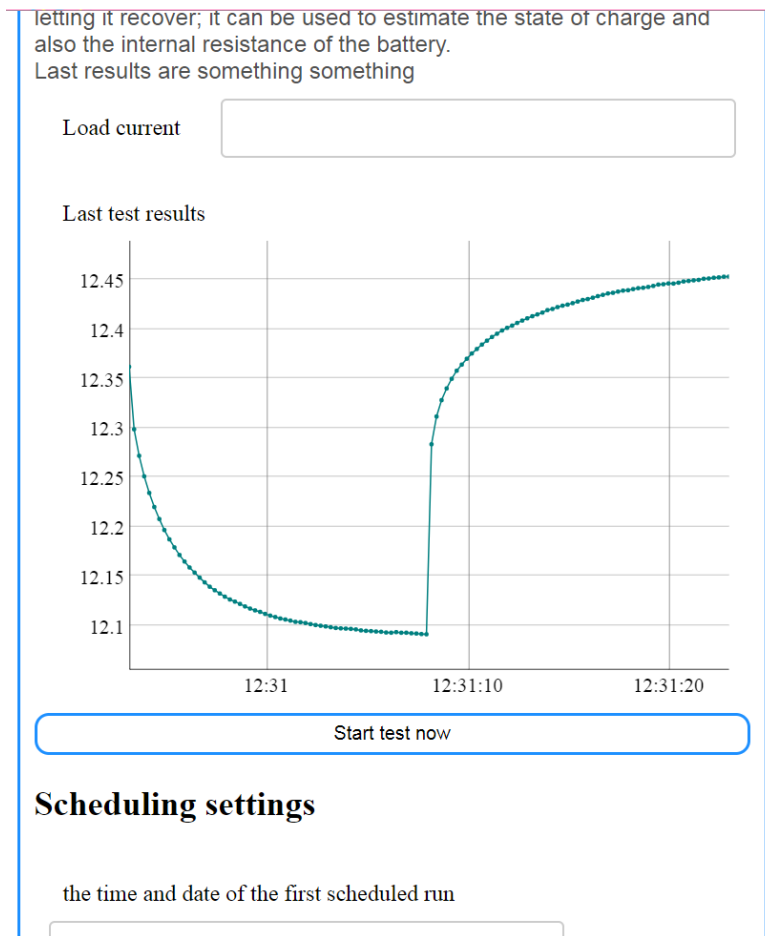


Figure 11.1: GUI built using my library

this day (28.4.2019), it has not been implemented. [ESPc]

For these reasons, I have decided to write my own GUI library. I wrote a library known as TGH-GUI, where TGH is a short version of my nickname “thegoodhen”. This library has many different user interface elements, ranging from informative texts through checkboxes, list boxes, and sliders to zoomable line charts with multiple lines within them. The elements can be organized into horizontal or vertical groups and also into tabs. The screenshot of the user interface generated by it can be seen in image 11.1.

■ 11.0.2 Operation principles

The data between the server (ESP) and the client (the device used to access the GUI, such as PC, smartphone, or tablet) is exchanged using the HTTP

protocol and the WebSocket protocol. Websocket protocol is a computer communications protocol providing full-duplex communication over TCP. It provides a way for the server to send content to the client without it being requested by the client (polling). [WS] Javascript in modern browsers allows the opening of a WebSocket channel and using it for communication.

The communication between the server and client works as follows:

- Client connects to the server; an HTTP connection is opened
- HTTP headers are sent as a response; these headers establish that the content of the webpage will be sent in chunks and that the content length is unknown. This allows the ESP to take the time necessary to send the webpage without danger of a timeout.
- The HTML page starts being built and sent piece by piece; first, the relative paths to javascript files and stylesheet (.css) files are sent
- Next, the body of the HTML page is built; this step consists of iterating over the individual GUI elements and sending their html code snippets. The snippets include information about callbacks; effectively, in this step, the ESP “tells” the browser, which events (i.e., button click, slider move) it wants to be notified about. The information about the visual organization of the page is also sent in this step.
- Once the information about all the GUI elements was sent to the browser, the webpage is properly terminated to form a valid HTML; this consists of sending the closing `</body>` and `</html>` tags.
- The HTTP connection is then closed; the browser now automatically sends a request to load the external content (such as the CSS stylesheet files and the javascript).
- The ESP responds to all the requests, sending the javascript and CSS files located in its SPIFFS memory
- The website has now been loaded; this launches a client-side javascript code, downloaded from the ESP in the previous step. The code opens a WebSockets connection between the two devices.
- The newly opened WebSockets connection is used by the client-side javascript to send information about its readiness.
- Once the ESP receives this information, it sends requests back to the client to set the default properties of the individual GUI elements - such as whether the checkboxes are checked, contents of the text fields, and more. For charts that have their data stored in the SPIFFS of the ESP,

the client then proceeds to send an HTTP request to download the historical data of the chart; the reason for the request being carried over HTTP is that the WebSocket library I use has a relatively large overhead and this way is much faster.

- Everything is ready now. From now on, events, such as the user pressing a button will trigger javascript code, which will send a JSON-encoded message over the established WebSockets tunnel; this message will contain the id of the element that the event belongs to, the type of the event and optionally some other data. All this information can then be used by the ESP to perform a lookup of a callback C++ function, which will then be executed. Similarly, the connection will, from now on, be used to update the different properties of the GUI elements, also by the use of JSON-encoded packets.

■ 11.0.3 Object oriented design basic concept

All GUI elements, such as sliders, buttons, and texts that are generated in the browser as javascript objects once the webpage is loaded have their counterparts in the C++ program running on the ESP.

Those C++ counterparts all subclass the class `GUIElement`. This class contains some common functionality for all the elements of the GUI. For instance, it provides utility functions for setting the text of the element or for retrieving an integer representation of the value of the element. It is also subclassed by the `Container` class, which is an abstraction for different containers - objects that can contain other GUI elements and affect their layout. Each GUI element has a string id, which can be used to refer to it. This id corresponds to the “id” property of the corresponding JavaScript element.

`GUI` class is, in a sense, the main class of my library. It handles all the HTTP and WebSocket communications, is used by the individual elements, and it can also be used to search between them by their ID.

■ 11.0.4 Multiple users and synchronization

Up to 5 clients can be connected to the server at a given time. When an event is fired, the ESP keeps track of the user who caused this event. This

way it is possible to keep a limited form of “session” for individual users. For instance, when some user clicks a button, some text on the UI can change, but only for them. The mechanism for distinguishing between the users is rather limited and is subject to future improvements that I didn’t feel were necessary for the purposes of this thesis. The main drawback is that the identifying numbers are assigned to the user once they connect, and when they disconnect, the number is freed and may later be assigned to a different user. It should therefore not be relied on outside the realms of callbacks, where we may want to only provide some information to the user who has just requested it.

It is possible to update the properties of the elements on a per-user basis. Alternatively, it is also possible to update them for all the currently connected users at once. This ties to the “synchronization” feature of the library. An element can be synchronized. When it is so, each time some property of it changes (that is, when an event is fired), the newly obtained value is automatically sent to all the other users connected. This effectively means that it is possible to have a checkbox that, when clicked by one of the clients, automatically gets checked on all the connected devices in real-time. This functionality has been implemented for checkboxes, sliders, and list boxes.

■ 11.0.5 Working with the library

In the Arduino programs, two functions are always present. The `setup()` function is called once the Arduino turns on, and then `loop()` is called periodically afterwards. A typical way to use this library is to set up all the necessary GUI elements in the `setup()`. This setup includes organizing the elements on the page, specifying their callbacks, and setting their properties. Then, the `GUI.loop()` function needs to be called inside `loop()`. The callbacks specified in the `setup()` are then automatically executed when applicable, the website is hosted, and everything works without additional setup. When necessary, specific functions can be called to change some of the properties of the GUI elements on the fly.

An example of a simple program using the library can be seen below. This program contains one textual label and a button. When any of the connected clients press the button, all clients will see the text of the label to change to a number that was previously assigned to the given client. This example demonstrates the initialization of the GUI elements, their organization into containers, creating callbacks, searching the GUI element by its respective ID, and changing its property.

Listing 11.1: Minimal code

```

#include "TGH_GUI.h" //We include the library
GUI gui;
//WiFi setup-related code omitted here for brevity

void setup() {
    //WiFi setup-related code omitted here for brevity
    //begin the communication over the serial port at 9600 bauds
    Serial.begin(9600);
    //properly initialize the GUI
    gui.begin();
    //add a new button called "btn" with text "Submit",
    //which runs the buttonCB C++ function when clicked
    Button* b = new Button("btn", "Submit", buttonCB);
    Label* l = new Label("lbl1", "Number of the last user who clicked the button");
    //create a container which organises all the GUIElements
    //inside it underneath each other
    VBox* vb=new VBox("vb");
    //add the newly created container to the GUI
    gui.add(vb);
    //add the label to the container;
    //since the container is vertical box,
    //all elements inside it will lay underneath each other
    gui.add(l);
    //Add the button to the container
    gui.add(b);
}

void loop() {
    //you have to call this function in loop() for this library to work!
    gui.loop();
}

void buttonCB(int user)
{
    Serial.println("User clicked the button! User number: ")
    //find the object with id "lbl1" and
    //set its text to the number of the user who clicked the button
    gui.find("lbl1")->setText(ALL_CLIENTS, (String)user);
}

```



Chapter 12

Other attempts

The device described under "Single-transistor-based-solution" was at least third of its kind; the previous ones were built around LM358 as the main operational amplifier, were built exclusively on a breadboard, and I have since forgotten much about them. The final version of the resistors-based device underwent several iterations; for example, the board the components are mounted on was previously wooden.

I wasn't very happy with any of the attempts previously described. They were unreasonably complicated and also had a few other problems. For this reason, I have ended up developing a device which is actually useful and meets the requirements, despite its simplicity. It can only measure the open-circuit voltages of the two batteries, report it over the GSM network and also report power outages.

Part V

Recommendations regarding the battery maintenance

The correct procedure regarding the maintenance of the batteries has been extensively covered by L. Dastych in his diploma thesis [Das] Having read it, I have to say that I agree with most of the points made by him.

Keeping the temperature around 20°C is good for the batteries, but such regulation is difficult to achieve. Finding the correct charging voltage is always a compromise. Typically, the batteries are charged to lower voltages for this sort of application. These voltages range from 2.3 to 2.35V per cell. This maximizes the service life. Higher voltages (Above 2.4V per cell) shorten the service life due to grid corrosion on the positive plate. [CTL] [Chab]

While Ing. L. Dastych suggests that lead-acid batteries should not be kept on the charger once they've been fully charged (float-charging) [Das, p. 76], I am unable to find such information either in the documents he has quoted or elsewhere. Moreover, the sources I have found explicitly state that float charging is NOT detrimental to the health of lead-acid batteries.[CTL] [Tem]

It is crucial not to store the batteries discharged for prolonged periods, as this causes sulfation. Sulfation is a phenomenon, where large crystals deposit on the negative plates of a battery, decreasing the active surface, lowering the performance. [Sul] L. Dastych also states that the batteries should be cycled every 4-6 months by discharging them to a recommended DoD before recharging them again. Later, he states that such a cycle should, in fact, be carried out once a month. He states that this is the best way to treat the lead-acid batteries. I was unable to confirm this, and the sources he quotes do not, in fact, discuss said topic at all. I imagine that the reasoning would be that it might prevent sulfation.

The self-discharge rate of lead-acid batteries is about 5% per month. [Chaa] Since storing the batteries at low charges leads to sulfation and the low charge of the batteries also affects the time for which the backup power will work during a power outage, it is recommended to keep the battery voltage over 12.4V for 12V batteries and over 24.8V for 24V batteries. Deep discharging of the batteries should be prevented whenever possible. See image 12.1 for the relationship between the depth of discharge and the number of cycles. This being said, the proper backup of the frequency standard is much more important than ensuring the highest life expectancy of the batteries. Therefore, no measures should be used to prevent the battery from deep discharge at the cost of lowered backup time.

It might be beneficial to run extensive tests sporadically to ensure a good state of the batteries. For example, testing the internal resistance of the batteries once they are recharged is advisable. This has the added benefit of

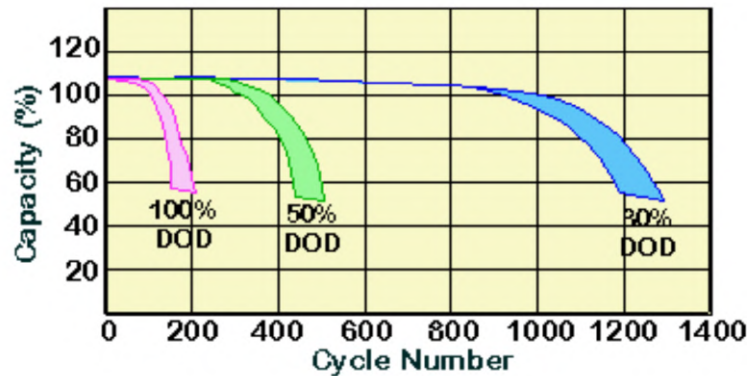


Figure 12.1: Number of cycles vs. depth of discharge (source: <https://pvcadrom.pveducation.org/BATTERY/charlead.htm>)

getting rid of surface charge, rendering subsequent SoC estimations based on OCV more accurate. [How] The system made by L. D. could be used for this purpose. An additional benefit to this would be that

In summary, the following procedure should be followed:

- The batteries should be charged to 2.3 to 2.35V per cell. This corresponds to 13.8V for a nominal 12V Battery.
- When the batteries get discharged below 25V, they should be recharged. This will be reported by the system over an SMS message.
- Once the batteries are charged, the system developed by L. Dastyeh can be used to measure their internal resistance and overall state of health.
- The battery temporarily removed from the system should be replaced by a different one to ensure continued operability of the system.



Part VI

Conclusion

The assignment required me to build a system capable of measuring the immediate state of lead-acid batteries, which would provide backup power to the frequency standard at all times.

My system displays the current state of charge of the two batteries on its small LCD screen, while also being capable of periodically reporting it over SMS messages. It also uses the SMS messages to automatically report critical states, such as low state of charge of the batteries, or a power outage. Optionally, the user can manually request the report of the current status by sending a message to the device. A loud siren is used to alert the user about such critical states when they are present. This system is currently already in use and I have periodically tested its functionality over the course of a few months.

I have written some recommendations regarding the proper maintenance of the batteries - those are available in part “Recommendations regarding the battery maintenance”.

I have built several other devices and prototypes before deciding for my final design. One was a MOSFET-based constant-current/constant-voltage/constant resistance load, interface of which consisted of a numeric keyboard and an OLED display. Another one was a constant-resistance load, capable of running various tests automatically. In the first iteration, it hosted a webserver and the user could use a WiFi-enabled device to control it. In the second iteration, it communicated over a serial port and telnet. At one point, I have also been using a combination of MOSFET-based constant-current sinks, each consisting of an op-amp in a feedback loop with a MOSFET.

All such devices are described in part aptly named “Previous attempts”.

I have also decided that it might be beneficial to design a PCB layout, so that the device (which is currently built on a perfboard) can be easily replicated. For the same reason, I have designed a 3D-printable enclosure. The design files can be found on the CD.

I believe that I have met all the criteria and that the final design of the device is superior to my prototype designs, despite its simplicity. The only downside I see is that it only measures the open-circuit voltage. In some cases (such as in case of heavy sulfation), the battery can appear to have a relatively high open-circuit voltage but struggles to keep this voltage under load. This is a very specific example, however, and I have concluded that such a case, should it appear, will be identified during offline tests carried

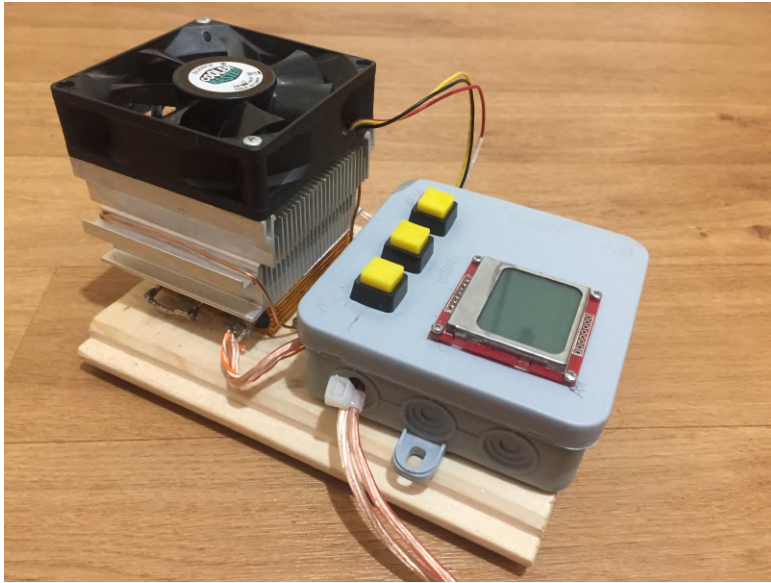


Figure 12.2: Battery tester

out as a part of regular maintenance.

In my free time, I am currently working on a device, capable of running such offline tests, which would mitigate the need to use complex lab equipment for improved user experience. The current progress can be seen in image 12.2.



Appendices



Appendix A

Unfinished previous diploma thesis

Before settling for my final design, I had been planning to hand out a completely different solution. As problems arised, I have concluded that it is overly complicated and doesn't serve the intended purpose well enough. This was, however, after I wrote most of the documentation for it.

This never-finished work in progress is present on the CD mostly for historical reasons.



Appendix B

Used abbreviations

AC	Alternating Current
ADC	Analog to Digital Converter
AP	Access Point
API	Application-Programmer Interface
ASCII	American Standard Code for Information Interchange
BJT	Bipolar Junction Transistor
CSS	Cascading Style Sheet
CZK	Czech crown
DAC	Digital to Analog Converter
DC	Direct Current
ESD	Electrostatic Discharge
GSM	Global System for Mobile communication
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IP (address)	Internet Protocol address
JSON	JavaScript Object Notation
LCD	Liquid Crystal Display
LED	Light Emitting Diode
MOSFET	Metal-oxide on Semiconductor Field-Effect Transistor
NTP	Network Time Protocol
OLED	Organic Light Emitting Diode
op-amp	Operational Amplifier
PC	Personal Computer
PCB	Printed Circuit Board
RTC	Real-Time Clock
SIM	Subscriber Identity Module
SMD	Surface-Mounted Device
SMS	Short Message System
SMTP	Simple Mail Transfer Protocol
SoC	State of Charge
SoH	State of Health
SPIFFS	Serial-Parallel Interface Flash File System
SSID	Service Set Identifier
TCP	Transmission Control Protocol
TV	Television
UART	Universal Asynchronous Receiver/Transmitter
UNIX	(not an acronym)
UPS	Uninterruptable Power Source
USB	Universal Serial Bus
VPN	Virtual Private Network
Wi-Fi	Wireless Fidelity

Appendix C

Bibliography

[ATM] ATMEL, *8-bit avr microcontroller with 32k bytes in-system programmable flash*, [http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf], Online; accessed 15-7-2019].

[Bat]

[BUZ] *1/5/10pcs sfm-27 high decibel alarm continuous sound electronic buzzer dc3-24v*, [https://www.ebay.com/itm/1-5-10Pcs-SFM-27-High-Decibel-Alarm-Continuous-Sound-Electronic-Buzzer-DC3-24V/142955956556?_trkparms=aid%3D555018%26algo%3DPL.SIM%26ao%3D1%26asc%3D20131003132420%26meid%3D9bb3b00ad3474bf392c916f044f44340%26pid%3D100005%26rk%3D6%26rkt%3D12%26sd%3D113359524668%26itm%3D142955956556%26pg%3D2047675&_trksid=p2047675.c100005.m1851], Online; accessed 15-7-2019].

[Chaa] *Charging sealed lead-acid batteries*, [https://www.silvertel.com/images/technical-articles/charging_sealed_lead_acid_batteries.pdf], Online; accessed 19-12-2019].

[Chab] *Charging the lead-acid battery*, [https://batteryuniversity.com/learn/article/charging_the_lead_acid_battery], Online; accessed 19-12-2019].

[Coo] Tyler Cooper, *3.3v conversion*, [https://learn.adafruit.com/arduino-tips-tricks-and-techniques/3-3v-conversion?utm_source=rb-community&utm_medium=forum&utm_campaign=

- powering-arduino-nano-from-a-3-7-v-lipo-battery, Online; accessed 15-7-2019].
- [CTL] *Can the lead acid battery compete in modern times?*, [https://batteryuniversity.com/learn/archive/can_the_lead_acid_battery_compete_in_modern_times), Online; accessed 19-12-2019].
- [Das] Lukas Dastych, *System pro testovani akumulatoru s vysokou kapacitou*.
- [dee19] *Deep cycle battery faq*, December 2019, [<https://techtutorialsx.com/2017/01/21/esp8266-watchdog-functions/>], Online; accessed 15-7-2019].
- [Eas] *Easyui*, [<https://github.com/ayushsharma82/EasyUI>], Online; accessed 3-1-2020].
- [ESPa] *Analog to digital converter*, [<https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/adc.html>], Online; accessed 18-8-2019].
- [ESPb] *Esp-dash*, [<https://github.com/ayushsharma82/ESP-DASH>], Online; accessed 3-1-2020].
- [ESPC] *Espui*, [<https://github.com/s00500/ESPUI>], Online; accessed 3-1-2020].
- [Fai] Fairchild, *50n50 500v n-channel mosfet*, [<https://pdf1.alldatasheet.com/datasheet-pdf/view/90595/FAIRCHILD/FDA50N50.html>], Online; accessed 15-7-2019].
- [GSM] *Jak jednoduse na gsm - iii.dil - jedeme pres arduino*, [<https://www.arduino-tech.cz/inpage/jak-jednoduse-na-gsm-iii-dil-jedeme-pres-arduino/>], Online; accessed 15-7-2019].
- [How] *Bu-903: How to measure state-of-charge*, [https://batteryuniversity.com/learn/article/how_to_measure_state_of_charge], Online; accessed 18-8-2019].
- [LCD11] *Low-power lcd smackdown*, 2011, [<https://www.bigmessowires.com/2011/06/07/low-power-lcd-smackdown/>], Online; accessed 15-7-2019].
- [Lin] Linear technology, *Lt1006 precision, single supply op amp*, [<https://www.analog.com/media/en/technical-documentation/data-sheets/LT1006.pdf>], Online; accessed 15-7-2019].
- [Low] *Topic: Arduino uno, lower clock to 8mhz [internal]*, [<https://forum.arduino.cc/index.php?topic=349818.0>], Online; accessed 18-8-2019].

- [Mic] Microchip, *Primary frequency standard*.
- [PH18] Hill Winfield Paul Horowitz, *The art of electronics, 3rd edition*, Cambridge University press, 2018.
- [SIMa] *Sim800l gprs gsm module*, [<https://www.kuongshun-ks.com/uno/uno-board-shield/sim800-sim800l-gprs-gsm-module.html>], Online; accessed 15-7-2019].
- [Simb] SimCom, "*sim800 series_embedded at_sleep_application_note_v1.0*", [https://simcom.ee/documents/SIM800x/SIM800%20Series%20Embedded%20AT%20Sleep%20Application%20Note_V1.01.pdf"], Online; accessed 15-7-2019].
- [Simc] SimCom, *Sim800l_hardware_design_v1.00*, [https://img.filipeflop.com/files/download/Datasheet_SIM800L.pdf], Online; accessed 15-7-2019].
- [SL] Jeff Cherry et. al. Sodik Lee, Joseph F. McDonald, *Modeling and validation of 12v lead-acid battery for stop-start technology*, [<file:///N:/downloads/2017-01-1211.pdf>], Online; accessed 19-12-2019].
- [Sul] *Sulfation and how to prevent it*, [https://batteryuniversity.com/learn/article/sulfation_and_how_to_prevent_it], Online; accessed 19-12-2019].
- [Tem] *Temperature effects on sealed lead acid batteries and charging techniques to prolong cycle life*, [<https://prod-ng.sandia.gov/techlib-noauth/access-control.cgi/2004/043149.pdf>], Online; accessed 19-12-2019].
- [TP4] *Tp4056 1a lithium lipo battery charger with or not protection diy arduino*, [<https://www.ebay.com/itm/Tp4056-1a-Lithium-Lipo-Battery-Charger-with-or-not-Protection-Diy-Arduino/264280367074?hash=item3d88565fe2:m:mfkMtVnfzleeQ6mREnZIX8A>], Online; accessed 15-7-2019].
- [Vis] Vishay Huntington, *Wirewound resistors, commercial power, aluminum housed, chassis mount*, [<https://www.vishay.com/docs/31883/ah.pdf>], Online; accessed 15-7-2019].
- [WS] *The websocket protocol*, [<https://tools.ietf.org/html/rfc6455>], Online; accessed 15-7-2019].