Technische Universität München
Fakultät für Elektrotechnik und Informationstechnik

Lehrstuhl für Messsystem- und Sensortechnik
Prof. Dr.-Ing. habil. Dr. h.c. Alexander W. Koch

Interdisciplinary Project

Further Development of an Interface Simulator for Experiments on the REXUS Sounding Rocket

Nicolas Beneš

August 31, 2015

Supervisor:
Dipl.-Ing. Max Rößner

Acknowledgements

After the launch campaign for the successful RX15/16 missions in June 2014, the initial ideas for providing every REXUS team with a simple service module simulator had settled sufficiently and my work on this project began. Soon the number of desired features grew, and more and more work was the result. Several time extensions and more than a year later, a small batch of simulators has been manufactured and is ready for deployment marking the first release and the official end of this project phase.

Although a lot of the circuit and software were designed from scratch, user interface concepts and the basic architecture are based on designs developed by Jakov Kholodkov, who worked on a service module simulator for the FOVS team of RX15.

Furthermore, Max Rößner supervised the project for the Institute for Measurement Systems and Sensor Technology and was always at hand if problems in the circuit design arose or a nice housing had to be designed. Moreover, he remained patient even though the project took much longer than initially planned.

Development of the simulator as such mostly was fun, but to have an actual user of the final product was a motivational boost: Torsten Lutz of ZARM provided valuable input for their needs and application, and he enabled the manufacturing of a small batch of simulators with which the following REXUS teams can play with.

Naturally, a service module simulator needs to imitate an original counterpart so information about the internals is important. Thanks to Markus Pinzer of DLR MORABA, questions regarding the implementation of the REXUS Service Module could be answered and a higher degree of similarity could be achieved.

Finally, I like to thank the members of the participating teams of the RX15/16 missions for a great time and life experience, as well as the various organisations that make the REXUS programme possible.

Abstract

The REXUS programme allows student teams from across Europe to design, build, test, and fly their own experiments aboard a sounding rocket. A crucial part of every space mission is verification of the correct behaviour, with testing being a prominent method to achieve said goal. Every team has to ensure the compatibility of its experiment with the directly attached modules and has to verify correct reactions to control signals.

This work presents RXSMS, a free hardware design and free software interface simulator of the REXUS Service Module. It aims to facilitate handling and testing of experiments without requiring the actual service module.

Contents

1. Introduction

The Rocket Experiments for University Students (REXUS) programme[1] allows student teams from European universities and higher education colleges to design, build, and test scientific experiments for sounding rockets. Two of such rockets are launched from Esrange Space Center, Sweden, every year, each carrying four to five experiments to an altitude of approximately 80 km to 90 km. After reaching the apogee, the payload falls back towards earth and lands using a parachute. It then gets recovered by helicopters and is returned to the student teams.

The experiments have to be designed [5] with regard to environmental constraints, like heat dissipation, ambient temperature, vacuum, milligravity, as well as rocket vehicle constraints, like total mass, power consumption, dimensions, and high acceleration in all directions. To ensure a successful mission and the safety of personnel, multiple reviews are held throughout the different stages of the project and several tests are conducted. Initially, these are per component or experiment, then with higher levels of integration for instance the assembled scientific payload.

The REXUS Service Module (RXSM) is a crucial component and provides basic functionality to experiments, for example switching the power supply on and off, or asserting the lift-off signal; however, the RXSM only exists in the order of one or two of a kind and it is not possible to provide one to every team. As a result, experiments get connected to the real REXUS interface in a comparatively late phase like integration test or bench test. Issues becoming apparent at this stage are a risk to the overall mission as time to find and fix the causes is limited and time of the other waiting teams is occupied. To be able to test their experiments in advance, year after year teams develop their own simple interface boards and simulators as an additional side project; still, the risk of common cause mistakes, like wrong pin assignments on connectors, persists.

The herein proposed REXUS Service Module Simulator (RXSMS) is a further development of the simulator designed by Jakov Kholodkov [4]. It is meant as a tool to be delivered to all participating REXUS teams to support them in handling of the experiment in the lab, testing the communication channels, and testing its compatibility with

[1]`http://rexusbexus.net/`

the RXSM. The simulator, moreover, is engineered to allow for multiple PCBs to be cascaded and assembled in a compact housing; thus, it can also be used to simulate the multi-channel operation of the service module and is suitable for integration testing.

1.1. Scope and Purpose

This document covers the REXUS Service Module Simulator (RXSMS), an interface simulator of the REXUS Service Module (RXSM) to be used as tool during development and for testing of REXUS experiments. It contains two main parts:

- a user manual for the student teams and the supporting engineers responsible for the scientific payload, and
- an extended manual for developers and maintainers of the simulator hardware and software.

Firstly, the RXSM is briefly introduced and put in context of the REXUS vehicle. Secondly, the interfaces of the service module simulator and its basic application are explained in the user manual. Thirdly, the extended manual details the architecture and internal operation of the simulator and informs the reader about advanced features like cascaded operation. And to conclude, an outlook on remaining work and possible future enhancements is presented.

1.2. License

The source code of the microcontroller program, as well as the circuit schematics and PCB layout are licensed under the GNU General Public License Version 3 or any later version. Users of the simulator and the software are invited to contribute and improve it where needed. Bugs should be reported to the author preferably via encrypted email (listing 1.1).

```
OpenPGP Key:
  2FA1 7660 4768 CC45 893C  9CF7 E24E 92CA E789 8C3B
  Nicolas Benes <benes@in.tum.de>
```

Listing 1.1: Contact information of the author.

2. REXUS Service Module

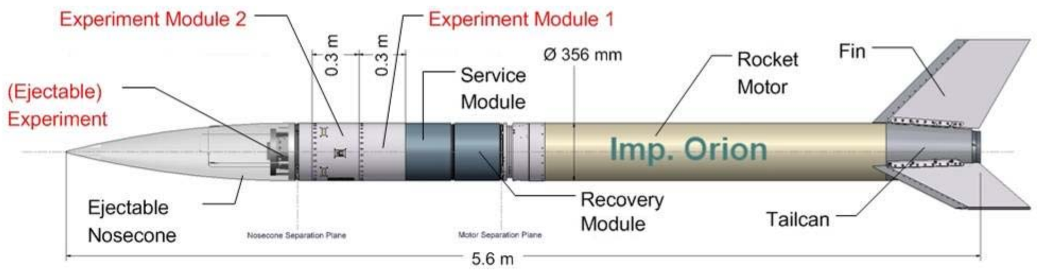


Figure 2.1.: REXUS Standard Configuration [5, p.22].

The REXUS vehicle [5] consists of a one-stage solid propellant motor and approximately 95 kg of payload (fig. 2.1). The latter is further divided into recovery module, service module, and some experiment modules representing the scientific payload. The REXUS Service Module (RXSM) is the central component connecting all the other modules of the payload and provides the following interfaces as seen from the experiment modules:

- Power Interface: an unregulated 24 V to 36 V battery power supply
- Charging Interface: a 28 V to 34 V/1 A supply for charging built-in batteries in a switched-off experiment
- Telecommand and Telemetry Interface (TM/TC): a RS-422 link to send data from the ground station via the RXSM to the experiment is provided before lift-off, and a RS-422 link to send data from the experiment via the RXSM to the ground station is provided before lift-off and during flight.
- control interface: three open-collector signals allow the RXSM to indicate statuses or trigger events
  - $\overline{\text{LO}}$: Lift-Off, shared by all experiments
  - $\overline{\text{SODS}}$: Start/Stop of Data Storage, separate for each experiment

- $\overline{\text{SOE}}$: Start/Stop of Experiment, separate for each experiment

Besides the interfaces to the scientific payload, the RXSM has other electrical connections, for instance for a GPS antenna and for RF links to the ground. For a complete overview of the RXSM, consultation of the REXUS manual [5] is advised.

3. REXUS Service Module Simulator – User Manual

The REXUS Service Module Simulator exists in differently populated PCB variants depending on the desired application and user group. This chapter covers the RX Variant (fig. 3.1) providing a stand-alone simulator for REXUS teams.

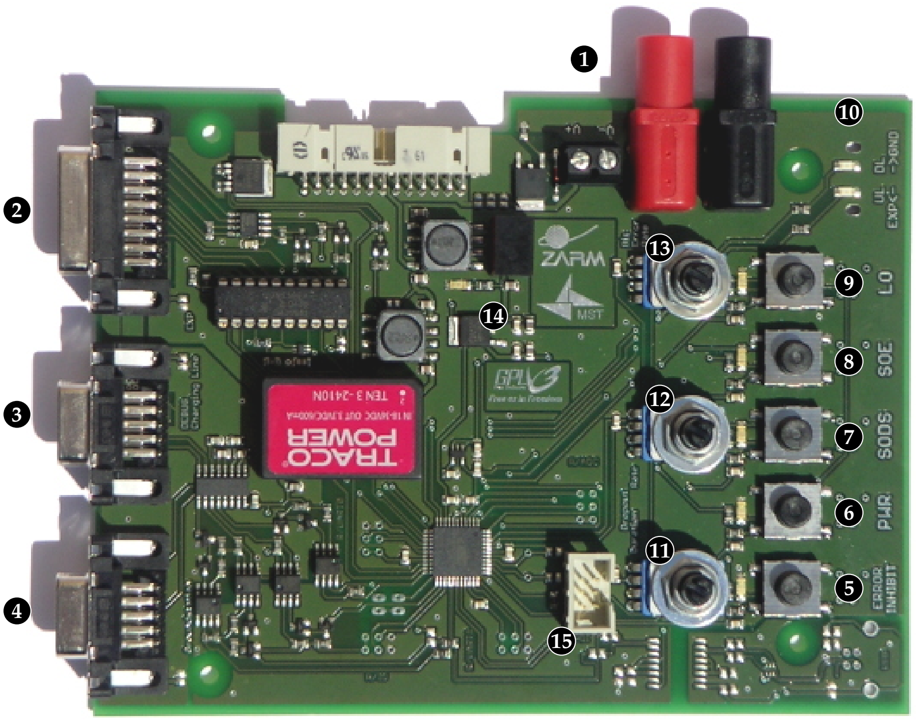


Figure 3.1.: REXUS Service Module Simulator in the RX Variant.

3.1. Experiment Power Up

In order to supply power to an experiment, it must be attached to the D-SUB 15 REXUS interface connector ❷. Then, a 28 V power supply is connected to the power interface ❶ on the simulator which are 4 mm jacks or a screw terminal.

When the power supply is switched on, a green status LED ⓮ lights up; however, the experiment remains powered-off. By pressing the *PWR* push button ❻, supply for the experiment is toggled and a status LED indicates the current state.

The simulator is reverse battery protected and does not power up in that case.

3.2. Control Interface

Every experiment is provided with the REXUS Control Interface consisting of three open-collector lines:

- $\overline{\text{LO}}$: Lift-Off
- $\overline{\text{SODS}}$: Start/Stop of Data Storage
- $\overline{\text{SOE}}$: Start/Stop of Experiment

A signal is asserted if it is set to GND, and is negated if it is floating. Both service module and simulator do not have a pull-up resistor for these lines; thus, it has to be added on the experiment side to pull the line to the 28 V supply voltage.

The signals can be toggled using the push buttons *SODS* ❼, *SOE* ❽, and *LO* ❾. A LED next to each push button is enabled if the corresponding signal is asserted.

3.3. Communication Link Test

The RXSM provides experiments with a bidirectional serial link to the team's ground station. Towards the experiment, this interface is exposed as differential RS-422 pins on the REXUS interface connector ❷, and ends in a RS-232 connection on the ground station ❹. Communication takes place at 38.4 kBd, 8 data bits, no parity, 1 stop bit.

While the rocket is at the launch pad, both uplink and downlink are provided using a wired connection. During flight, the uplink is disabled and data can only be sent via RF transmitters to a tracking antenna on the ground. As a result, there can be communication dropouts and bit errors, especially shortly after lift-off when the antenna looses tracking for a short time.

The user can test up- and downlink by attaching the ground station to the RS-232 connector ❹. As soon as data is transmitted, a pair of LEDs ❿ starts flashing. There is one LED for uplink and one for downlink.

The uplink is automatically disabled by the simulator and all incoming data from the ground station is dropped, if $\overline{\mathrm{LO}}$ is asserted.

3.3.1. Error Generation

To test the robustness of the protocol in the experiment, for instance the use of sufficient error detecting codes to prevent accidental telecommand execution, or in the ground station, for example to detect bugs in the protocol parser that could cause the software to crash, the simulator allows for injecting errors in the byte stream. Two types of errors can be generated:

- Bit Error:

  A single bit in the data stream is flipped with probability p_e.

- Byte Dropout

  All communication is dropped with probability p_d and each dropout event takes time t_d.

For configuring the error generators, the three parameters can be adjusted using linear potentiometers ⓫ ⓬ ⓭. Their mechanical angle of approximately 300° is divided into 16 equally sized bins for p_e and p_d, and is divided into 32 bins for t_d.

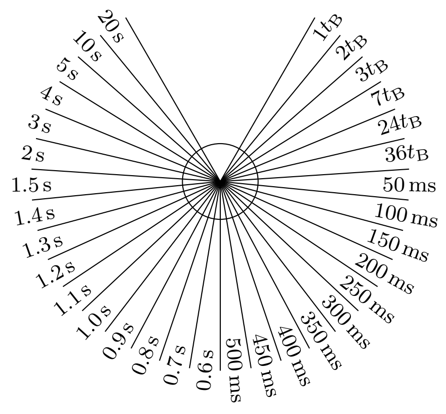
A predominantly base-two logarithmic scale is used to express the event probabilities (fig. 3.2c and fig. 3.2b) allowing for a wide range of values to be covered, whereas the scale for the duration (fig. 3.2a) has several linear regions.

By default after power up, error generation is disabled regardless of potentiometer positions and can be toggled using the *ERROR INHIBIT* push button ❺. The current status is shown by an LED distinguishing three states:

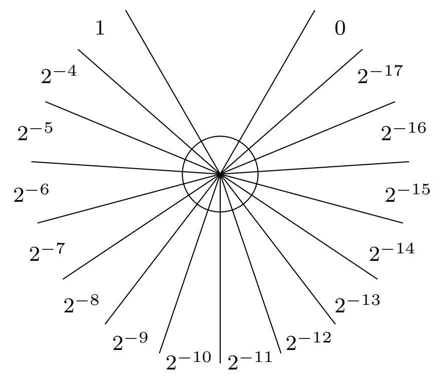
- LED continuously switched off: error generation is enabled and errors are generated;

- LED flashing: error generation is enabled, but the current potentiometer settings disable all errors;

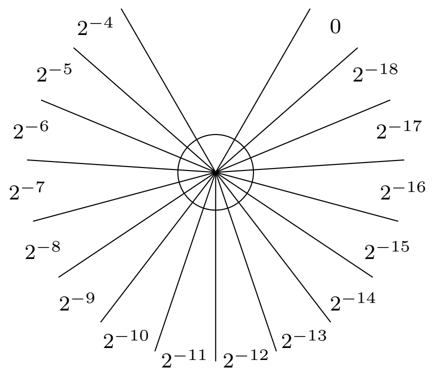
- LED continuously switch on: error generation is inhibited and the potentiometer settings are ignored.



(a) Dropout duration.



(b) Dropout rate.



(c) Bit error rate.

Figure 3.2.: Mapping of potentiometer mechanical position to parameters for error generators.

3.4. Charging Line

Some experiments contain internal batteries that need to be charged during times experiment and service module are switched off. Therefore, a charging line separate of the normal power supply interface is provided via a D-SUB 9 connector ❸.

The charging line does not embody a reverse polarity protection and is routed directly to the REXUS interface connector.

The D-SUB 9 connector (fig. 3.3) is shared with the debug interface. If both features need to be used, a breakout cable must be added to separate the three lines of the RS-232 debug interface and the six lines of the charging line.

Otherwise, equipment might be permanently damaged.

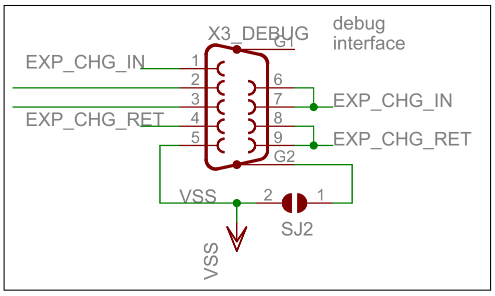


Figure 3.3.: Debug and Charging Line connector.

3.5. Debug Interface

Since the exact error generator properties only can be guessed visually from the potentiometer positions, the simulator regularly sends basic status information to the D-SUB 9 connector ❸.

If the charging line is not used by the experiment, it is safe to use a RS-232 cable to connect to a computer.

Otherwise, a breakout cable must be used to separate the three lines of the RS-232 debug interface and the six lines of the charging line, or equipment might be permanently damaged.

The information is sent as ASCII strings at 115.2 kBd, 8 data bits, no parity, 1 stop bit and uses Windows style CRLF line endings. The strings can be read with an ordinary terminal application, such as using GNU/Linux standard tools (listing 3.1).

```
[u1@debian ~]$ /usr/sbin/ldattach -s 115200 -8 -n -1 tty /dev/ttyUSB0
[u1@debian ~]$ cat /dev/ttyUSB0
Error Inhibit: OFF        Byte Dropout Rate: 2^-11        Dropout
   ↪ Duration: 800ms         Bit Error Rate: 2^-18
# many more lines
```

Listing 3.1: Reading the debug interface from Debian GNU/Linux.

4. REXUS Service Module Simulator – Extended Manual

4.1. System Architecture

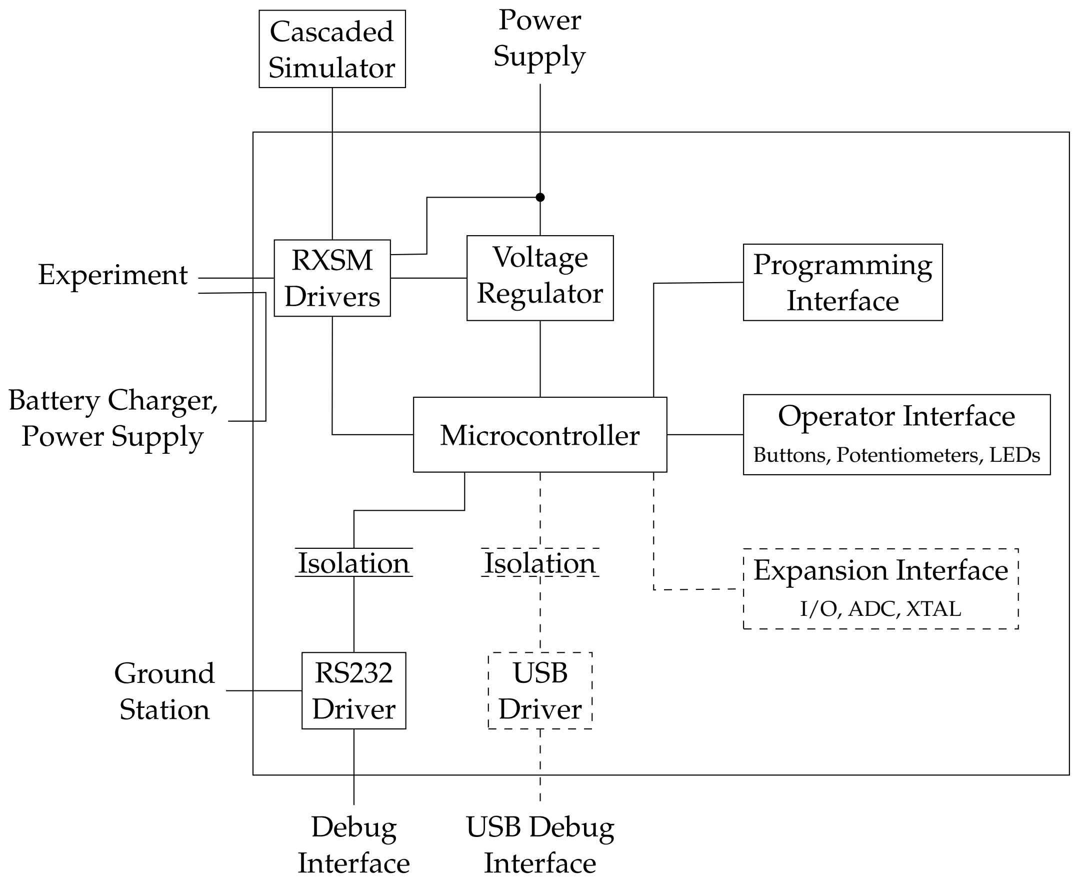


Figure 4.1.: Overview of the simulator system architecture (dashed parts: optional).

The REXUS service module simulator can be seen as a small system (fig. 4.1) that is driven by an Atmel ATxmega32A4U [1][2] microcontroller operated at 32 MHz. The software (section 4.5) implements a simple time-triggered scheduling of high level tasks which each handle a certain aspect of the problem.

The microcontroller interacts with the experiment though a driver unit that uses the same type of parts as the real RXSM. These are an Infineon BTS6143D high-side switch to enable or disable the experiment power supply, a SP490 transceiver to convert between the RS-422 signals of the experiment and the 3.3 V signals of the microcontroller UART, and a UDN2596A sink driver for the REXUS Control Interface. The latter part has been discontinued by the manufacturer and is getting increasingly rare, yet using these parts allows for a greater similarity of physical and electrical characteristics between simulator and service module.

By pressing the push buttons or by adjusting the potentiometers, the user can toggle the respective signals or can influence the generation of errors in the communication. The current status of the signals is indicated by various LEDs nearby the push buttons. In general, an LED is lit, if the signal is asserted and is switched off, if the signal is negated; however, the LEDs for $\overline{\text{LO}}$ and ERROR INHIBIT also support a flashing state. In the former case this is used in the cascaded simulator configuration (section 4.3) to show $\overline{\text{LO}}$ being asserted from another simulator. In the latter case it highlights error generation being enabled in general, but the current potentiometer settings disable all errors.

To exchange data with the ground station, an RS-232 transceiver converts the voltage levels. Nonetheless, the driver unit is not attached directly to the microcontroller, but via optocouplers to galvanically separate ground station and simulator. This prevents ground loops, for example when the power supply GND is removed and the current flows into the ground station's GND. Also for the real service module, there is galvanic separation between service module and ground station.

Since the used RS-232 transceiver has a second channel, it is used to emit internal debug information from the simulator (section 3.5). Due to the debug interface sharing the same connector with the charging line, a breakout cable to separate the two interfaces must be used.

The charging line, on the other hand, is directly connected to the respective pins on the REXUS interface connector and does not embody reverse polarity protection or any other safety measures.

As an option for the future, the PCB contains footprints for an isolated USB circuit and expansion interfaces for spare I/O channels and similar. These circuits are not tested and are not supported in the current microcontroller software.

4.2. Hardware Variants

The simulator PCB can be populated in four variants with slight differences:

- RX Variant

  This is the basic variant to be handed out to teams participating in the REXUS programme. It contains all potentiometers and SMD push buttons, as well as the 4 mm jacks.

- RX/MST Variant

  This variant is used for testing and development of the simulator at the Institute for Measurement Systems and Sensor Technology and is an extension to the RX Variant. The additions it may contain are extra connectors and the USB circuitry.

- ZARM Variant

  Several simulators (fig. 4.2) can be cascaded to allow for testing multiple experiments (section 4.3), for example as needed during integration testing done by ZARM[1]. To simplify handling of the various simulators, they can be put in a compact housing. Consequently, the 4 mm jacks and potentiometers are omitted and other push buttons suitable for the housing are chosen.

- ZARM/MST Variant

  Like the RX/MST Variant, this variant is for simulator testing and development and may contain additional components otherwise not present on an ordinary ZARM Variant simulator.

4.3. Cascaded Operation

Cascaded or multi-channel operation is an important feature to verify compatibility of experiments with each other. Most of the signals provided via the REXUS interface are separate to the specific experiment; however, the $\overline{\text{LO}}$ line and the power supply is electrically shared. Relying on individual experiment tests only would imply to miss events like an experiment accidentally asserting the $\overline{\text{LO}}$ line for all other experiments.

Consequently, cascaded simulators have to be powered from the same source and have to share the same $\overline{\text{LO}}$ line on the output side.

[1]`https://www.zarm.uni-bremen.de/`

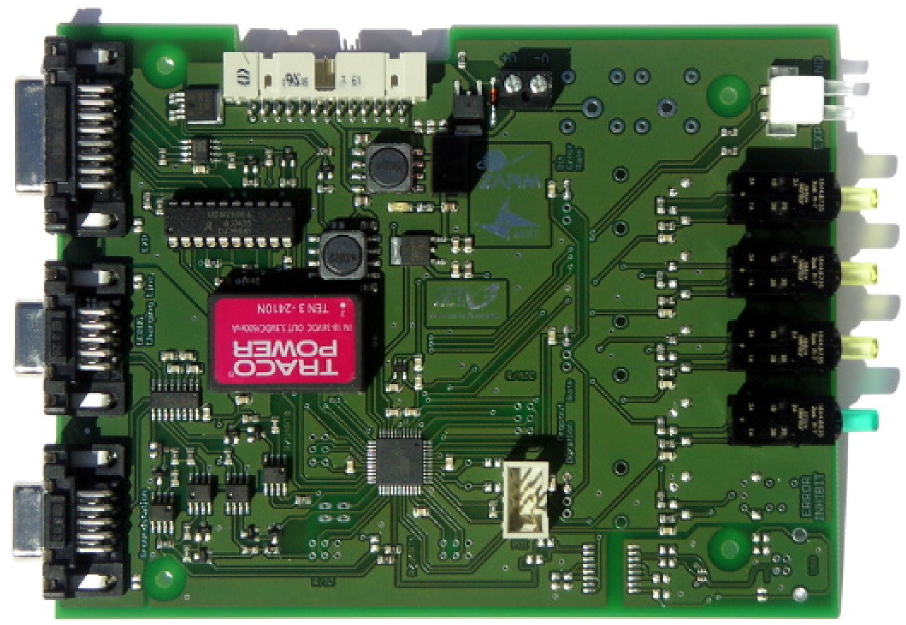


Figure 4.2.: Top view of a simulator PCB populated as ZARM Variant.

In addition, the simulators need to detect when $\overline{\text{LO}}$ is asserted for blocking the uplink. The following approaches were considered:

1. a backward channel sensing the state of $\overline{\text{LO}}$ to be added to every simulator, or
2. simulators sharing the internal signal for the REXUS interface drivers.

The first option could be realised using an optocoupler, just like it is recommended for the experiments to sense $\overline{\text{LO}}$ [5]. On the other hand, the RXSM does not provide the means to sense the line, so adding the optocoupler interferes with the electrical characteristics of the driver. Moreover, it also distinguishes the behaviour between simulator and RXSM by having the former block the uplink in the broader case of $\overline{\text{LO}}$ being asserted on the output side, and the latter only if an electrical connection to the ground structure is separated.

The second option is more complex to implement (fig. 4.3) because every simulator contains its own voltage regulator affected by tolerances and signal lines cannot be connected directly without risking damage to IO ports.

Each microcontroller is connected via the $\overline{\text{SIM\_LO}}$ signal with up to six other simulators. To allow the line to be pulled to GND, every attached IO pin is set to a wired-AND configuration, instead of totem-pole, and is not actively driven HIGH.

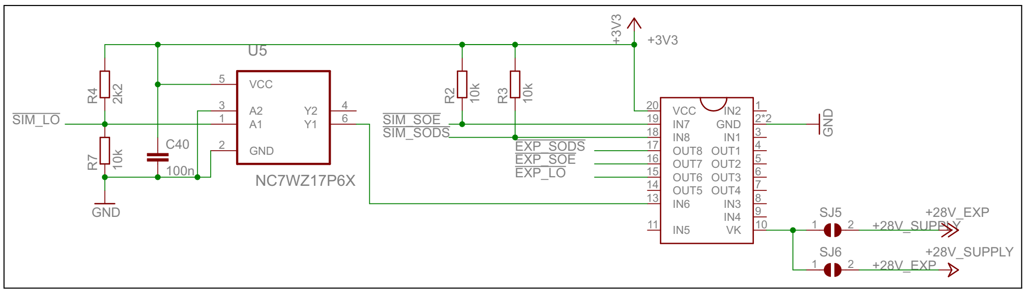


Figure 4.3.: Communicating the internal $\overline{\text{SIM\_LO}}$ signal to cascaded simulators.

Due to the previously mentioned tolerances of the 3.3 V regulators, a voltage divider is used to reduce the signal to approximately 2.7 V (eq. 4.1), which is safe for the IO pins even for a worst case combination of tolerances. At the same time, the level is above the IO pin input Schmitt trigger high threshold and can be read safely.

$$U_{\overline{\text{SIM\_LO}}} = \frac{R7}{R4 + R7} \cdot U_{3.3\,\text{V}} = \frac{10\,\text{k}\Omega}{10\,\text{k}\Omega + 2.2\,\text{k}\Omega} \cdot 3.3\,\text{V} = 2.7\,\text{V} \tag{4.1}$$

In contrast, the UDN2596A has no built-in Schmitt trigger; thus, a dedicated IC needs to be added.

Using this approach, several simulators can cooperate just by connecting them to the same power supply, the experiment side $\overline{\text{LO}}$ line and the internal $\overline{\text{SIM\_LO}}$ line.

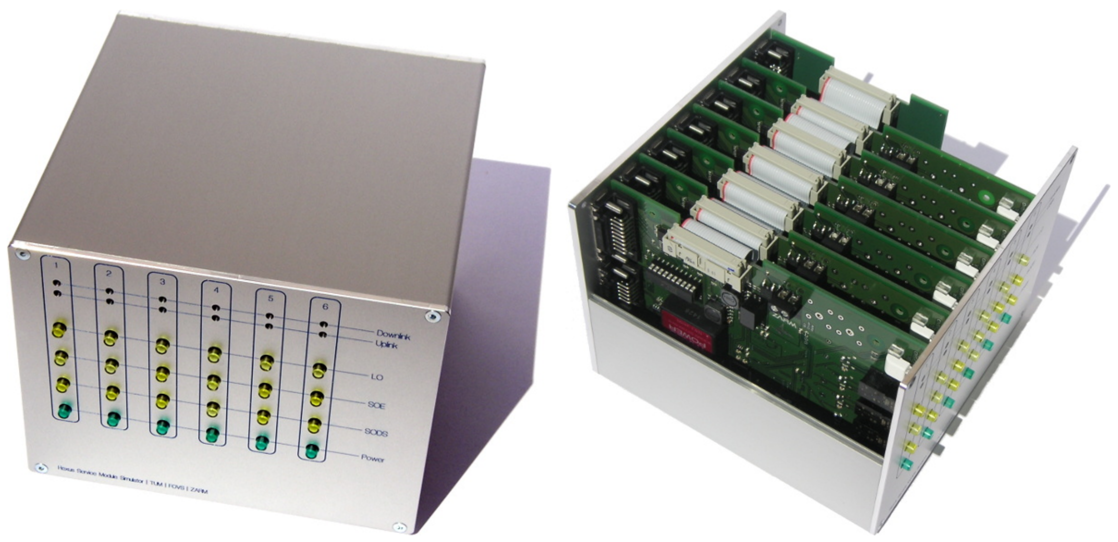
4.3.1. Housing

The simulator PCBs for each channel can be slit-in a housing (fig. 4.4) designed by Max Rößner and form a compact portable assembly. The structure is built from off-the-shelf top and bottom U-shaped parts, and front and back planes with custom engravings.

A ribbon cable connects all simulators with the power supply and the $\overline{\text{LO}}$ line pair. Furthermore, an additional PCB contains the reverse polarity protection and the 4 mm jacks for the complete assembly.

4.4. Subsequent Hardware Modification

Specific test setups and scenarios, as well as future upgrades might require to alter the circuit. To facilitate some of the expectable modifications, the simulator PCB already contains some footprints which are not populated by default in any variant.



(a) Front view with push buttons. (b) Exposed simulator PCBs in the housing.

Figure 4.4.: Compact housing for the assembled 6-channel simulator for ZARM.

4.4.1. Shield to GND Connection

Connectors with a shield, for example D-SUB connectors or the addable USB connector, have the shield disconnected from GND. This can be bypassed via SMD jumpers, for example in case a specific cable setup requires so. However, care must be taken not to accidentally violate the isolation barriers, for instance by connecting multiple shields to supposedly different GND domains through a conductive housing.

4.4.2. UDN2596N VK Pin

The UDN2596N sink driver for the REXUS Control Interface provides a diode from each open-collector output to a VK pin. In both RXSM and simulator, this pin is left floating. If a different behaviour is desired, it can be connected directly to the general 28 V power supply, or to the 28 V experiment power supply enabled or disabled by the high-side switch.

4.4.3. External Crystal Oscillator

In the current design, the microcontroller runs from an internal 32 MHz clock source. If this turns out to be too inaccurate for a specific feature, then it is possible to solder an external crystal and the needed capacitors.

4.4.4. Disabling the Reverse Voltage Protection

Some circumstances may require the on-board reverse voltage protection to be disabled. This can be achieved using SMD jumpers to bypass the protection circuit.

4.4.5. Expansion Connectors

Not all pins of the microcontroller are in use by the current design and can be exposed by soldering additional connectors. These allow for attaching expansion circuits for instance extra potentiometers, a separate push button matrix, or serial links to other microcontroller PCBs or simulators.

4.4.6. USB Support

A later version of the simulator could make use of the USB controller in the microcontroller, for example to replace the RS-232 debug interface and to allow higher data rates. In that case, two $0\,\Omega$ resistors need to be removed and the USB circuitry added.

4.5. Software

The PCB contains an Atmel ATxmega32A4U [1][2] microcontroller that executes the various tasks for transferring bytes between ground station and experiment, generating errors, or updating the operator interface. Still, the software only accumulates to slightly more than 1000 source lines of C code and is compatible with all hardware variants.

4.5.1. Time-Triggered Scheduling

In order to provide a high level of abstraction and separation between logically distinct tasks, the software realises a static, deterministic and provable time-triggered scheduling strategy (fig. 4.5). The same approach has already been demonstrated successful in the on-board data handling unit of the FOVS experiment [6] launched on RX15 in May 2014 and also in a physical intrusion detection system for home routers [3].

The available CPU time is split into equally sized slots of $25\,\mu s$ and eight slots in total. The tasks mapped to each slot are executed one after the other and are repeated from the first slot after each $200\,\mu s$ pass has been completed. The base interval between

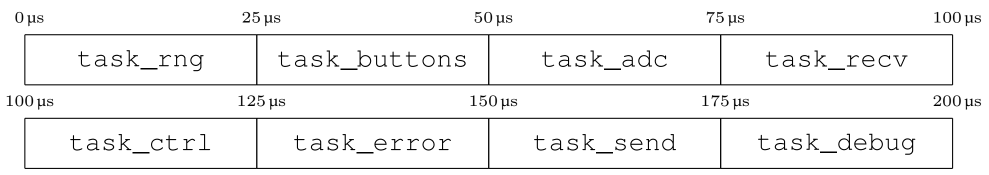


Figure 4.5.: Mapping from time slots to tasks.

slot beginnings is generated from a timer triggering the invocation of an interrupt service routine (ISR). Inside the ISR, an array of function pointers is accessed to read the address of the task handler, and the respective function is called (listing 4.1).

```
static const struct task *const scheduling_map[SCHED_SLOT_COUNT] = {
    &task_rng, &task_buttons, &task_adc, &task_recv,
    &task_ctrl, &task_error, &task_send, &task_debug
};
ISR(SCHED_TIMER_OVF_vect)
{
    uint8_t slot = next_slot;
    const struct task *t = scheduling_map[slot];
#if SCHED_SLOT_COUNT % 2 == 0
    next_slot = ((uint8_t) (slot + 1)) % SCHED_SLOT_COUNT;
#else
    if (++slot >= SCHED_SLOT_COUNT)
        slot = 0;
    next_slot = slot;
#endif

    if (NULL != t)
        t->run();
}
```

Listing 4.1: Fragment of the scheduler implementation.

Due to the ISR running regularly in constant-time and only one interrupt source being enabled, the remaining time jitter until the start of a task is caused by the length of the instruction to be completed when the interrupt occurred, that is, 1 to 5 CPU cycles or $31.25\,\mathrm{ns}$ to $156.25\,\mathrm{ns}$ respectively.

Furthermore, it can be proven that deadlines are met by counting the longest execution path for each task. The execution of the ISR already requires approximately 100 cycles out of the 800 cycles available for each slot (eq. 4.2) so it has a remarkable but acceptable overhead.

$$n_{\text{slot}} = 25\,\mu\text{s} \cdot 32\,\text{MHz} = 800 \tag{4.2}$$

Compared to this strategy, an event-based approach differs in several aspects and has various disadvantages. Using more than a single interrupt source, for example, makes it difficult to verify whether deadlines are met and potentially decreases readability of the code by splitting and mixing tasks to different ISRs. Moreover, the order of ISRs called is not deterministic so data dependence between tasks must be tracked manually.

4.5.2. Tasks

In total, there are eight different tasks which are providing and/or consuming information from other tasks. This data dependence (fig. 4.6) is already included in the task order in the scheduling plan, and therefore, is always guaranteed.

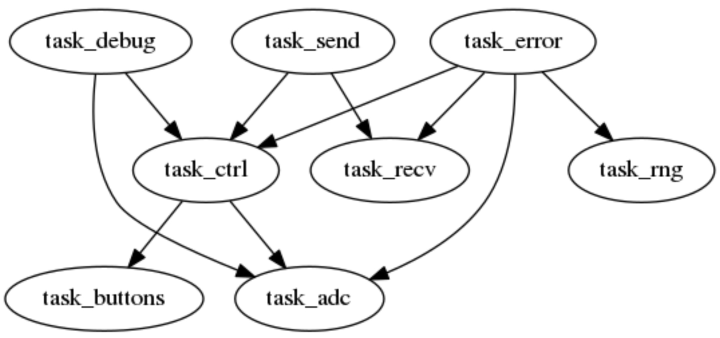


Figure 4.6.: Data dependence graph for tasks; $A \rightarrow B \Leftrightarrow A$ reads from B.

task_rng

The first task in the scheduling plan determines 16 pseudo-random bytes, which are later used for bit error and communication dropout generation. The initial implementation of this task called the `random()` function multiple times but it exceeded the available time of a slot. As a result, the implementation now uses the built-in AES-128 cryptographic peripheral in the microcontroller and obtains the pseudo-random numbers by repeatedly encrypting a block of 16 bytes.

task_buttons

This task continuously queries the state of the push buttons and digitally debounces the signal.

task_adc

The analog inputs, for instance attached to the potentiometers, are sampled sequentially, and then, a hysteresis is applied to the results. This removes noise from the measurements which otherwise could make the error generator jump between two error probabilities.

task_recv

The task checks and reads the UART receive buffers for the ground station and the experiment. If new data arrived, it is stored in an internal buffer in order to be forwarded.

task_ctrl

By evaluating data from the previous tasks, `task_ctrl` keeps track of the push button presses and the $\overline{\text{LO}}$ and ERROR INHIBIT states. It then asserts or negates the respective output signals of the REXUS Control Interface and updates the status LEDs.

task_error

This task combines data from several other tasks to generate the bit error patterns and to calculate the communication dropout parameters. The current error probabilities are derived from the potentiometer positions reported by `task_adc` and the pseudo-random numbers are taken from `task_rng`. In case `task_ctrl` requests errors to be inhibited, this task does nothing.

task_send

As the complement to `task_recv`, `task_send` forwards the previously received bytes from the ground station or experiment to the respective opposite UARTs. If $\overline{\text{LO}}$ is reported assert by `task_ctrl`, data from the ground station is dropped instead.

task_debug

To provide the user with some information on the current error generator parameters, this task continuously emits a string on the debug interface.

5. Conclusions and Future Work

This document described a new version of a REXUS Service Module Simulator that can be used by individual REXUS teams but also by engineers of the involved organisations for testing various aspects of the real REXUS Service Module. It explains the structure of the REXUS vehicle and some important features of the RXSM. Then, information on the interfaces and general usage instructions for the simulator are presented in a user manual, followed by a closer look at the simulator architecture and the implementation in the extended manual.

For the next steps, the manufactured simulators need to be deployed to the participating teams and organisations, and have to be proven usable and functional in a day to day work situation.

Furthermore, some of the prepared but not yet implemented features can be realised, for instance support for a bootloader to permit self-programming of firmware updates, or the addition of the USB interface. Also, due to the simulator being free software and free hardware design, users can add support for their specific use cases and needed functions.

6. Bibliography

[1] ATMEL CORPORATION. *Atmel AVR XMEGA AU MANUAL*. `http://www.atmel.com/Images/Atmel-8331-8-and-16-bit-AVR-Microcontroller-XMEGA-AU_Manual.pdf`, Apr 2013. Accessed: 2014-10-10.

[2] ATMEL CORPORATION. *ATxmega16A4U/32A4U/64A4U/128A4U*. `http://www.atmel.com/images/Atmel-8387-8-and16-bit-AVR-Microcontroller-XMEGA-A4U_Datasheet.pdf`, Oct 2014. Accessed: 2014-10-10.

[3] BENEŠ, N. *An Approach for Home Routers to Securely Erase Sensitive Data*. Bachelor Thesis, Technische Universität München, Oct 2014.

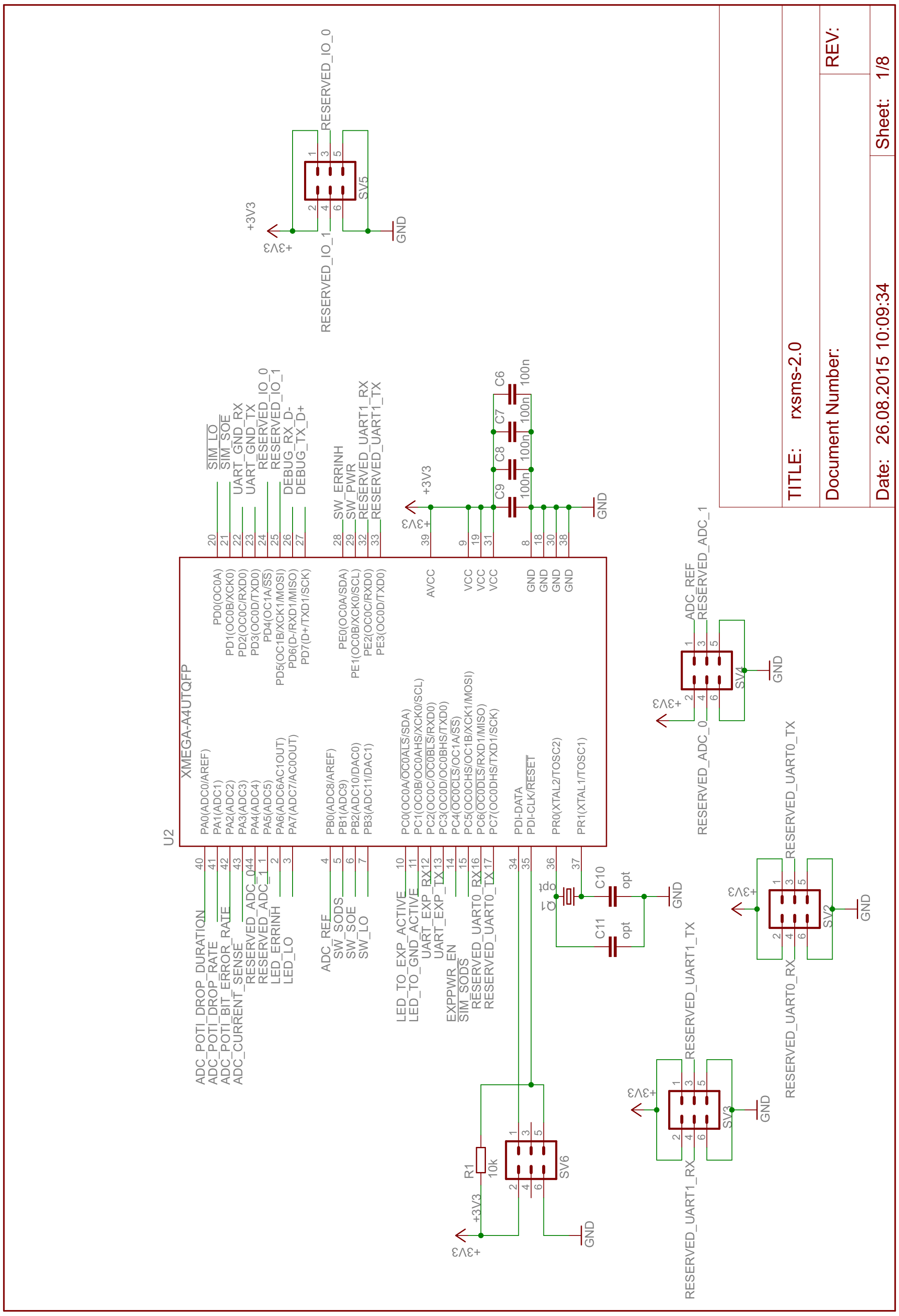
[4] KHOLODKOV, J. *Rexus Sevice Modul Simulator*. Forschungsarbeit, Technische Universität München, Jan 2014.

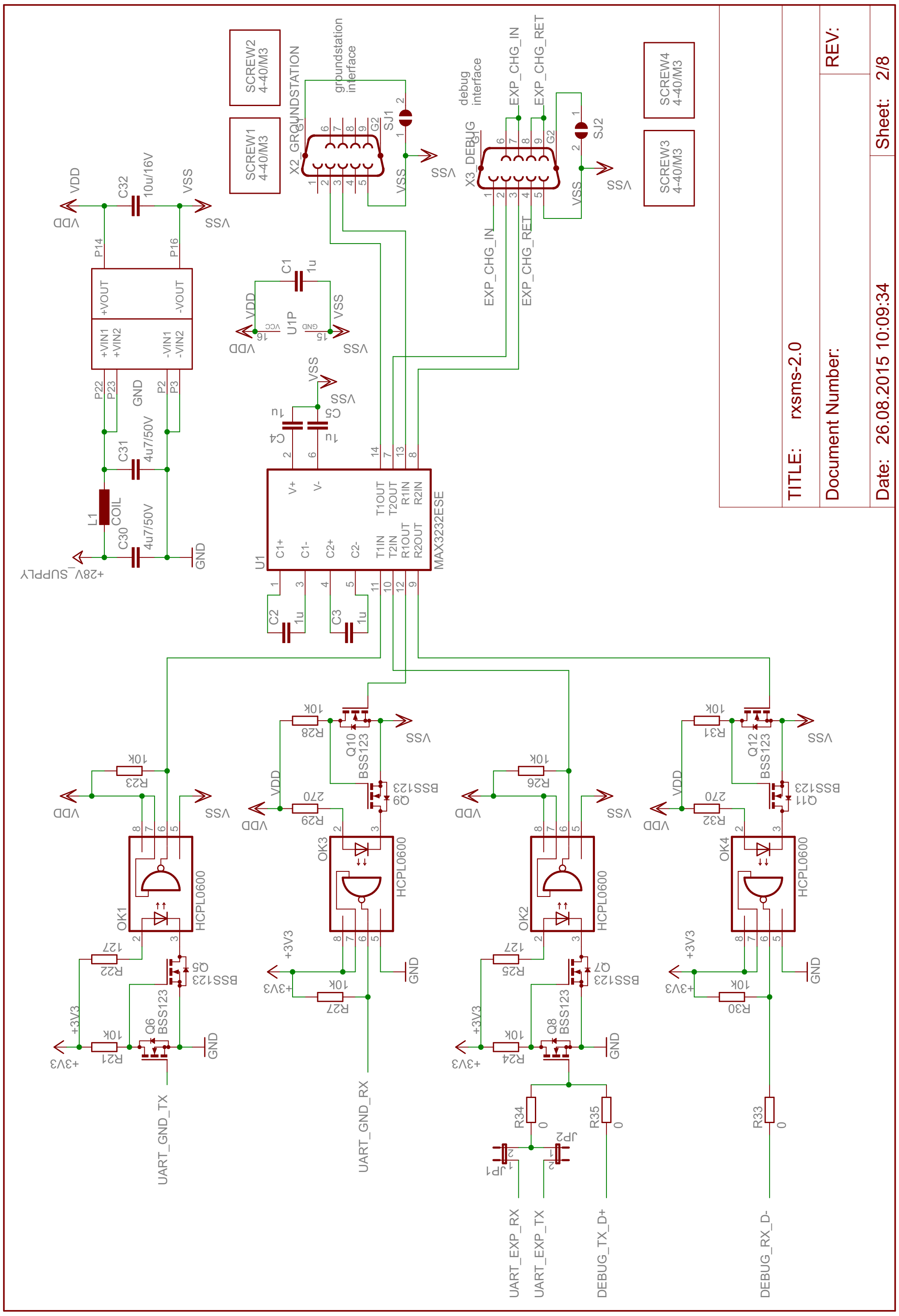
[5] MAWN, S. ET AL. *REXUS User Manual*, 7.13 ed. EuroLaunch, `http://rexusbexus.net/wp-content/uploads/2015/06/rx_usermanual_v7-13_15dec14.pdf`, Dec 2014. Accessed: 2015-08-29.

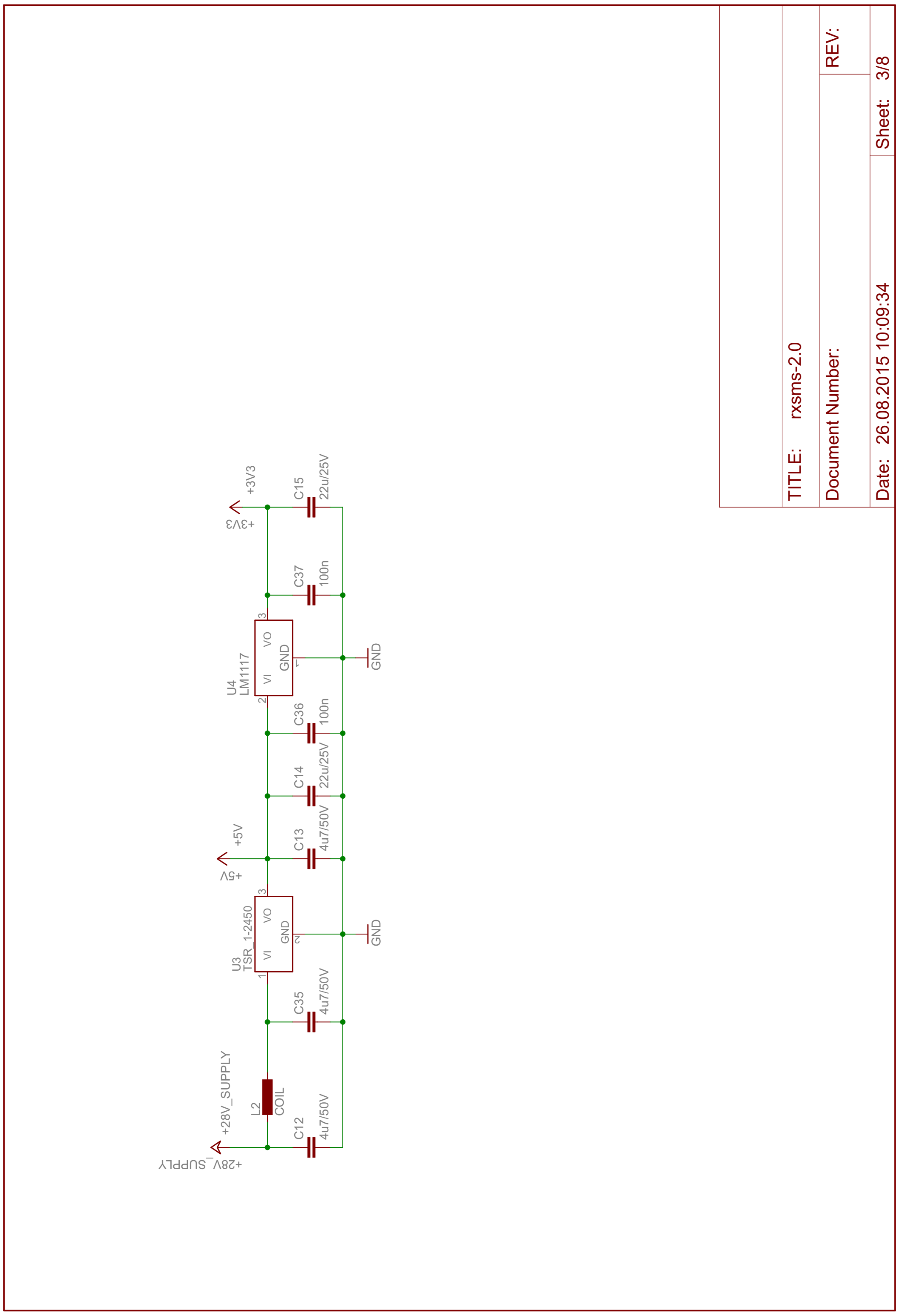
[6] RÖSSNER, M., BENES, N., GRÜBLER, T., HESSKE, A., HORNUNG, A., KHOLODKOV, J., KIN, V., PLAMAUER, S., POPP, A., AND UHLENKAMP, T. *Student Experiment Documentation – Fiber-Optic Vibration Sensing Experiment*, 5 ed., Nov 2014.

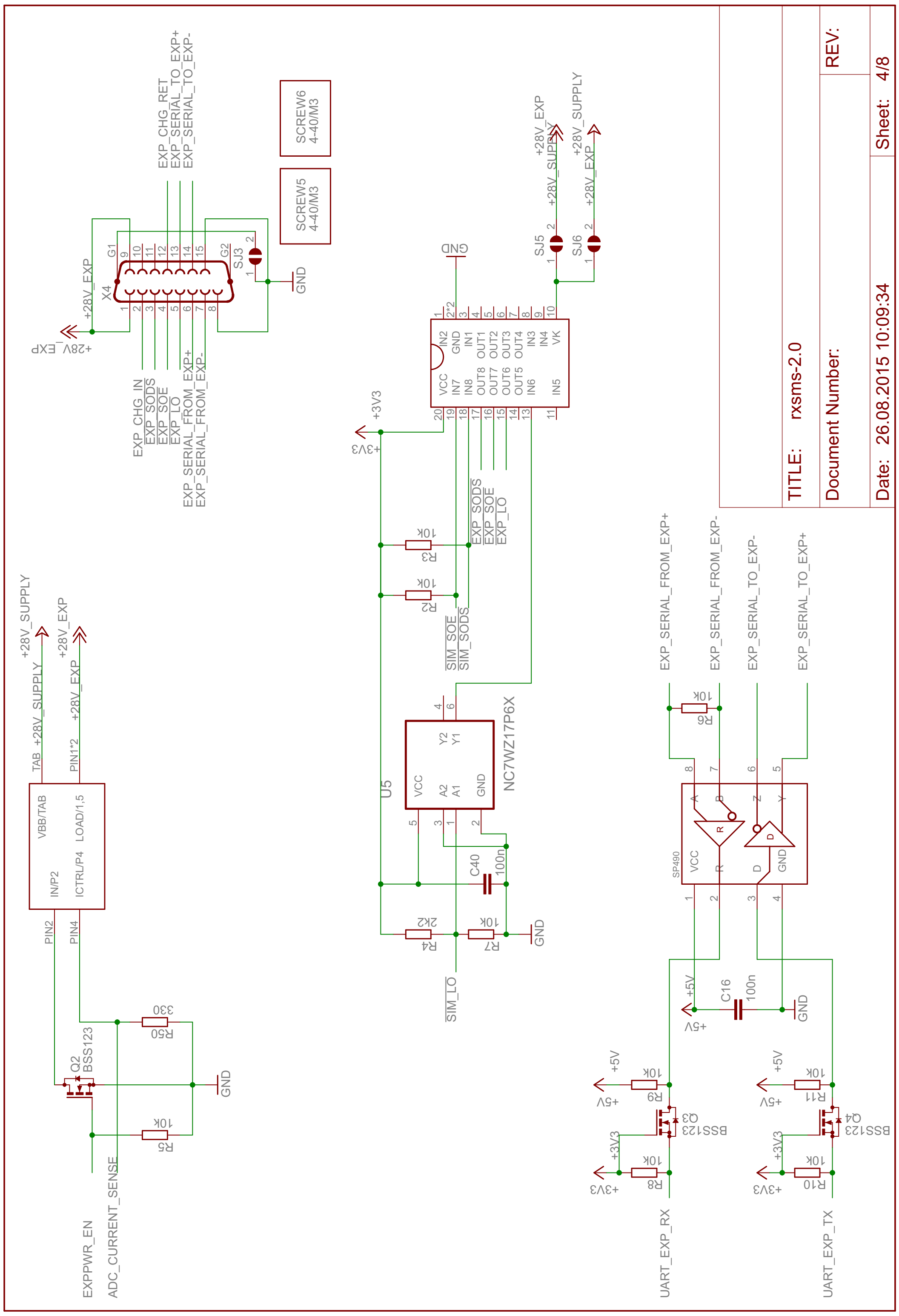
A. Appendix

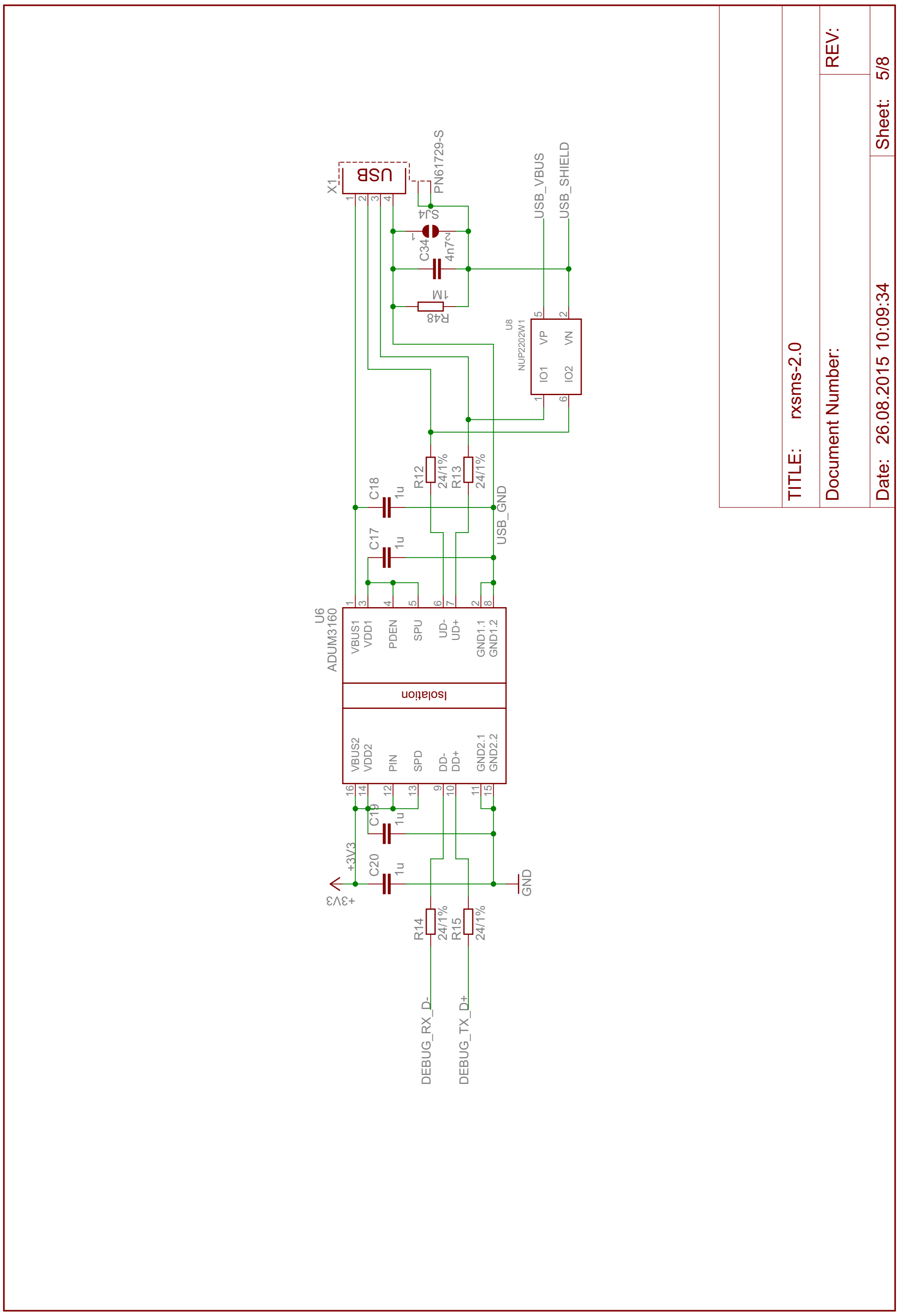
A.1. Schematics

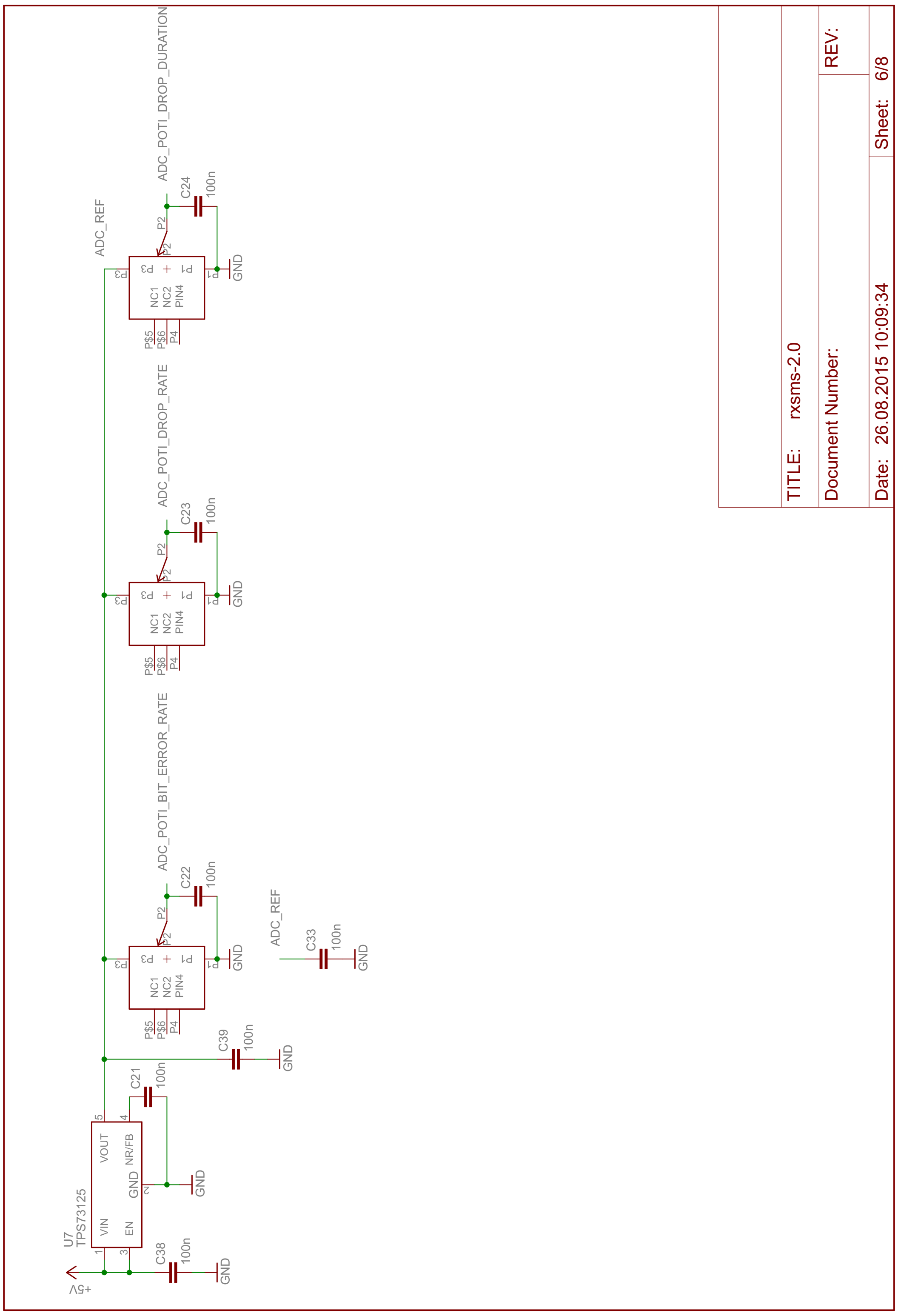

TITLE: rxsms-2.0
Document Number:
REV:
Date: 26.08.2015 10:09:34
Sheet: 1/8
U2
XMEGA-A4UTQFP
+3V3
GND
SV2
SV3
SV4
SV5
SV6
R1
10k
C6
C7
C8
C9
100n
C10
C11
opt
Q1
ADC_POTI_DROP_DURATION
ADC_POTI_DROP_RATE
ADC_POTI_BIT_ERROR_RATE
ADC_CURRENT_SENSE
RESERVED_ADC_0
RESERVED_ADC_1
LED_ERRINH
LED_LO
ADC_REF
SW_SODS
SW_SOE
SW_LO
LED_TO_EXP_ACTIVE
LED_TO_GND_ACTIVE
UART_EXP_RX
UART_EXP_TX
EXPPWR_EN
SIM_SODS
RESERVED_UART0_RX
RESERVED_UART0_TX
SIM_LO
SIM_SOE
UART_GND_RX
UART_GND_TX
RESERVED_IO_0
RESERVED_IO_1
DEBUG_RX_D-
DEBUG_TX_D+
SW_ERRINH
SW_PWR
RESERVED_UART1_RX
RESERVED_UART1_TX

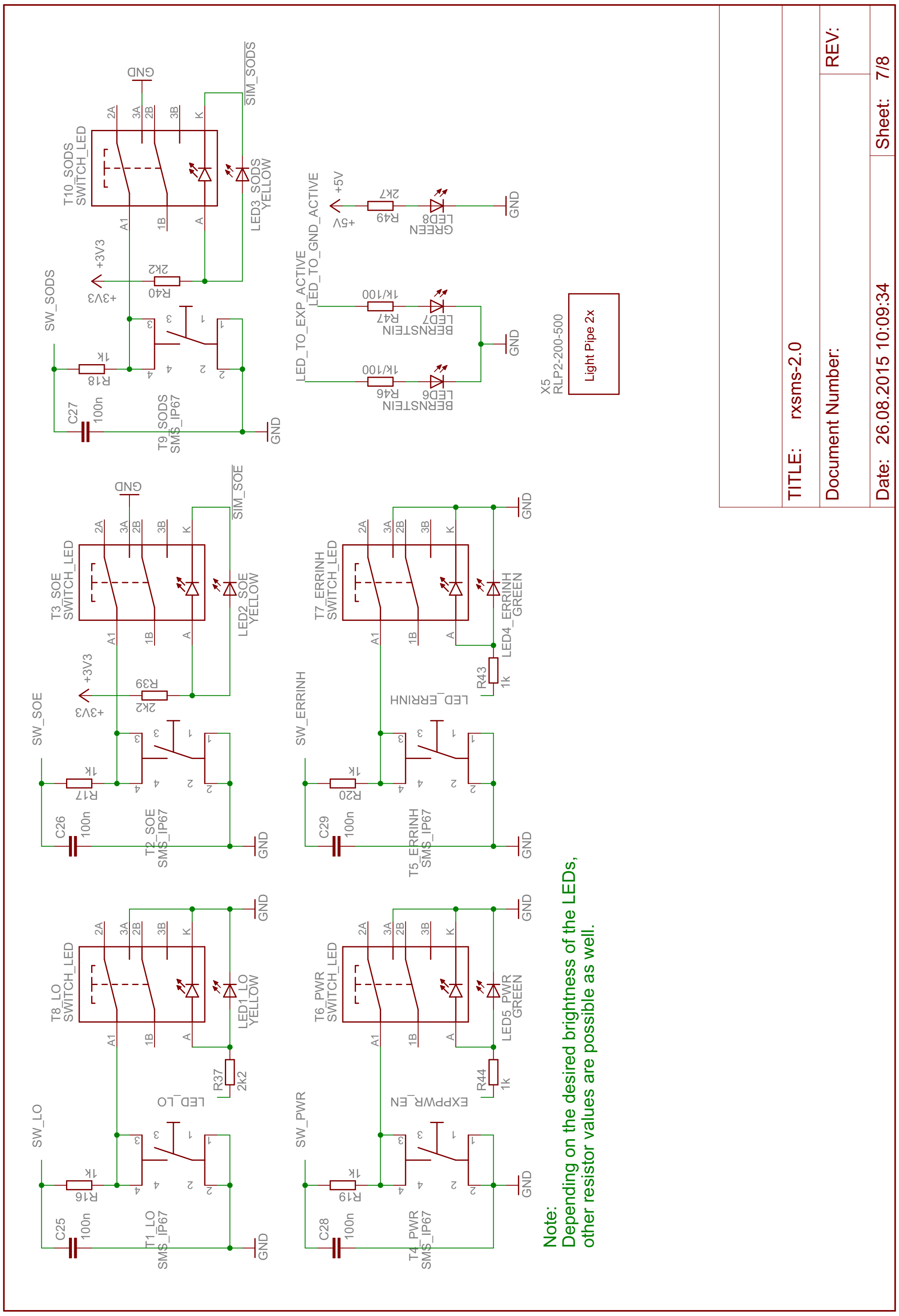

TITLE: rxsms-2.0
Document Number:
REV:
Date: 26.08.2015 10:09:34
Sheet: 2/8
SCREW1 4-40/M3
SCREW2 4-40/M3
SCREW3 4-40/M3
SCREW4 4-40/M3
X2_GROUNDSTATION
groundstation interface
X3_DEBUG
debug interface
SJ1
SJ2
EXP_CHG_IN
EXP_CHG_RET
VDD
VSS
GND
+3V3
+28V_SUPPLY
C32 10u/16V
C31 4u7/50V
C30 4u7/50V
L1 COIL
C1 1u
C2 1u
C3 1u
C4 1u
C5 1u
U1 MAX3232ESE
U1P
+VOUT
-VOUT
+VIN1
+VIN2
-VIN1
-VIN2
T1IN
T2IN
R1OUT
R2OUT
T1OUT
T2OUT
R1IN
R2IN
C1+
C1-
C2+
C2-
V+
V-
OK1 HCPL0600
OK2 HCPL0600
OK3 HCPL0600
OK4 HCPL0600
Q5 BSS123
Q6 BSS123
Q7 BSS123
Q8 BSS123
Q9 BSS123
Q10 BSS123
Q11 BSS123
Q12 BSS123
R21 10k
R22 127
R23 10k
R24 10k
R25 127
R26 10k
R27 10k
R28 10k
R29 270
R30 10k
R31 10k
R32 270
R33 0
R34 0
R35 0
JP1
JP2
UART_GND_TX
UART_GND_RX
UART_EXP_RX
UART_EXP_TX
DEBUG_TX_D+
DEBUG_RX_D-

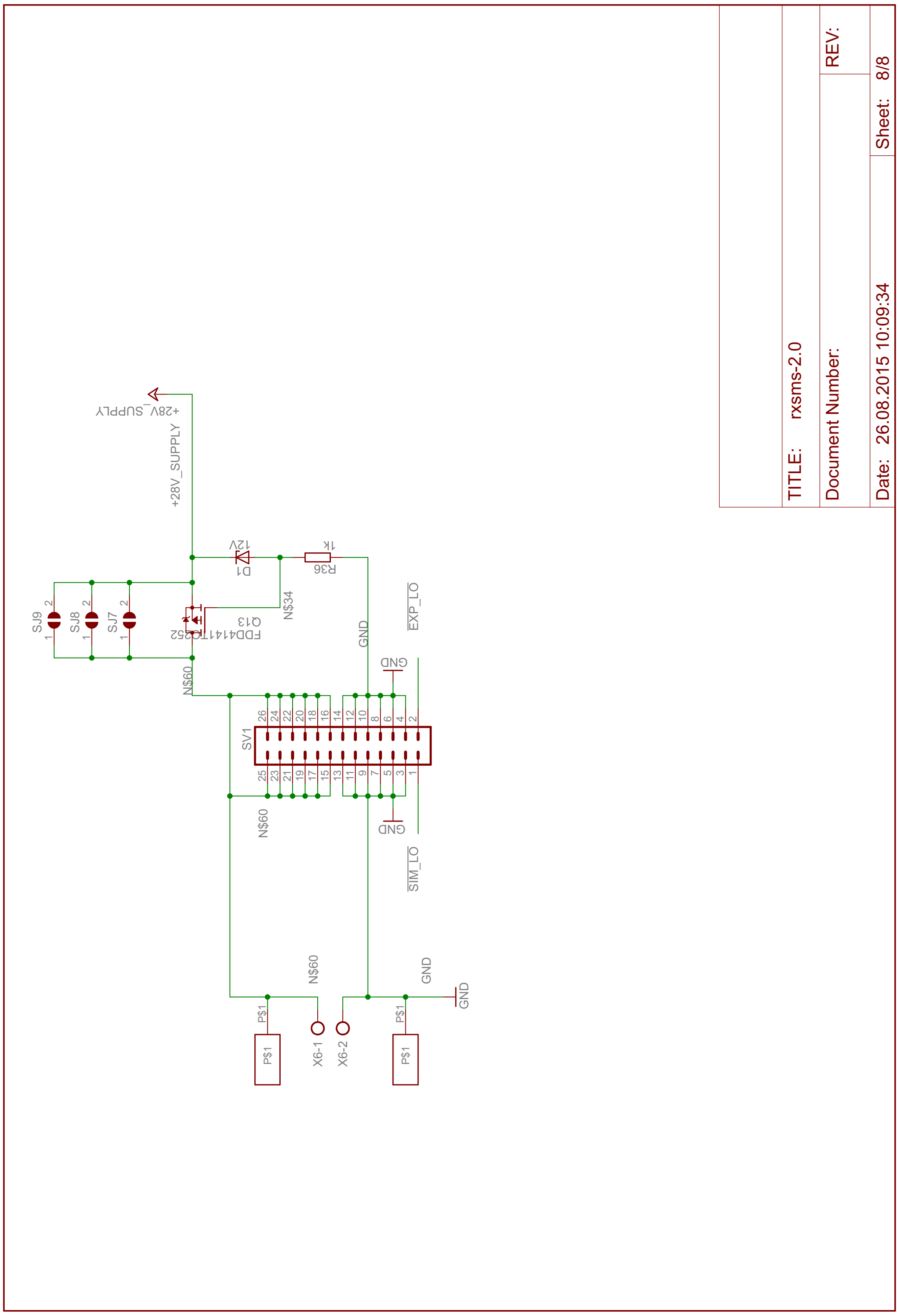

+28V_SUPPLY
L2
COIL
C12
4u7/50V
C35
4u7/50V
U3
TSR 1-2450
VI
GND
VO
GND
+5V
C13
4u7/50V
C14
22u/25V
C36
100n
U4
LM1117
VI
GND
VO
GND
C37
100n
+3V3
C15
22u/25V
TITLE: rxsms-2.0
Document Number:
REV:
Date: 26.08.2015 10:09:34
Sheet: 3/8


TITLE: rxsms-2.0
Document Number:
REV:
Date: 26.08.2015 10:09:34
Sheet: 4/8
X4
SJ3
SJ5
SJ6
SCREW5 4-40/M3
SCREW6 4-40/M3
U5
NC7WZ17P6X
SP490
Q2 BSS123
Q3 BSS123
Q4 BSS123
C40 100n
C16 100n
R2 10K
R3 10K
R4 2K2
R5 10K
R6 10K
R7 10K
R8 10K
R9 10K
R10 10K
R11 10K
R50 330
+28V_EXP
+28V_SUPPLY
+3V3
+5V
GND
EXP_CHG_IN
EXP_SODS
EXP_SOE
EXP_LO
EXP_SERIAL_FROM_EXP+
EXP_SERIAL_FROM_EXP-
EXP_CHG_RET
EXP_SERIAL_TO_EXP+
EXP_SERIAL_TO_EXP-
SIM_SOE
SIM_SODS
SIM_LO
EXPPWR_EN
ADC_CURRENT_SENSE
UART_EXP_RX
UART_EXP_TX

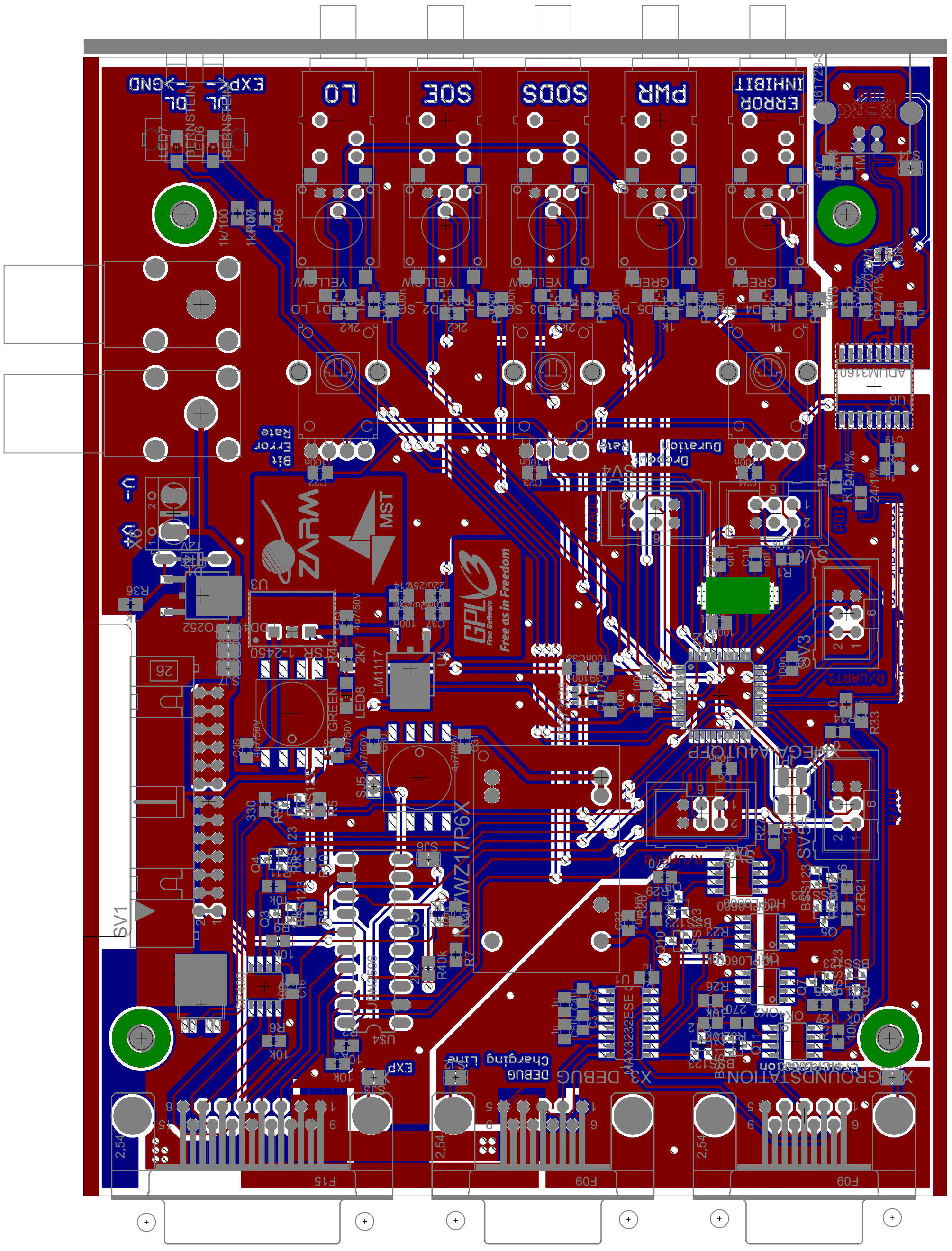

X1
USB
PN61729-S
SJ4
C34
4n7
R48
1M
U8
NUP2202W1
IO1
IO2
VP
VN
USB_VBUS
USB_SHIELD
USB_GND
C18
1u
C17
1u
R12
24/1%
R13
24/1%
U6
ADUM3160
VBUS1
VDD1
PDEN
SPU
UD-
UD+
GND1.1
GND1.2
Isolation
VBUS2
VDD2
PIN
SPD
DD-
DD+
GND2.1
GND2.2
C19
1u
C20
1u
+3V3
GND
R14
24/1%
R15
24/1%
DEBUG_RX_D-
DEBUG_TX_D+
TITLE: rxsms-2.0
Document Number:
REV:
Date: 26.08.2015 10:09:34
Sheet: 5/8


ADC_POTI_DROP_DURATION
ADC_POTI_DROP_RATE
ADC_POTI_BIT_ERROR_RATE
ADC_REF
C24
C23
C22
C21
C33
C38
C39
100n
P1
P2
P3
P4
P$5
P$6
NC1
NC2
PIN4
GND
U7
TPS73125
VIN
EN
GND
VOUT
NR/FB
+5V
TITLE: rxsms-2.0
Document Number:
REV:
Date: 26.08.2015 10:09:34
Sheet: 6/8


TITLE: rxsms-2.0
Document Number:
REV:
Date: 26.08.2015 10:09:34
Sheet: 7/8
Note:
Depending on the desired brightness of the LEDs,
other resistor values are possible as well.
SW_LO
SW_PWR
SW_SOE
SW_ERRINH
SW_SODS
T1_LO SMS_IP67
T8_LO SWITCH_LED
T4_PWR SMS_IP67
T6_PWR SWITCH_LED
T2_SOE SMS_IP67
T3_SOE SWITCH_LED
T5_ERRINH SMS_IP67
T7_ERRINH SWITCH_LED
T9_SODS SMS_IP67
T10_SODS SWITCH_LED
LED1_LO YELLOW
LED5_PWR GREEN
LED2_SOE YELLOW
LED4_ERRINH GREEN
LED3_SODS YELLOW
LED_LO
EXPPWR_EN
LED_ERRINH
SIM_SOE
SIM_SODS
LED_TO_EXP_ACTIVE
LED_TO_GND_ACTIVE
LED6 BERNSTEIN
LED7 BERNSTEIN
LED8 GREEN
R46 1k/100
R47 1k/100
R49 2k7
R16 1k
R17 1k
R18 1k
R19 1k
R20 1k
R37 2k2
R39 2k2
R40 2k2
R43 1k
R44 1k
C25 100n
C26 100n
C27 100n
C28 100n
C29 100n
+3V3
+5V
GND
X5 RLP2-200-500
Light Pipe 2x


+28V_SUPPLY
+28V_SUPPLY
D1
12V
R36
1k
Q13
FDD4141TO252
N$34
SJ9
SJ8
SJ7
N$60
GND
EXP_LO
SV1
GND
GND
SIM_LO
N$60
N$60
GND
GND
P$1
X6-1
X6-2
P$1
TITLE: rxsms-2.0
Document Number:
REV:
Date: 26.08.2015 10:09:34
Sheet: 8/8

A.2. Layout


LO
SOE
SODS
PWR
ERROR INHIBIT
EXP<-->GND
UL DL
Bit Error Rate
Duration Dropout
MEGA4AUTOFP
MAX3232ESE
DEBUG
Charging Line
EXP
GROUNDSTATION
ADUM3160
LM1117
F15
F09
SV1
X3