PREcup-1: An Embedded System Platform for Prototyping ECU Power Management

Andreas Barthels*, Florian Ruf†, Alexander Schlenk†, Gregor Walla‡, Hans-Ulrich Michel§, and Uwe Baumgarten*
*Institute for Informatics, Technische Universitaet Muenchen, Munich, Germany
Email: barthels@in.tum.de
†Institute for Energy Conversion Technology, Technische Universitaet Muenchen, Munich, Germany
‡Institute for Integrated Systems, Technische Universitaet Muenchen, Munich, Germany
§BMW Group, Research and Technology, Munich, Germany

***Abstract*—This paper presents an embedded system platform for investigating power management methods relevant to future automotive systems. The platform is built around an ARM Cortex A8 System on Chip (SoC). It allows for measuring the power consumption of the SoC and its peripherals. The SoC has a defined set of power states, namely frequency and voltage operating points, as well as retention and off-modes. During runtime, the power states are managed by a novel Linux-based scheduler. It allows the execution of plans combining both timed software and power state switches. The design is evaluated using a single ECU and its local scheduling.**

I. Introduction

Modern vehicles feature a lot of functions driven and controlled by electronics. Having innovative functions well integrated in a vehicle is a major selling point. This has lead to an abundance of interconnected electronic control units (ECUs) in luxury class vehicles. The process of engineering the architecture of such complex systems is supported by using test benches.

To this date, automotive test benches are mostly used for investigating power net architectures. These test benches are either used for investigating voltage stability in power nets [1]–[3], or used for investigating power generation efficiency in today's cars [4]. Beside the efficiency of power generation, the efficiency of ECUs becomes more and more important. Any amount of saved energy leads to an increased range and mileage for both electric and conventional vehicles. In order to investigate automotive related power management strategies in an academic environment, a test bench based upon a flexible ECU platform interconnected with automotive buses is essential.

This paper presents an embedded system platform which resembles an ECU in the automotive domain. The platform allows for testing different power management paradigms. It runs Linux, thus it inherits all capabilities for traditional desktop and embedded systems. The system is extended by modifying the device drivers and the system scheduler, allowing for a mixture of event triggered, and flexible time-triggered operation modes including power state setting. Power states are set for both for an ARM Cortex A8 based system on chip (SoC), but also the peripherals.

The ECU features circuitry for powering attached sensors and actuators as well as for measuring the consumption of the different components. The platform thus combines hardware and software features allowing to emulate different use case scenarios of vehicular ECUs. Traditional ECUs for controlling the engine, chassis or driving assistance systems have hard real-time properties. There are a large number of body functions which are activated by the driver or passengers–these functions have reaction time constraints which are less critical than those essential for driving the car. More recently, vehicles became equipped with entertainment systems, which are less critical and could typically be emulated by unmodified Linux.

A. Power Management Paradigms

For the automotive view on power management, the standardization initiative AUTOSAR [5] has to be noted. The initiative distinguishes three power management mechanisms:

- *ECU degradation* aims to allow for switching off subsystems of an ECU. This approach is not yet accepted as a standard but in draft status.
- *Pretended Networking* allows to turn off ECUs and offload their communication to a dedicated hardware unit [6]. Pretended networking is still in draft status.
- *Partial Networking* is a network centric approach which is accepted as standard. It covers the explicit communication protocol for allowing to disable network packets, shut down ECUs and whole bus systems [7].

Besides these AUTOSAR concepts, there exists a lot of related work in terms of system-level power management. Benini et al. surveyed design techniques for system-level power management [8]. More recent work includes power-aware multi core scheduling [9] and cyber-physical systems [10], [11].

B. Structure

The paper is structured as follows. Section II describes the motherboard and circuitry relevant to emulating ECU functionality. Section III lists the extensions to the Linux operating system as a software platform which were implemented. For allowing to emulate the functions with real-time properties, the system scheduling was extended. The paper continues by describing the test setup and results in sections IV and V.

TABLE I
POWER STATES OF THE OMAP-BASED ARM PLATFORM [13]

Power State	ARM MPU	ARM Core
C1	WFI	ON
C2	WFI	inactive
C3	CSWR	inactive
C4	OFF	inactive
C5	OSWR	OSWR
C6	OFF	OSWR
C7	OFF	OFF

Finally, the paper concludes by reflecting on the contribution and providing an outlook.

II. HARDWARE PLATFORM

The motherboard is a custom designed PCB featuring CAN-Bus and Ethernet for automotive connectivity, power supply and control of external sensors or actuators. The main SoC is not directly mounted on the board. For simplicity reasons, a module (called Gumstix Overo [12]) has to be plugged in, which includes the SoC and all its necessary peripherals like RAM or local power supply. The design includes a flexible programmable gate array (FPGA) that can be used for offloading hardware related tasks from the main CPU. This allows for example to monitor or control analog values at a high sampling rate of the integrated analog-digital and digital-analog-converters. The whole design is depicted in figure 1.

A. Computational Sub-system

The main processor on the motherboard which is used for simulating a generic ECU is represented by a Texas Instruments OMAP 3503 processor [14]. This SoC is based on an ARM Cortex-A8 based processor, capable of running at frequencies of 600 MHz and below. The SoC is the small sibling of the widely used OMAP3530, which features an additional graphics accelerator and a discrete DSP core. For simulating especially computational loads these feature were not necessary. Overall the small variant is still powerful enough to implement almost every common ECU task that is present in an automotive environment.

B. Power Management Techniques

The chip supports three different types of power management [15] of which two are actively used in this experiment:

- DVFS (Dynamic Voltage and Frequency Scaling),
- AVS (Adaptive Voltage Scaling), and
- DPS (Dynamic Power Switching).

The most common mechanism is DVFS, which allows the scaling of the CPU frequency and its core voltage in accordance to the current processing load. The second, but not actively used technique is AVS, which allows to measure the degradation of the silicon and to adjust the operating voltage accordingly. Another possibility for power saving is DPS (dynamic power switching) which allows to explicitly switch off certain parts of the IC. The OMAP processor is divided up into separate power domains, which can be controlled

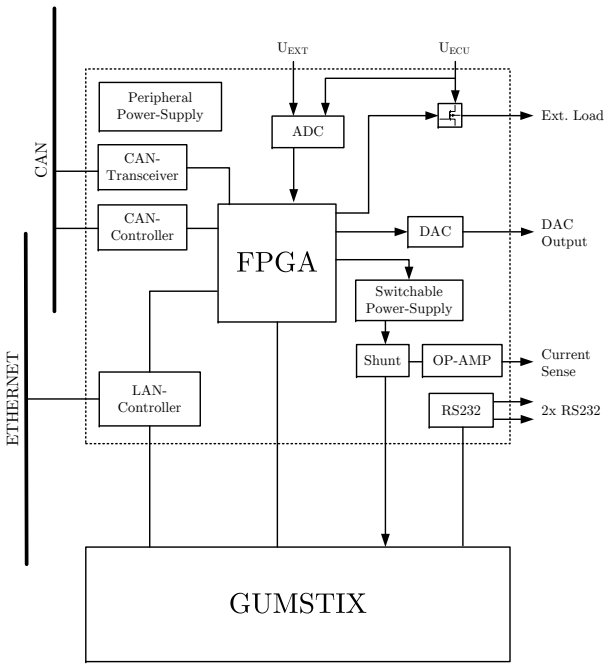


Fig. 1. Schematic Structure of the Custom Platform

independently. The SoC features approximately 10 of these domains, depending of what is counted. The most important ones (by the names used in the Linux kernel) are: CORE, MPU, DSS, IVA, SGX, NEON. The SGX power domain is not applicable for this IC, because it refers to the optional graphics accelerator. NEON and IVA are multimedia subsystems and DSS is the Display Subsystem. For this experiment the focus is one the CORE and MPU domain. MPU is the actual ARM processor, while CORE is the peripheral circuitry. Each power domain can enter one of the following states:

- ON: All circuits are fully operational.
- WFI (Wait for Interrupt): Equals a halt command. The logic is fully powered, but not actively working.
- RET/CSWR (Closed Switch Retention): The logic is in retention mode, but still powered
- RET/OSWR (Open Switch Retention): The logic is switched off, but the state is preserved in dedicated static RAM.
- OFF: The component is completely powered off

In the Linux kernel it is possible to refer to a combination of these power domains as power states. The mapping is shown in table I.

C. Peripheral Circuitry

The ARM SoC is paired with some generic peripherals for interconnection with other ECUs. This includes an Ethernet port and an SPI based CAN controller. Those two are solely used by the ARM SoC. Furthermore, the motherboard contains an analog-digital- and digital-analog-converter, as well as some general purpose pins and a power MOSFET

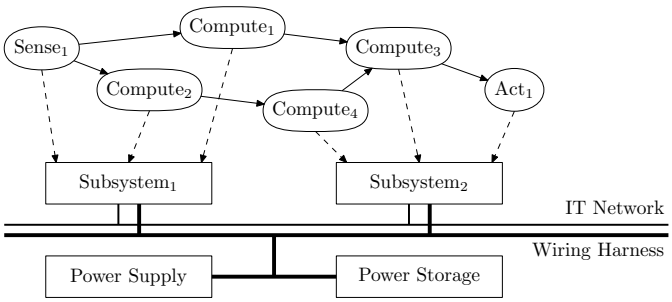


Fig. 2. Functional chain together with mapping onto system components. Dashed lines indicate mapping to hardware. Taken from [11].

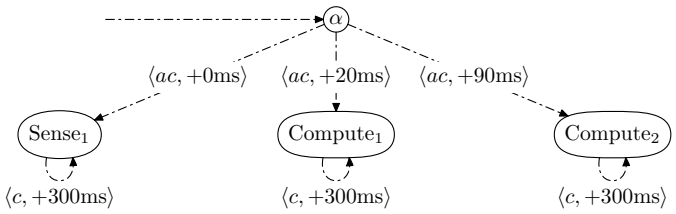


Fig. 3. Fixed cycle scheduling as a power management plan, taken from [11]. α denotes the change of a power state, i.e. the system start-up itself. The arcs are annotated with ac and c, which stands for after completion and concurrently, respectively.

for switching external loads. For a flexible usage of these components, they are all connected to the FPGA.

D. Measurement Circuitry

For evaluating the effect of the different power states of the OMAP SoC, the motherboard provides an adequate measurement circuit. The consumed energy can be evaluated by measuring the current flow and multiplying it with the fixed voltage. The current is determined with a shunt resistor in series with the load. The voltage drop of the resistor is then amplified and provided as an external measurement output. Which can be evaluated by an oscilloscope or multimeter. To allow further investigations of the power usage of the total system it is possible to include the Ethernet and CAN-Bus circuitry into the measurement.

The exact power consumption of the computational subsystem cannot be precisely measured due to the usage of the SoC as a module which includes its own power supply and peripherals. The overhead can be approximated by a 15 % efficiency loss of the switching supply and a static energy consumption of about 100 mW for powering the on-module peripherals.

III. Software Platform

The key contribution of this work is the extension of the existing Linux paradigms; combining the scheduling of software and power.

A. Actor-based Software

As a software development paradigm, real-time tasks are assumed to be actor-oriented. Furthermore, the interdependencies of the software blocks are assumed to be given by a graph structure. Each software block has its own set of requirements. Together, they provide the functionality itself. These functions can span multiple ECUs, as depicted in figure 2.

How this software paradigm can work together with the other components of the power management scheduling is described subsequently.

B. Mainline Linux

Linux tasks are distinguished into three groups and prioritized from 1 to 130. Runnable tasks are being multiplexed in a time-division manner to the computational resources. The scheduling of this multiplex depends highly on the group of tasks. Within groups, tasks with higher priority supersede tasks with lower priority. For every priority, there is one linear queue, from which runnable tasks are taken.

1) Idle Tasks: For each CPU core in the system, one idle task exists. It is a distinguished, hardware dependent task which calls up the CPU idle framework. It has the least possible priority and only gets switched to, if no other task is in a runnable state.

2) Completely Fair Tasks: The Linux default task group is completely fair. Fair meaning the CPU time of tasks featuring the same priority is kept in balance. This group can be used to simulate tasks which are not highly critical in timing–e.g. in-car entertainment.

3) Real-Time Tasks: In mainline Linux, real-time tasks are catered to soft-real-time, meaning they are guaranteed to reach a certain bandwidth, meaning CPU time per second. In order to be able to emulate hard real-time tasks, extensions to the system scheduler were implemented.

C. Power Management Scheduling

As an extension to the existing Linux scheduler, the description of the software timing relations as non-linear data structures (so-called power management plans, defined in [11]) is allowed. Power management plans are graphs relating power states with functional software units, and their timing relations. Figure 3 depicts an exemplary plan which shows a strictly cyclical schedule. The cycle is instantiated after establishing the power state α, which is necessary to run the tasks.

Derived from the hardware platform properties, the implementation distinguishes the following power states:

$$(\alpha_{\text{anyC}}), (\alpha_{\leq \text{C7}}), (\alpha_{\leq \text{C6}}), \ldots, (\alpha_{\leq \text{C1}})$$

for restricting the usable idle states as in table I.

For the system under load, there are additionally the power states

$$(\alpha_{975\,\text{mV}}), (\alpha_{1075\,\text{mV}}), (\alpha_{1200\,\text{mV}}), (\alpha_{1270\,\text{mV}}), (\alpha_{1350\,\text{mV}}),$$
$$(\alpha_{125\,\text{MHz}}), (\alpha_{250\,\text{MHz}}), (\alpha_{500\,\text{MHz}}), (\alpha_{550\,\text{MHz}}), (\alpha_{600\,\text{MHz}}),$$

which correspond to the operating points of the microprocessor unit.

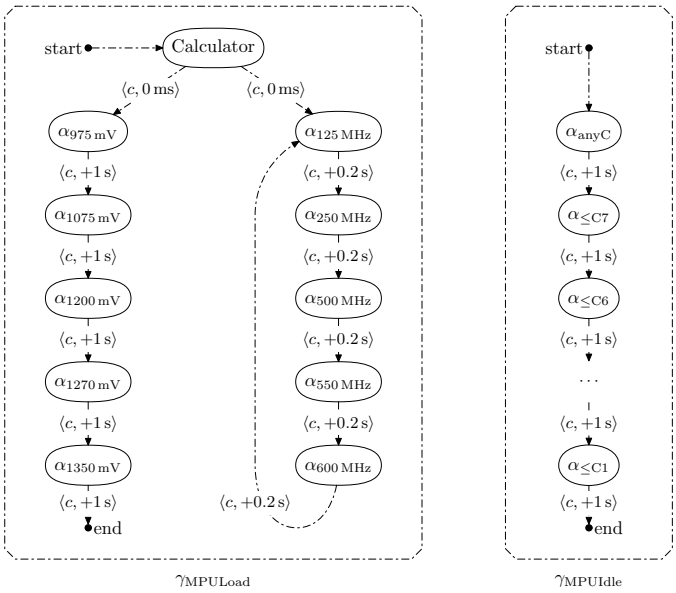


Fig. 4. Power management plans for testing both the microprocessor unit under load and idle. All supported operating points are cycled.

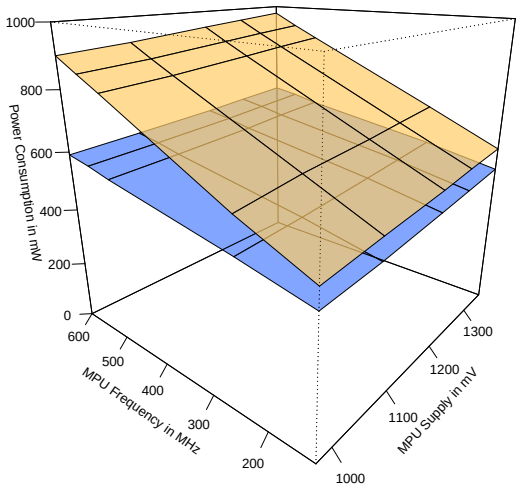


Fig. 5. Typical power consumption of MPU over frequency and supply voltage. The lower surface depicts the consumption during C1, and the higher surface the consumption during load.

D. CPU idle framework

Essential to power management of the computing system is the CPU idle framework. It is invoked from within the idle task. It allows the so-called governor to select an idle state of the hardware, which the idle driver actually sets. The idle framework had to be adapted to the new scheduling paradigm.

1) CPU idle governor: The CPU idle governor selects and reflects on the system status and the idle state to be chosen. This is done on the basis of an estimate of the idle time. It is calculated by the known next timer interrupt to occur in the system and by a measured average interrupt frequency for in- and output (I/O).

The power management mechanism restricts the available idle states and thus helps to investigate the effects of different idle state selections over time.

2) CPU idle driver: The CPU idle driver provides interfaces to query and describe possible hardware states. For the hardware platform at hand, the driver implements the power states listed in table I.

E. Device Drivers

For the CPU idle but also for CPU frequency adjustments, hardware dependent device drivers must register with their respective frameworks and configure the hardware as requested. Furthermore, Linux drivers should allow the system to enter different low idle states which involve reducing the power consumption of the peripherals, like dropping connectivity or configuring a communication hardware to do Wake-on-LAN.

IV. Test Setup

The test setup covers a single ECU and its computational subsystem. All available load and idle operating points are cycled and the power consumption measured. The measurement was done using a multimeter and an Oscilloscope with USB interface, for doing both a static and dynamic analysis.

The power management plans used during this experiment are depicted in figure 4. To the left is the test case for system load, and to the right the test case for the system idle experiment.

1) System Load: In order to put load onto the system, bc [16] was run to calculate thousands of digits of:

$$\tan^{-1}(1) = \frac{\pi}{4} \approx 0.7854.$$

During the calculation, all available combinations of microprocessor frequency and supply voltage are cycled. The frequencies are held for 0.2 s each, while the voltages are held for 1 s. In the end, every frequency and voltage combination is established.

The system ends up in the highest operating mode with 600 MHz and 1350 mV supply, in which the calculation finishes.

2) System Idle: For system idle testing, the Linux idle governor is incrementally restricted in the idle state selection phase. In the first second there is no restriction, to the next second applies the same, allowing maximally C7. Each following second, one more C state is marked as unavailable. During the test setup, the ECU was connected to an internal network and ran a set of other Linux background tasks, which both induce system wake-ups during the system idle test case.

V. Experiment Results

A. Static Results

Figure 5 depicts the static results which were gathered reading a multimeter. It shows the typical power consumption values of the computational subsystem over all possible frequency and voltage combinations. The higher plane is for the system during load, while the lower plane is during idle (C1).

The figure indicates, that with this piece of hardware, the power loss increases almost linearly with the supply voltage level.

B. Dynamic Results

The results for the system load test case are depicted in figure 6. The spike at second 0.4 marks the start of the experiment. The calculator is invoked and the first power state combination is immediately activated. The graph is divided into intervals of equal microprocessor supply voltage. Within these intervals, the different operating frequencies are cycled. After 5 seconds, the calculation finishes in the highest operating point and drops to idle afterwards.

The results for the system idle test case are depicted in figure 7. Figure 7a shows an oscillogram of the power consumption of the computational subsystem. It can be seen, that after a preparation phase, the average power consumption is about the same level during the intervals Unconstrained Idle to Max C4. This interval is characterized by decreasing spikes. These spikes are partly induced by the overhead of saving and restoring the registers to and from SRAM. The power state transitions are the more costly, the deeper the idle state is. In C2, the sporadic interrupts can be handled without incurring large spikes.

Figure 7b shows the amount of time which the computational subsystem spent in the respective power states. The numbers indicate, that during the idle test case, the system was at most 0.5 % of the time busy and thus 99.5 % idle. Furthermore, the bar chart shows, that besides C7, the governor most often selects the deepest idle state it is allowed to go.

VI. Conclusion and Outlook

In this paper, a platform for evaluating power management mechanisms is presented. Measurement results for both load and idle scenarios were presented. Future work is to combine ECUs built upon this platform into a holistic test bench, consisting of a real car body, the wiring harness, distributed electronic loads and measurement equipment. The ECUs will be interconnected via bus systems. Thereby, power management mechanisms can be prototyped and evaluated in the context of a complete car test bench setup.

In the complete car test bench, different distributed approaches to power management can be investigated. These approaches should combine both coordinating energy efficient measures but also reliability and operational safety of the automotive system. Additionally, the effect of changing the partitioning of software to hardware can be investigated.

Acknowledgment

This work is part of an interdisciplinary research project of the Technische Universitaet Muenchen and BMW Group Research and Technology. It is organized under the CAR@TUM initiative, where the decades long-standing cooperation between the Technische Universitaet Muenchen and the BMW Group is intensified and restructured.

References

[1] L. Brabetz, M. Ayeb, A. Niknejad, and J. Wang, "Architekturoptimierung und leistungsmanagement für robuste bordnetzsysteme," in *Energy Management & Wire Harness Systems*, 2011.

[2] T. Kohler, R. Gehring, J. Froeschl, D. Buecherl, and H.-G. Herzog, "Voltage Stability Analysis of Automotive Power Nets Based on Modeling and Experimental Results," in *New Trends and Developments in Automotive System Engineering*, M. Chiaberge, Ed. InTech, Rijeka, Crotia, 2011, vol. 1.

[3] T. Kohler, T. Wagner, A. Thanheiser, C. Bertram, D. Buecherl, H.-G. Herzog, J. Froeschl, and R. Gehring, "Experimental investigation on voltage stability in vehicle power nets for power distribution management," in *Vehicle Power and Propulsion Conference (VPPC), 2010 IEEE*, sept. 2010, pp. 1 –6.

[4] L. Brabetz, M. Ayeb, D. Tellmann, and J. Wang, "Smart power control and architecture for an efficient vehicle alternator - capacitor load system," in *Advanced Microsystems for Automotive Applications 2010*, ser. VDI-Buch, G. Meyer and J. Valldorf, Eds. Springer Berlin Heidelberg, 2010, pp. 51–60. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-16362-3_6

[5] "Automotive open system architecture," http://www.autosar.org/.

[6] C. Schmutzler, A. Kruger, F. Schuster, and M. Simons, "Energy efficiency in automotive networks: Assessment and concepts," in *High Performance Computing and Simulation (HPCS), 2010 International Conference on*, 2010, pp. 232–240.

[7] M. Fuchs, P. Scheer, and A. Grzemba, "Selektiver Teilnetzbetrieb im Fahrzeug: Eine Realisierung für den CAN-Bus und Adaption auf andere Bussysteme," in *AmE 2010 - Automotive meets Electronics (GMM-FB 64)*, 2010.

[8] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Transactions on very large scale integration (VLSI) systems*, vol. 8, no. 3, June 2000.

[9] W. Y. Lee, "Energy-efficient scheduling of periodic real-time tasks on lightly loaded multicore processors," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 3, pp. 530 –537, march 2012.

[10] E. Lee, "Cyber physical systems: Design challenges," in *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, May 2008, pp. 363 –369.

[11] A. Barthels, F. Ruf, G. Walla, J. Fröschl, H.-U. Michel, and U. Baumgarten, "A model for sequence based power management in cyber physical systems," in *1st International Conference on ICT as Key Technology for the Fight against Global Warming – ICT-GLOW*, 2011, pp. 87–101.

[12] GUMSTIX, "small open source hardware," http://www.gumstix.com/.

[13] "Linaro ti kernel," http://www.linaro.org/.

[14] T. Instruments, "OMAP3513/03 Applications Processor," http://www.ti.com/lit/ds/symlink/omap3503.pdf.

[15] ——, "OMAP Power Management Whitepaper," http://www.ti.com/lit/an/sprt495/sprt495.pdf.

[16] P. Nelson, "bc - an arbitrary precision calculator language," http://www.gnu.org/software/bc/.

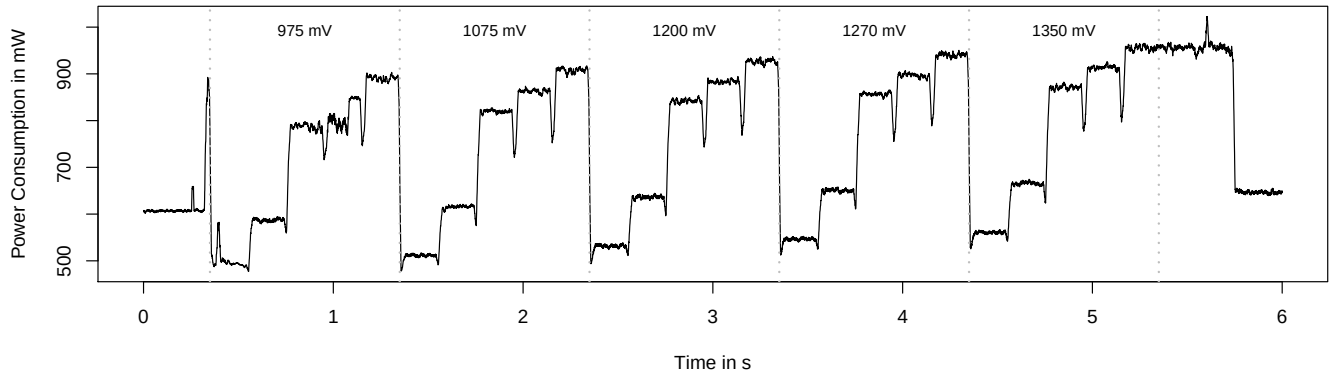
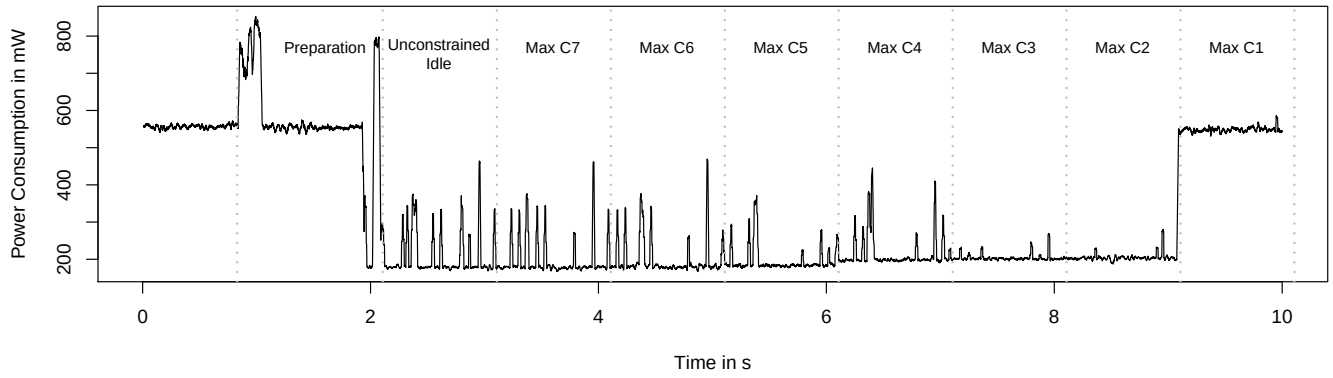
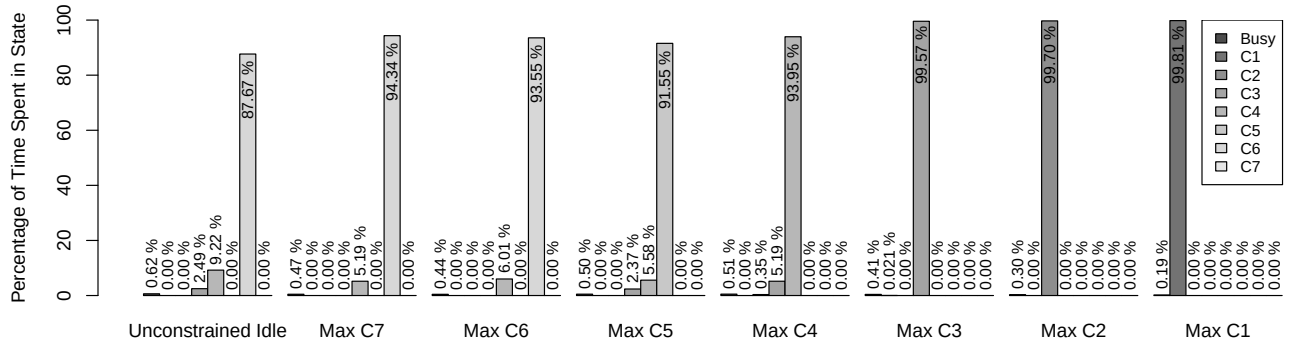


Fig. 6. Oscillogram of power consumption during system load. Low pass filter with 100 Hz. The intervals are tagged with the used supply voltage level. Within each interval, all 5 frequency operating points were used.



(a) Idle Measurements as in figure 4, low pass filter with 70 Hz. There is a significant change in power consumption from C1 to C2, as well as a small one in between C4 and C5–7.



(b) Time Spent in Idle during the Time Intervals Depicted Above

Fig. 7. The oscillogram indicates, that changing to the deeper idle states on this platform does not bring a lot of benefit. Additionally, the cost of waking up and preparing for sleep is higher (more visible spikes).