UHH Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Bachelor Thesis

Optimizing ArduPower

submitted by

Daniel Bremer

Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
Arbeitsbereich Wissenschaftliches Rechnen

Studiengang:	Informatik
Matrikelnummer:	6834136
Erstgutachter:	Dr. Michael Kuhn
Zweitgutachter:	Prof. Dr Thomas Ludwig
Betreuer:	Dr. Michael Kuhn Mohammad Reza Heidari

Hamburg, March 7, 2018

Abstract

In the recent years the performance of computation systems is not rated only by their FLOPS anymore but also by their power consumption. With the aim to design energy aware computing systems and software, monitoring systems have to be used to gather information on the power consumption. By making improvements to the existing ArduPower platform, a highly modular, efficient and easy to use internal power monitoring system is build and tested. After evaluating different aspects of power monitoring systems, the hardware design of a probe based, self-configuring system is described utilizing probes for current measurement and a shield for the Arduino Mega 2560. For increased efficiency in data transmission a sophisticated protocol is designed to ensure the highest possible sampling rates and enable detailed analysis of a system's power consumption.

Contents

1 Introduction

For a long time the performance of High Performance Computation (HPC) systems was only measured by their computational performance acquired by benchmarks and the resulting number of Floating Point Operations per Second (FLOPS) was one of the only parameters for comparison. But in the recent years another factor slowly gained influence in the rating: the power consumption. While users often only focus on higher performance and faster run times, the operators of the systems also face the rising cost of maintaining and running the system and pursue to reduce these. So a balance between performance and power consumption has to be found achieve best possible performance at a reasonable cost.

With the introduction of the Green500 list in 2013 this focus was also supported by the TOP500 list. Not only does the Green500 list use the FLOPS to rate a system's performance like the original TOP500 list, but it rates using a FLOPS/Watt metric, therefore favors energy efficient systems.

The goal to achieve higher efficiency also is pursued by the hardware manufacturers. Using new manufacturing techniques like 14 nm lithography allows to not only make the components smaller, therefore allowing more transistors and higher performance on a chip with the same size, but also reduces the power consumption. Better controlling of clock frequencies and voltages with mechanics like "Dynamic Voltage and Frequency Scaling" (DVFS) and "Dynamic Concurrency Throttling" (DCT) can optimize the energy efficiency for a given system load. The addition of new circuits and instruction set architectures like AVX (Advanced Vector Extensions) or FMA (Fused Multiply Add) allow for better concurrency of instructions on a hardware level, increasing the execution efficiency per operation [HSI$^+$15].

Energy optimization for a given HPC system unfortunately can not be done by just comparing data sheets. While it is possible to calculate the theoretical peak performance of a CPU, the real workload of a specific HPC application often does score a lower performance due to its specific calculations and instructions.

Possible applications could be mathematical calculations on huge matrices, simulation of physical environments and objects, like e. g. wind tunnels, and biological processes like simulation of protein-ligand-bindings. While physical simulations could benefit from better load distribution with more, but not from better performing processing nodes, a brute force application like ligand-simulation could benefit from better node performance. These parameters could lead to a decision that compromises on slightly worse energy efficiency but a better runtime efficiency.

To increase energy efficiency the system's performance and power consumption need to be inspected. The goal is to increase the first while reducing the latter. Tools like power

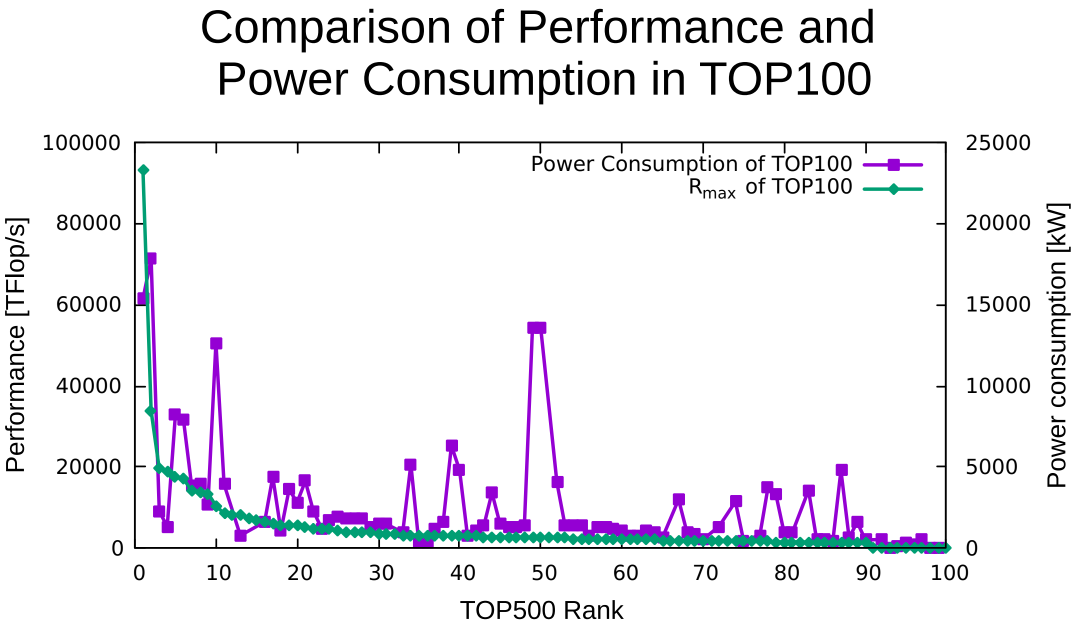


Figure 1.1: Plot of the first 100 places in the TOP500 List of November 2017 [uSG] comparing the performance and power consumption. The green plot shows the the performance values R_{max} that are used for performance comparison. R_{max} is acquired by running the High Performance Linpack benchmark. It quantifies the maximal achieved performance in a run, measured in Floating Point operations per second. A monotonically decreasing logarithmic function is expected and can be explained with Moore's Law. The purple graph shows the power consumption of each system. Here no trend can be identified as no direct relation exists between power consumption and performance.

monitoring systems allow to capture the energy used by machines and enable assessment processes in system design for use cases. Imagine a HPC system that is used mostly for a specific task like weather simulation. Although there will be times in that the system is not used for weather simulation, this application will be the main workload and therefore the system and its architecture should be optimized for that specific application.

To compare the energy efficiency of different system architectures a test system could be used and the power consumption with this specific task could be tested. Monitoring power consumption requires integration of a power monitoring system (PMS) into such a test system. Two different types of PMS's exist and can be differentiated: external and internal ones. External PMS's can monitor power consumption from the wall. A whole computation node is connected to the device and all mains voltage is delivered by the PMS. This way the power consumption of the computation nodes as a whole can be measured and a precise overall result can be inspected. Internal PMS on the other hand require preparation before the measurement and can not be done with any system. To be able to measure internally the power wiring needs to be modified, either using a measuring capable power supply unit (PSU) or adding a type of sensor to the wiring harness/power delivery circuit of the node. This requires the system to use extra power wires to each internal component, like in a consumer grade computer, and therefore does not allow measurement on systems that use power lines integrated into the mainboard.

With an internal PMS the power consumption of each internal component is being measured independently and a much more detailed, component based analysis can be performed. Thus the efficiency of each component can be compared. This enables comparisons of performance per Watt between different devices, e. g. CPUs and GPUs. The only component that is not monitored is the PSU itself, so for this an additional external PMS has to be used to gather data for its efficiency and correctness.

1.1 Content of this Thesis

Building a highly modular, efficient and easy to use internal power monitoring system was the task of this bachelor's thesis. ArduPower [DHK$^+$15], a system developed by Germán Fabregat from Universitat Jaume I and Manuel F. Dolz, Mohammad Reza Heidari and Michael Kuhn from Universität Hamburg was used as a basis and redesigned to increase its modularity and improve the usage experience, while maintaining a low cost. Also modifications on the software level were done to achieve a higher saturation of the serial interface and increase communication efficiency.

This thesis describes the ArduPower platform in its first design and after the modifications were made. The modifications were done by developing a sophisticated Arduino shield and probes to sense current flows of a computation node. To complement the hardware changes, a firmware for the Arduino was written to drive the ArduPower shield and send data to a monitoring computer with the new 6 bit protocol.

2 Fundamentals

This chapter provides all details necessary to understand the ArduPower platform. Specialized integrated circuits are used to create a cyberphysical system that connects the power distribution of a computer system to its software. The ArduPower platform consists of three components, an Arduino Mega 2560, that features a programmable processor to allow control of additional hardware, an ArduPower Arduino shield, that extends the functionalities of the Arduino platform, and probes designed for integration into the power distribution, to allow for monitoring of current flows.

2.1 The Arduino Platform

Arduino is an open source hardware and software platform [LLCb]. Originally developed by Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino and David Mellis at the Interaction Design Institute of Ivrea in 2005, it was designed to help design students to develop cyberphysical systems. Soon the open source community developed several variants featuring different Atmel processors.
Meant for beginners the usage is made easy by allowing writing code with a specialized Arduino IDE, which provides many libraries for common uses, and a dialect of the C++ programming language. Flashing the board is as easy as connecting an USB cable to the Arduino board. A pre-flashed bootloader on the Arduino device allows flashing using a standard serial connection (UART) with the standard RS-232 interface.

This project uses an Arduino Mega 2560, a microcontroller board built around the 8-bit ATmega2560 processor [LLCa]. Providing 70 programmable pins, which include 15 PWM outputs, 16 analog inputs which are connected to the processors 10-bit AD converter with an internal multiplexer and 4 serial connections, it is the biggest board in the Arduino lineup in terms of features. For programming 256 KB of flash memory can be used, whereby 8 KB are required by the bootloader.
To connect sensors and actuators to an Arduino the platform is equipped with female pin headers. As the layout of each Arduino is specified it is possible to design printed circuit boards (PCB) with these specifications. PCBs following these specifications are generally described as shields for they can be plugged directly onto the Arduino microcontroller board and expand the functionalities.

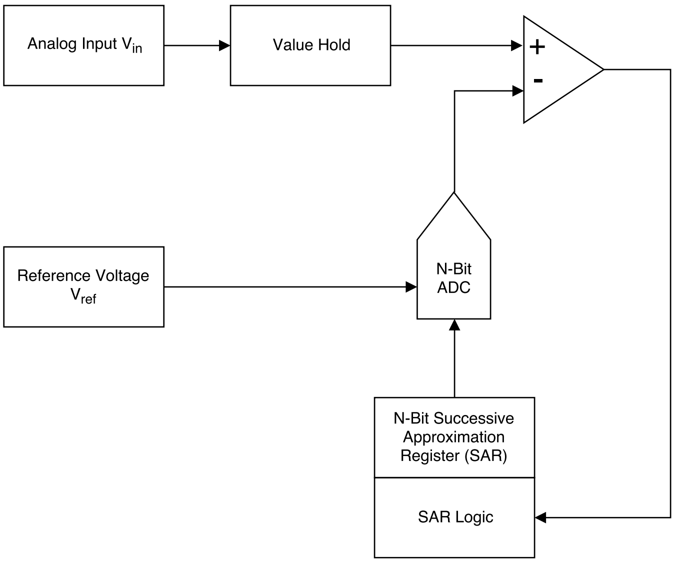

Analog Input V_{in}
Value Hold
+
-
Reference Voltage V_{ref}
N-Bit ADC
N-Bit Successive Approximation Register (SAR)
SAR Logic

Figure 2.1: A block diagram of a common Successive Approximation hardware implementation. The voltage V_{in} is compared with a voltage $\frac{V_{ref}}{2^n}$ to approximate V_{in} in a binary search like matter.

2.2 Analog Digital Conversion

While digital signals can be represented with two discrete voltages (e. g. 0 V and 5 V for LOW/HIGH states or 0/1 in binary), analog signals are represented by their voltage which can be any real value. The ATmega2560 processor, which is used on the Arduino Mega, can handle analog signals between 0 V and 5 V, where the upper bound can be customized by an external or internal reference voltage $V_{ref} \leq 5\,\text{V}$. Internal reference voltages are the supply voltage 5 V, 2.56 V and 1.1 V, while the external reference voltage can be any desired voltage up to 5 V. Note that the 5 V reference is the voltage supplied by the USB interface and USB voltages are normally unregulated and range from 4.40 V to 5.25 V, therefore are not a precise reference voltage [CHPI+00].

Analog Digital Conversion (ADC) is a process of mapping an analog voltage to a digital representation. As the ATmega2560 features a 10-bit ADC, the result is a value between 0 and 1023, allowing a precision of ($V_{ref}/1024$). The algorithm used to determine the digital representation is called "Successive Approximation". This algorithm is an effective regarding the speed-cost-trade-off, as it allows high accuracy at reasonable speeds with low hardware cost. On the other hand the speed is directly coupled to the desired accuracy, resulting in a speed penalty with higher accuracy. The implementation of this is done by a comparator based circuit like shown in figure 2.1 that is used successively to increase the precision of the approximation in each iteration.

The process is similar to a binary search. The Successive Approximation Register (SAR) is initialized with a 1 as the MSB. This produces an output of $V_{comp} = V_{ref}/2$ for the Digital Analog Converter (DAC). Now a comparison is performed whether V_{in} is bigger then the output of the DAC. In case of V_{in} being greater than the comparison voltage generated by the DAC, a 0 is written into the current bit, otherwise the 1 is kept and the next iteration starts. These steps are repeated successively until the 10 Bits of the SAR are filled and the approximation is completed.

The ATmega2560's ADC has a "Sample and Hold" circuit integrated, which ensures a constant value of the input voltage during the conversion regardless of changes at the input. This allows omitting the fact that the probes might change their output during the sampling phase of the Arduino.
Also the accuracy and speed of the ADC can be changed by oversampling and changing the clock speeds, but this is not further discussed in this work. Further information can be found in the ATmega2560 data sheet [Cor14].

2.3 Voltage Divider

A voltage divider is a simple way of mapping higher voltages to lower ones while maintaining a linear relation. This is for example required to be able to differentiate between voltages from 0 V to 12 V with a 5 V capable ADC. Using a voltage divider the voltage can be dropped by a Assuming Vcc is 12 V and $R1$ and $R2$ have a value of 330 Ω and 220 Ω the divided Voltage V_d can be calculated with the formula $V_{d,12\,\mathrm{V}} = \frac{Vcc}{R1+R2} \cdot R2 = \frac{12\,\mathrm{V}}{330\,\Omega+220\,\Omega} \cdot 220\,\Omega = 4.8\,\mathrm{V}$. Doing this with 5 V and 3.3 V gives $V_{d,5\,\mathrm{V}} = 2.0\,\mathrm{V}$ and $V_{d,3.3\,\mathrm{V}} = 1.3\,\mathrm{V}$. Like the far too big range of 12 V was mapped into the allowed 5 V range and now ca be used it with the 5 V ADC.
Note that the value of the resistors is important as it produces a current flow, therefore has to be high enough to not draw too much power.

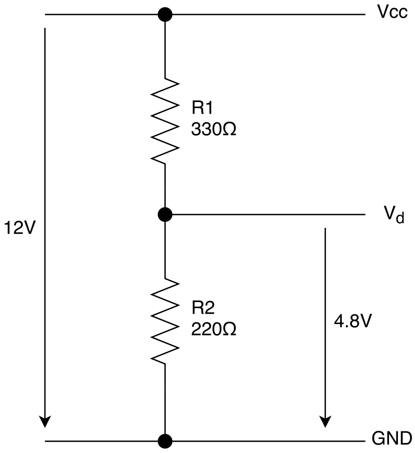


Figure 2.2: Schematics of a voltage divider built from two resistors. This shows the given example of Vcc=12 V, R1=330 Ω, R2=220 Ω. The divided voltage Vd is then used by the Arduino to determine the voltage Vcc.

2.4 Multiplexers

Multiplexers are a type of hardware switches. While a switch can be simple and just opens or closes a connection line, like transistors do, they also can be designed to switch between multiple inputs and alter the internal connections to allow the output on an output pin. Such switches are called multiplexers. Multiplexers often feature 2^N inputs, N control inputs for output determination and a single output pin. Internally the chips consist of a specific circuit, built from transistors and inverters, but being integrated into silicone they allow a smaller footprint and decomplexify the circuits needed on a PCB. While digital multiplexers can be designed easily, because they only use two voltage level, analog multiplexers need a dedicated control logic to be independent of the passed analog voltage levels.

2.5 Hall Effect Sensors

The Hall effect is an electromagnetic effect that occurs when a magnetic field traverses a conductor. Because of the Lorentz force that applies to electrons in magnetic fields the electrons are diverted perpendicular to their travel path, producing a small voltage in the conductor orthogonally to the current flow.
Hall effect sensors use these principles to measure the current flow. As figure 2.3 shows, a Hall voltage V_H can be measured perpendicular to the electrical current flow in an magnetic field. Hall sensors also use the fact that electrical current flows induce a magnetic field. This magnetic field is linear to the current flow and therefore measurement of a current flow in a conductor by using a Hall effect sensor is possible.

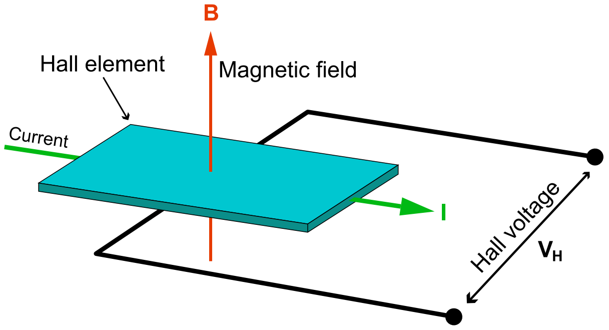


Figure 2.3: Visual representation of the Hall Effect. Electrons in the Hall element are affected by the magnetic field B and are diverted to the sides of the Hall element. Because electrons now are distributed unequally in the element a Hall voltage can be measured between both sides. This voltage is used for calculation of the current flow I.

2.6 ACS723 Hall Sensor

The ACS723 linear Hall effect sensor by Allegro MicroSystems is an integrated circuit for current measurement. The sensor uses the magnetic field induced by the current flow to produce an output voltage corresponding to the current flow. Multiple variants are available that are capable of different maximum current flows. This project uses the 20 A and 40 A variants. As it is rated to a root mean square voltage of 297 V it can be used with any voltage that might be present in current computer systems.

2.7 Wattmeters

Wattmeters are devices used for power measuring. As the name suggests they measure an electrical wattage. This can be done in two different ways by distinguishing wattmeters for computers in external and internal devices.
External wattmeters are connected in series with the mains power and the computer that is analyzed. This allows a measurement of the overall system's power draw by sampling the electrical current and multiplying the sample with the mains voltage.
Internal wattmeters on the other hand are connected with the power supply unit inside the analyzed computer. Like this every power line can be sampled individually, which allows a higher resolution of the analysis as different power consumers can be identified, e. g. the CPU or graphics cards.
The main benefit of external wattmeters is the fast setup as they just require the mains cables to be reconnected while internal wattmeters need to be integrated into the computer's power supply circuit, therefore rewiring of all power lines inside the computer is necessary. This effort is compensated by a far more precise, component based power draw measurement.

2.8 Power Distribution in Computers

To provide power to computer components a power supply is needed. Normally the voltages required for normal operating computer systems are 12 V, 5 V and 3.3 V. But not every component needs every voltage to operate, e. g. processor units and graphics units only require 12 V lines as they need more power and transform the voltage internally, while mainboards and SATA devices require all three voltages. Also different connectors are used for each component and power requirements to eliminate the risk of damage from reverse voltages. As each plug type is specified it is possible to easily identify the voltages on each line in every system.

3 ArduPower Version 1

To be able to differentiate between the states of ArduPower before and after the modifications made in this thesis they will be referred to as “Version 1” or “v1” and “Version 2” or “v2”.

The main goal of the ArduPower project was to design a cost efficient, scalable power monitoring system. While commercial systems often are quite expensive, early in the development the focus also was laid on low cost. Integration into the running software stack ought to be easily possible on every platform. Additionally a high sampling rate for a high quality analysis was to be achieved. These requirements lead to the Arduino platform, as it is a feature rich, easy to customize basis. Furthermore integration into the open source PMlib [CDKL14] was planned.

3.1 The ArduPower v1 Shield

Version 1 of ArduPower is designed by using a single shield that can be attached to an Arduino Mega. It is capable of monitoring up to 16 power lines by saturating all analog inputs on the Mega. Each analog input is directly connected to a 20 A capable ACS712 linear Hall effect sensor by Allegro.
ArduPower’s version 1 is designed as an internal wattmeter. Connection to the system is made by connecting the power lines to the orange terminals of the ArduPower unit (consisting of the ArduPower v1 Shield and the Arduino device), either in a destructive way by cutting the wires and directly attaching them them or by building an adapter. Each terminal represents a channel that is sampled. As each ACS712 sensor could sense up to 20 A a theoretical maximum of 240 W could be measured on a 12 V line. This allows for merging of lines that are wired to the same component and have the same voltage, as they are only split to allow for more flexible, thinner wires.
To report the measurements the ArduPower unit is connected to a monitoring computer, which could be the analyzed device itself, with an USB link and the built-in serial connection function by the Arduino. Each measurement is reported to the monitor computer that can use the PMlib program to capture data [CDKL14]. To increase the performance of the serial connection the sent values are combined in a custom protocol to better saturate the 8 bits of the serial connection.

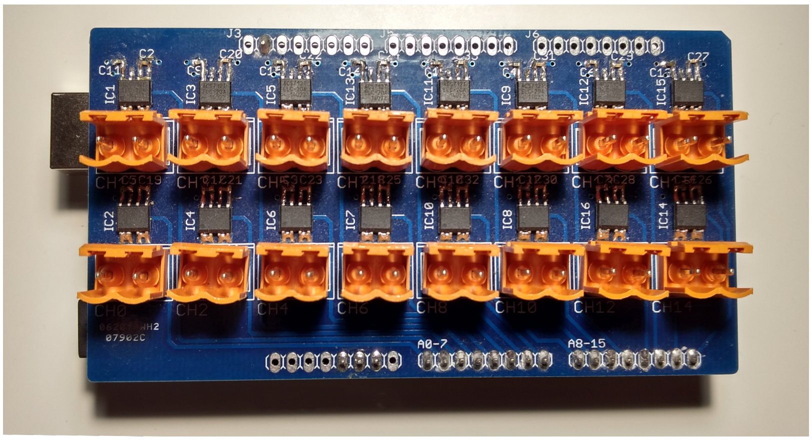


Figure 3.1: The ArduPower version 1 shield. The orange terminals feature two connection points to allow routing the existing power distribution circuit through the shield. To do this the wiring harness of a node has to be modified so a line for measurement can be connected to the ArduPower shield. This can be done by building an adapter or by cutting the line and connecting both ends of the cut wire to the shield. The PCB contains internal lines for 5 V and GND to power the ACS712 chips and a line to transfer the output voltage from the sensors to the analog input pins on the Arduino.

3.2 Protocol for Data Transmission

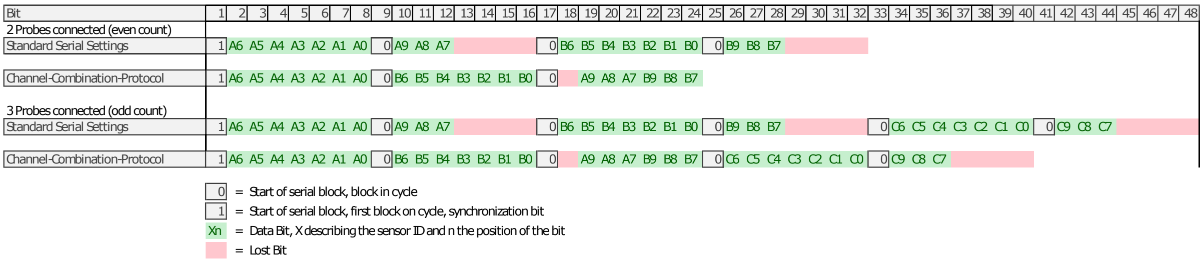
For better saturation of the serial connection a special protocol was to combine probe data.

The RS-232 serial interface, also called UART (Universal Asynchronous Receiver Transmitter), allows customizable baud rates, different data bit lengths and the use of parity. As the interface is asynchronous the baud rate has to be known by the sender and receiver. When a level change (start bit) on the receive line is detected the receiver can start monitoring the line and with the baud rate (that equals the bitrate) arrival time for each bit can be determined and each following byte of the sent packet can be received.

Data is send in packets. Packets are normally 8 bit wide, but the packet length can be customized as long as sender and receiver can handle the width. Optionally a parity bit can be used for error detection. After the optional parity bit a stop bit is send and one clock cycle of silence is appended. Using the standard parameters, that consist of 8 data bits and no parity, 11 cycles are used to send one data packet, 8 for data transmission, 2 control bits and one silence bit.

The Arduino's ADC samples voltages with an accuracy of 10 bits. Serial transmission with the standard parameters would require 2 packets, the first packet would contain 8 bits of the converted analog value, the last two bits are sent in the in the following packet. To transmit 2 values a total of 4 packets is required.

To reduce the amount of packets a protocol was designed to compress two values and only use three packets instead of four. Like shown in figure 3.2, two packets are used



Bit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
2 Probes connected (even count)																																																
Standard Serial Settings	1	A6	A5	A4	A3	A2	A1	A0	0	A9	A8	A7	🟥	🟥	🟥	🟥	0	B6	B5	B4	B3	B2	B1	B0	0	B9	B8	B7	🟥	🟥	🟥	🟥																
Channel-Combination-Protocol	1	A6	A5	A4	A3	A2	A1	A0	0	B6	B5	B4	B3	B2	B1	B0	0	🟥	A9	A8	A7	B9	B8	B7																								
3 Probes connected (odd count)																																																
Standard Serial Settings	1	A6	A5	A4	A3	A2	A1	A0	0	A9	A8	A7	🟥	🟥	🟥	🟥	0	B6	B5	B4	B3	B2	B1	B0	0	B9	B8	B7	🟥	🟥	🟥	🟥	0	C6	C5	C4	C3	C2	C1	C0	0	C9	C8	C7	🟥	🟥	🟥	🟥
Channel-Combination-Protocol	1	A6	A5	A4	A3	A2	A1	A0	0	B6	B5	B4	B3	B2	B1	B0	0	🟥	A9	A8	A7	B9	B8	B7	0	C6	C5	C4	C3	C2	C1	C0	0	C9	C8	C7	🟥	🟥	🟥	🟥								

0 = Start of serial block, block in cycle

1 = Start of serial block, first block on cycle, synchronization bit

Xn = Data Bit, X describing the sensor ID and n the position of the bit

🟥 = Lost Bit

Figure 3.2: A graphical comparison of the protocol that uses combination of different bytes to increase efficiency and sending the raw data. The red areas show lost bits. The combination of data allows for a significantly lower waste of bits, increasing the performance, as loop times are getting lower and the sampling rate increases.

and filled with the first 7 bits of the converted values acquired from the probes. The last bit in each packet is used for synchronization to identify ArduPower's measurement cycles. The third packet is filled with the remaining bits of each value, a total of 6 bits. While no protocol needs 2 packets per channel in each measurement cycle, the data combination of two channels in the third packet reduces the number to 1.5 packets for each channel. While no compression looses 4 bits per channel due to no usage, the combination only looses a maximum of $\frac{n}{2} + 3$ bits, n being the number of active channels.

3.3 Sensor Calibration

The ACS712 data sheet characterizes the Total Output Error with ±1.5%. To lower the margin of error each sensor on the shield was calibrated by using a regulated power supply. After all sensors were connected in series, a current flow was generated with an adjustable resistor. The power supply was able to display the output current that then was entered into a python script for data collection and the outputs of the sensors were collected by the program. With the collected data it is possible to calculate the slope and offset of each sensor's output by using linear regression. This data then was provided to the PMlib collector to be used for calculation of the power consumption while measurement.

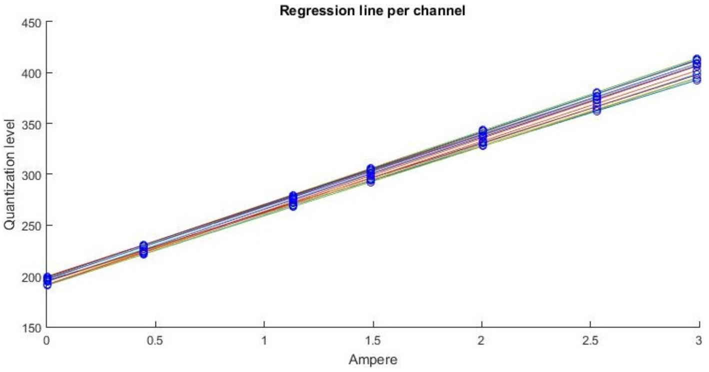


Figure 3.3: Example regression lines from the values obtained by the calibration process. A set of linear functions is expected because the ACS712 sensor is specified as a sensor with linear output.

4 ArduPower Version 2 Design Approaches

The main goals for ArduPower's version 2 were to improve the hardware and usability of its predecessor. The small form factor that allows the unit to be easily stored inside a server node and the low power consumption should be maintained while the usability and feature set should be improved. Early on two different design approaches, the "Central Box" and the "Probe" approach were identified and evaluated.

4.1 Concept of the "Central Box" Approach

The first revision of ArduPower relied on a shield that was connected with the power lines that were measured. This needed the wires to be cut and reassembled. The "Central Box" approach is based on a redesign of the shield to allow the direct connection of all desired plugs from the power supply unit of a system.

This way the ArduPower unit would consist of the Arduino Mega, the shield and an enclosure offering the standard connectors for power like an ATX plug, 6/8-pin PCI-e device power plugs, SATA plugs, a 4 or 8 pin CPU power plug or even Molex connectors. Also for each input connection an output connector is required with the opposite gender plug to allow a connection between the PSU and computer component.

Using this approach would have vastly increased the size of the ArduPower unit. Approximate sizes for standard connectors are:

- ATX power connector: 50 mm x 12 mm
- SATA power connector: 25 mm x 4 mm
- CPU power connector: 18 mm x 12 mm
- PCI-e power connector: 18 mm x 12 mm

As each connection is required at least twice the minimum size can not be lower than 22.5 cm^2 without any spacing between the connectors and about 40 cm^2 with a clearance of 2.5 mm in every dimension. For comparison the Arduino Mega has a size of 55 cm^2. Also the low distance between lines could lead to a significant noise floor, therefore distorting the Hall effect sensors.

Also the feature set of the new version of ArduPower should contain the possibility of an Automatic Configuration [see 5.3. Automatic Configuration]. This feature would have further increased the total size of the shield by a significant amount.

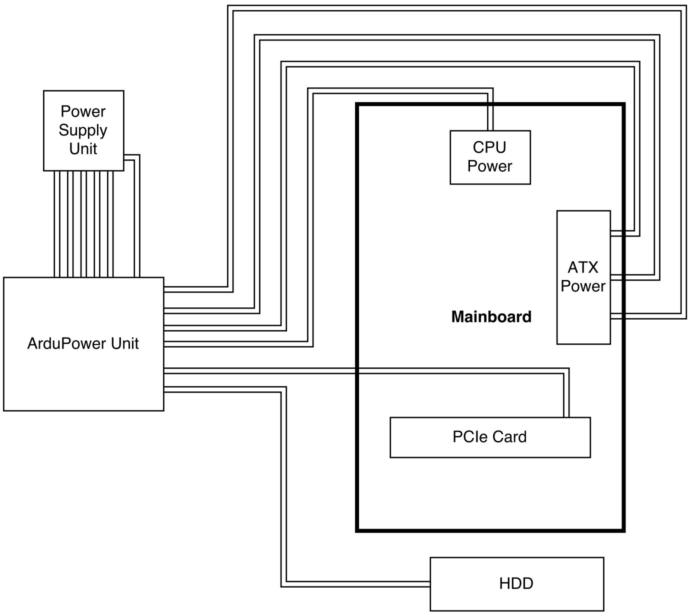


Figure 4.1: ArduPower as described with the “Central Box” approach. This is the current usage with ArduPower v1. Each power carrying line from the Power Supply Unit is routed through the ArduPower shield, allowing measurement on the lines. The GND lines do not have to be rerouted as otherwise the lines would be measured twice. Also lines negative voltage with negative voltages can be ignored.

4.2 Concept of the “Probe” Approach

As the name suggests the “Probe” approach makes use of probes. While the concept of the first revision and the “Central Box” approach were to use a shield with Hall effect sensors on the Arduino and reroute the power wiring through the ArduPower unit the “Probe” approach uses a number of small probes with the Hall effect sensors directly integrated into the power circuit.

The usage of probes allows to distribute the functions of the system. Using the “Central Box” design the ArduPower collector unit (consisting of the Arduino Mega and its shield) is in charge of measuring the Amperage on the power lines of the analyzed system, collecting the results from the Hall effect sensors by using the ADC, integrating the Automatic Configuration circuit and transmitting the values to a computer. When using probes the ArduPower unit can be split into multiple components. While it still requires a shield for the Arduino to extend its functionalities, this shield now has a reduced feature set containing only the power distribution for the probes and the Automatic Configuration circuit. All sensing is done with attachable probes. The Arduino and shield combination will be referred to as the “ArduPower collector unit”, while the probes will simply be called probes or sensors. Note that all probes contain an ACS723 sensor and some identification wiring, but as the function is mostly reduced to the function of the ACS723 Hall effect sensor the words will be used synonymously.

This function splitting has a huge impact on the accomplishment of the set goals. As the Hall effect sensors are a huge factor in the overall cost of the ArduPower platform it is now possible to reduce the number of sensors connected to the system as their number can be freely chosen. This reduces the cost of the system to the minimum, while the old approach had the fixed number of 16 probes built onto the shield. As the power consumed by the sensors is a maximum of 0.07 W with 16 sensors in use the fact of power saving can be neglected.
If ArduPower is meant to be used as a permanently integrated wattmeter the Automatic Configuration on the shield can also be omitted, only requiring a shield for power distribution.

On the downside this design increases the number of wires in the system and introduces probes into the system that slightly extend existing power lines. As each probe requires five connections for data and power and two for the Automatic Configuration each probe would introduce five wires to the system. These could be bundled to a cable with roughly the diameter of a USB 2.0 cable.

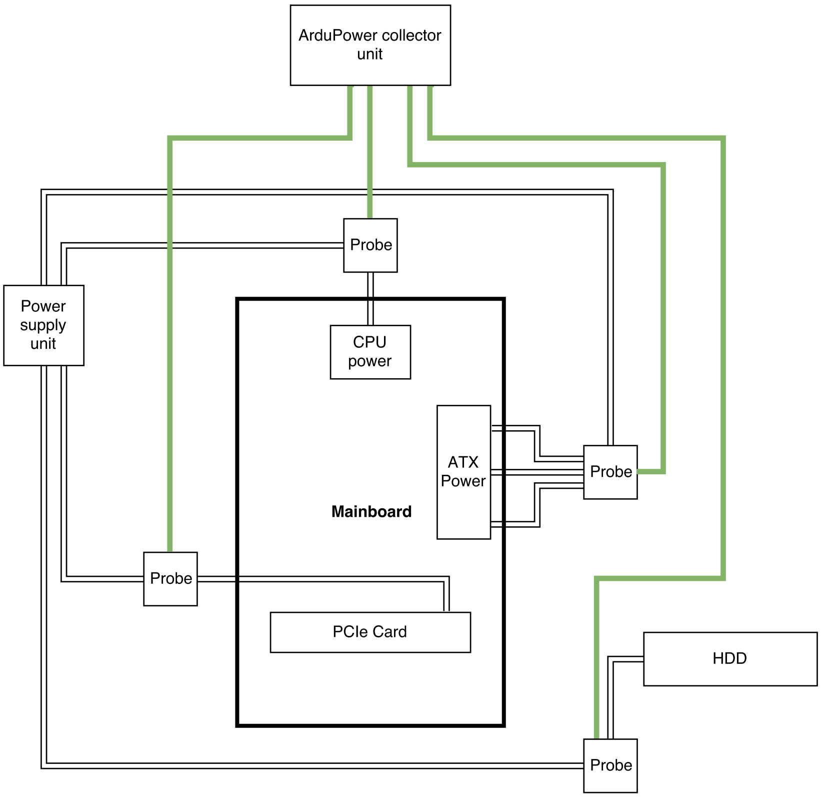


Figure 4.2: Diagram of the Probe integration into the power distribution of the analyzed node. In contrary to the "Central Box" design the wiring will be interrupted on each power transferring connection individually. This allows for selective measurement and has a smaller impact on wire routing in the node. Additionally to the other approach wires between the collector unit and the probes are required for power delivery and data transmission.

4.3 Comparison of both Design Approaches

As for modularity the “Probe” approach is clearly the one to be chosen. Not only is it possible to connect sensors specifically for the components that should be monitored, this can also be done without swapping a shield. While the “Central Box” approach requires a box with all connections needed for a set of components, the “Probe” approach can be used with any set of components for monitoring. As an example one could imagine a data center that uses ArduPower to analyze a GPU computation node. Here many PCI-e power connectors are needed. To now use ArduPower on a storage node a lot of SATA power lines have to be measured. The “Central Box” design would have to feature a sufficient number of PCI-e power connectors to allow usage in the computation node, but also enough SATA power connectors to be used on the other node, with the limit of 16 connections available. Also the whole power distribution circuit of both nodes would have to be rewired to disconnect and connect the ArduPower unit. The “Probe” approach on the other hand would just require to unplug the probes from the ArduPower collector unit connected to the GPU node and plug in the probes from the storage node onto the shield. In addition it would be possible to keep the probes connected and just disconnecting the Arduino and its shield, reducing the effort to rewire the system for another measurement.

Cost wise the “Probe” design also benefits from the high modularity, as the number of probes is freely selectable between none and fifteen (fifteen, because one analog input is needed for the 5.3. Automatic Configuration), while the “Central Box” design would have a fixed amount of sensors built in and only a dedicated connector set on the shield.
Also the size of the ArduPower collector unit can be relatively small as the probes can be distributed in the system and do not increase the size of the shield.

For the implementation the “Probe” approach was chosen, because of its superiority in terms of size and modularity.

5 Implementation

ArduPower's version 2 required modifications of the original ArduPower platform. Using the "Probe" design, probes for integration into the power distribution circuit are needed. To connect these probes to the Arduino Mega a shield is required, which also carries the dedicated hardware for the new Automatic Configuration feature. To support the new design and features a sophisticated firmware for the Arduino is needed. This firmware also needs to implement the new protocol for serial communication that uses a packet size of 6 bits instead of 8 bits.

5.1 Probes

ArduPower's version 2 requires probes to be integrated into a systems power circuit to allow for current measurement. These probes consist of a ACS723 Hall effect sensor by Allegro and a piece of PCB to allow connection for standardized computer power plugs. A probe can contain up to three sensors on a single PCB. While CPU and PCI-e power only need a 12 V line to be monitored, therefore can be built with a single sensor, power for the mainboard is delivered by 3.3 V, 5 V and 12 V lines, so a probe with three sensors, one for each voltage level, is required.

A probe is a simple, active device. The PCB is equipped with 5 pin headers per ACS723 sensor for connection to the collector unit (see figure 5.2). The connections are required for powering the sensor (5 V and GND), data transmission, for the sensor type and the voltage of the measured power line. Also connection points on the PCB are required for the male and female computer power connectors, so the power can be routed through the sensor and the probe has an adapter like appearance.

Communication between the ArduPower collector unit and the probe is done by three wires. Two wires provide configuration information like the maximum amperage measurable by the sensor used on the probe or the voltage of the analyzed system's line that is measured by the probe. The third line is used for passing the output voltage of the ACS723 sensor to the ADC on the Arduino.

The sensors are powered by the USB voltage through the Arduino, which normally is 5 V. Problems could arise as USB power is not regulated, being as low as 4.4 V [CHPI+00], while the ACS723's minimum voltages is rated 4.5 V. The output signal of the ACS723 is not affected by different USB voltages, as the zero-current-output-voltage is relative to Vcc ("Voltage common collector", positive supply voltage), so it can be measured without an error. Also the analog signal is converted by using the Arduino's 5 V level, which is offset with the same value, as the 5 V level of the sensor. If the ArduPower collector unit is connected to another machine than the analyzed, this could lead to

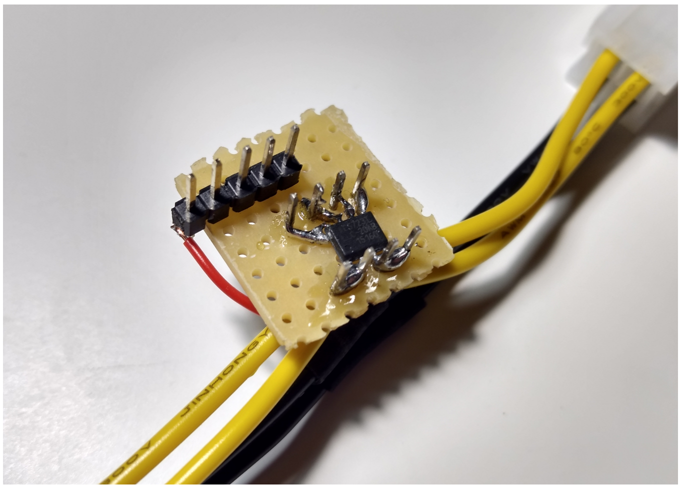


Figure 5.1: Image of a probe with a single sensor. The shown probe is used to capture current flow on a CPU power input, therefore the probe only is connected to a 12 V line. The row of pin headers is used for connection to the ArduPower collector unit.

problems, as the 5 V and GND levels in the systems may differ.

Allegro points out that external magnetic fields could introduce noise and errors to the readings of the ACS723 [DB13]. Version 1 of ArduPower was not affected by this as the device was placed away from system components with oscillators and the power lines were connected from the top, so the magnetic fields from the wires had no effect on the ACS712 sensors.

The new version brings the probes closer to the nodes components, possibly allowing interference from the generated magnetic fields. Additionally other power lines might pass the ACS723 sensor on the probes closely, introducing a noise source. If the evaluation shows a lot of noise, shielding of the sensors should be considered.

5.2 Serial Communication

To achieve higher data transmission efficiency, ArduPower version 1 used a protocol that allowed combination of two sensor output values to fit three serial packets, where each packet had a length of 8 bits. The best case performance allowed for only one unused bit in every three packets. To improve usage of the serial interface, ArduPower v2 uses a new protocol that is based on 6 data bits per serial packet.

By reconfiguring the serial connection, it is possible to shorten the length of a serial byte to 6 bits. To prevent confusion, this 6 bit byte will be referred to as a serial packet. As every value from the ADC consists of 10 bits, it is possible to split this value into two 5 bit blocks. For stability the protocol still uses the synchronization bit, adding one more bit to each serial packet. This saturates all available 12 data bits in the two packets, utilizing 100% of the data bits on the serial interface.

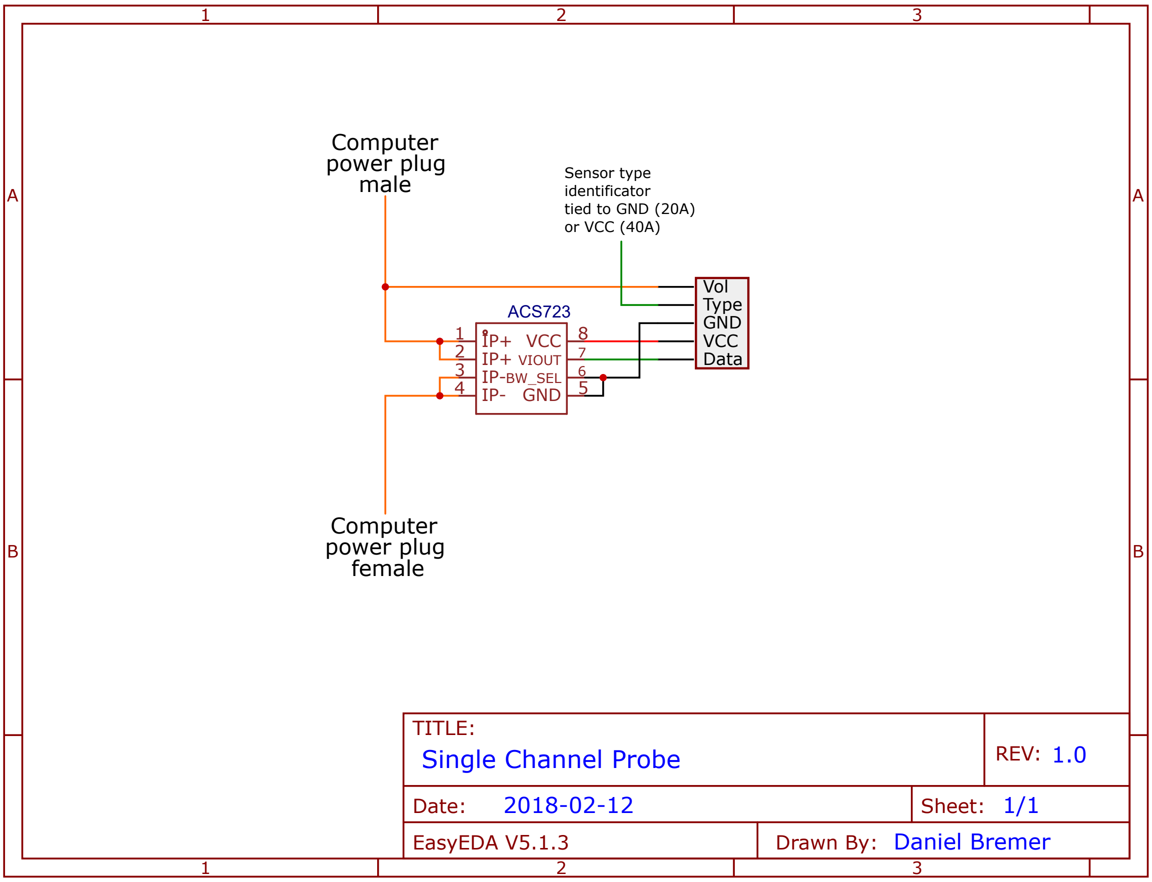
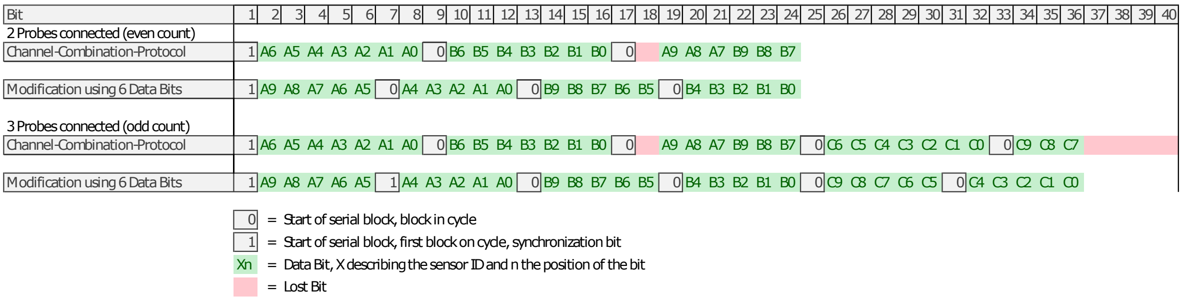


Figure 5.2: Schematic of a probe that is connected to a single voltage/line. IP+ and IP- are the terminals to connect the line that will be measured. GND and Vcc are used for powering the chip. BW_ SEL controls the sampling rate of the chip. Connecting it to ground sets the sensor in high bandwidth mode, allowing a sampling rate of 80 kHz. VIOUT outputs the sensors measured value as an analog signal. The “Vol” pin on the probe is connected to the measured line, forwarding the lines voltage to the ArduPower collector unit. Another pin named “Type” is used to provide information on the used ACS723 sensor. As a 20 A capable and a 40 A capable subtype of the chip can be used on a probe, it is required to know which sensor is used for correct calculation of the flowing current. This is done by connecting the pin to GND or VCC, depending on the sensor’s type.



Bit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2 Probes connected (even count)																				
Channel-Combination-Protocol	1	A6	A5	A4	A3	A2	A1	A0	0	B6	B5	B4	B3	B2	B1	B0	0	🟥	A9	A8
Modification using 6 Data Bits	1	A9	A8	A7	A6	A5	0	A4	A3	A2	A1	A0	0	B9	B8	B7	B6	B5	0	B4
3 Probes connected (odd count)																				
Channel-Combination-Protocol	1	A6	A5	A4	A3	A2	A1	A0	0	B6	B5	B4	B3	B2	B1	B0	0	🟥	A9	A8
Modification using 6 Data Bits	1	A9	A8	A7	A6	A5	1	A4	A3	A2	A1	A0	0	B9	B8	B7	B6	B5	0	B4

Bit	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
2 Probes connected (even count)																				
Channel-Combination-Protocol	A7	B9	B8	B7																
Modification using 6 Data Bits	B3	B2	B1	B0																
3 Probes connected (odd count)																				
Channel-Combination-Protocol	A7	B9	B8	B7	0	C6	C5	C4	C3	C2	C1	C0	0	C9	C8	C7	🟥	🟥	🟥	🟥
Modification using 6 Data Bits	B3	B2	B1	B0	0	C9	C8	C7	C6	C5	0	C4	C3	C2	C1	C0				

0 = Start of serial block, block in cycle

1 = Start of serial block, first block on cycle, synchronization bit

Xn = Data Bit, X describing the sensor ID and n the position of the bit

🟥 = Lost Bit

Figure 5.3: Graphical representation of the 6 bit protocol used in ArduPower version 2. The red areas again show the lost blocks. The 100% utilization of the protocol allows for no wasted bits. Also, when using an odd number of probes with ArduPower, it is slightly shorter in overall bit-length.

When an odd number of probes is connected to the system, this protocol also is four bits shorter, increasing the data throughput, as shown in figure 5.3. In reality, the 6 bit protocol is only three bits shorter, as in this case one more stop bit, that is used by the serial connection to mark the end of a serial packet, is required.

5.3 Automatic Configuration

Power consumption is measured in Watt and electric power is calculated by $P = U \cdot I$, P being the power consumption, U the voltage and I the amperage. To be able to calculate the power drawn by a component in the measured system the monitoring computer needs information about the connected probe. Three values need to be provided: The voltage on the measured line, the subtype of the sensor and the current.

Version 1 of ArduPower used a configuration file that had to be provided manually to provide this information. Introducing the Automatic Configuration this step handled automatically, removing the need for a configuration file for each ArduPower configuration. ArduPower's version 2 is designed to allow two different types of sensors for better accuracy on lines that consume different amounts of power, unlike version 1, which only made use of a 20 A capable sensor. While a mainboard normally has a low power consumption, a GPU can consume several hundred watts. Current generation graphics cards like NVIDIA's Titan V can consume more than 250 W of power [NVI], while a mainboard will use only a low amount of current on its 3.3 V line. The graphics card therefore needs a 40 A rated sensor, as it will not be possible to safely measure more than 240 W with a 20 A sensor otherwise. By allowing two types of sensors this problem can be solved with low effort.

As distinguishing two sensors can be done by a binary signal, the Arduino's digital pins can be used for sensor identification. If the type pin is connected to Vcc (5 V) the sensor can be determined as a 40 A capable sensor, when set to ground it is a 20 A sensor.

Normal computers use three different voltages provided by the power supply: 3.3 V, 5 V and 12 V. While 3.3 V and 5 V are mostly used for driving low powered chips, providing common voltage levels for hardware level communication or powering peripherals, the 12 V lines deliver power to components that require lots of power. Because the Arduino platform can just differentiate two voltages on its digital input pins, 0 V (GND) or 5 V, this introduces a barrier for the Automatic Configuration, as an analog digital converter pin is required to determine the voltage. Since it is not possible to connect two inputs to a single pin directly without interference a solution has to be found.

A switch consisting of two transistors could be connected to each analog input, allowing switching between the Automatic Configuration voltage and the probe's data voltage. As transistors produce a resistance and need a current on the base for switching this would increase the power consumption and also make the shield design more complex by introducing 32 new components. A more simple approach using an analog multiplexer was chosen.

An analog multiplexer works like a n-ary switch, allowing to use 2^i inputs with one output. The decision which channel is put out is made with i control inputs in a binary counter fashion. To keep the circuit simple one analog channel is permanently occupied by the Automatic Configuration, reducing the possible amount of attachable sensors to 15.

An important component in the circuit is the voltage divider. The Arduino's ADC is not capable of measuring more than 5 V and will be damaged otherwise. A voltage divider is used to map the voltages into the possible 5 V range. The resulting voltage levels then can be read and the original voltage can be identified.

After these operations the voltage of the measured line, the used subtype of sensor and the current flow are known and the wattage can be calculated.

5.3.1 Hardware Implementation

The Arduino platform is only capable of handling voltages up to 5.5 V without damaging any components. As computers use 12 V lines to provide much power at reasonable currents it is not possible to connect all lines directly to the ArduPower collector unit. To map all voltages into the 5 V range a voltage divider with the values $R1 = 330\,\Omega$ and $R2 = 220\,\Omega$ was used. Like this it is possible to reduce the voltages {3.3 V, 5 V, 12 V} to {1.3 V, 2.0 V, 4.8 V} which allows measurement on the Arduino's ADC. The threshold values defined for identification of each voltage should allow for a range of ±0.2 V, as voltages between different PSUs may vary due to the voltage specifications defining ranges for the standard voltages [Cor13]. Also a slightly lower voltage has to be expected as the used multiplexer adds a resistance of 90 Ω. So the expected voltages are {1.13 V, 1.72 V, 4.13 V} for the corresponding voltages {3.3 V, 5 V, 12 V}.

As there are three voltages, an identification cannot be done with a binary signal using one wire and no additional device for sending the ID. Voltage sensing on the other hand requires the Arduino's ADC which also is required for reading the output of the sensors. So a permanent connection of all the line voltage to an ADC pin is not possible, because the ADC inputs are needed to read the current flow from the sensors.

Two implementations to solve this problem were possible: a transistor based and a multiplexer based.

The transistor based design would have required two complementary transistors (N-channel and P-channel MOSFET transistor or NPN and PNP transistor) and a digital pin for switching. Like this an exclusive switch on each ADC input could have been realized that either would have passed the voltage of the probed line or the output of the probe representing the current to the input. This design would have increased the complexity of the overall design as it would have introduced many transistors, more digital outputs of the Arduino Mega would have been saturated and each switch also would have required a separate voltage divider.
A multiplexer, being a N-to-one switch, allows to quickly switch between 16 inputs and therefore the whole Automatic Configuration can be handled with one ADC pin. Also only one voltage divider, for voltage mapping, and four digital pins, to control the multiplexers output, are required. As the probe's output cannot be connected to the Arduino while the Automatic Configuration is ran, the ADC pin A15 is handled as a designated Automatic Configuration pin, reducing the number of connectable sensors to 15 sensors. The goal of low cost was impacted by this decision as a 16-to-1 multiplexer is far more expensive than the corresponding design with complementary switches made from individual transistors, but for this case the focus was laid on a simple circuit to allow for easy modification and customization.

An important "limitation" of the design is given by the power supply for the multiplexer. The used ADG406BNZ is designed to be powered with a voltage that is at least the highest switched voltage, so a Vcc of 12 V is required. This is can be achieved by connecting a probe that is connected to a 12 V line to the first channel of the ArduPower device, because the voltage sensing input is also connected to the Vcc pin of the multiplexer on the shield. Probes with a 12 V voltage can be found on the CPU line, any PCI-e line or the ATX plug that is connected to the mainboard.

5.3.2 Software Implementation

The Automatic Configuration feature requires a specialized routine to configure the Arduino. This routine can be used to achieve two things: it can be used to sense the voltage of a line that a probe is connected to and it also allows to count the connected probes to optimize the sensing loop.
The ADG406BNZ multiplexer requires four control inputs, that are used for channel selection, and an enable-input pin, that switches the multiplexer on or off. The control inputs are used in a binary counter way. If every control input is switched to low (0000 in binary representation) channel 1 is connected to the output, if a binary 1 (0001) is present on the control inputs channel 2 is switched, and so on. As only 15 switch inputs of the ADG406BNZ are used one channel will be unused, in the current design this is channel 1.
Counting of active probes can be done by sensing the analyzed system's line voltage of each probe's input. If the voltage is 0 V, no probe is connected and the input can be skipped in the sampling loop. For this an array with all active inputs is stored in the memory. Like this only the ADC pins that have a probe connected will be read. This improves the sampling loop time as the analog digital conversion is a limiting factor

since it is quite slow compared to all other actions performed by the processor in the ArduPower firmware.
To acquire the type of sensor a digital pin is read, as the type of sensor can be described with a digital signal of HIGH or LOW. The type of sensor is only read if the probe is connected and another array with indexes corresponding to the active probe array is held to allow quick determination of the sensor type.

5.4 ArduPower Shield

ArduPower consists of three elements: An Arduino, an ArduPower shield for the Arduino and probes that are connected to the ArduPower shield.
Because the pin layout of the Arduino does not allow connection of multiple wires in a side by side manner, a shield was designed to allow for convenient connection terminals to the Arduino. With this all wires from the sensor, providing power to the sensor and data to the collector unit, are being placed topographically close on the shield and it is easy to connect a probe to the ArduPower collector unit. The shield then uses internal wiring to route the data to the pins on the Arduino that process the data. To expand the capabilities of the Arduino with the Automatic Configuration circuit (see 5.3.1 Hardware Implementation), the shield was equipped with the ADG406BNZ multiplexer and a voltage divider.
For easy debugging and on the fly status checking a RGB LED was implemented to show the status of the ArduPower device.

A schematic of the shield can be found in the appendix, page 50. The shield contains 5 rails for connection of probes. These rails are labeled “5 V rail”, “Voltage identify inputs”, “Sensor type inputs”, “GND rail” and “Sensor data inputs”. The 5 V rail and GND rail supply power to the probes and are connected to the VCC and GND pins on the probes and the Arduino, the remaining rails are used as inputs to gather data from the probes. The “Sensor type inputs” rail is connected to the probes’ “Type” pins. This probe pin outputs either GND or VCC, allowing determination of the ACS723-sensor on the probe. The rail is connected directly to the ATmega2560’s digital pins on the Arduino device and the numbers on the rail in the schematic represent the Arduino’s input pins that the rail pin is connected to. The non-deterministic order comes due to the fact that errors while building the prototype were made and is corrected with the `sensor_type_input_pin` array in the firmware, that provides the pin order on the shield. The Arduino then can use the voltage levels on the pins to determine whether the connected probe is equipped with a 20 A capable sensor or if it is rated for measurement of up to 40 A.
The “Sensor data inputs” rails is connected to the Arduino’s analog inputs. The input is used to forward the ACS723 sensor’s output from the “Data” pin to the Arduino so it can be converted to a digital value and the flowing current can be calculated.
The remaining “Voltage identity inputs” rail is connected to the probe’s “Vol” pin. It feeds the analyzed line’s voltages to the ADG406BNZ multiplexer whose output D is connected to a voltage divider. By controlling the output when setting the control pins

A0, A1, A2 and A3 to HIGH or LOW, all inputs can be mapped to the analog pin A15 on the Arduino. The value of the input gets mapped to a measurable range by the voltage divider to prevent damage of the ATmega2560 chip. Pin 42 of the Arduino is connected to the enable-pin EN on the ADG406BNZ for on and off switching.
The Pins 24, 26, 28 and 30 on the Arduino are used to drive a RGB LED for optical status representation.

The rails are placed side by side on the shield, in the following order:

Sensor data inputs	VCC	GND	Sensor type inputs	Voltage identify inputs

The "Sensor data inputs" rail is plugged directly into the Arduino's analog inputs A0-A15. This allows convenient connection of the probes to the ArduPower collector unit.

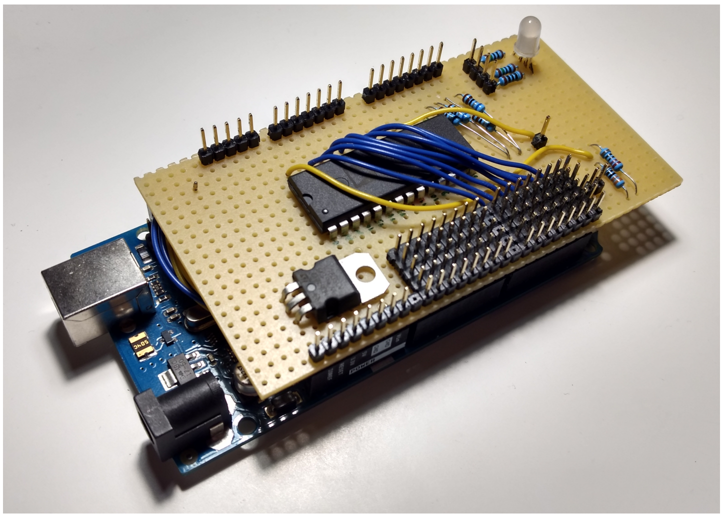


Figure 5.4: Image of the ArduPower collector unit. The chip in the middle of the shield is the ADG406BNZ multiplexer, used for the Automatic Configuration. The large area equipped with pins is used for probe connection to the collector unit. Behind that, the two voltage divider resistors can be seen. The lines of pins on the side of the LED, are not used for any connection with external devices, they are used to connect the shield with the Arduino's input headers. The wires seen on the top transfer the voltage on the analyzed system's lines for the Automatic Configuration to the multiplexer's pins. Further wires and soldered connections can be found on the bottom side of the shield.

5.5 Firmware

Firmware describes software that is embedded into a device's hardware and supports its specific functionality. Writing firmware for Arduino devices can be done by using an IDE that supports the C++ dialect which is used by the e. g. avr-gcc compiler.

ArduPower's firmware was designed in the Arduino IDE and uses the provided standard libraries.
Programming for embedded systems often is done in a loop. As flash memory in these devices is limited, a sequence of code is written and repeated. To alter the program flow internal and external events can be used. For convenience a setup routine often is implemented to allow for one time configuration of pins, sensors, etc. at boot time.
A basic Arduino program also implements these routines, named `setup()` and `loop()`.
ArduPower's setup routine is used to initialize all pins, setting their mode to input/output and their default states, initializing the serial connection and running the Automatic Configuration.
The loop of the firmware consists of only a single `if` conditions to check if the measurement is enabled. Incoming serial data is handled by the `serialEvent()` function that automatically is called by an interrupt when data arrives. This allows to have a clean `loop()` that is dedicated only to the measurement process. If the measurement is enabled the `collectAndSendData()` routine is called that collects the data of each sensor and sends it to the monitoring computer.
For the full, annotated source code, please see page 51 in the appendix.

5.5.1 Controlling ArduPower

ArduPower can be controlled by sending characters via the serial connection. Five commands can be used for basic control of the measurement process as shown in Table 5.1. With these commands the measurement can be started or stopped, the Automatic Configuration can be triggered after a physical reconfiguration of the device and the information of the Automatic Configuration can be returned to the monitoring computer. Sending the character '2' will return the current configuration state, gathered by the Automatic Configuration, which will consist of of the probe count and three additional

Character	**Function**	**Response**
0	Disables the measurement routine by setting measurement to 0	0x00
1	Enables the measurement routine by setting measurement to 1	0x01
2	Sends the sensor information gathered by the Automatic Configuration	
3	Disables the measurement and calls the Automatic Configuration routine for reconfiguration of ArduPower	0x03
4	Executes a single measurement cycle and sends the data	
any other	Ignored	

Table 5.1: Table of all commands that can be used to control ArduPower.

values per connected probe. The data will be transmitted probe wise, so the first value will be the pin that the probe is connected to, minus an offset of 54, to be able to fit the value into 6 bits (value range [0,14]). The second value will represent the type of the ACS723 sensor on the probe (values {0,1} for {20 A,40 A} sensors), while the third serial packet contains the voltage of the line that the sensor is connected to (values {1, 2, 3} representing {12 V, 5 V, 3.3 V}). It is required to re run the Automatic Configuration after every configuration change on ArduPower (connection of new probes or removal of probes). Also the configuration has to be read, so the power consumption can be calculated correctly.

The LED on the ArduPower shield also provides state information. A green LED signals a running measurement, a red LED is shown during the standby status (no measurement running).

5.5.2 Measurement Process and Data acquisition

The measurement process is done by using the Analog Digital Converter of the ATmega2560. The provided analog value of each connected sensor is read, split, to fit the 6 bit serial blocks and sent by writing it to the serial connection.

The measurement process was designed simple to be very fast. The `loop()` consists of only a single if and the check whether measurement is enabled. When enabled the `collectAndSendData()` function is called. It uses a `for` loop to iterate over each connected probe and starts an analog digital conversion to read the value that the ACS723 sensors outputs.

The next steps consist of bit shifting to split the 10 bit integer returned from the conversion. The shift is a 5 bit right-shift and the result is stored in a temporary variable. These five bits are the high-bits of the 10 bit integer, the sixth bit is the synchronization bit and can be set to 1 for synchronization in each first packet of an iteration. After this the data, the high-bits in the temporary variable and the remaining low bits, is sent to the monitoring device with two `Serial.write()` calls. The low bits are masked directly in the function call, setting every bit except the last five bits to zero.

Data acquisition on the monitoring device is done in reverse. The first received value is stored in a variable and the synchronization bit gets set to zero. After receiving the second value, the original value can be restored by left-shifting the first received bits, so the high-bits are in the correct position again, and OR-ing with the remaining low-bits. Before any data is received for accumulation, a synchronization loop is implemented. This loop checks the incoming serial packets for the synchronization bit and ensures that the first block of a measurement iteration is used as a starting point. After one synchronization loop reading from the serial interface can be done in a infinite loop by rechecking the synchronization bit on every iteration start and dropping iterations with bad synchronization bits.

The monitoring device now has the result from the analog digital conversion and needs to calculate the current in the line. This is done by calculating $I = 10\,\mathrm{A} \cdot \left(\frac{\text{analogRead value} \cdot 5\,\mathrm{V}}{1024\,\mathrm{V}} - 2.5\right)$ with a 20 A capable sensor. For 40 A capable sensors the formula changes, as the output signal changes from 100 $\frac{\text{mV}}{\text{A}}$ to 50 $\frac{\text{mV}}{\text{A}}$, the current is now calculated by $I = 20\,\text{A} \cdot \left(\frac{\text{analogRead value}\cdot 5\,\text{V}}{1024\,\text{V}} - 2.5\right)$. The type of sensor is acquired from reading the configuration of ArduPower. To calculate the power consumption, the current has to be multiplied with the voltage on the corresponding line.

For a annotated code reference for data collection on a monitoring device, please see page 57 in the appendix.

5.5.3 Support for Monitoring Suites

Monitoring Suites are software suites that implement monitoring functionalities for services and hardware devices. To enable monitoring a status providing service on the monitored device is required. Such a service could be a simple script that sends data, e. g. the system's load or RAM usage, to the monitoring suite. Communication and data exchange can be based on autonomous sending of data in a specified frequency or by a polling message from the data collection service.
To keep the performance impact low, the frequency of data accumulation often is lower than 10 Hz.
An example monitoring suite for HPC systems is Diamond [Pub], a python daemon for system metric collection.
Because of the low frequency of data reading, an extra function was implemented to the ArduPower firmware. Sending multiple hundreds of samples per second would lead to a higher system load, due to serial interrupts. Also, most of the received data would be discarded, as the receive buffer just would overflow, overwriting unread data.
To complement this, a single set of data can be acquired from ArduPower, by sending a '4' character. This triggers a single loop of probe reading and sending the data.
With this function a polling script can be written to poll ArduPower for new power consumption data, only when necessary, supporting low impact, low time resolution system monitoring.

5.6 Sensor Calibration

ArduPower v2 omits the sensor calibration. Using version 1 it was observed that the slopes and offsets were within the sensors specified 1-2% range. Because of this small impact the calibration was discarded, as there would be very hard to perform the measurements required for the calibration with a hardware circuit that would fit an Arduino shield. Doing a manual calibration the Automatic Configuration would have to read the slope and offset values from the probe. This would require a storage chip on the probe and increase the cost massively.

6 Evaluation of ArduPower v2

This thesis made changes to ArduPower on the software and hardware level to achieve better usability and performance. To evaluate the changes both levels will be inspected independently. The hardware design of the shield and the probes will be tested and compared against the first version of ArduPower, which already was evaluated with an external LMG450 Power Analyzer by ZES Zimmer [Zim]. To determine the efficiency and speed of the 6 bit packet size and the new protocol, a comparison with the old protocol using 8 bits per packet will be drawn.

6.1 Hardware Evaluation

For the tests a single computation node was used, equipped with two Intel Xeon X5560 processors and a total of 8 cores with deactivated hyper-threading. Each processor's Thermal Design Power (TDP) is rated at 95 W [Cor] resulting in a total power draw of up to 190 W. The real power consumption can be higher, because Intel's Turbo Boost feature is active. Load generation was done by using the High Performance Linpack (HPL) [PCWDC08] on all cores, which is a benchmark used to assess a systems peak performance. The tested node allowed for interception of the two CPU power inputs and the mainboard power supply connection. While ArduPower version 1 is connected to the system with all 16 Hall effect sensors, only three probes, two CPU probes and one ATX-connector mainboard, with a total of five ACS723 sensors were connected to version 2, improving the sampling rate, as less analog-digital-conversions had to be made to acquire data.

Figure 6.1 and figure 6.2 show a HPL run over a timespan of approximately 11 min. Both graphs show a sawtooth wave. In the beginning the initialization phase can be recognized.

While both graphs have a similar course, they show slightly different results. ArduPower version 1 has sampled 55000 samples, whereas version 2 sampled more than 720000 data points in the same time. This is due to the different number of analog-digital-conversions each platform has to conduct. As version 1 has all analog inputs saturated, version 2 only is connected to 3 probes with a total of 5 sensors. While ArduPower's version 1 shows power consumption in the range of 150 W to 250 W, version 2 shows higher peaks of up to 267 W and falls down into valleys of less than 100 W.

These valleys seem to be normal to the HPL, as they occur in each phase of high power consumption. Comparing the results to the analysis made in [SGFC09, Fig. 4b] with the PowerPack wattmeter, these valleys are natural and can be explained with the communication phases in the HPL. As the benchmark is ran on a single system

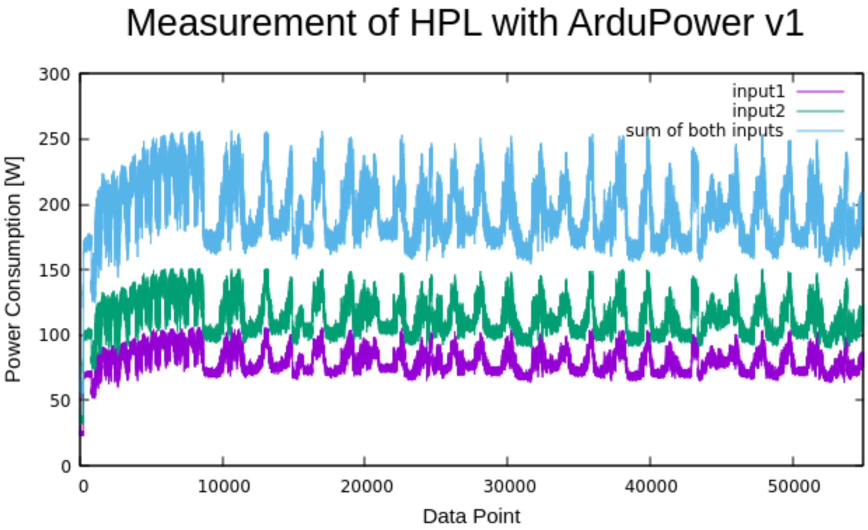


Figure 6.1: Measurement of the HPL with the ArduPower v1 device. Input 1 and Input 2 plot the measurements for both CPU inputs, while the sum shows the overall power consumption of both CPUs.

these phases are very short an can not be captured by ArduPower v1, as it has a lower sampling rate.

When comparing with the LMG450 Power Analyzer a peak wattage of 416 W is measured, while valleys have a value of about 286 W, which is a difference of 130 W. Being an external wattmeter, the values are captured from the mains voltage that powers the power supply of the node, so it includes the power consumption of the whole system and normally should be higher than results from an internal wattmeter due to losses while converting the voltages in the PSU. Like ArduPower's v1 the LMG450 device has a lower sampling rate than version 2 of ArduPower, only capturing 10700 data points over the whole run. The difference between peak power draw and lowest power consumption when the HPL was running captured with ArduPower v1 is 104 W. Ignoring the very low peaks in the measurement of v2, the difference is similar with approximately 134 W.

In conclusion the data captured with ArduPower v2 provides a better representation of the power consumption in the system because of the higher sampling rate. This allows a more detailed analysis of an application's power consumption, enabling the possibility for energy usage optimization. In addition the values can be classified to be correct as both comparison devices show similar power consumptions and value ranges. Differences in measurements are the valleys caused by the communication in the HPL run that are only captured with version 2 of ArduPower. This is due to being the only device with a sampling rate that is high enough to measure in the time span of the shared memory communication of MPI.

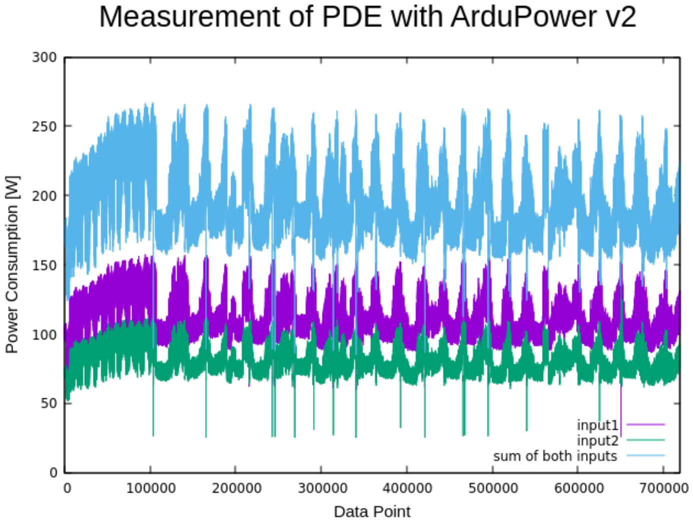


Figure 6.2: Measurement of the HPL with the ArduPower v2 device. Input 1 and Input 2 plot the measurements for both CPU inputs, while the sum shows the overall power consumption of both CPUs. The asymmetrical power draw of both CPUs is due to 4 pin plugs on the power inputs, which results in this behavior of power distribution. Using 8 pin plugs eliminates this effect.

6.2 Protocol Evaluation

To improve serial communication and achieve a higher saturation of the serial interface, a protocol that uses a packet size of six bits was implemented. This reduces the number of unused bits to none, allowing to save bits and increase the sampling rate.

To examine the impact of the changes, 500000 data transmissions were simulated. To further analyze the impact of the number of connected probes, the tests were conducted with an even and an odd number of probes, as the benefits of the new protocol should be higher when using an odd number of probes.
To test an even number of probes, two probes were simulated, and for an odd count one more probe was added. This allows to compare the results, as higher counts of probes will increase the timing differences linearly.

#Probes	Time with 8 Bits per Packet [s]	Time with 6 Bits per Packet [s]
2	127.736	136.202
3	212.633	204.287

It can be seen that the speed of both protocols depends on the used amount of probes. Using 8 bits per packet is faster with a even amount of probes, when the overall transmitted number of data bits is the same. On the other hand, when ArduPower is used with an odd number of probes, the 6 bit variant is faster.
This effect can be explained with the stop bits that mark the end of a serial packet. While with an even number of probes the amount of data bits is the same with both

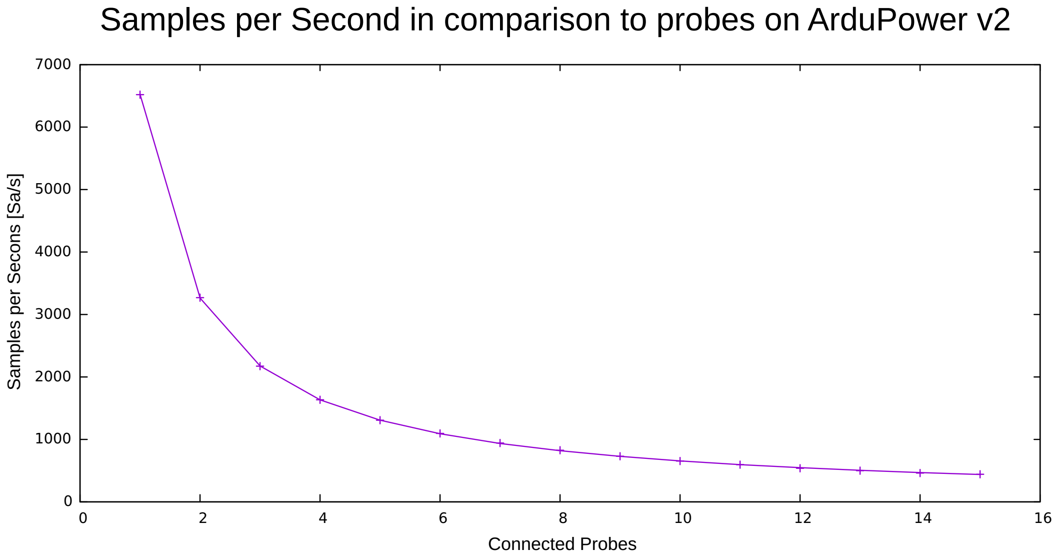


Figure 6.3: Comparison of the average sampling rate versus the number of probes connected to ArduPower v2. The sampling rate is highest with one connected probe at 6517 Sa/s, falling to 433 Sa/s when 14 probes are connected.

packet sizes, one more stop bit is send with a packet size of 6 bit, as one more packet is used to transmit the data. This makes the overall data from one measurement iteration one bit longer. On the other hand, when using an odd count of probes, the 6 bit packet size requires one additional stop bit, but saves four data bits per iteration.

Comparing the sampling rates of both ArduPower versions, v2 manages to achieve a rate of 6517 Sa/s (Samples per second) with one connected probe, bottoming out at 425 Sa/s. This is both better and worse than the version 1 sampling rates. A low number of probes allows for a higher sampling rate than version 1, which only achieves about 5900 Sa/s at maximum. With a higher amount of probes, version 2 looses its lead, only sampling with 433 Sa/s, while v1 still manages a rate of 480 Sa/s.

6.3 Evaluation

While both versions of ArduPower allow high sampling rates, the performance of the 6-bit protocol seems to suffer when many probes are connected to the system in comparison to the used 8-bit combination protocol of version 1.

The results of version 2 seem to be noisy. This noise could be a result of other wires being close to the sensor. These wires could be the ground wires of each connector, that are close to the bottom side of the probe boards. It also is possible that other power wires are close to the top side and the ACS723 sensor, as there is no top cover on the probe to prohibit such. Impacts of magnetic fields induced by the nodes components can be eliminated, as the probes were placed several centimeters away from the mainboard. Closing the node's cover and forcing the probes to be close to the node's hardware, as it would be in a real world use case, could increase noise even further.

On the other hand, the noise could be virtual and explained with the high sampling rate of ArduPower v2. Further tests with other PMS's with high sampling rates have to be conducted to see whether the measurements are affected by external magnetic fields or not.

As proposed by Allegro, shielding the ACS723 sensors could be tried to reduce these impacts.

7 Related Work

Power Monitoring is a topic for many HPC providers. While commercial power monitoring systems exists, their capabilities may not be enough for a detailed analysis of a specific application. Often the possibility for customization is wanted. While systems for global monitoring are available, capturing the power consumption of a whole node or reading the CPU's returned values if supported, a finer resolution, down to a component based level is wished. Moreover the system often is only used temporarily, so flexibility is desired to allow for unmeasured runs or even runs with low and high sampling rates. Furthermore the time resolution is often low, integrated IPMI functions to allow power consumption monitoring often only have low sampling rates of <10 Sa/s.

One of the most recent developed power monitoring systems is the High Definition Energy Efficiency Monitoring (HDEEM) project, a cooperation between the Technische Universität Dresden and Bull S. A. S. [HIS+14]. The goal was to implement a highly accurate power monitoring system with a high sampling rate based of existing IPMI circuits. The results are processed in a way to allow integration into existing analysis tools like VampirTrace [STIH11] or the Score-P [KRM+12].
Knobloch et al. did a similar project with IBM's Power7 processors which feature integrated circuits to monitor the system's power consumption on a component level. [KFH+14]

PowerPack by Ge et al. is an internal power monitoring system that support for dynamic power management configurations [GFS+10]. It provides component based power monitoring in a system by introducing a dedicated data acquisition system. In its latest version it supports multicore processors and allows for analysis of performance changes from DVFS changes.

Design-wise PowerInsight, a project by Scandia National Laboratories and Penguin Computing [LPD13], is pretty close to ArduPower v2. The PowerInsight system also is a probe based monitoring system, connected to a BeagleBone for data collection and forwarding, providing more than 1 kSa/s from user space. The design not only allows the connection of standard power plugs, but also implements PCI-e risers to monitor PCI-e devices like GPUs or accelerator cards.

Using power monitoring systems allows for the creation of power-performance profiles for HPC applications. Song et al. used the PowerPack system for a detailed power analysis of the HPC Challenge benchmarks [SGFC09]. G. Da Costa and J.-M. Pierson did similar experiments with the NAS Parallel Benchmark with the aim to identify which specific test of the benchmark suite was ran only by its power-performance profile [**?**]. The ability to identify running programs could as well lead to side channel attack on applications that use cryptography [KJJR11]. This allows attacks against DES and AES which can be "cracked" by monitoring the power consumption. By inspecting the changes over time the key used for decryption can be recovered.

To enable power constrains in applications Marathe et al. introduce Conductor, a system for intelligent power distribution [MBL+15] at run-time. It aims to distribute power in HPC systems to enable power delivery balancing and enforces power bounds.

8 Conclusion

The goal of this thesis was to improve ArduPower to achieve better usability and improve the modularity, both while maintaining its low cost and high performance. Furthermore a new protocol for data transmission was implemented to increase the saturation of the serial interface and increase communication speed.
Introducing the Automatic Configuration eliminated the need for setting up the device manually, allowing to read the configuration state from the ArduPower device itself. This enables on the fly usage and fast reconfiguration.
The implemented probe based design allows fast reconfiguration of the device. Unlike in ArduPower's first version, the power consumption of components can now be captured by plugging a probe directly between the power delivery circuit and the component without the need to reroute power lines to the ArduPower device. Testing different component configurations does also no longer require rerouting of wires, probes simply can be disconnected or connected. This also allows for measurements with higher sampling rates as the number of connected probes directly corresponds to the possible rate.
To further increase the usability the Automatic Configuration feature was implemented, removing the need for manual configuration of the platform. This enables plug and play in every system that does not use a power delivery circuit integrated into the mainboard. On the downside one measurement channel had to be sacrificed to enable this feature.
The new protocol and 6 bit packet size in the serial connection both increase the maximum sampling rate, enabling a more detailed analysis. The optimization only has positive effects when using an odd number of probes in the new system, using an even number decreases the performance and makes it slower than the old protocol with 8 data bits.
Comparing the cost of both systems the modularity allows for a significant cost reduction. The computation node used for evaluation only allowed capturing two CPU power inputs and the input on the mainboard. While the shield used in version 1 still would have been equipped with 16 ACS723 sensors, version 2 now only has a total of 5 sensors on all probes. With the addition of the Automatic Configuration the cost of the ArduPower collector unit slightly increased, because the ADG406BNZ multiplexer is needed to implement this feature.

Overall every goal was accomplished, the modularity and Automatic Configuration both increase the usability of the ArduPower platform and allow for modularity. Not only were the costs kept on a low level, most cases even allowed for cost reductions (when not using more than 10 probes per collector unit). Only the modifications to the communication between the ArduPower collector unit and the monitoring computer deliver mixed results: While an improvement was made when using an odd number of probes, the performance is worse with an even amount of probes, but it allows for a higher maximum sampling rate, whereby only one sensor can be used.

9 Further Considerations

This chapter lists further modifications that could be made to the ArduPower platform by modifying the hardware and software. It also discusses why the Automatic Configuration could be overengineered.

9.1 Improvement on Hardware Level

The most simple improvement for the hardware design that massively benefits the usability would be swapping the used pin headers on the probes and the shield for reverse-proof plugs. Currently it is easily possible to destroy the Arduino by accidentally plugging a 12 V wire into the ADC input. With the use of special plugs this can be prevented.

To improve cost effectiveness a redesign of the shield could be done to become a standalone device, omitting the need for an Arduino Mega. It would allow to reduce the size, as a dedicated printed circuit board could be designed, which could include terminals for a processor and a SMD version of the used ADG406BNZ multiplexer. Like this a specialized "Arduino" could be designed, which only allows software customization. This on the other hand would eliminate every possibility of expanding the platform with additional shields and functions.

The prototypes are built with prototyping methods that include manual soldering on perfboard. A dedicated printed circuit board could be designed that contains all lines, replacing a design based on solder traces and wires. This would allow easy reproduction as only surface mounted components as the ACS723 sensors, the ADG406BNZ multiplexer or pin-headers need to be soldered onto the PCBs.

Building an autonomous device would require two further modifications. While a connection with a control device has to exist, this connection could be used only for control signals. This would allow to use ArduPower without performance penalties in self-monitoring systems. To achieve this the measured data has to be saved for later analysis. The Arduino Mega's capabilities for data storage are quite limited, only having a size of several kB. Storage could be added with SD cards, which are cheap and can be used with the Arduino libraries. This however would slow down the measurement cycle, decreasing the performance of ArduPower.

As can be seen in the Evaluation the ArduPower system suffers from noise in the system. The probe design currently connects the BW_SEL pin on the ACS723 sensors to a GND level, setting them into high bandwidth mode. Changing the connection to Vcc would result in a lower sampling frequency of 20 kHz which is less affected by magnetic noise and still is more than the ATmega2560 could sample with its ADC. Also shielding the ACS723 sensors with a ferro-magnetic material could be tested, as suggested by Allegro.

Most wattmeters do not allow to be used internal and external simultaneously. By designing a special probe this limitation can be broken, making ArduPower a wattmeter capable of monitoring the power consumption of a whole node, including the PSU. This is discussed in detail in 9.3.

9.2 Improvement on Software Level

In the current design ArduPower relies on a serial connection with a system to transmit the collected data. Further ArduPower's original use case is constructed to be used by a computation node that monitors itself. This raises the question of what impact the monitoring process has on the running application.

The RS-232 interface was designed in the 1960s. As many technologies from such early times the serial interface relied on processor interrupt to be processed in time. This rivals with the goal of optimization in high performance computing that aims to reduce these interrupts as they require the processor to switch contexts which is a slow operation. Removing the ability to transmit data with a serial connection would require a storage medium that could be used to save the data while a measurement is running and that could be read afterwards. Adding a SD card to the ArduPower unit could solve this problem but would reduce the sampling rates on the other hand.

Using an external computer for monitoring could be another solution for this problem, but would require special software for integration of captured data into other monitoring and tracing environments like Score-P.

9.3 Discussion on the Automatic Configuration

A goal of ArduPower v2 was better usability. To achieve this the Automatic Configuration feature was implemented by adding a specialized circuit to the existing platform. This circuit was designed to allow sensing the computers different voltages that are connected to the probes and omit the need to maintain a configuration file. While this allows easy and fast use of the system, it introduces a new, complex component to the system. When thinking about power consumption in different system parts, it becomes obvious that most components that require much power use the 12 V line to receive much power while keeping the amperage on a low level. The components then use internal, specialized converters to reduce the voltage to the desired level, that often is between 3.3 V and 1.1 V.

Inspecting the plugs on a standard computer power supply, it become obvious that 3.3 V lines are only used on mass storage and the mainboard, which often do not require much power at all. This raises the question if the measurement of 3.3 V lines is necessary in most cases.

In spite of this the Automatic Configuration could be used in a not yet explored use case. While this thesis only focused on the aspect of ArduPower being an internal power monitoring system, it could be altered to also work as an external power monitoring system. This modification can be made with low effort by connecting mains voltage to a probe. This allows to have a fully monitored system with the possibility to calculate the electrical efficiency of the system. The used ACS723 Hall effect sensors could be used in these probes, as the basic isolation of the sensor is rated to a root mean square of 297 V, with a peak voltage of 420 V. The probes PCB has to be altered to avoid a breakdown by using prototyping hardware like perfboard.

Bibliography

[CDKL14] Konstantinos Chasapis, Manuel Dolz, Michael Kuhn, and Thomas Ludwig. Evaluating Power-Performace Benefits of Data Compression in HPC Storage Servers. In Steffen Fries and Petre Dini, editors, *IARIA Conference*, pages 29–34. IARIA XPS Press, 04 2014.

[CHPI$^+$00] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, and Philips. *Universal Serial Bus Specification*, 04 2000.

[Cor] Intel Corp. Intel Xeon Processor X5560 Product Specifications. `https://ark.intel.com/products/37109/Intel-Xeon-Processor-X5560-8M-Cache-2_80-GHz-6_40-GTs-Intel-QPI`. [Online; accessed 14th February, 2018].

[Cor13] Intel Corporation. *Power Supply Design Guide for Desktop Platform Form Factors Revision 1.31*, 04 2013.

[Cor14] Atmel Corporation. *8-bit Atmel Microcontroller with 16/ 32/64KB In-System Programmable Flash Datasheet*, 02 2014.

[DB13] Richard Dickinson and William Bentley. *Managing External Magnetic Field Interference When Using ACS71x Current Sensor ICs*, 2013.

[DHK$^+$15] M. F. Dolz, M. R. Heidari, M. Kuhn, T. Ludwig, and G. Fabregat. Ardupower: A low-cost wattmeter to improve energy efficiency of hpc applications. In *2015 Sixth International Green and Sustainable Computing Conference (IGSC)*, pages 1–8, Dec 2015.

[GFS$^+$10] R. Ge, X. Feng, S. Song, H. C. Chang, D. Li, and K. W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel and Distributed Systems*, 21(5):658–671, May 2010.

[HIS$^+$14] D. Hackenberg, T. Ilsche, J. Schuchart, R. Schöne, W. E. Nagel, M. Simon, and Y. Georgiou. Hdeem: High definition energy efficiency monitoring. In *2014 Energy Efficient Supercomputing Workshop*, pages 1–10, Nov 2014.

[HSI$^+$15] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer. An energy efficiency feature survey of the intel haswell processor. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 896–904, May 2015.

[KFH+14] Michael Knobloch, Maciej Foszczynski, Willi Homberg, Dirk Pleiter, and Hans Böttiger. Mapping fine-grained power measurements to hpc application runtime characteristics on ibm power7. *Computer Science - Research and Development*, 29(3):211–219, Aug 2014.

[KJJR11] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, Apr 2011.

[KRM+12] Andreas Knüpfer, Christian Rössel, Dieter an Mey, Scott Biersdorff, Kai Diethelm, Dominic Eschweiler, Markus Geimer, Michael Gerndt, Daniel Lorenz, Allen Malony, Wolfgang E. Nagel, Yury Oleynik, Peter Philippen, Pavel Saviankou, Dirk Schmidl, Sameer Shende, Ronny Tschüter, Michael Wagner, Bert Wesarg, and Felix Wolf. Score-p: A joint performance measurement run-time infrastructure for periscope,scalasca, tau, and vampir. In Holger Brunst, Matthias S. Müller, Wolfgang E. Nagel, and Michael M. Resch, editors, *Tools for High Performance Computing 2011*, pages 79–91, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[LLCa] Arduino LLC. Arduino Mega 2560 website. `https://store.arduino.cc/arduino-mega-2560-rev3`.

[LLCb] Arduino LLC. Arduino Website. `https://www.arduino.cc/`.

[LPD13] J. H. Laros, P. Pokorny, and D. DeBonis. Powerinsight - a commodity power measurement capability. In *2013 International Green Computing Conference Proceedings*, pages 1–6, June 2013.

[MBL+15] Aniruddha Marathe, Peter E. Bailey, David K. Lowenthal, Barry Rountree, Martin Schulz, and Bronis R. de Supinski. A run-time system for power-constrained hpc applications. In Julian M. Kunkel and Thomas Ludwig, editors, *High Performance Computing*, pages 394–408, Cham, 2015. Springer International Publishing.

[NVI] NVIDIA. NVIDIA TITAN V product page. `https://www.nvidia.com/en-us/titan/titan-v/`.

[PCWDC08] Antoine Petitet, R C. Whaley, Jack Dongarra, and A Cleary. Hpl – a portable implementation of the high-performance linpack benchmark for distributed-memory computers. *no journal*, 01 2008.

[Pub] Diamond Publishers. Diamond metric colelction suite. `https://github.com/python-diamond/Diamond`.

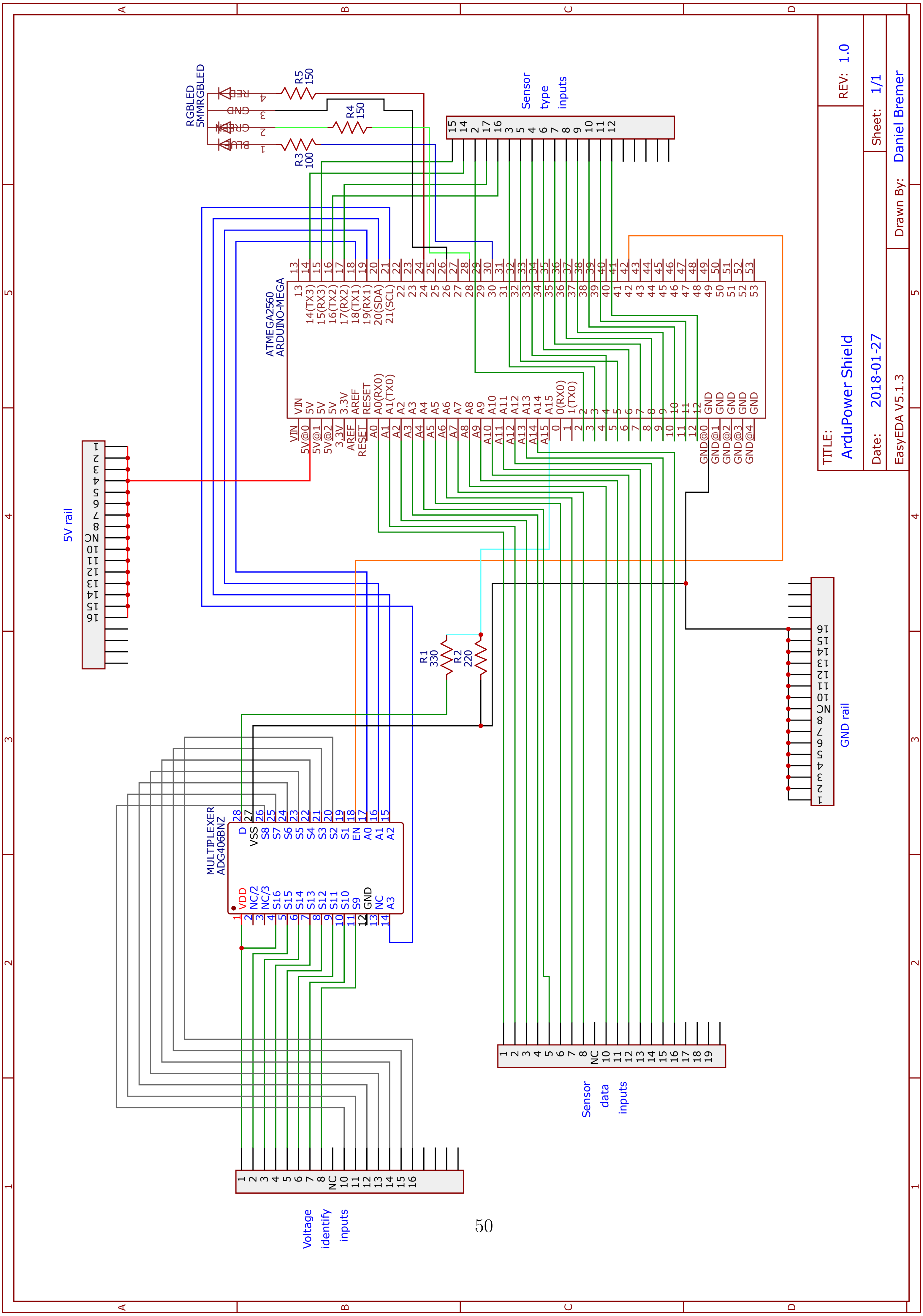
[SGFC09] Shuaiwen Song, Rong Ge, Xizhou Feng, and Kirk W. Cameron. Energy profiling and analysis of the hpc challenge benchmarks. *Int. J. High Perform. Comput. Appl.*, 23(3):265–276, aug 2009.

[STIH11] Robert Schöne, Ronny Tschüter, Thomas Ilsche, and Daniel Hackenberg. The vampirtrace plugin counter interface: Introduction and examples. In Mario R. Guarracino, Frédéric Vivien, Jesper Larsson Träff, Mario Cannatoro, Marco Danelutto, Anders Hast, Francesca Perla, Andreas Knüpfer, Beniamino Di Martino, and Michael Alexander, editors, *Euro-Par 2010 Parallel Processing Workshops*, pages 501–511, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[uSG] PROMETEUS Professor Meuer Technologieberatung und Services GmbH. TOP500 List of November 2017. `https://www.top500.org/lists/2017/11/`. [Online; accessed 14th February, 2018].

[Zim] ZES Zimmer. LMG450 4 Channel Power Analyzer. `https://www.zes.com/en/Products/Precision-Power-Analyzers/LMG450`.

Appendices


TITLE:
ArduPower Shield
Date:
2018-01-27
EasyEDA V5.1.3
REV:
1.0
Sheet:
1/1
Drawn By:
Daniel Bremer
ATMEGA2560
ARDUINO-MEGA
MULTIPLEXER
ADG406BNZ
RGBLED
5MMRGBLED
RED
GND
GREEN
BLUE
R1
330
R2
220
R3
100
R4
150
R5
150
5V rail
GND rail
Sensor type inputs
Sensor data inputs
Voltage identify inputs
VIN
5V
3.3V
AREF
RESET
GND
VDD
VSS
EN
D
NC

Codes

ArduPower v2 Firmware

```c
/*
 *  ArduPower v2 Firmware
 *  Code by Daniel Guenther Bremer
 *  Part of bachelor's thesis "Optimizing ArduPower"
 */

#define ANALOG_PINS 16

// Declaration of pins
// All possible pins for type-sensing (ordered)
int sensor_type_input_pin[] = {12, 11, 10, 9, 8, 7, 6, 4, 5, 3, 16, 17, 2, 14, 15};
int sensor_type[16];
int sensor_pin[] = {A14, A13, A12, A11, A10, A9, A8, A7, A6, A5, A4, A3, A2, A1, A0};
// Probe voltage ADC pin in Automatic Configuration
int voltage_sens_pin = A15;
// Multiplexer (MUX) enable pin
int E_mux_pin = 42;
// MUX channel selection pins
int A_pin[] = {18, 19, 20, 21};
// Array that contains list of connected Probes, their types and voltages
int* active_sensors = (int *) malloc((ANALOG_PINS-1)*sizeof(int));
int* active_sensors_types = (int *) malloc((ANALOG_PINS-1)*sizeof(int));
int* active_sensors_voltages = (int *) malloc((ANALOG_PINS-1)*sizeof(int));
// Pins for the RGB-LED
int r = 22, g = 26, b = 28;
int readData;
int sendcontainer = 0x00;
// #Connected_Probes
```

```
int sensor_pin_count = sizeof(sensor_type_input_pin) / sizeof(int);
// Measurement enable/disable var; 0 == false, 1 == true
short measure = 0;
// Receiving buffer
char recv_msg;

/*
 * Function ran once at the program start to set up the device and variables
 * Used to prepare all pins and start the serial connection
 */
void setup() {
  Serial.begin(115200, SERIAL_6N1);
  setPinMode(sensor_pin, ANALOG_PINS, INPUT);
  setPinMode(A_pin, 4, OUTPUT);             // setup channel sel
  pinMode(r, OUTPUT);                          // setup LED
  pinMode(b, OUTPUT);                          // setup LED
  pinMode(g, OUTPUT);                          // setup LED
  pinMode(24, OUTPUT);                         // setup LED
  digitalWrite(24, LOW);                       // setup LED
  pinMode(E_mux_pin, OUTPUT);           // setup MUX enable
  digitalWrite(E_mux_pin, LOW);             // Disable the MUX
  // Read initial configuration
  sensorConfiguration(active_sensors, active_sensors_types, active_sensors_voltages);
  digitalWrite(r, HIGH);
  digitalWrite(g, LOW);
  digitalWrite(b, LOW);
}

/*
 * Main loop
 * Provide data to monitor if measurement is running
 */
void loop() {
  if (measure == 1) {
    collectAndSendData();
  }
}

/*
 * Handling for arriving serial data
 * Triggered automatically by an interrupt
 * Reads data, modifies the control variable measure depending
 * on the input and or handles funciton calls
 */
void serialEvent() {
  recv_msg = Serial.read();
  switch (recv_msg) {
    case '4':                  // Send a set of data once
      measure = 0;
      collectAndSendData();
      break;
    case '2':                  // Send probe configuration
      measure = 0;
      Serial.write(sensor_pin_count);
      for (int i = 0; i < sensor_pin_count; i++) {
        Serial.write(active_sensors_types[i]);
        Serial.write(active_sensors_voltages[i]);
        //min(active_sensors) = A0 = 54 => mapping to 0-14
        Serial.write(active_sensors[i]-54);
      }
      break;
    case '1':                   // start measurement
      Serial.write(1);          // send feedback
      measure = 1;              // enable measurement
      digitalWrite(r, LOW);     // switch LED to green
      digitalWrite(g, HIGH);
      break;
    case '0':                   // abort measurement
      Serial.write(0);          // send feedback
      measure = 0;              // disable measurement
      digitalWrite(g, LOW);     // switch LED to red
      digitalWrite(r, HIGH);
      break;
    case '3':                 // rerun Automatic Configuration
      measure = 0;
      sensorConfiguration(active_sensors,
         ↪ active_sensors_types, active_sensors_voltages);
      Serial.write(3);          // send feedback
      break;
    default:                    // do nothing if undef
      break;
  }
}
```

```
/*
 * Helper function to set an array of pins to a desired mode
 */
void setPinMode(int pins[], int len, int mode) {
  for (int i = 0; i < len; i++) {
    pinMode(pins[i], mode);
    if (mode == 1) {
      digitalWrite(pins[i], LOW);
    }
  }
}

/*
 * The Automatic Configuration function
 * Read information of the probes
 * Parameters are global accessible memory to save data
 */
void sensorConfiguration(int* sensorptr, int* typeptr, int*
   ↪ voltageptr) {
  int areadVal = 0;
  int active[ANALOG_PINS];
  int types[ANALOG_PINS];
  int voltages[ANALOG_PINS];
  int active_count = 0;

  //Enable the MUX
  digitalWrite(E_mux_pin, HIGH);

  //Set sensor_type_input_pins to INPUT_PULLUP
  setPinMode(sensor_type_input_pin, sensor_pin_count,
     ↪ INPUT_PULLUP);

  for (int i = 0; i < ANALOG_PINS; i++) {
    setMuxChannel(i+1);          // offset of +1 because of
       ↪ shield-design, MUX-Channel 1 is not used
    areadVal = analogRead(voltage_sens_pin);  // read voltage
       ↪ of probed line
    if (areadVal > 716) {           // Voltage > 3,5V => 12V
      voltages[active_count] = 1;
      active[active_count] = sensor_pin[i];
    } else if (areadVal > 306) {    // Voltage > 1,5V => 5V
      voltages[active_count] = 2;
      active[active_count] = sensor_pin[i];
    } else if (areadVal > 184) {    // Voltage > 0,9V => 3,3V
```

```
      voltages[active_count] = 3;
      active[active_count] = sensor_pin[i];
    } else {                          // Voltage 0V => NC
      continue;                       // don't mark active
    }

    // Read the sensor type
    if (digitalRead(sensor_type_input_pin[i]) == LOW) { //sensor_type-line connected to ground => 20A sensor
      types[active_count] = 1;
    } else {                    //sensor_type-line connected to 5V => 40A sensor
      types[active_count] = 0;
    }
    active_count++;
  }

  // reset to INPUT to prevent leak currents
  setPinMode(sensor_type_input_pin, sensor_pin_count, INPUT);

  // disable the MUX
  digitalWrite(E_mux_pin, LOW);

  // Save the results
  sensor_pin_count = active_count;
  //truncate the tmp arrays and save data
  for(int i = 0; i < active_count; i++) {
    sensorptr[i] = active[i];
    typeptr[i] = types[i];
    voltageptr[i] = voltages[i];
  }
}

/*
 * Control the multiplexer, write the desired channel to the control pins
 */
void setMuxChannel(int channel) {
  digitalWrite(A_pin[0], (channel & (1 << 0)) == 1 ? HIGH : LOW);
  digitalWrite(A_pin[1], (channel & (1 << 1)) == 2 ? HIGH : LOW);
  digitalWrite(A_pin[2], (channel & (1 << 2)) == 4 ? HIGH : LOW);
  digitalWrite(A_pin[3], (channel & (1 << 3)) == 8 ? HIGH : LOW);
}

/*
 * Read the output of every connected probe and send the result to the monitoring computer
 */
void collectAndSendData() {
  for(int i = 0; i < sensor_pin_count; i++) {
    readData = analogRead(active_sensors[i]);  //Read probe's output
    sendcontainer = (readData>>5) & 0b011111;  //store HIGH bits
    if(i == 0) {                               //set SYNC bit
      sendcontainer |= 0b100000;
    }
    Serial.write(sendcontainer);               //write HIGH bits
    Serial.write(readData & 0b011111);         //write LOW bits
  }
}
```

Reference Python Script for Benchmarking and Data Collection

```
import serial
import struct
import time
import signal
import sys

"""
 *  ArduPower v2 Reference Data Collector and Benchmarking Code
 *  Code by Daniel Guenther Bremer
 *  Part of bachelor's thesis "Optimizing ArduPower"
"""


# global variable for the serial interface
ser = None

# function to handle quitting the program with Ctrl+C
# closes the serial connection, stops the measurement and resets the input buffer
def ctrlc_handler(sigcode, frame):
    global ser
    print('Aborting!')
    ser.write(struct.pack("!b", 0x30))
    ser.flush()
    ser.reset_input_buffer()
    ser.close()
    print('Stopped measurement and closed Serial Connection!')

def collector():
    global ser
    voltages = []
    types = []
    channels = []

    try:        # open the serial connection
        ser = serial.Serial(port='/dev/ttyACM1',
                            baudrate=115200,
                            bytesize=serial.SIXBITS,
                            timeout=None)
    except serial.SerialException, e:
        print('Serial Connection Problem: %r' % e)

    time.sleep(3)           # wait for Arduino reboot
    ser.write(struct.pack("!b", 0x32))  # send '2' to get config
    print("reading configuration")
    tmp = struct.unpack("!b", ser.read(1))  # read the count of connected probes
    n_probes = tmp[0]
    print("n_probes = " + str(n_probes))

    volt = 0
    for i in range(n_probes):  # read the probe configurations
        tmp = struct.unpack("!b", ser.read(1))
        types.append(tmp[0])
        tmp = struct.unpack("!b", ser.read(1))
        if tmp[0] is 1:
            volt = 12;
        elif tmp[0] is 2:
            volt = 5;
        elif tmp[0] is 3:
            volt = 3.3
        else:
            print('Unknown voltage on probe ' + str(i))
            volt = 0
        voltages.append(volt)
        tmp = struct.unpack("!b", ser.read(1))
        channels.append(tmp[0])

    for a in range(n_probes):    # print configuration
        print('Channel '+str(channels[a])+'@'+str(voltages[a])+'V, type '+str(types[a]))
    time.sleep(5)
    ser.write(struct.pack("!b", 0x31))  # enable measurement

    var = True
    value = 0x0000
    data = 0x00

    # Start sync, search for 1 in MSB
    while var:
        tmp = struct.unpack("!b", ser.read(1))
        data = tmp[0]
        if data & 0b100000 is 0b100000:
            print('Synched - starting data collection\n')
            var = False
            # ignore first data set
            for i in range(n_probes-1):
                tmp = struct.unpack("!b", ser.read(1))
        else:
            print('Searching for next sync bit - dropping
               ↪ block\n')
            continue

    index = 0

    while True:   # start reading loop
        sys.stdout.write(str(time.time())+';'+str(index)+';')
        index = index+1
        for i in range(n_probes):   # read a whole iteration
           ↪ of data collection
            tmp = struct.unpack("!b", ser.read(1))
            if i == 0 and tmp[0] & 0b100000 != 0b100000:
                # Connection out of sync!
                print("Out of sync!")
                while True:   # read until synched again
                    tmp = struct.unpack("!b", ser.read(1))
                    if tmp[0] & 0b100000:
                        # Found sync
                        break;

            data = tmp[0] & 0b011111   # save HIGH bits
            tmp = struct.unpack("!b", ser.read(1))
            value = (data << 5) | tmp[0]  # recover value
            if sys.argv[2] is 'r':  # print the raw
               ↪ ADC-results
                sys.stdout.write(str(value)+';')
                if i == (n_probes-1):
                    sys.stdout.write('\n')
                else:
                    pass
            elif sys.argv[2] is 'p':  # print the power
               ↪ consumption values
                if types[i] is 1:     # calculate power
                   ↪ consumption for 20A sensors
```

```
                    sys.stdout.write('%.2f' % (-10.0 * ((value * 5) / 1024.0 - 2.5) * voltages[i]))
                    sys.stdout.write(';')
                else:                 # calculate power for 40A sensor
                    sys.stdout.write('%.2f' % (-20.0 * ((value * 5) / 1024.0 - 2.5) * voltages[i]))
                    sys.stdout.write(';')
            else:
                print('Unknown argument: '+sys.argv[2])
                sys.exit(0)
        sys.stdout.write('\n')


def printusage():
    print('Usage:')
    print('-c type      Show the collected data, type={r,p} (r shows raw ADC results, p for calculated power)')


def main():
    signal.signal(signal.SIGINT, ctrlc_handler)
    if sys.argv[1] == '-c':
        if sys.argv[2] == 'r' or sys.argv[2] == 'p':
            print('Starting data collection')
            collector()

    printusage()
    sys.exit(0)

if __name__ == "__main__":
    main()
```

List of Figures

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Studiengang Bachelor of Science Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ort, Datum

Unterschrift

Veröffentlichung

Ich bin damit einverstanden, dass meine Arbeit in den Bestand der Bibliothek des Fachbereichs Informatik eingestellt wird.

Ort, Datum

Unterschrift