Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bachelor's Thesis Nr. 335b

Systems Group, Department of Computer Science, ETH Zurich

Real-time Board Management using an FPGA

by

Sarah Tröndle

Supervised by

Dr. Michael Giardino
Daniel Schwyn
Prof. Dr. Timothy Roscoe

October 2020 – April 2021

D INFK

Abstract

Server-level computers usually have a BMC (baseboard management controller), that is responsible for setting up and maintaining an environment where all of the components of a system can function. This includes time sensitive tasks like triggering an emergency shutdown of a component or the whole system. The Enzian BMC runs based on OpenBMC, one of the few open source BMC projects. It is based on python scripts communication over D-Bus. This entails that there are no strong timing guarantees. Thus, the goal is to achieve real-time board management with the BMC's FPGA.

This thesis examines the feasibility of that goal by implementing and testing a PID-based fan control on Enzian's BMC FPGA. The functioning PID controller shows that FPGA board management is possible. It also exposes difficulties that still have to be overcome, when board management tasks should be handled exclusively by the FPGA. The main difficulty lies in communication between the BMC FPGA and other devices on Enzian.

Contents

List of Figures

1 Introduction

1.1 The BMC

For a computer to run it requires a certain range of voltage, temperature control and more. This is not related to the execution of a program or the operating system, but setting up and maintaining an environment where all of the components of the computer can function. In server systems this is usually handled by the Baseboard Management Controller (BMC), which is placed on its own chip.

The exact workings of the BMC, including the software, are typically manufacturer specific and proprietary. This has lead to there being little research on an essential part of increasingly complex computers. There are, however, known tasks the BMC is commonly responsible for: [2]

Health monitoring and management The BMC monitors the health of the system, for example with temperature sensors and fault monitors. The aquired information can be used for recovery or response mechanisms. Often the information is accessible through a web interface.

Data collection and logging Beyond what is necessary for health management, the BMC collects data about the state of the hardware. Based on this data it often can produce alerts or notifications.

Booting During booting, the BMC can be responsible for managing the clock and power resources and installing software, like firmware, a CPU needs to boot.

Remote access Services of the BMC are usually available over a network or interface.

The work for this thesis is done on Enzian's BMC. Enzian is a research computer built by the Systems Group at ETH Zürich. It has a 48-core CPU, a large FPGA, 640 GiB of DDR4 RAM and up to 460Gb/s network bandwidth. The BMC that manages all of the components was also developed by the Systems Group. It is a Enclustra Mercury SoM with a Xilinx Zynq MPSoc CPU running Linux. Its software is based on OpenBMC which runs with python scripts and D-Bus.

One of the BMC's main task is to manage the power supply to the CPU and FPGA by configuring devices like clock generators and voltage regulators before the CPU and FPGA are started. The devices themselves also have to be started in a specific sequence. This too is handled by the BMC. Furthermore, it also controls fans and monitors sensor readings sent from devices over I2C. Additionally, the BMC is responsible for handling SMBus alerts that are sent from devices. During booting, the BMC can program the FPGA with an initial bitstream and access the flash memory, which is used to boot the CPU. [2]

1.2 Goal

Since the BMC software is based on python with D-Bus, there are no timing guarantees. This software, however, has to handle critical tasks such as triggering an emergency shut down of a component or the whole system. With the absence of timing guarantees, there is also no guarantee, that the BMC can react fast enough to prevent faults from occurring. The lack of such guarantees is concerning, especially for a research computer that was built with a high-end CPU and FPGA. To be able to have timing guarantees, the goal is to achieve real-time board management with the BMC's FPGA. This thesis examines the feasibility of that goal with fan control as a test case. In doing so, the intended approach can be tested for its applicability and potential difficulties of doing board management with an FPGA can be discovered.

2 Related Work

To the best of our knowledge, there has not been any work published that specifically looks at using an FPGA to do real-time board management. Other aspects of a BMC have, however, been researched. Work on using the FPGA for some of the BMC's tasks has also been published. After presenting an article on the state of BMC software, work on different aspects of power management and using an FPGA to do tasks of a BMC are presented.

In his article Frazelle gives an overview over the current state of BMC software. The article mentions that the intelligent platform management interface (IPMI), which BMCs usually use to communicate to the outside world, was designed with the idea that data center control networks would be segregated and trusted. They are, however, not segregated and hence, cannot be trusted. This has lead to IPMI being notorious for security vulnerabilities. In the context of security vulnerability the article also discusses BMC software mostly being proprietary. The author describes it as "an alarming problem that the code running with the most privilege has the least visibility and inspectability". There are, however, open source BMC software projects, as Franzelle writes. One of them is OpenBMC, which is based on two projects with the same name, one founded by Facebook and the other by IBM and Rackspace. As mentioned in the Introduction, the Enzian BMC uses openBMC. [3]

Dynamic power management regulates how much power a component of a system gets, depending on the required performance. There are multiple known methods to achieve this. An example is HP's PaperClip, an electronic clipboard. Its system-level dynamic power management policy has been described in [4]. A power management module, that runs on the CPU, predicts when the power should be reduced. It then writes commands to memory-mapped I/O locations. The power management commands are then decoded by control circuit implemented on an FPGA and distributed from there. Many of the approaches to power management algorithms are described with finite state machines. These describe the different power states a machine can be in, for example idle, run and sleep, and when to transition between those. These different power states can be reached with clock gating, where the clock frequency or the voltage supply is reduced. Since power is proportional to the frequency and the square of the supply voltage, often reducing the voltage is preferred. However, this is usually combined with frequency downscaling. Alternatively, the power supply can be completely shut down to individual components. [5]

Pozniak et al. proposed a FPGA based platform control board (PCB) for low level radio frequency control systems. [6] The platform controller (PC) is the main part of the PCB and consists of three modules:

- VME-bus controller: passive controller for VME-bus communication
- PC-embedded controller: optional communication module for control operations over LLRF, signal processing and monitoring
- Data & timing controller: distribution of fast and synchronous signals.

The data & timing controller is implemented in a programmable chip. The distributed synchronous signals are divided into clock signals and triggering signals. They have separate lines that supply all module slots. For the data signal the data & timer controller has a bus connection to each slot, however, the slots are also connected between each other. [6]

Another aspect of board management is ensuring that the supplied voltage to each component stays in a range that allows it to function without any faults occurring. One way this has been done is with a combination of numerical relays and FPGAs. A numerical relay can monitor analogue inputs like current or voltage by transforming them to digital signals and processing the digital signals, commonly with a microprocessor. The FPGA is then programmed with algorithms that have conditioned signals from the numerical relay as inputs, calculate a cut-off and generate control signals for the relay unit. The relay unit can then, if necessary, cut the unit off from the power supply. [7]

Fan Control, another task of the BMC, has also been done with an FPGA. Daboul and Nouman implemented a DC fan controller on a FPGA, outputting a PWM (pulse width modulation) signal. This functions on the basis of comparing the value describing the required fan speed with a triangular wave signal. The focus of this work is on producing a signal to directly power the fan. [8]

Other work focuses on calculating the required fan speed based on a measured temperature. Chan et al. present a modular feedback controller, which can be adapted for different use cases. The presented application is a temperature control system. An ADC (analogue-to-digital converter) interface converts an analogue signal from a thermistor and passes it to the PID (proportional-integral-derivative) controller. The PID controller is based on distributed arithmetic and implemented on an FPGA. Its output are PWM signals for the fan motor and the lamp to lower or increase the temperature. [9]

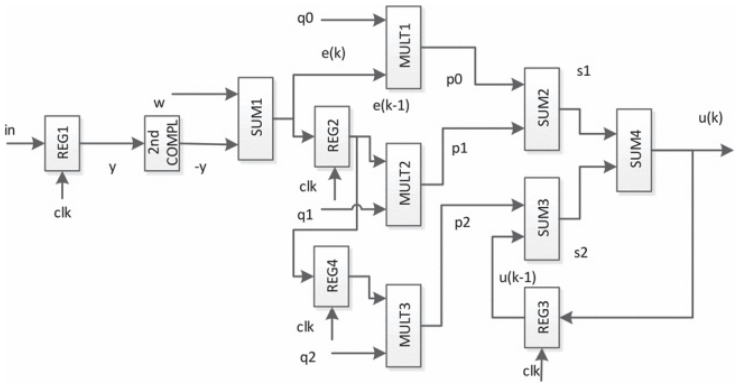


Figure 1: Parallel design of an FPGA PID controller [1]

Another approach to FPGA based PID controllers is presented by Kocur, Kozak and Dvorscak. Here the focus is not the whole control loop but mainly the PID controller on the FPGA. In comparison to the previously mentioned controller there is less focus on obtaining a space efficient algorithm and more focus on implementing a known discrete PID algorithm. The design they propose is parallel, visible in figure 1, and described in more detail than the design presented by Chan. [1]

3 Approach

Previous works related to the goal of achieving real-time board management with an FPGA only covers aspects of it in isolation. The elements of the presented control circuits are directly connected. On the Enzian, however, even the different tasks of the BMC, like voltage or temperature control, cannot be viewed in isolation. Most sensors and regulators the BMC relies on are not located within the BMC, but across the whole system. Therefore, to examine the feasibility of real-time board management on the FPGA, different tasks of the BMC can be explored separately with regards of using the FPGA to handle them. However, they always have to be viewed in the context of the whole system. Considering that fan control with an FPGA has already been done, we decided on using this example to explore what is necessary to realise board management on an FPGA on a system like the Enzian. Controlling the temperature with a fan has also the advantage that testing it is much more feasible than testing what happens if the power supply behaves in unexpected ways.

Since the goal is to test how control logic on the FPGA can interact with the whole system, we use a PID controller to determine the required fan speed. It is a reasonable approach to do fan control with an FPGA as it uses an input to calculate an output value, that is sent on as a control signal. Allowing to see how a controller can be integrated into the system, communicating with other components.

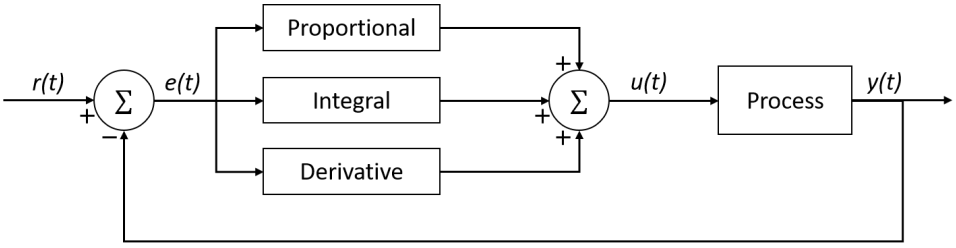


Figure 2: Block diagram of a PID controller, with feedback loop

4 PID Controller Background

A proportional-integral-derivative (PID) controller is a commonly used controller to bring a system towards an objective, the set point. Depending on the system, that could for example be a temperature, a certain speed or a position. A PID controller uses a closed-loop system. Its input is the feedback-signal from the process under control. As the name suggests the PID controller is based on three parts:

Proportional Controller The P-part results in an output proportional to the current error $e(t)$, where the error is the difference between the set-point and the feedback.

Integral Controller The integral controller is responsible for eliminating the possibility of a steady-state error. This can occur when with the control value of the proportional part the system ends up in a steady state before the set-point is reached. By adding a value proportional to the integral of previous error, $\int_0^t e(\tau)d\tau$, the I-part of the control value keeps increasing, preventing the system from being stuck in a steady state.

Derivative Controller The integral controller cannot predict the future behaviour of the error. As long as there is an error, the I-part increases. This causes an overshoot when the set-point is reached, since at the set point the P-part is 0 but the I-part is not. This is countered with the differential part. With the P-part proportional to the derivative of the error, the output of the differential controller gets more and more negative, the faster the error decreases.

As shown in figure 2, the results of the three parts are added together, resulting in the command signal $u(t)$. The PID equation can then be written as

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de}{dt}$$

where K_p, K_i, K_d are the proportional, integral and derivative gains. [10] It

can also be written as

$$u(t) = K_p \left((t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt} \right)$$

where T_i, T_d are the integral and derivative time constants. [1]

These are continuous equations, hence they are suited for controlling analogue signals. An FPGA, however, processes digital signals, needing a discrete equation.

Both PID controller papers presented in section 2 have direct connections from the analogue to digital converter to the FPGA, allowing them to be synchronized. In the Enzian the fan controller is connected to the FPGA over an I2C bus and current temperatures have to be requested. This makes the problem of providing the PID controller with the input signal in the correct form more complex. Also, sending out the control signal is non-trivial.

For this reason implementing a PID fan controller on the Enzian BMC FPGA is viewed as two problems. Firstly, the communication between the FPGA and the fan controller (MAX31785) and then, based on the constraints this entails, implementing a PID controller on the FPGA.

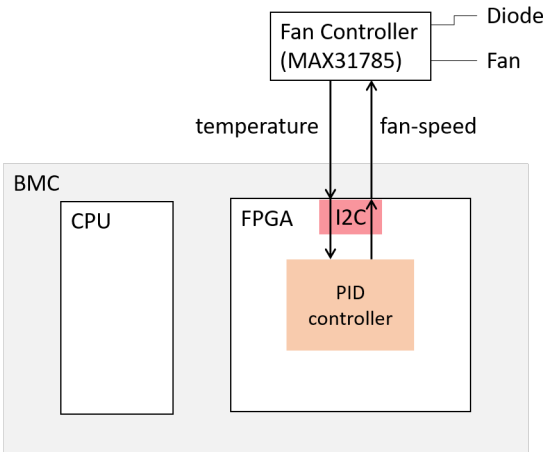


Figure 3: Diagram of the ideal integration of the PID controller into Enzian

5 Integrating a PID controller into the Enzian BMC

The PID controller should take a temperature value as an input and output the according fan speed. The fan's speed should be in percentage for the PWM duty cycle, describing during how much time of one cycle the fan should be supplied with power. This will then control the fan speed, since with PWM a device is supplied with power at a specific frequency. Unless the device is reconfigured this frequency does not change. What changes is the duty cycle, which can also be described as the ratio between a pulse width and the period. By increasing the duty cycle, the time during which, for example, a fan is supplied with voltage is increased. This increases the total power supplied, which in turn increases the fan speed.

As mentioned, the BMC is not directly connected to the fan controller. Temperature requests and fan speed commands have to be sent over an I2C bus. Ideally this could be handled from the FPGA, resulting in a design shown in figure 3.

This would, however, require the commands for requesting the temperature and setting the fan speed to be produced on the FPGA. This also includes extracting the temperature from the message sent back by the fan controller and setting the right channel of the fan controller to reach the desired diode or fan. This has to be done since 6 fans and diodes can be connected to the fan controller MAX31785. While there is an I2C implementation on the FPGA, implementing this would require a lot of work, going beyond the scope of this bachelor thesis, while not being its main focus. For this reason communication with the fan controller is handled by the BMC software on the CPU. There, this is already implemented for the telemetry and the power service from the power management tools. While this can no longer result in a real-time system and seems to contradict the goal of using

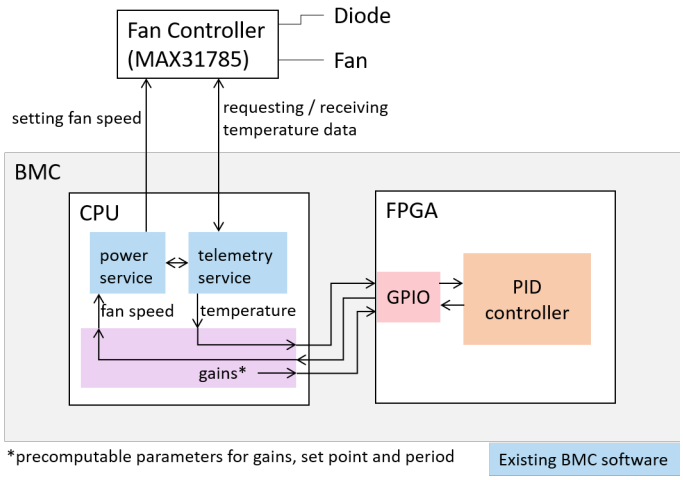


Figure 4: Diagram of the integration of the PID controller into Enzian

the FPGA instead of the CPU to have more reliable speed, it still allows to examine what is necessary in order to reach that goal. One such necessity is having the I2C communication protocol with device specific commands implemented on the FPGA.

Handling I2C communication from the CPU results in a system as shown in figure 4. A python script (represented by the purple rectangle), that is added to the power manager, uses the existing power management software to communicate with the fan controller and the FPGA. The additional script uses the telemetry service to receive a temperature update every second. The temperature is then converted to a signed integer format and written to memory-mapped AXI GPIO. After writing the temperature, the script waits for 0.1 seconds before reading the calculated fan speed from GPIO. This delay of 0.1 seconds has been set generously and could still be optimized. The power and telemetry service communicate based on D-Bus. The D-Bus commands then get translated to I2C by another part of the power manager.

Constants like gains and set points for the PID controller can be hard-coded into the FPGA. With there already being a python script that writes values to the FPGA it made, however, sense to also write them from the CPU. For the eventual execution of the PID controller this does not make a difference, but it makes it easier to change constants and allows for a more convenient tuning of the PID controller. Changing values in a python script is less time-consuming than regenerating a bitstream.

6 Implementation of the PID Controller

The requirements of the PID controller are to take in a temperature value and then calculate a fan speed with that. The fan speed should be in percentage for the duty cycle, describing during how much time of one cycle the fan should be supplied with power. How the PID controller is supplied with temperature values and how the calculated fan speed is sent to the fan controller was discussed in section 2. Some of the choices described in the previous section affect how the PID controller itself has to be designed. How the values are exchanged between the CPU and the FPGA has the biggest effect on how the PID controllers presented in the previous work section have to be adapted to the Enzian BMC. The AXI GPIOs get written asynchronously meaning that either the whole PID controller has to work asynchronously or there has to be a step to synchronize the signal. This will be discussed later in this section.

6.1 Calculations done by the PID controller

Since the goal for the PID controller is not to implement it in the most effcient way but to use it to see how board management on the FPGA can work, the PID controller is based on the ones presented in the previous work, as they have been tested and are known to work. The first approach was to use the controller presented in [9], since it has been tested for temperature control. This FPGA PID controller is based on distributed arithmetic and is space optimized. It calculates the output $u(k)$ the following way:

$$u(k) = P(k) + I(k) + D(k)$$

where k is the current instant and $P(k), I(k)$ and $D(k)$ are:

$$P(k) = K\left(bu_c(k) - y(k)\right)$$
$$I(k) = I(k-1) + \frac{K}{T_i}\left(u_c(k-1) - y(k-1)\right)$$
$$D(k) = \frac{T_d}{T_d + NT}D(k-1) - \frac{KT_dN}{T_d + NT}\left(y(k) - y(k-1)\right)$$

Where $y(k)$ is the feedback signal, $u_c(k)$ is the command signal at the current instant, K, b, T_i, T_d, N are controller parameters and T is the sampling period.

Using a PID controller that was tested with temperature control made sense, especially when the formulas behind the PID controller are not the main focus of the project. However, during an implementation attempt of the PID controller for this project, the u_c was accidentally handled as a constant. This prompted a revaluation of the idea that using a PID formula that has been tested with temperature control is better than using a less

optimised, more general PID formula. The conclusion, in line with previous experience, was drawn that a simpler concept is likely faster to implement and less prone to mistakes.

For this reason the PID controller presented in [1] was then chosen as a basis for the PID controller on the Enzian BMC FPGA. The control output $u(k)$ at instance k is calculated as

$$u(k) = u(k-1) + q_0 e(k) + q_1 e(k-1) + q_2 e(k-2)$$

where $e(k)$ is the error of the current instance, $e(k-1), e(k-2)$ are errors of the previous 2 instances and q_0, q_1, q_2 are

$$q_0 = P\left(1 + \frac{T_d}{T}\right) \quad q_1 = -P\left(1 - \frac{T}{T_i} + 2\frac{T_d}{T}\right) \quad q_2 = P\frac{T_d}{T}$$

T is the sample time, P is the proportional gain and T_i, T_d are the integral and derivative time constants. The paper also provides the decomposition of the equation in a form that can be directly implemented on the FPGA:

$$\begin{gathered} e(k) = w(k) - y(k) \\ p_0 = q_0 * e(k) \quad p_1 = q_1 * e(k-1) \quad p_2 = q_2 * e(k-2) \\ s_1 = p_0 + p_1 \quad s_2 = p_2 + s_1 \quad u(k) = s_2 + u(k-1) \end{gathered}$$

Where $w(k)$ is the set point, $y(k)$ is the feedback at the current instance, $u(k-1)$ is the control output of the previous instance and p_0, p_1, p_2, s_1, s_2 are intermediate results. For fan control the error was redefined as $e(k) = y(k) - w(k)$, since a temperature that is higher than the set point should result in a higher fan speed.

The sample time is defined by the telemetry service running on the CPU and set to one second. This allows to assume T as a constant, making q_0, q_1, q_2 precomputable. Instead of setting the gains and handling the computation of q_0, q_1, q_2 on the FPGA, they are precomputed on the CPU and written to an AXI GPIO before starting the fan control. Since there is already a python script, that writes temperatures to the FPGA (see previous section) the same script is used to write the q values. Setting the gains and also the set point in a python script allows them to be changed easily. Which is especially useful while tuning the controller. This results in the controller shown in figure 5.

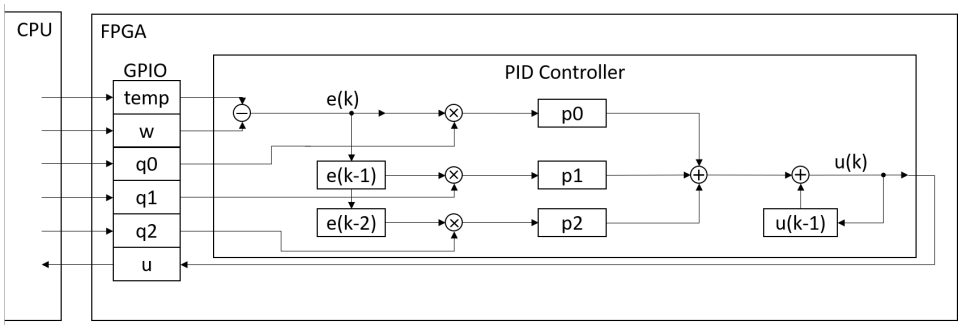


Figure 5: Simplified block diagram of the PID controller

6.2 Handling Asynchrony

With the temperature value being written to AXI GPIO once the telemetry service returns a value, a way had to be found to only calculate a new output when a value is written. Simply recalculating every clock cycle would not have worked. The result of recalculating the output once or multiple times is different, even if the input values are the same, because every time the output of the previous iteration, $u(k-1)$, gets added to the new output. Simply waiting for the temperature value on the GPIO to change is also not an option. If the temperature does not change, the output still has to be recalculated. For this reason an extra GPIO register is added, which holds a value that switches between 0 and 1 every time a temperature is written. This alone is, however, not enough. Waiting for an AXI GPIO value to change cannot by itself trigger a process on an FPGA. While it can be implemented in VHDL, the FPGA would need a mechanism to actively observe a value C on GPIO to be able to tell when the value changes.

By adding a register that stores the last read C, the register and GPIO can be compared every clock cycle. This then allows to start the calculation of the output only when a temperature value has been written. Since not all of the computation fits in one clock cycle, the p_0, p_1, p_2 values are saved in registers and in the next cycle the fan speed u is calculated.

Doing this as a pipeline would not work, since then the output would be calculated twice instead of once. Instead, during the first cycle only the first part of the calculation is done and during the second cycle only the second part of the calculation is executed. To trigger the execution of the second part, an additional register is introduced, whose value is set to 1 at the end of the cycle that recalculated p_0, p_1 and p_2. As with the C register at the beginning of each cycle it is checked if this register is set to the value 1. If so, the second part of the computation is executed and the register is set to 0 again.

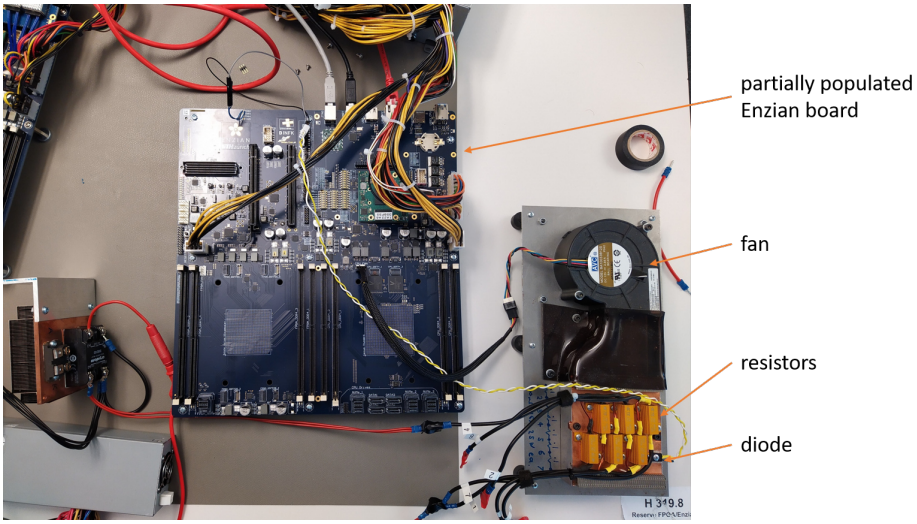


Figure 6: Set up for testing the PID fan controller

6.3 Tuning and Testing the PID controller

First, it was tested, that the PID controller integrated into Enzian can measure temperatures and set fan speeds. This was done by reading the BMC's internal temperature sensor and controlling a fan that was connected to a partially populated Enzian board, the partpop2. This could however only test if single components of the fan control system were working, not if it would react as expected to changing temperatures. Also, this could not be used to tune the PID controller. To tune the PID controller, both low and hight temperatures have to be reached controllably and the fan has to set up to cool the component that is heating up.

For this reason the fan was set up to cool resistors, which were used to generate heat and were connected to an external power source. The temperature at the resistors was measured with a diode connected to the partpop2.

The resistors were supplied with 2.97 volt and produced 0.4455 watt. When the resistors were not supplied with power and the fan was turned off, the diode measured room temperature at approximately 25°C. With the resistors heating, a maximal temperature of 45.75°C was measured. When also running the fan at full speed, the temperature stabilised at around 31°C.

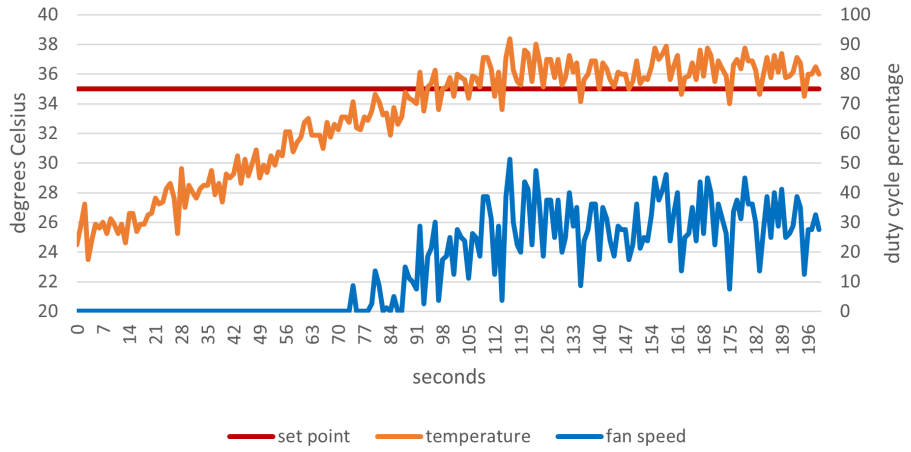
This is not the temperature range required to fully tune a PID controller before it is used to control the fan speed, but it is enough for testing purposes. Once it is established that the PID controller functions as expected inside the Enzian BMC FPGA, the only things that would have to change, before using it to cool a running system, are the set point and the

gains. Whenever the PID controller is used with a new fan, these have to be adapted anyway, since the set point depends on the position of the diode and what exactly has to be ventilated. The gains depend on the specific fan. A powerful fan needs different gains than a fan that, with a 100% duty cycle, can only achieve halve or a fourth of the power.

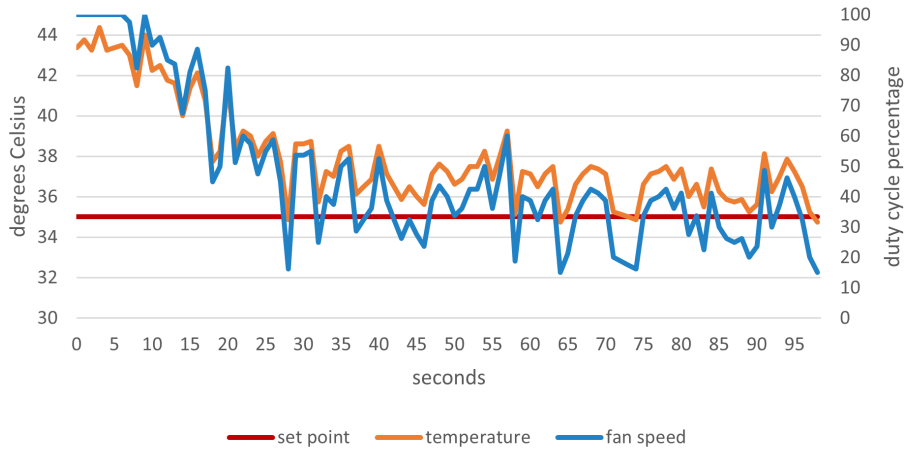
Since the range of controllable temperature was between 31°C and 45°C the set point was set to 35°C. The proportional gain was set to 0.00001, such that the fan would receive maximal power at 45°C. Since in this temperature range the temperature reacted quickly to the changes in fan speed, the integral part could be kept small with setting the integral time constant to 10. The derivative time constant was set to 0 because for fan control overshooting is not an issue, although it might not result in optimal efficiency.

How fan speed and temperature behave when first the PID controller and the fan are activated and then the resistors are supplied with power, is visible in figure 7a. The temperature rises and then stabilises slightly above the set point. In figure 7b it is shown, how temperature and fan speed behave when the fan is only activated once the resistors have heated up. The temperature decreases quickly and as in figure 7a it stabilises slightly above the fix point.

This shows, that the PID controller on the Enzian BMC FPGA behaves as expected.



(a) Fan controller is activated before resistors



(b) Fan controller is activated after resistors

Figure 7: Temperature in °C and fan speed in duty cycle over time

7 Conclusion

To examine the possibility of doing real-time board management with an FPGA, a PID controller was implemented and tested on the Enzian BMC, to control the fan speed. The resulting controller could, in theory, directly be used. However, as mentioned in the previous section, the PID fan controller has to be tuned for every fan and diode position. For this, a tuning set up would have to be built first.

Implementing a PID fan controller on the BMC FPGA revealed some challenges that arise when using the FPGA to handle tasks of the BMC.

A big one is the communication between the FPGA and other components of the system. In this project this was not handled by the FPGA, allowing the focus to be on the testing of the general concept of doing board management with an FPGA. On the Enzian BMC FPGA I2C is however already implemented. What remains to be done is generating the device specific commands and extracting the desired values from the responses.

A difficulty that was solved when it appeared, is handling asynchrony. The source of asynchrony was the CPU writing the temperature value to the AXI GPIO. If the FPGA handles all communication with other devices over I2C, this will likely have to be handled differently. Mainly because with the I2C implementation on the FPGA synchronisation will have to happen at a different stage.

Handling of faults or alerts from the system is an aspect of board management, that has not been examined in this project. However, this are tasks the FPGA would be suited for well, since depending on the alert, fast handling is crucial. Here, it is likely that as with the GPIO writes, signals would also reach the FPGA asynchronously.

References

[1] M. Kocur, S. Kozak, and B. Dvorscak, "Design and implementation of fpga-digital based pid controller," in *Proceedings of the 2014 15th International Carpathian Control Conference (ICCC)*. IEEE, 2014, pp. 233–236.

[2] C. Heimhofer, "Towards high-assurance board management controller software," Master's thesis, ETH Zürich, 2021.

[3] J. Frazelle, "Opening up the baseboard management controller," *Communications of the ACM*, vol. 63, no. 2, pp. 38–40, 2020.

[4] L. Benini, R. Hodgson, and P. Siegel, "System-level power estimation and optimization," in *Proceedings of the 1998 International Symposium on Low Power Electronics and Design*, ser. ISLPED '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 173–178. [Online]. Available: https://doi.org/10.1145/280756.280881

[5] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," 2000.

[6] K. Pozniak, R. Romaniuk, and K. Kierzkowski, "Modular and reconfigurable common pcb-platform of fpga based llrf control system for tesla test facility," 2005.

[7] B. Venkateshmurthy and K. Nataraj, "Design and implementation of high speed fpga for under & over voltage protective relay," in *2017 International Conference on Recent Advances in Electronics and Communication Technology (ICRAECT)*. IEEE, 2017, pp. 76–80.

[8] M. Daboul and Z. Nouman, "The control of fan speed using fpga boards," *Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska*, 2014.

[9] Y. F. Chan, M. Moallem, and W. Wang, "Design and implementation of modular fpga-based pid controllers," *IEEE transactions on Industrial Electronics*, vol. 54, no. 4, pp. 1898–1906, 2007.

[10] "Introduction: Pid controller design," https://ctms.engin.umich.edu/CTMS/index.phpexample=Introduction&section=ControlPID, accessed: 10.04.2021.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Real-time Board Management using an FPGA

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Tröndle	Sarah

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Windlach, 12.04.2021	Sarah Tröndle

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.