Department of Physics and Astronomy
University of Heidelberg

Bachelor Thesis in Physics
submitted by

Daniel Kutny

born in Kędzierzyn-Koźle (Poland)

2017

Development of a Modern Monitoring Platform for the BrainScaleS System

This Bachelor Thesis has been carried out by Daniel Kutny at the
Electronic Visions Group in Heidelberg
under the supervision of
Prof. Karlheinz-Meier

Abstract

Ongoing monitoring of is an integral component of any large-scale project. This thesis aims to enhance the monitoring of the BrainScaleS system by improving on the existing infrastructure as well as developing a modern visualization for the system state. A new "event pipeline" has been introduced which streamlines the process of aggregating, processing and storing event data received from the BrainScaleS system using ElasticStack.
Using Grafana multiple dashboards were created which allow a fast visualization of the state of BrainScaleS system, the internal Visions network and the connected devices. A new status code for FPGAs has been introduced which unites all information regarding its state in a three digit number. This status code is used for the central dashboard giving the user an overview of all FPGAs and Reticles.
Security measures regarding the access to the Elasticsearch database and the monitoring software were introduced and the ground work for alerting has been laid down in this thesis.

Zusammenfassung

Kontinuierliche Überwachung ist ein integraler Bestandteil eines jeden großskaligen Projekts. Diese Bachelorarbeit zielt auf die Verbesserung der Überwachung des BrainScaleS-Systems durch Verbesserung der bereits existierenden Infrastruktur sowie der Entwicklung einer modernen Zustandsvisualisierung ab.
Eine "Event Pipeline" wird eingeführt welche die Aggregation, Verarbeitung und Speicherung von Ereignissdaten ("event data") des BrainScaleS-Systems durch Nutzung des ElasticStack vereinfacht. Mittels Grafana werden verschiedene Dashboards zur Überwachung des Zustands des BrainScaleS-Systems, des internen Visions Netzwerk sowie der angebundenen Geräte erstellt. Für die FPGAs der Wafer wird ein Statuscode eingeführt welcher die zentralen Zustandsinformationen in einer dreistelligen Zahl vereinigt. Dieser Statuscode wird zur Entwicklung des zentralen Dashboards genutzt welcher dem Nutzer eine Übersicht über alle FPGAs und Reticles ermöglicht.
Sicherheitsmaßnahmen betreffend dem Zugang zur Elasticsearch Datenbank sowie der Visualisierungssoftware werden eingeführt. Ebenso wird die Grundlage für die Alarmierung geschaffen.

Contents

Chapter 1

Introduction

1.1 Motivation

Slawek Ligus defines *monitoring* in his book *"Effective Monitoring & Alerting"* as follows [4]:

> Monitoring is the process of maintaining surveillance over the existence and magnitude of state change and data flow in a system. Monitoring aims to identify faults and assist in their subsequent elimination.

Ligus' definition already shows the importance of monitoring. Today's physics experiments often run for many weeks or months and rely on many electronics, computers, servers and other components. Large-Scale projects are frequently platforms for other scientists who rely on those components. As such, errors may occur of which the system's designers did not think of. Examples of such projects would be DESY, which allows scientists from all over the world to perform their experiments with synchrotron radiation, but also the Electronic Visions Group with the Neuromorphic Platform, which aims to allow scientists to use neuromorphic hardware.
The goal of this bachelor thesis is to build a *monitoring system* for the neuromorphic platform which allows the user and the administrators to visualize the behavior of its different component, study patterns and recognize potential failures easily.

1.2 Different Types of Monitoring Data

In general, one can differentiate between two types of data: *event data* and *time series data*. Timeseries data are continuous lists of pairs of values, in which one is a point in time and another one a numeric value. They are meant to represent time-dependent, continuous metrics. An example of such data would be the his-

tory of the temperature of a CPU.

On the other hand, we have event data. In the context of monitoring, event data is a data set whose existence is triggered by an external event and is attached to a point in time. These datasets do not need to be continuous nor do they need to have a specific value attached to them. An example of such an event is the booting of a computer, crashing of a program or the failure of a computer component. These events are not necessarily numeric values but can contain logs, errors or some structured data, i.e. a JSON-object. [7]

1.3 Monitored Hardware and Software

In this section the BrainScaleS neuromorphic platform and its monitored hardware and software components will be outlined.

1.3.1 BrainScaleS System

The BrainScaleS system consists of twenty wafer modules with the neuromorphic wafer as the central component of a module.
The wafer itself is divided into 48 parts called *reticles*. Each of those reticles contains 8 HICANN chips, resulting in a total number of 384 HICANN chips for each wafer. Every reticle is connected to a *FPGA communication PCB* (FCP) which handles the communication between the user and the reticle. All reticles are also connected to the *Analog Breakout PCB* (AnaB), which send membrane voltages to the analog readout modules.
The wafer module is connected to the PowerIt module which, as the name suggests, powers all its components. The power to the reticles is maintained by the *Wafer Module Main PCB* (MainPCB), which can switch the power on and off on a per-reticle basis. basis.
Six boards called cure boards are installed on the MainPCB board which provides the monitoring of all wafer voltages. All the data from the sensors are collected by the *Single-Board Control Computer*, which is a Raspberry Pi connected by an Ethernet link. [6] [3] A schematic view is provided in Figure 1.1, an explosion view is provided in Figure 1.2.

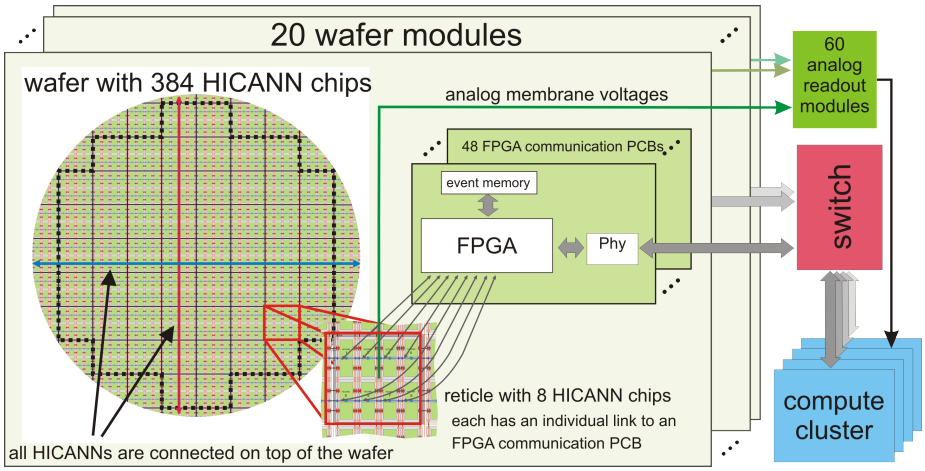

Figure 1.1: Schematic overview of a wafer module. [6]

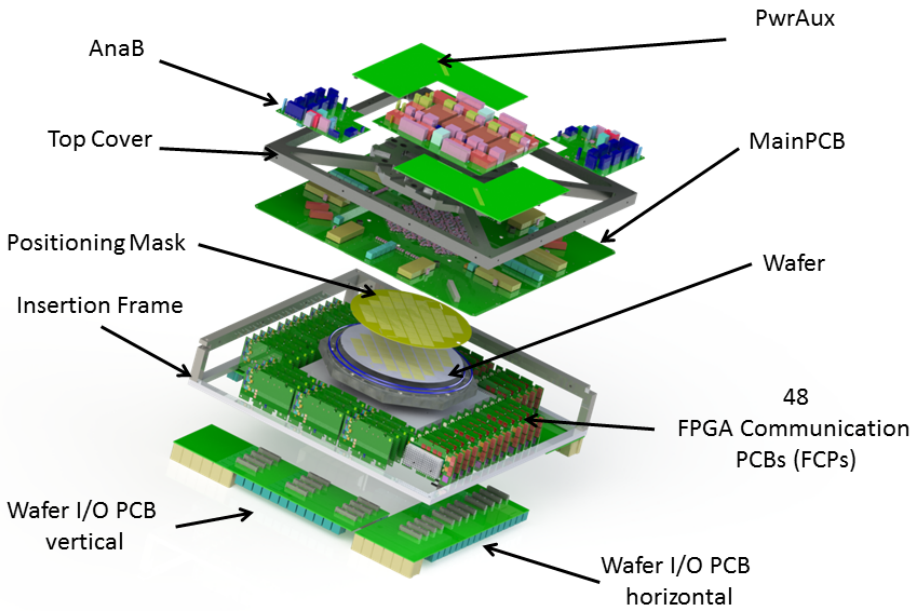

Figure 1.2: Exploded-view drawing of the module. [6]

1.3.2 Electronic Visions Network

Servers and clients from the internal network are all monitored by the *Ganglia Monitoring System*, or short *Ganglia*. Ganglia is a daemon which collects metrics such as CPU temperature, work load, network activity and memory allocation and sends them to a storage such as Graphite, which is described in 1.4.2.

1.4 Storage and Visualization

1.4.1 Elastic Stack: Elasticsearch, Logstash and Filebeat

The software bundle consisting of Elasticsearch, Logstash and Filebeat is referred to as "Elastic Stack". In the following section all the components of the Elastic Stack will be presented.

Elasticsearch

Elasticsearch is a database with an integrated, powerful fulltext search. It can store structured data as JSON documents and allows an access to saved documents via a RESTful API. Documents are organized in so called *indices*. For example, one could put all logs belonging to the wafers in an index called log. Elasticsearch supports the star symbol (*) as a wildcard, therefore one can save files in indices with the today's date, ie. "logs-2017.08.13". Using the wildcard, one can still query across all logs using "logs-*".
Documents are classified by so called types. For example, logs could be classified in the types "error", "warning", and "information". In the following section the queries for writing and deleting these documents will be presented. All the documents are also carrying an unique identifier.

Queries Queries are handled with simple HTTP-Requests. To save a document, one would send a PUT request to the Elasticsearch server with the URL of the form `http://host/index/type/id`. As an example, we will put an error message into Elasticsearch:

```
PUT http://host/logs-2017.08.09/error/5423
{
    "@timestamp": "1507637388000",
    "from":  "Server_13",
    "error": "Booting failed"
}
```

To access this error message, we can simple use GET on the above url. We can also simply open

```
http://host/logs-2017.08.09/error/5423
```

in our browser. The answer from the server will be the error from above and some metadata. The example response is shown in 1.1.

```
{
    "_id": "5423",
    "_type": "error",
    "_version": 1,
    "_score": null,
    "_source": {
        "@timestamp": "1507637388000",
        "from":  "Server 13",
        "error": "Booting failed"
    }
}
```

Listing 1.1: Example response for a GET query

Searching through documents can be carried out in two ways. Either through a URL with a Lucene Query or with a JSON object.

Lucene Query To search with a URL, one can simply append "_search" to an index and the search query with "q=". The following request will look through the index "logs-*", in which the source of the errors is the Server 13.

```
GET http://host/logs-*/error/_search?q=from:Server_13
```

Request Body Search The request body search allows a more sophisticated search, as it is possible to perform full-text search, combine multiple queries, weight different queries and receive a score based on how well the document matches the information. For the purpose of this bachelor thesis, however, it is sufficient to know that you can query through Elasticsearch with a HTTP-body like in 1.2.

```
GET http://host/logs-*/error/_search
{
    "query" : {
        "term" : { "from" : "Server_13" }
    }
}
```

Listing 1.2: Example of a document request in Elasticsearch

Filebeat

Filebeat is a daemon which runs on a device such as a server and observes its log files and sends them to Logstash.

Logstash

Logstash is a service that can process log files to useful JSON objects which can be put into Elasticsearch. A typical log file from syslog is in the following format:

```
Apr 12 18:43:35 host script.py: Error while Init
```

Logstash can process this error message into a JSON file, but the user has to define the pattern of the log format against which it is supposed to match. The end result that we want for the above message is a structured document which can easily be queried:

```
{
    "@timestamp": 1002891515000,
    "host": "host",
    "source": "script.py",
    "message": "Error while Init"
}
```

Listing 1.3: Example of a transformed log message

To start processing, we need to define a "pipeline" along which the log is processed. Such a pipeline is saved in /etc/logstash as a.conf file and is loaded automatically when starting Logstash. Since logstash has a built-in filter for syslog messages, the listing 1.4 file would transform the log into the JSON format we want and insert it into Elasticsearch.

```
input {
    # Listen to port 5044 for
    # incoming logs from filebeat
    beats { "port" => "5044"}
}

filter {
    grok {
        match => {
            # Match incoming log against known
            # pattern and structure it to JSON
            "log" => "%{SYSLOGBASE} %{GREEDYDATA:message}"
        }
    }
}

output {
    #Send structured file to Elasticsearch
    elasticsearch {}
}
```

Listing 1.4: Example of a configuration file for Logstash

1.4.2 Graphite

Graphite is a fixed-size time-series database. This means that for each time-series metric, a fixed period is stored (for example, one year). The resolution in time

can also be set up, so that one month metrics only have a resolution of 30 minutes, while the metrics of the last 24 hours have a resolution of one minute. The metrics are stored in a folder-like structure. Graphite provides a JSON object for retrieving the stored data. It can be queried using `http://shinviz.kip.uni-heidelberg.de/render?target=QUERY&format=json`, where QUERY is replaced with the target metric. An example of such a target would be "WaferModule.*.Wafer.temperatures.south". This will retrieve all temperatures from the "southern" part of the wafer from all wafers. Like in Elasticsearch, the star symbol (*) is used as a wildcard.
An example response from graphite is provided in 1.5.

Listing 1.5: Example response from graphite for wafer temperature

```
[
    {
        "target": "WaferModule.36.Wafer.temp.south",
        "datapoints': {
            [34.3, 1509724940],
            [35.2, 1509724960],
            [34.7, 1509724980],
            ...
        }
    },
    ...
]
```

1.4.3 Grafana

Grafana is a web-based visualization tool for monitoring and analysis. The software supports the creation of dashboards. A dashboard is divided into rows which can be created, rearranged or deleted. The user can add so-called panels to the rows. These panels can be different types of visualizations like the graph panel and the single status panel which will be described in this chapter.
The user can choose the time range he wants to view. Grafana supports multiple data sources for visualization such as MySQL, Graphite, InfluxDB or Elasticsearch. [2] In this bachelor thesis only Graphite and Elasticsearch are used.

1.4.4 Kibana

Kibana is a browser-based visualization and management tool for Elasticsearch. Kibana provides a GUI for trying out queries, but also the creation of different types of graphs such as histograms.

Chapter 2

Deployment & Development

This chapter covers the creation of an event processing pipeline, the dashboards and the access security to the platform. Figure 2.1 shows the parts of which the current monitoring platform consists.
On the left side we have the BrainScaleS system, covered in section 1.3, which sends metrics to Graphite. We also have the "Visions Network" in the top center consisting of all the servers and computers in the network which interact with the BrainScaleS System (i.e. job allocation, power cycling,..). This network itself is also monitored and sends events and metrics to the monitoring storage represented below the Visions Network box. The aggregation is described in the next chapter. Finally, Grafana accesses data from monitoring storage and visualizes it. This part is covered in Section 2.2.
During the precedent internship Elasticsearch as a new event database and Grafana as a visualization tool were deployed. This bachelor thesis focused on the creation of an event pipeline for the interaction between the network and the BrainScaleS System as well as the creation of different dashboards for monitoring using Grafana.

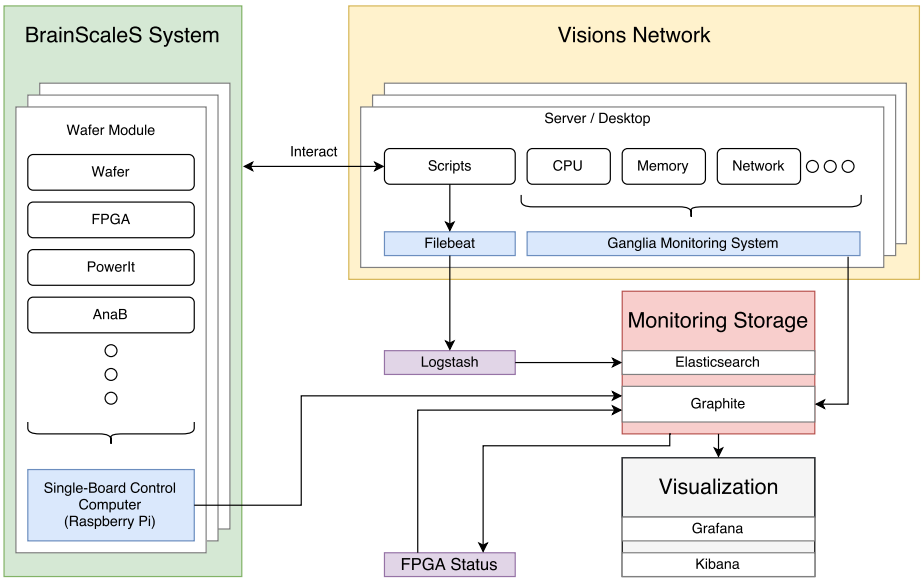

Figure 2.1: Diagram of the monitoring system showing different parts of the monitoring system.

2.1 Event Processing Pipeline

During the bachelor thesis the need for aggregating and storing different events emerged. For this, a pipeline for processing events was designed. Figure 2.2 portrays the pipeline which is explained in this chapter.

2.1.1 Aggregation

In order to simplify the aggregation it was decided to write events as a system log on a server and send the logs to Logstash using Filebeat.
Events like the on and off switching of a FPGA or its allocation to a SLURM job are written to the server *hel* into a file */var/log/elasticsearch/*.log*, where the star can be an arbitrary valid file name.
A Filebeat daemon was installed on *hel* which watches all .log files in the directory */var/log/elasticsearch*. Whenever a new line is noticed by Filebeat it is send to Logstash.
For an easy and understandable processing of the events logs by Logstash a logfile format which should be obeyed was introduced. The format demands

1. Every event type, like the FPGA switching, should have their own file
2. Every event should be written in one line

3. Every line starts with the syslog base
   `MMM dd hh:mm:ss {SOURCE} {SCRIPT}: {LOGGER}`
   An example of such a base is
   `Aug 24 14:06:38 hel fpga_remote_init.py: hwlog`

4. The events are written after the syslog as key-value pairs.

5. The keys and values are separated by an equal sign (=) and all pairs are separated by a space ( ). Keys should not contain any special characters and especially spaces. Values which are strings should be surrounded by double quotation marks ("). Newlines and quotation marks have to be escaped.

An example of full log file line is in Listing 2.1.

Listing 2.1: Example of a log file line

```
Aug 24 14:06:38 hel fpga_remote_init.py: vislog
Wafer=20 FPGA=07 State=1
```

This format not only allows an easy implementation for the user wanting to log information, it also allows an easy processing of the event data by Logstash described in the next chapter.

2.1.2 Processing

The Logstash daemon was deployed on monviz and listens on port 5044 for incoming log lines. The incoming line is split into the syslog part and the remaining part, which contains all the key-value pairs. Using the kv-Filter from Logstash the remainder is split into key-value pairs. The timestamp from the syslog is converted into a elasticsearch timestamp. As such the example from listing 2.1 would translate into the following JSON-object.

```
{
    "@timestamp: 2017-08-14T12:06:38,
    "logsource": "hel",
    "program": "hwlog",
    "Wafer": "20",
    FPGA: "07"
}
```

This JSON-object is then send to Elasticsearch. The configuration file for the log file processing can be found in the Appendix 4.1.

2.1.3 Storage

Once the JSON-object is received from Logstash it is saved to the Elasticsearch storage and is now queriable. The full pipeline is represented in 2.2.

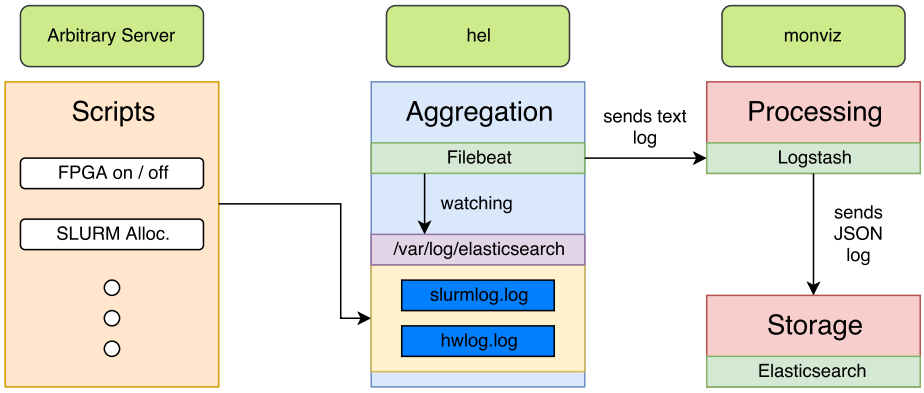

Figure 2.2: Diagram of the monitoring system. [6]

2.2 Creation of Monitoring Views

Having a clean visualization of a system allows an easy and fast administration. Therefore, developing views for various components were one of the key goals of this bachelor thesis. This section covers the development and motivation of different views. All views were created in Grafana, described in subsection 1.4.3. The entry point into Grafana is the "Overview"-Dashboard. As the name suggests, this dashboard allows the user to see the status of all Reticles and FPGAs of all wafers at a glance. The user can click on an FPGA / Reticle / Wafer which leads them to the corresponding dashboard, where they receive more information. All dashboards are described in this section, except for the "Overview"-Dashboard which, due to its nature and complexity, received its own section in 2.3.
For a better readability the screenshots for all dashboards were moved to the end of this section. To improve the printing the dashboard colors were switched to a white theme. In production the standard background color of Grafana is black.

2.2.1 Wafer Module

The "Module"-Dashboard gives the user an abstract overview of the status of the wafer module. The dashboard is divided into four rows. The first row features the selection of a wafer and links to more detailed dashboards of different components, i.e. AnaB or PowerIt.
The second row shows the most important temperatures of the wafer module, i.e. minimal and maximal temperature of the wafer or MainPCB temperature. Depending on the value of the temperature, the background color of the panel changes to yellow (warning) or red (critical) if certain thresholds exceeded. The threshold values depend on the component. For example, the wafer's tempera-

ture is deemed to be critical above 65°C, while an air temperature above 40°C is already considered to be critical.
The third row shows the power status of the reticles as well as the status of the FPGAs. Next to it, the temporal evolvement of the PowerIt current at the 48V and VDD points is displayed. Next to it, a graph of the wafer's minimal, maximal and average temperature is shown. The third graph counts the number of job allocations for the Reticles.
The fourth and last row shows a bar chart with the temperature of each FPGA. They are sorted depending on whether they are off, on or in a critical status (temperature above 60°C).

2.2.2 Wafer Temperatures

The "Wafer Temperatures" dashboard gives an overview of the key temperatures of the wafer system and can be reached by clicking on a temperature panel in the "Module"-Dashboard. In the top row on the left the temperature of the surrounding air is displayed whilst on the right five temperatures of the MainPCB board.
The second row features the temperatures of the wafer itself as well as the temperatures measured from the top cover.
The third row includes the temperature of the AnaB module as well as the speed of the fans mounted on the wafer system.

2.2.3 AnaB

Like the "Wafer Temperatures" dashboard the "AnaB" dashboard can be reached from the "Module" dashboard. It features the three panels with one presenting the AnaB temperature, one presenting the main voltages and the third one presenting corresponding voltages at the shunts.

2.2.4 PowerIt

The "PowerIt" dashboard consists of six graphs. Again, they show the most important voltages and currents of the PowerIt module as well as its temperature. As of now, many of these sensors are not yet calibrated (i.e. the temperature of the PowerIt is at constant 3°C) so the use of the dashboard is limited. Seeing relative changes might still be useful, though.

2.2.5 FPGA

The FPGA dashboard can be accessed from the "Module" dashboard. Its purpose is to show in-depth information about the state of FPGAs where the user can choose the wafer and one or multiple FPGAs. Three metrics which are measured from FPGAs are its temperature, the RTA(round time average, average time a package needs from the client to the FPGA and back) and the package loss. These metrics are displayed in three separate graphs in the second row.

Additionally, the temperature of the corresponding FCP is displayed in the temperature graph.
The user can choose to display events concerning the FPGA. As of now, only the power state change is supported. These events are displayed as vertical lines in the graphs. An application of this is shown in Figure 2.14. Additionally, the current temperature and the current RTA of all FPGAs are displayed at the bottom.
Using the button at top right the user can switch to the linked Reticle. Since the Reticle number and FPGA number are different one can not simply use the build-in linking feature of Grafana, which normally allows to link between two dashboards. Therefore a link to a webpage which routes the user to the correct reticle has been created.

2.2.6 Single Reticle

The "Singe Reticle" dashboard can be accessed from the FPGA dashboard. The top row is a panel showing the power status of the currently viewed reticle. The second row offers an overview over all recorded voltages. The left half shows a plot of voltage drains (V_{DD}) while the right side shows the voltage at common collectors and the high / low output voltages. Additionally, the minimal, maximal, average (over the chosen time period in the upper right side) and current voltage are listed in table.
The left plot shows seven voltages. The user, however, can choose to click on one or multiple of these to filter these out.

2.2.7 Cluster

The "Cluster" dashboard is not related to the wafer system. It is supposed to give a general overview of the state of the servers and nodes in the Visions network. The dashboard is separated into two rows. In the first row two graphs are featured showing the load on the frontend (servers and desktop computers) and the network nodes. These graphs might look confusing but abnormalities are easy to spot. Additionally, two tables listing the top users of SLURM (ordered by number of SLURM jobs) are displayed.
In the second row a graph showing the traffic throughput of all connected devices is included. Next to it the RTAs (round time averages) of all KIP nodes are displayed. Two more graphs plot the package loss of all cluster devices and kip nodes which have a package loss of more than 0%. This allows a fast identification of problematic nodes and devices.

2.2.8 Device Information

The purpose of the "Device Information" dashboard is to give the user an overview of the state of a device. The first graph in the upper left side shows the load on the device as well as the 5- and 15- minute weighted average of it so

that trends can be easier identified. Next to this a graph showing the CPU usage (in percent) of the user, system and IO (input/output) processes is placed. Furthermore, key characteristics of the device such as RAM size, clock speed, number of cores and the total disk size as well as its occupation in percentage are shown.
In the next row on the left the composition of the memory usage (in percent) and the CPU temperature are plotted. The last row features important information regarding the device's network activity such as the number of bytes and packages coming in and out as well as information about the Round Time Trip (average, maximum, minimum).
Most devices do not deliver all metrics at the same time. In such a case, the graphs are left out empty.

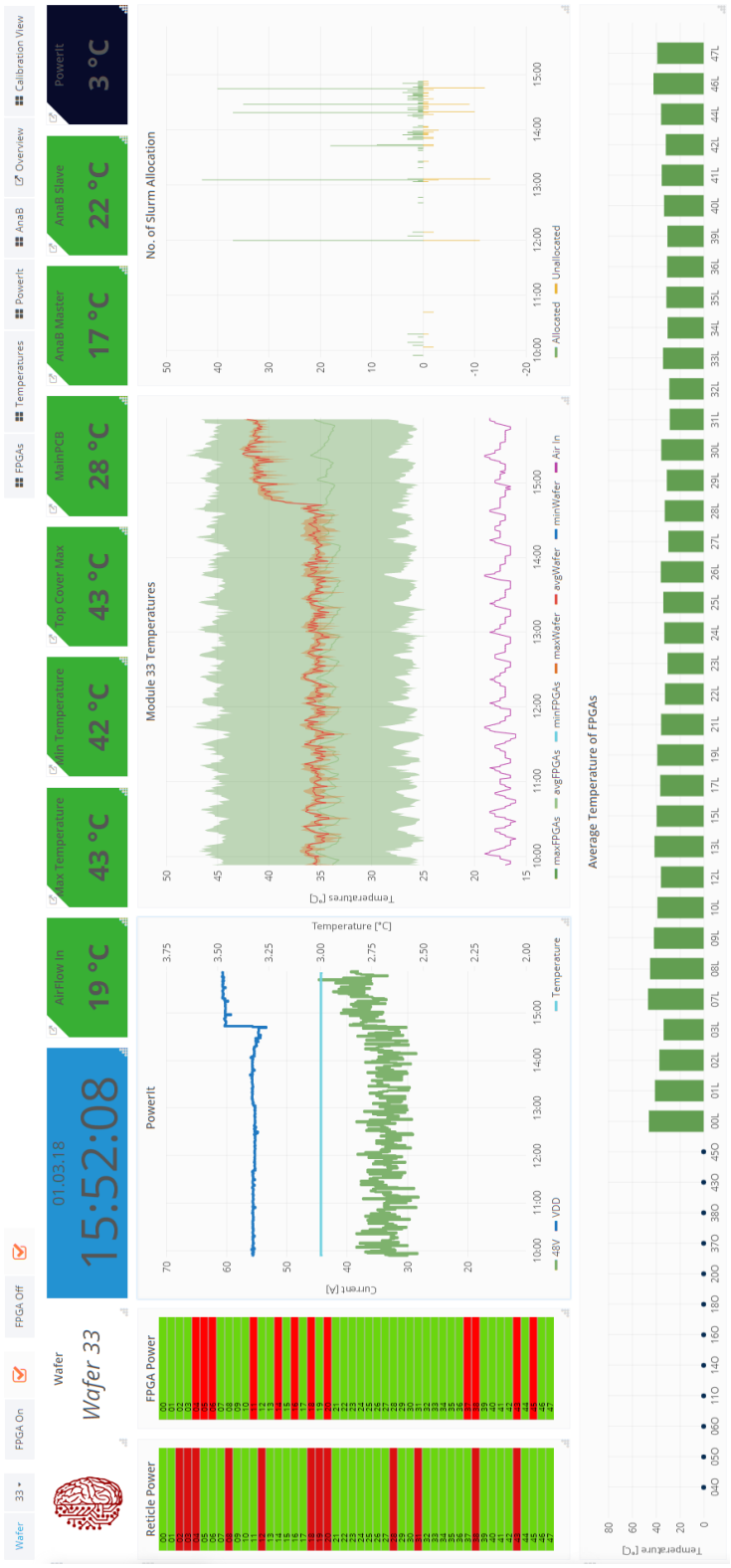

Figure 2.3: Screenshot of the module view showing all graphs and panels which give the user an abstract view of the wafer state.

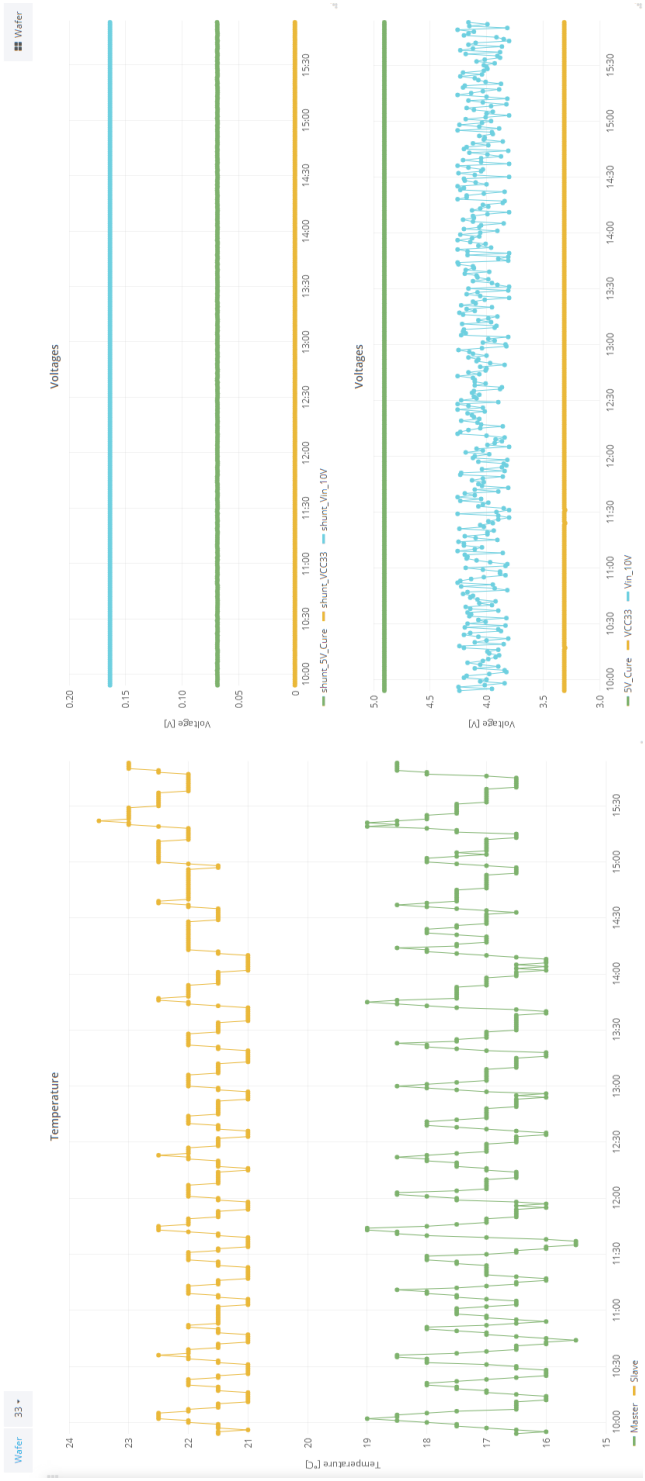

Figure 2.4: Dashboard of the AnaB module with key voltages being displayed.

Figure 2.5: This screenshot shows the important metrics for the PowerIt module. As we can see the temperature sensor is not calibrated as it shows a constant temperature of 3°C

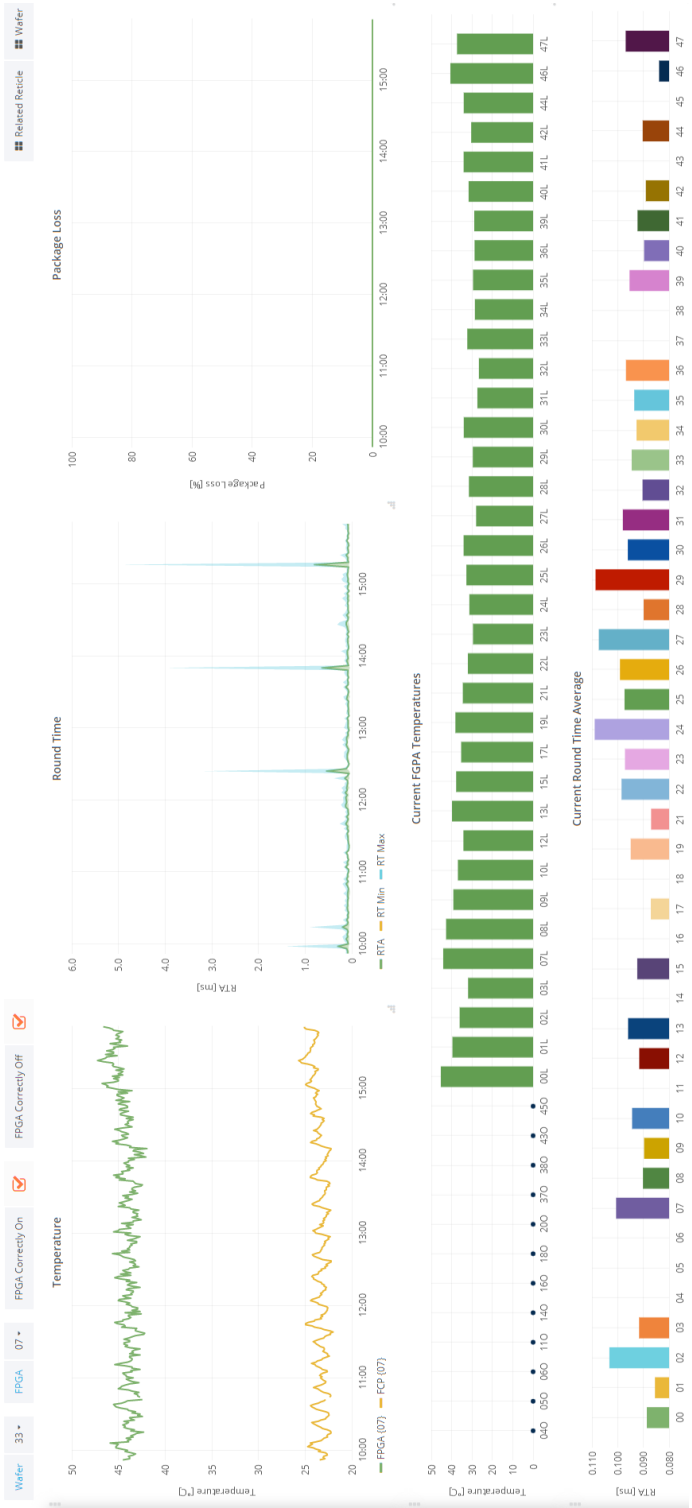

Figure 2.6: Exmple of an FPGA dashboard with one activated FPGA. We can see a zig zag pattern in the temperature. Interestingly, this pattern can be found across many different devices. Some assume that these come from the air conditioning powering on and off depending on the temperature.

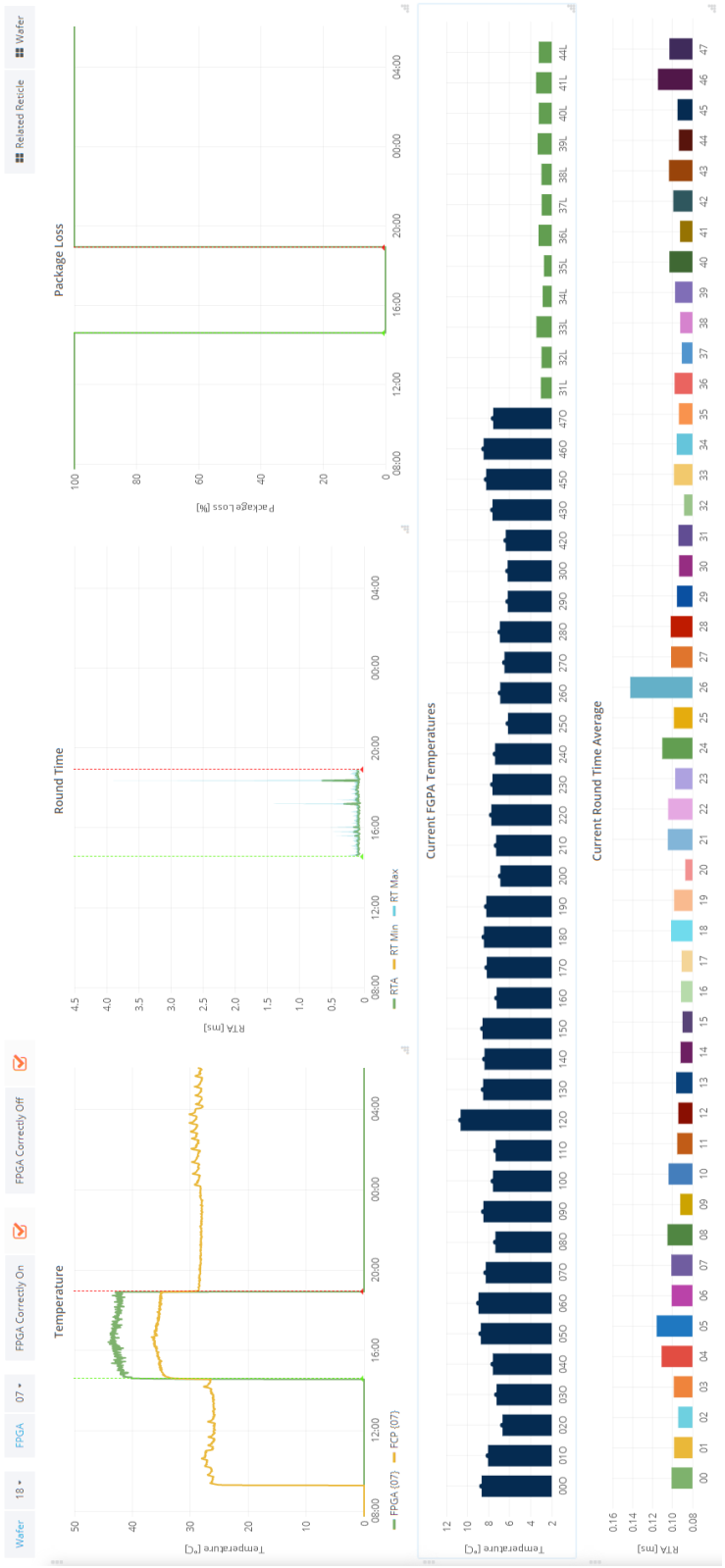

Figure 2.7: Another screenshot of FPGA dashboard with events (power up and shut down) being marked by green and red line. Round Trip Time stops being measured and temperature of FCP goes down when FPGA shut down.

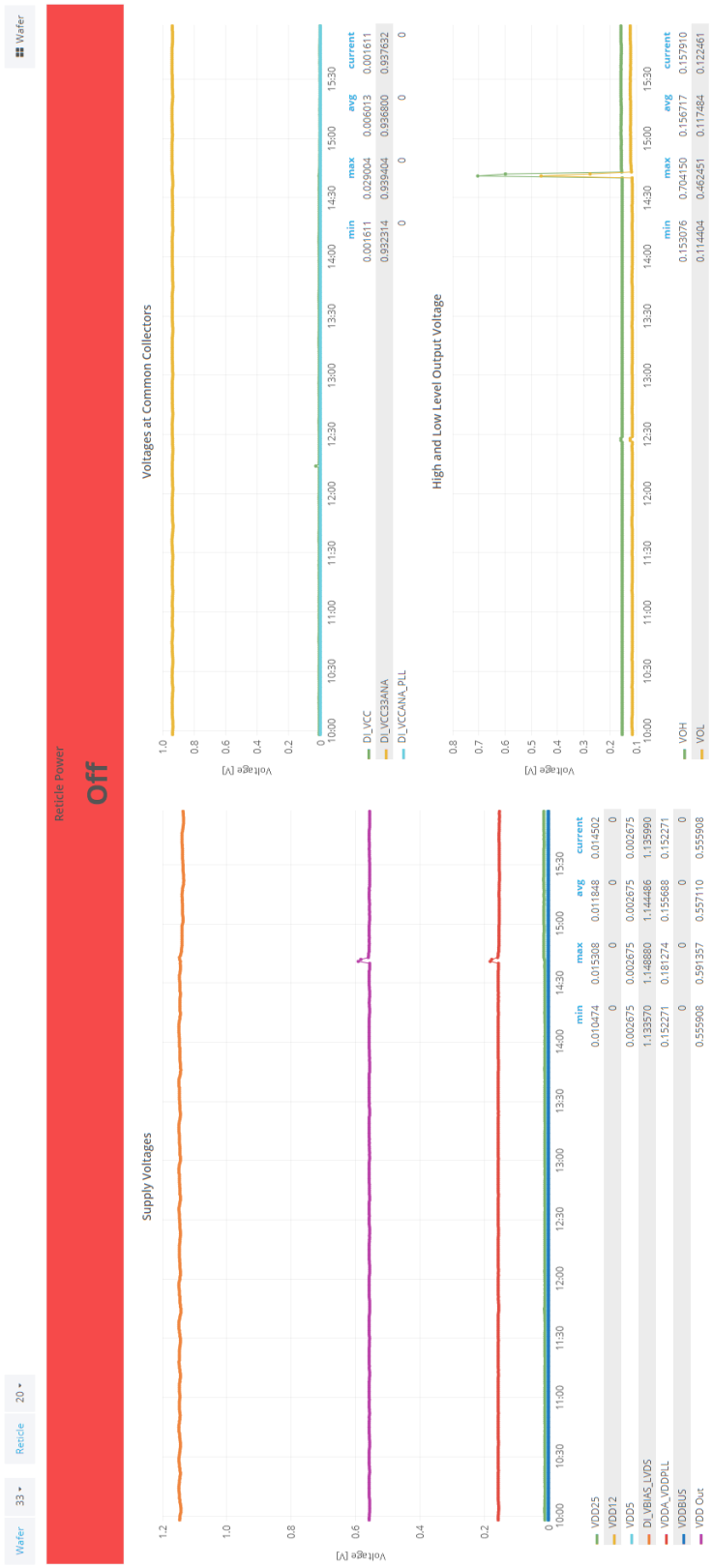

Figure 2.8: "Reticle" dashboard showing voltages of an reticle which is off.

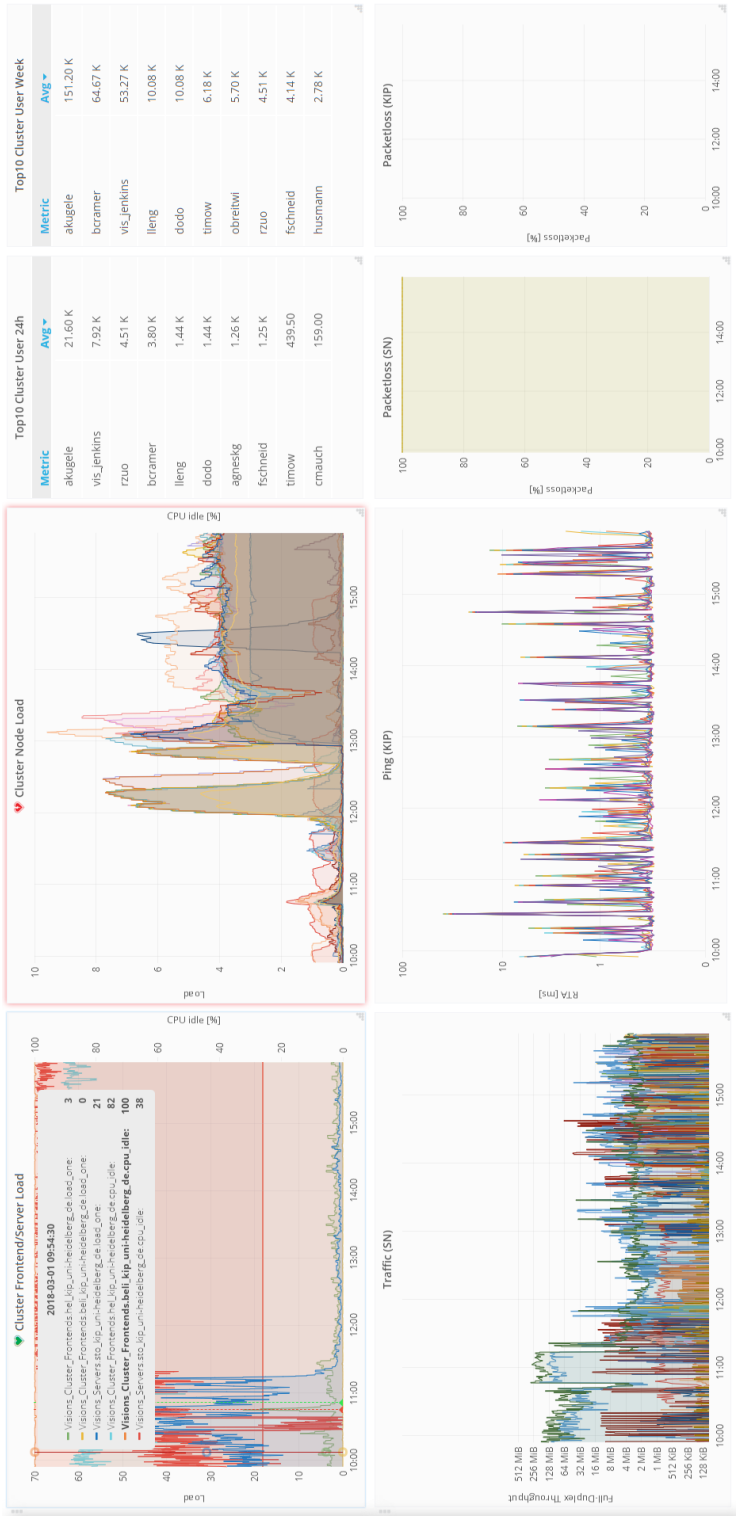

Figure 2.9: "Cluster" dashboard showing the state of the cluster. Interestingly, the load of some nodes exhibit the behavior of a leaky integrator

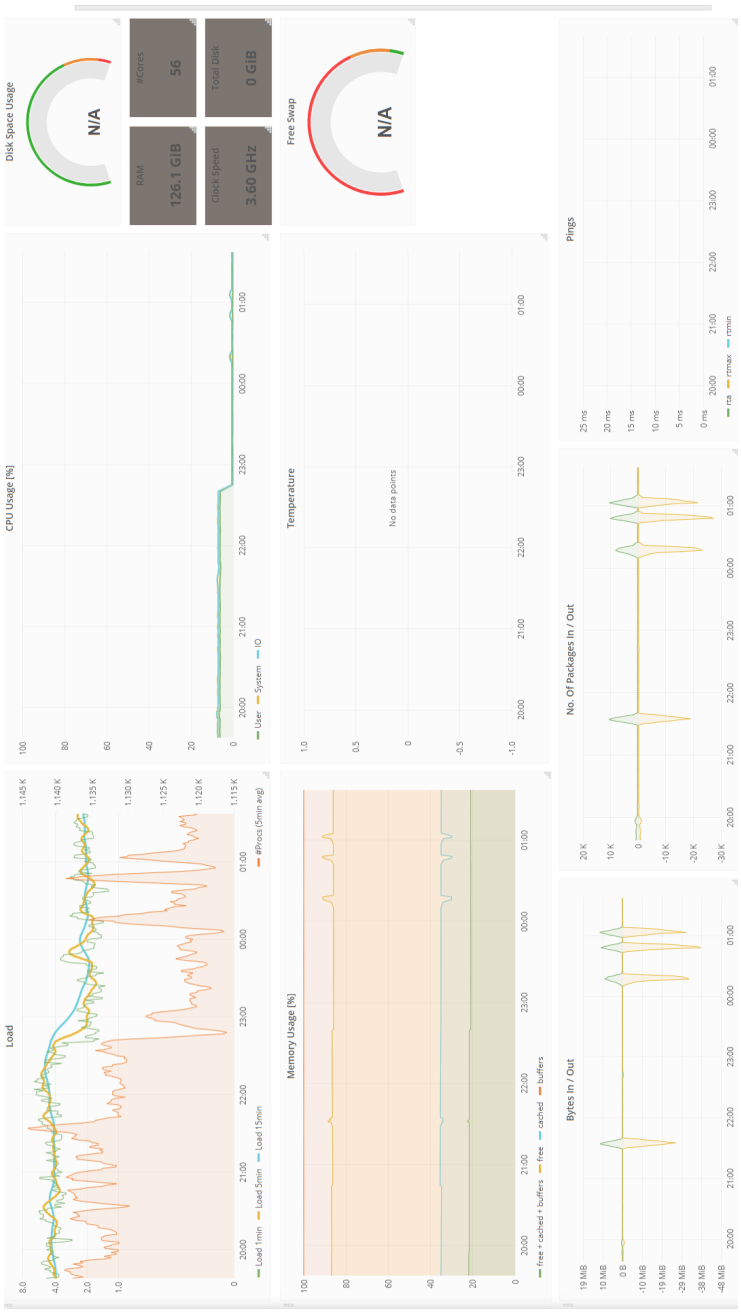

Figure 2.10: "Device" dashboard showing different information regarding the activity of a desktop computer. Not all information was available for display for this device.

2.3 Overview Dashboard

One of the key views that needed to be developed was an overview for the status of all FPGAs and Reticles of all wafers. However, none of the available panels for Grafana were suitable for this. Therefore, a new panel had to be developed.

2.3.1 First iteration

In the first version a Python script was started using a CRON job every minute which queried the package loss and the temperature of the FPGA and also the power status (on/off) of the reticles from *shinviz* and saved the results as files. This reduced the workload on the database as they were only queried once a minute instead of being queried every time a user started the overview.
The queried information was then processed in the browser. If the temperature was above 0 (meaning that the FPGA sensor is actively sending the temperature metric) and the package loss was 0%, the FPGA was on. The color green was assigned for this status. If the package loss was 100% or less but above 0% and a temperature value was received, a problem with the network or FPGA was assumed. This was coded with the color yellow.
If the temperature was received but the package loss was below 100% an error in the temperature sensor was assumed, being displayed as blue. If the package loss was 100% and the received temperature was 0%, the FPGA was considered to be offline.
The status were aligned rectangular, with the FPGA's number going in direction X and the Wafer's number going downwards Y-direction.
Additionally, the power status of the corresponding reticle was added. A screenshot of the first overview was added in Figure 2.11.

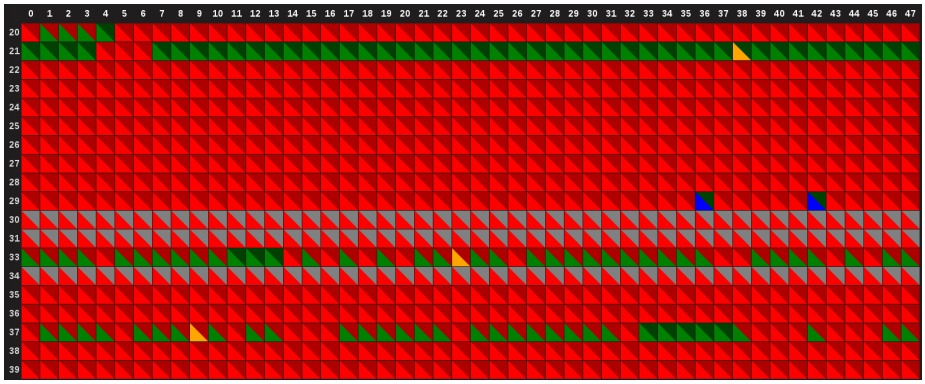

Figure 2.11: Screenshot of the first iteration of an FPGA / reticle overview

2.3.2 Second Iteration

The first iteration was only capable of showing status quo of an FPGA. It did not take into account that there might be a desired status for an FPGA, i.e. the FPGA should be on or off (*PowerState*) and whether it initialized this state successfully (*InitState*). For this, Christian Mauch developed a script which, whenever an FPGA is supposed to be on or off, records this event in Elasticsearch using the pipeline described in 2.1.1.
As many combinations are possible regarding the temperature values, package losses and the initiation state, a new metric *status code* was introduced. Each FPGA now has a status code assigned, a three digit number encoding its state.

- The first digit encodes the availability of the FPGA. All FPGAs are routinely pinged. If all packages are lost, the first digit of the status code becomes "0", if all packages are received the digit becomes "1", and if there is some loss it becomes "2".

- The second digits encodes the temperature of the FPGA. If the temperature of an FPGA is unavailable the digit becomes "0", if it is below 50°C it becomes "1" and if it is above 50 °C it becomes "3".

- The third digit shows if was shut down ("0") or powered on ("1") correctly. If the *PowerState* and *InitState* do not match, the digit becomes "2". If such an event was not registered for the FPGA, the digit becomes a "3"

For example, "000" means the FPGA shut down correctly, whilst "111" means it powered on correctly and is running. The code "211" would mean that it was powered on correctly and the temperature is below 50°C, however there is some package loss regarding the FPGA.
Like in the first version a python script queries the databases and determines the status for all FPGAs every minute. To reduce the workload on the database and speed up the script only the states for the current day are queried whilst states from the previous days are taken from a cache. At the same time, if a new desired state is received during the current day, it is added to the cache. The caching reduced the time to determine all status code from an average of 40 seconds to 6 seconds. All states are written to a JSON file which is exposed to the internet.
On the front-end, at the internet browser of an user, a JavaScript script retrieves the status codes. Like in the first iteration, the wafer and FPGAs numbers are aligned rectangular, with each rectangle representing an FPGA in the top half and its corresponding Reticle in the lower half. The top half is colored conformable to its state, i.e. "111" becomes green(no package loss, temperature below 50°C, correct initialization), "000" becomes red (full package loss, no temperature sent from Raspberry Pi, shut off correctly) and "211" becomes yellow indicating a problem with the FPGA connectivity. Additionally, the status code is displayed as a number if it is not in an expected state. Like in the first version, the lower half of the rectangle shows the power state of the Reticle.

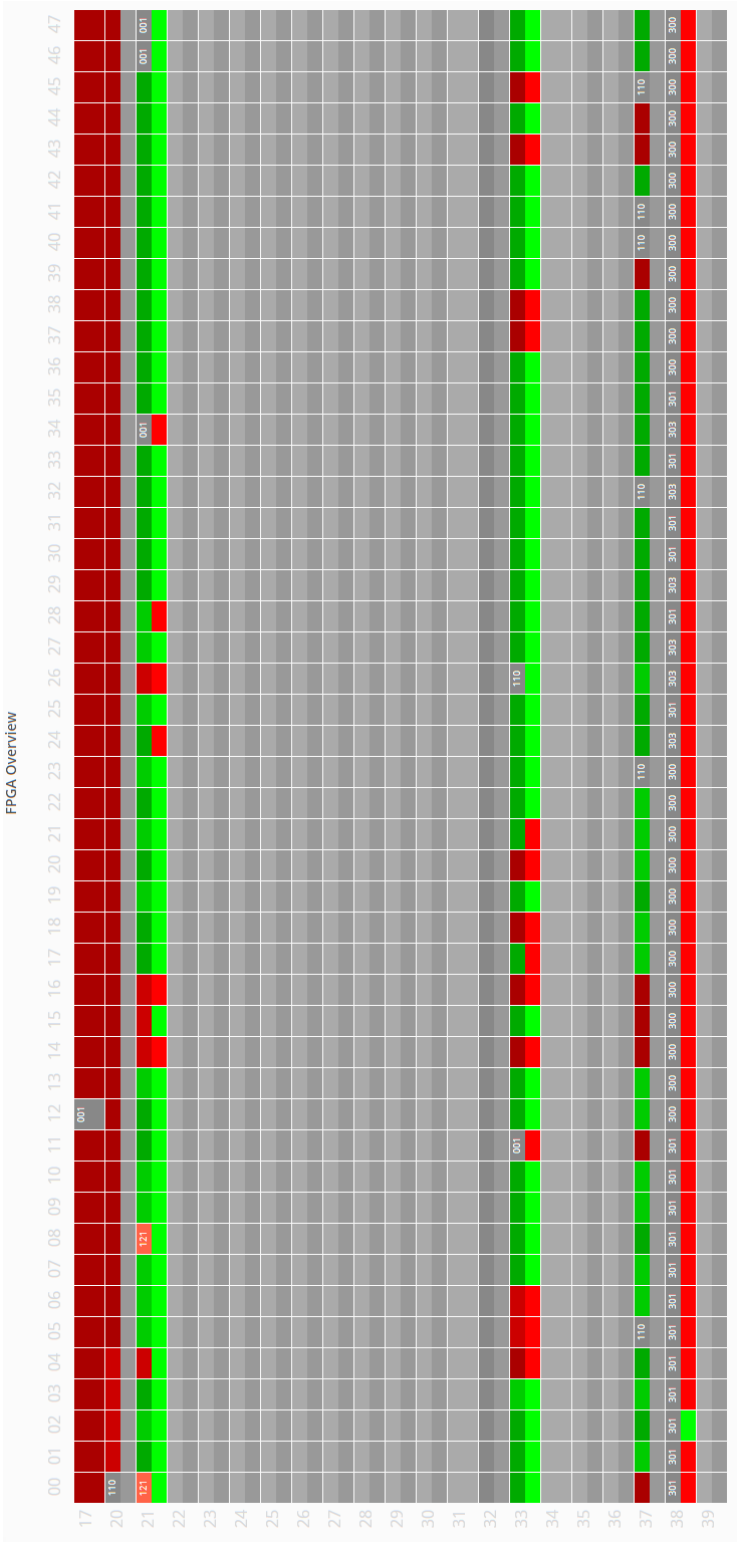

Figure 2.12: Second and final design of the new FPGA / reticle overview.

2.4 Exposition and Security

For administrators and users it is important to be able to access monitoring not only from inside the internal network of KIP, but also from the "normal" internet. For this, Eric Müller set up a reverse proxy from `monviz.kip.uni-heidelberg.de` to `brainscales-r.kip.uni-heidelberg.de:12443`. This however, meant that the Elasticsearch database and our monitoring software was exposed to the general public and therefore also to malicious users. Attacks on unsecured Elasticsearch databases were already reported. [8]

Grafana

Grafana has a built-in LDAP support. Grafana was configured to use the BrainScaleS LDAP server for login, which allows the user to use the same credentials as for GitViz or Gerrit.

2.4.1 Elasticsearch and Kibana

Elasticsearch and Kibana do not offer built-in security. Therefore, any user who has access to the server running Elasticsearch can perform any kind of operation he likes, including deleting or creating datasets. [1] There are two different security solutions, though, which allow the creation of users and management of their rights.
The first security solution is called "X-Pack", developed and maintained by the company behind Elasticsearch. Unfortunately, it is not free. Therefore, it was decided to use SearchGuard, an open source security plugin. SearchGuard license allows the use of Basic HTTP Authentication, but not LDAP. Therefore each use case gets its own user with the rights it needs in accordance with the *principle of least privilege* [5].

2.5 Alerting

A notification channel for Mattermost was created (called "Monitoring Alerts") and a Grafana Bot which can send notifications about status of the hardware was set up. An example notification is shown in Figure 2.13. Right now, only notifications about exceeding temperatures are implemented.

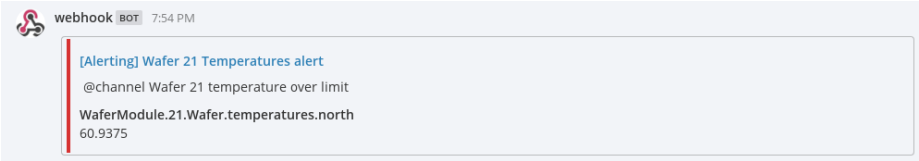

Figure 2.13: Example of a notification for exceeding wafer temperature in Mattermost.

In the long run more notifications about the components should be implemented, as well as an email notification and, in critical cases, even an SMS notification is conceivable. Grafana, however, should not be used for critical alerting as there are too many points of failure in the alert chain, i.e. connection error, server crash or loss of power.

2.6 Calibration Overview

Alexander Kugele developed during his master research phase a calibration overview for the wafer. These files now have been integrated into Grafana. They are copied regularly from Kugele's folder to the monviz server so they can be viewed publicly. Then they are included in the "Calibration Overview" in Grafana using an iframe.
The use can choose the calibration file which corresponds to the wafer of his interest.

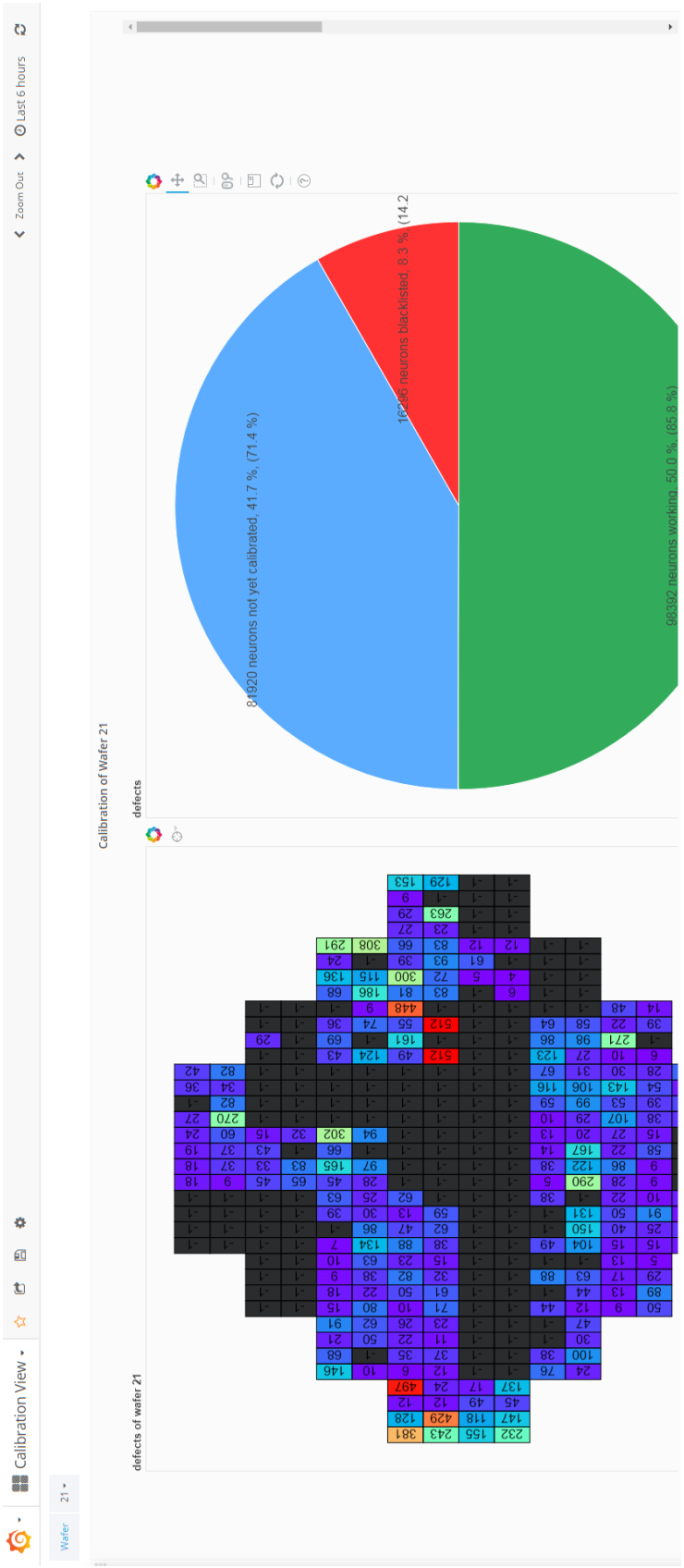

Figure 2.14: Alexander Kugele's calibration data included in Grafana.

Chapter 3

Summary and Conclusion

3.1 Summary

In this thesis and the preceding internship the groundworks for a modern monitoring platform, with Grafana as the central visualization tool, were developed. An event pipeline was developed which allows a streamlined aggregation, processing and storage of event data. Using Grafana, multiple dashboards for the different components of the BrainScaleS system were designed which allow a faster and simpler administration. An overview of the states of all FPGAs and Reticles was developed which uses a newly introduced status code. This status code encodes the state of an FPGA by evaluating different metrics such as network availability and temperature.
Additionally security measures were introduced which control the access to the database and Grafana. A basic alerting has been setup which notifies administrators of unexpected behaviors. It is, however, rather limited in the amount of supported metrics and alerting threshold.

3.2 Conclusion

Several things can still be improved on this platform. As of now the "Overview dashboard is just an iframe which binds an HTML-file, served by an nginx server. This solution is more of a hack rather than a proper Grafana plugin. A better way of implementing this would be the creation of a Grafana plugin using the official API. At first, the iframe solution was supposed to be used for prototyping, but due to an incomplete API documentation and time constraints this solution stuck around. A second thing that could have been improved is the determination of the FPGA status code. As of now, this code is simply determined by a Python script being started by CRON. Unfortunately, CRON can only start a program once a minute. If the Python script was made as a *systemd* Service, shorter refreshing periods would be possible. The main challenge during this work very often was not the creation of the dashboards but to learn the

many ways the databases and tools like the security plugin work in the backend.

3.3 Outlook

This bachelor thesis is only the beginning of a centralized monitoring system. With *monviz*, a foundation for a solid monitoring framework was laid down, which allows administrators to build upon it. Monitoring is never finished and always a work-in-progress, as the needs evolve with the development of the observed system. Therefore, the dashboards should be viewed as something to be expanded and built upon. As the BrainScaleS is becoming an open platform for scientists, errors will arise which were not thought of, as they might use the platform in a way not yet thought of. Therefore, the platform will need an extensive monitoring so that administrators will be able to respond to problems quickly. Several things are already planned for the time after this bachelor thesis.
For example, an overview for the job allocation of Reticles is already planned. The job allocation data has been made available using the developed pipeline and now needs to be visualized. There were already plans on doing the visualization in this thesis. These plans were postponed as the time to develop such an visualization would only allow the creation of a suboptimal solution. Plans to move *monviz* to a more stable and more powerful server are already in work. Additionally, the relocation of the time-series database Graphite from *shinviz* to *monviz* is planned which would unify monitoring-oriented databases on one server. Some users started experimenting with the Elasticsearch database as a storage for event data and hopefully more users will start to use it in the future. Monitoring profits from a large and meaningful dataset which still needs to be expanded.

Chapter 4

Appendix

4.1 Abbreviations

Reticle	FPGA
API	Application Programming Interface
FPGA	Field Programmable Gate Array
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
REST	Representational State Transfer

Table 4.1: Lookup table for the FPGA to Reticle connection and also the corresponding IP

4.2 Logstash Configuration File

Listing 4.1: Configuration file for Logstash

```
input {
    // Listen for log events coming from Filebeat
    beats {
        port => "5044"
    }
}

filter {
    grok {
        # Split the message into the syslog part and the rest of the text
        match => {
            "message" => "%{SYSLOGBASE} %{GREEDYDATA:remainder}"
```

```
        }
    }
}

filter {
    # The text is split into key value pairs.
    kv {
    }
}

filter {
    # Use the timestamp from the Syslog-part
    date {
        "match" => ["timestamp",
                    "MMM dd HH:mm:ss",
                    "MMM  d HH:mm:ss",
                    "ISO8601"]
        remove_tag => ["beats_input_codec_plain_applied"]
    }
}

output {
    # Save the data under the wafer-* index
    elasticsearch {
        index => "wafer-%{+YYYY.MM.dd}"
    }
}
```

Bibliography

[1] Elastic Documentation. `https://www.elastic.co/guide/index.html`.

[2] Grafana Documentation. `http://docs.grafana.org/`.

[3] Neuromorphic platform specification - public version. *Human Brain Project*, 2017.

[4] Slawek Ligus. *Effective Monitoring & Alerting*. O'Reilly, 2013.

[5] Jerome H. Saltzer. Protection and the control of information sharing in multics. *Communications of the ACM*, 1974.

[6] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. *Proceedings of the 2010 IEEE International Symposium on Circuits and Systems (ISCAS"10)*, pages 1947–1950, 2010.

[7] James Turnbull. *Art of Monitoring*. Self-published, 2016.

[8] Steven J. Vaughan-Nichols. Elasticsearch ransomware attacks now number in the thousands. `http://www.zdnet.com/article/elasticsearch-ransomware-attacks-now-number-in-the-thousands/`, 2017.

Statement of Originality

I certify that this thesis, and the research to which it refers, are the product of my own work. Any ideas or quotations from the work of other people, published or otherwise, are fully acknowledged in accordance with the standard referencing practices of the discipline.

Ich versichere, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, 05.03.2018 _____

(signature)