Electrical High Rate Setup

Master Thesis by

Stephan Wiederkehr

Department of Physics, ETH Zürich

Supervisor:
Prof. Dr. Roland Horisberger

Co-Supervisor:
Dr. Wolfram Erdmann

Sunday 15th February, 2015

Abstract

To gain deeper insight into particles it becomes inevitable to increase the center of mass energy of particle collisions. Unfortunately, huge and complex experimental setups have to be built for this purpose. However, to accurately test such detectors the real circumstances must be reproduced as good as possible while consuming as little space and time as possible. This thesis presents the most recent attempt to do so in the particular case of the CMS pixel detector.

A recently built prototype showed the possibility to induce charge in a read out chip via pulse generators in a controlled manner without the necessity of a sensor. Thus, it laid the path to build a more sophisticated electrical high rate setup. To gain control over the signals to be induced and to be independent of an external pulse generator, a field programmable gate array chip was included in the setup. Such a highly tunable facility is of great use to test specific situations at will, such as e.g. chip efficiency.

The control over the cluster pattern could be demonstrated in tests mapping the cluster position on the read out chip. Current measurements revealed a slope of the digital current consumption of roughly 0.083 mA per $\frac{MHz}{cm^2}$ which corresponds well to estimates performed earlier. Furthermore, the efficiency tests revealed a lower efficiency for rates up to 150 $\frac{MHz}{cm^2}$ than obtained with X-rays, but a lower gain with increasing rate. The efficiency was found to peak around the design WBC values before decreasing for higher and smaller WBC values.

Summarising, the new setup was found to deliver data within the estimates for data provided by the actual experiment which could be verified particularly well in the case of the current measurements.

Contents

1 Introduction

1.1 LHC and CMS

The Large Hadron Collider (LHC) was built to find and study hitherto inaccessible but theoretically predicted particles or to disprove their existence. It was built during the years 1989 – 2008 by the European Organization for Nuclear Research (CERN - Conseil Européen pour la Recherche Nucléaire). The two beam pipes accelerating bunches of protons in opposite directions were designed to provide proton energies of up to 7 TeV per beam. In April 2012, a centre of mass energy of 8 TeV was achieved, i.e. an energy of 4 TeV per proton. To increase the amount of data produced by the LHC, the centre of mass energy is set to be increased even further to about 14 TeV after the first long shutdown (LS1). In addition, the separation of particle bunches will be reduced from 50 ns to 25 ns.
In the LHC synchrotron, the two beam pipes carrying the particles traveling in opposite directions cross at four points to provoke particle collisions. The ALICE, ATLAS, Compact Muon Solenoid (CMS) and LHCb experiments are located at those crossing points, whereas CMS and ATLAS are the two largest and the most versatile detectors. LHCf, MoEDAL and TOTEM are further detectors situated close to the collision points and were build for very specific tasks.

The CMS experiment consists of several subsystems, one of which is the tracking system consisting of a pixel detector and a silicon strip detector. Its task is to provide information about the particle's path through the detector. Since the magnetic field penetrating the experiment is known, the path curvature can be used to calculate the momentum of the particle.
To cope with the high particle flux close to the beam pipe, the innermost detector is made of silicon pixels instead of silicon strips [1].

1.2 Pixel Detector

Due to the proximity of the pixel detector to the beam pipe, the detector must be able to withstand radiation damage as well as deliver accurate path information to distinguish the particles. Additionally, it must consist of as little material as possible to avoid multiple scattering of the particles. The detector currently consists of a barrel part with three layers cylindrically enclosing the beam pipe and 2–disk endcaps on each side yielding a total length of 1 m.
It is foreseen to replace the current system by a four layer barrel (with radii

30 mm, 68 mm, 109 mm and 160 mm) and a 3–disk endcap during the winter break 2016/2017.
The detector is built from modules, each consisting of 16 Read Out Chips (ROCs) and a silicon sensor to generate measurable currents for each particle passing through it. Both, the ROC and the sensor are divided into pixels. Each ROC contributes 52x80 channels of the dimension 100 μm x 150 μm to a total of 78.8 million channels for the whole barrel. Figure 1 shows schematically the current pixel detector and its upgrade [1].

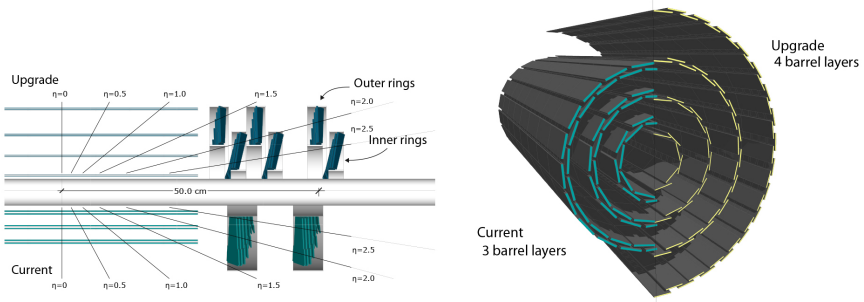

Figure 1: Left: A comparison of the current pixel detector (bottom) and the upgrade (top).
Right: A schematic drawing of the barrel part comparing the current and the upgraded detector [2].

1.3 Measurement Methods

Until the LS3 in 2022, the peak luminosity is planned to be increased by more than a factor two with respect to the current peak luminosity to reach $2 \cdot 10^{34}\ cm^{-2} \cdot s^{-1}$. This means, that the pixel detector must withstand even more radiation as well as being able to read out the data more efficiently. To deal with the increasing requirements, a new ROC is currently being tested to be installed in the layers 2, 3 and 4. At the time of writing, a ROC is being designed for the layer 1 to deal with the particularly high hit rates very close to the beam pipe.
To be able to test the ROC, the need for hit rates similar to those in the LHC emerges. Two presently available methods to generate high rates are X–rays and test beams. The latter are similar to the LHC but on a somewhat smaller scale, such that only X–rays are applicable within a typical laboratory. Yet, also X–rays are not optimal for testing purposes as one

photon only induces a hit in one pixel while in the CMS experiment particles penetrating the sensor typically induce clusters of pixel hits.
The Electrical High Rate (EHR) is the latest addition to those methods. It does not require a silicon sensor to generate hit induced voltages but sends electrical pulses directly to the ROC. The EHR setup can easily be placed in any laboratory and enables the user to program cluster patterns to be sent to a ROC.
As particle collisions inside the LHC happen randomly also the test methods should provide a hit rate in a random fashion. The two previously mentioned methods, namely X–ray setups and beam pipes, do fulfil this requirement. Programmable instruments, however, suffer from the impossibility of true randomness and have to rely on pseudo random number generators. With enough computational power, though, it is possible to minimise the gap between pseudo randomness and true randomness to a scale which makes this flaw irrelevant compared to other influences.

For testing purposes, though, randomly injected hits may not suffice. Only the ability to program specific hit patterns can make certain tests possible or decrease the measurement duration significantly.
The Pixel Unit Cell (PUC) already allows the injection of a calibration signal to emulate a hit. However, the calibration signal was not designed to emulate hit patterns. As a result, the programming of pixels can not be done fast enough to inject hit patterns in a timescale similar to the actual experiment.

1.4 Electrical High Rate

A first prototype of an EHR setup was built at the Paul Scherrer Institute (PSI) and tested at the Eidgenössische Technische Hochschule Zürich (ETHZ). It was presented in the thesis *Electrical High Rate Test* [3], which also explains the basic principles. Here only a short description of the basic principle will be given.
To induce hits on a ROC, electrical pulses are sent through wires touching a Kapton foil which lies on top of the ROC. The foil acts as a dielectric medium within a capacitor such that the charge induced on top of it will likewise be induced inside a PUC. The electrical pulses are generated by a Field Programmable Gate Array (FPGA) chip. The FPGA not only creates the electrical pulses but also manages the pulse patterns. It is controlled by a computer via an USB connection.
The wires are soldered to a Printed Circuit Board (PCB) which is connected

via high speed connectors to a separate PCB, which contains the FPGA chip as well as the USB connection.
The device is intended for a large variety of tests. In particular testing the power consumption and the efficiency, both as a function of the hit rate, delivers important information during the development of a ROC. Hence, while developing and eventually testing the device, the focus was on these measurements, beside some device related measurements.

The prototype was not able to address wires separately and hence control clusters individually. It also depended on an external pulse generator such that the signal quality was hampered by the connection between the pulse generator and the prototype. In the following a setup is presented which intends to erase both issues.

1.5 Field Programmable Gate Arrays

FPGA chips are designed to be reprogrammable after they were manufactured in contrast to Application Specific Integrated Circuits (ASICs). Commonly, an FPGA consists of an array of (configurable) logic blocks ((C)LB), which in turn contain a few logic cells, and Input/Output (I/O) pads. The LBs are wired to programmable switch blocks which are connected to each other as well. Therefore, the programming of the switch blocks determines the connection of LBs and I/O pads.
Typically, a logic cell contains a lookup table (LUT), an arithmetical logic block, and a D–type flipflop. Multiplexers installed in between these components enhance the versatility of the LBs.
On the user side, designing a FPGA configuration can be done either via a hardware description language (HDL) or by drawing a schematic. While the former has advantages for large or repetitive structures, the latter can be easier to visualise the design. It is also possible to include components designed with a HDL in a schematic.
In general, whenever a choice has to be made regarding the chip type, the difference in the production cost and the power consumption between FPGAs and ASICs play an important role in the decision. For small productions, however, the ability to reprogram the chip is a decisive advantage in favour of the FPGA [4].

2 Experimental Setup

Following the introduction in section 1.4, this section describes the experimental setup. Whereas the whole setup includes a computer, a digital test board (DTB) and the EHR device, the components which are particular to the EHR device are the following:

- The FPGA board controlling the signals.
- The metal block focusing the wires.
- The PCB connecting the wires to the FPGA board.

Figure 2 and 3 show pictures of the EHR device from different angles. Its components are described in more detail in the following sections. The FPGA board used in the EHR setup was the model ZEM4310 developed by Opal Kelly [5] who also developed the USB interface called FrontPanel [5]. Using the FrontPanel application program interface (API), a Python class was written to further simplify the usage of the EHR setup. The class is described in detail in the subsection 2.4 and a link to download it is given in the appendix. Since all functions are concentrated on the FPGA chip, a detailed description of its design is given in section 2.3.
The digital test board controls a single ROC or a module and its read out process. A brief description is given in section 2.5, yet a more detailed description lies beyond the scope of this report but can be requested from the designer [6]. Pictures of the EHR device are shown in the figures 2 and 3.

The single ROC is glued to a chip board which is fixed with two screws to the block of metal focusing the wires on the ROC. An adapter to connect the chip board to the DTB is contained in the EHR setup. Two springs leave the adapter some space for movement. Otherwise, the adapter would exert a force on the chip board which was observed to influence the connection between the chip and the wires.
The ROC can be exchanged by following the instructions explained in the appendix.

Please note that the FPGA class in the FPGA.py file is actually called ZEM, but to avoid confusion, the particular ZEM4310 [5] board used for the measurements is simply called FPGA board, as well as the ZEM class written to use the presented setup is called FPGA class throughout the whole thesis.

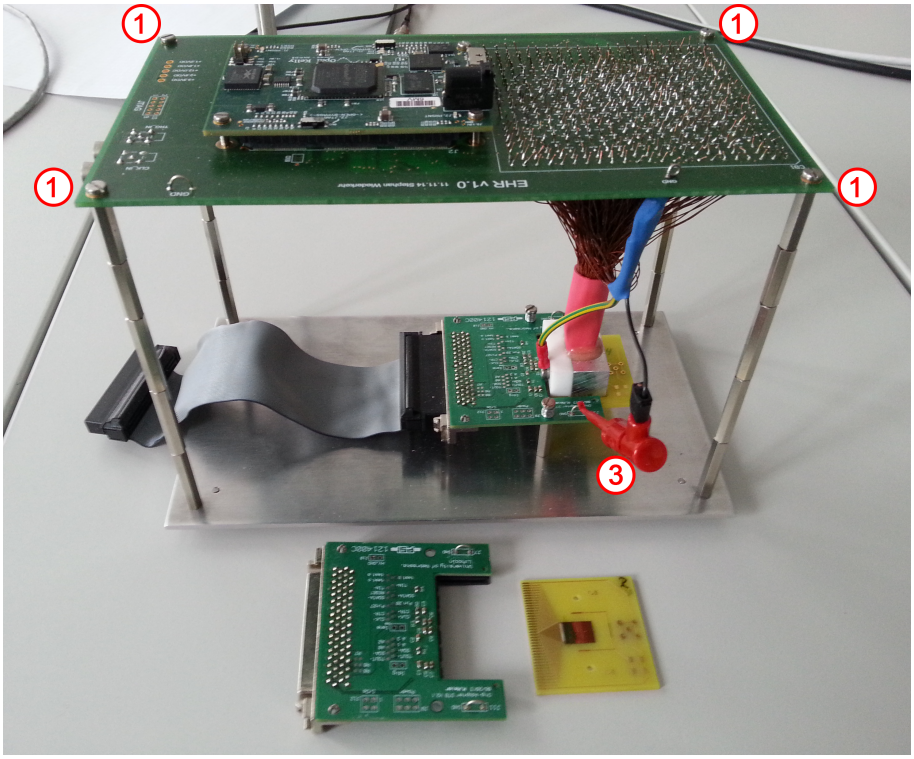

Figure 2: A picture of the EHR device. A single ROC adapter and a chip board are shown separately. The inserted numbers are mentioned in the text in the sections 2.2 and 6.

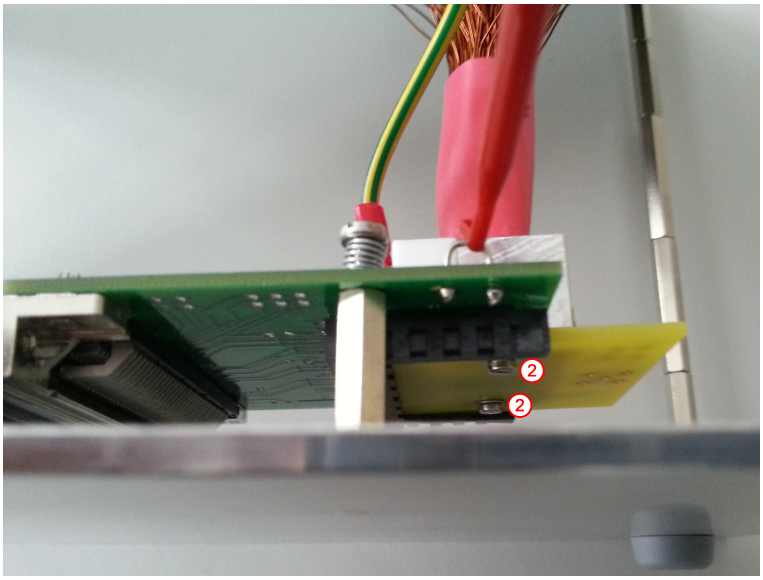

Figure 3: A picture of the EHR device showing the bottom side of the chip board while fixed to the setup. The inserted numbers are mentioned in section 6.

2.1 FPGA Board

The FPGA board features several components including an ALTERA Cyclone IV E FPGA, two Samtech connectors (QTH series), a 50 MHz low–jitter crystal oscillator and two user configurable LEDs.
The connectors supply voltage, which determines the output voltage, was set to 3.3 V by performing the operations described in the user's manual [7]. Even though the FPGA offers a maximum of 374 user I/Os and the Samtech connectors have 180 pins each, the FPGA board does only offer connections to 160 user I/Os. The remaining connector pins are use to transport other signals, e.g. clock signals, or are left unused.
To communicate with the device, FrontPanel introduces a concept based on *endpoints*. Each endpoint is essentially a data bus to transfer data from a computer to the FPGA and vice versa. The width of the bus depends on the USB version of the FPGA board, which in this case was USB 3.0 leading to a bus width of 32 bit. The endpoint concept extends to three connection modes: wires, triggers and pipes. Wires transport data asynchronously between host and target while triggers are synchronised to an endpoint clock. Pipes simplify the transfer of large data packages. In the following, these

wires, triggers and pipes related to the FrontPanel interface will be called ep_wires, ep_triggers and ep_pipes, respectively, to avoid confusion. In the EHR setup only ep_wires were used to transfer data [8].

2.2 Connection PCB

The Connection PCB offers 529 through-hole connections foreseen for wires. On the other hand, the FPGA board only offers 160 user configurable I/Os. One of these was foreseen as a trigger input, such that only 159 user configurable I/Os were used for the wires. Therefore, 159 of the 529 PCB through-holes, foreseen for the wires, were connected to the FPGA while the remaining through-holes were connected to ground. The through-holes connected to FPGA I/Os are labelled using the name of the FPGA board connection, e.g. J2_62.
A complete list linking the through-holes in the PCB to the wire number and to the FPGA connection can be found in the appendix. Connection pads for two 50Ω 0806 SMD resistors were designed to terminate the trigger input and the clock input. However, when used with the DTB, these pads are redundant due to the output termination of the DTB.
Some wires were connected to the ground connection linking the PCB and the DTB adapter grounds marked as 3 in figure 2. The motivation was to facilitate the substitution of wires in case one wire was found not working or not pointing to the active area of the ROC. The latter problem existed because the bundle of wires was chosen to be wider than the active area of the ROC to guarantee its whole coverage.
Summarising, the PCB features the following connections (figure 4 shows a schematic of the PCB):

- 529 through-hole connections, for the wires pointing on the ROC. 159 connections are linked to the FPGA I/Os, while the remaining connections are linked to ground.
- A trigger input.
- A clock input.
- A Joint Test Action Group (JTAG) connection foreseen for debugging.
- Several reference power out connections offered by the FPGA board.

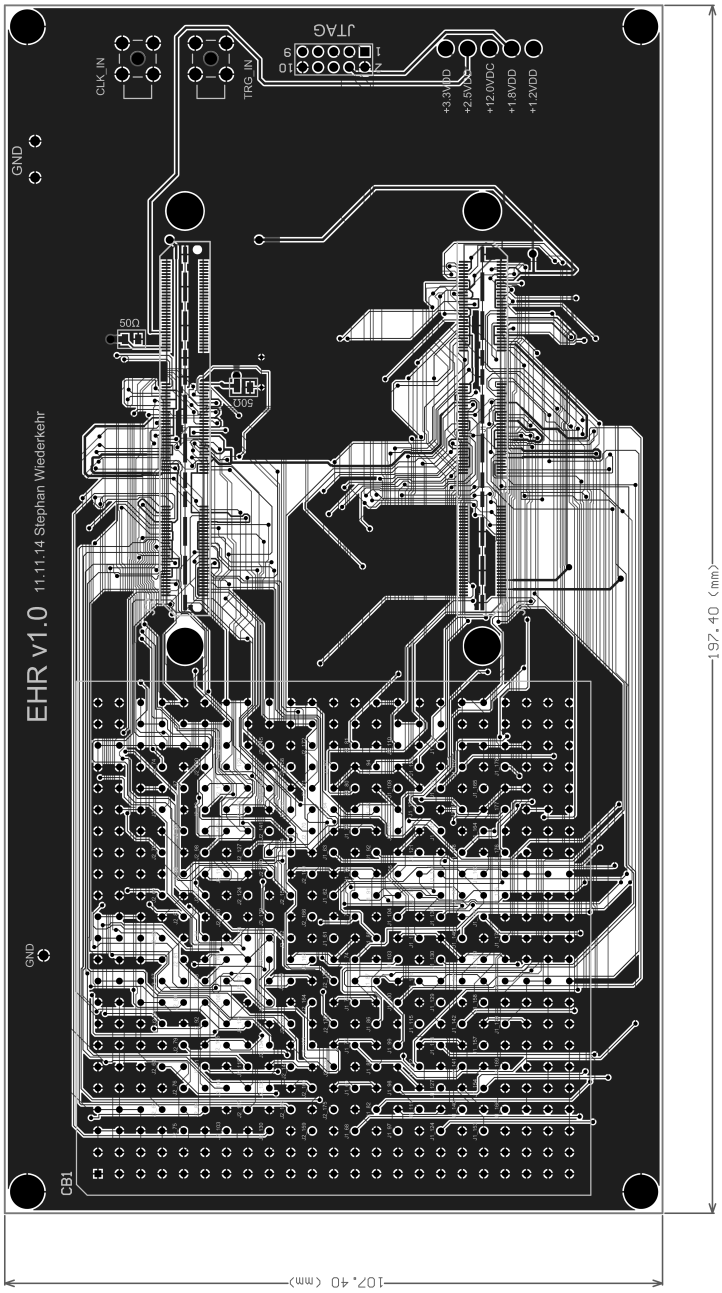

Figure 4: The copper top layer of the connection PCB. The JTAG and the reference power out connections are foreseen for debugging. The CLK_IN and TRG_IN connections were designed for LEMO connections for the clock input and the trigger input, respectively. The connections labeled J1 and J2 are designed for the SAMTEC connectors of the FPGA board.

2.3 FPGA Design

This section briefly describes what was implemented in the FPGA and, therefore, also illustrates the possibilities the EHR setup offers. The full Verilog file can be found in the appendix.
The FPGA design consists of four modules, namely the random pattern generator, the fix pattern generator, the phase shift module and the main module. Figure 5 depicts the conceptual design including the most important inputs and outputs. In the following the mentioned modules are introduced.

The main module features a phase locked loop (PLL) which takes either an external clock of roughly 40 MHz or the internal clock of 50 MHz provided by the FPGA board and outputs it to all modules. If no external clock is provided, the PLL automatically switches to the internal clock. However, if an external clock is provided, the PLL does not switch to it automatically. In this case one can manually switch between the clocks.
Furthermore, a memory is implemented which stores 256 one bit words. Its purpose is to activate or deactivate individual wires, where 1 corresponds to active and 0 to not active.
The main module also manages the USB connection using the FrontPanel interface, i.e. it accepts or sends all data, transferred via ep_wires, from or to the computer. Additionally, the 2 user programmable red LEDs on the FPGA board are programmed such that the LED 0 indicates the configuration status, i.e. it is turned on whenever the FPGA is fully configured, while the LED 1 indicates the use of the internal clock, i.e. it is turned on during the use of the internal clock.

The random pattern module generates a random pattern for each wire individually, such that there are in fact 159 modules. The pseudo random number generator is based on linear feedback registers and has a fixed starting point. The starting points are, of course, different for each instance of the module.
The generated pseudo random number is then compared to the rate parameter, henceforth denoted as r_p, which is the same for all random pattern modules. The comparator eventually triggers the state machine which forms the pulse. It is designed such that the pulse is asserted during three clock cycles and deasserted during the following three clock cycles as the ROC cannot process pulses in a shorter se-

quence. Further incoming triggers from the comparator are blocked while the state machine is running.

The fix pattern module contains a RAM memory which stores the pattern information for all wires. The memory is configured to store 4 096 256-bit words, where each word contains the pulse information of one clock cycle for all wires. This means that the maximum length of a fix pattern can be 4 096 clock cycles long. By default the memory restarts with the first word once the last word was output.
For future hardware as well as software manipulations it is important to note that the minimum input word length is 32 bit, precisely matching the length of an ep_wire. Hence, eight input words are needed to write one output word.
A few details regarding this module:

- A modulus parameter was implemented to restart the read out at any point within the maximum memory depth instead of restarting the output after the last memory word.
- The module reacts to an external trigger restarting the memory read out. Additionally, the pattern_max parameter can be set to limit the number of patterns sent after a trigger is received.

The phase shift module offers the possibility to shift the phase of the output clock with respect to the input clock of the PLL. The phase is shifted in steps of $2.08\overline{3}$ ns as determined by the PLL configuration, whereas the maximum number of steps is 127. It is possible to apply a shift in both directions. When resetting the PLL, also the phase shift will be reset to zero.

All inputs and outputs are listed in the appendix except the ones necessary to set up the FrontPanel interface. To see how the latter is set up refer to the *FrontPanel User's Manual* [8] or to the Verilog file, also in the appendix, containing the whole design.

It should be noted that the version of the FPGA configuration file which can be found in the appendix was installed as boot profile. This means that by powering-up the FPGA board, the FPGA will automatically be configured using the mentioned configuration file. The boot profile was saved in the system flash memory of the FPGA board. For more information about the available flash memory on the FPGA board please consult the ZEM4310 User's Manual [7].

The boot profile is configured to initialise the endpoint connections (cf. section 2.1) such that:

- The run mode is set to the fixed pattern mode. As the pattern memory is initialised to be all zero, there will be no pulse sent by default.
- The rate parameter is set to 256.
- All memory related signals are set to zero.
- The pattern modulus is set to 100.
- All signals related to the phase shift module as well as the PLL reset and the clock switching signal, which are all part of the same ep_wire, are set to zero.

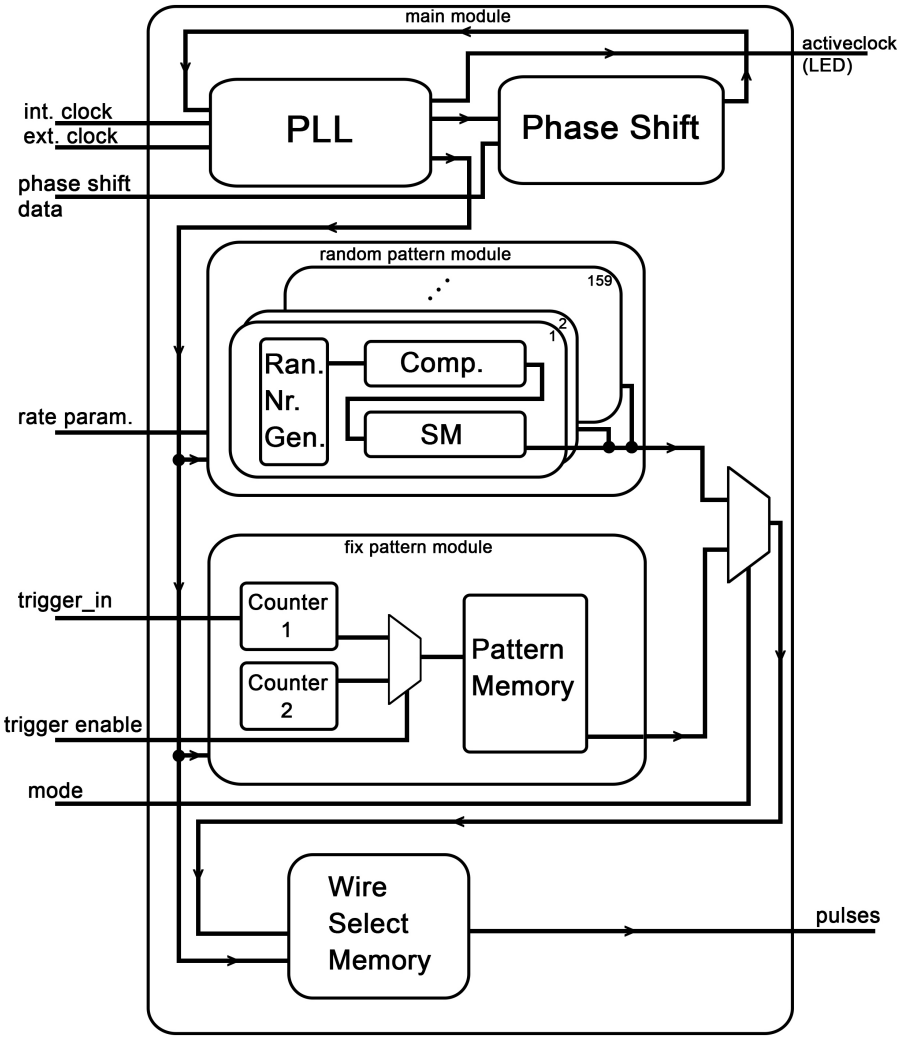

Figure 5: The simplified schematic design of the FPGA used for the EHR setup. The picture does not reflect the exact design.

2.4 FPGA Class

The FPGA class was written in Python to facilitate the use of the EHR setup as well as to complete some operations, e.g. the writing of the RAM memories, on the software side instead of implementing them on the FPGA. Alternatively, the FPGA can also be controlled using only the FrontPanel software. This section describes how to use the class, all functions included in the class and provides some additional information regarding the FPGA reaction. A link to download the FPGA.py file can be found in the appendix.

The class was written in Python 2.7.3, i.e. this or a compatible version of the interpreter is required. Additionally, the FrontPanel class (ok.py) is required to use the USB communication. It is included in the FrontPanel software development kit version 4.4.2. The FrontPanel software is available for several operating systems and was tested on Windows 7 (64 bit) and Ubuntu 12.04 (32 bit). To install the software please follow the instructions in the README file provided for each version.
All programs written and used are listed in the appendix. They can serve as examples of how to use the FPGA and the FrontPanel class. The FPGA class members are described in the following:

__init__(serial, filepath, max_modulus)
: During the initialisation the FPGA is accessed and checked for FrontPanel support which is crucial for the USB support. The initialisation takes three arguments. If more than one setup is attached to the computer, any specific setup can be opened by its unique serial number which can be given as input. The initialisation also offers to reconfigure the FPGA if a path to a valid configuration file is given as input. The only tested format for configuration files were .rbf files. Finally, also the *max_modulus* parameter is defined in the input as the memory is only initialised up to its value. It is therefore not recommended to raise the modulus parameter above *max_modulus*.
The __init__ function then initialises the wire selection memory all to 1 and the pattern memory all to 0, up to the *max_modulus* value. The remaining memory remains untouched. After the memory is initialised also the PLL is being reset. To conclude, the ep_wires are set to put out the pattern memory, i.e. zero, while a small rate parameter is set along a modulus of 100.

configureFPGA(filepath)
: This function allows to configure the FPGA, if the file path is pointing

to a valid configuration file. The reconfiguration was implemented for debugging purposes, but it is a valuable option in case the FrontPanel support is not enabled. In this case it is recommended to reconfigure the FPGA with the finalv3_1.rbf file given in the appendix.

reset_pll()
Resets the PLL. Note that this function also resets the phase shift.

reset_ran()
Resets the random pattern generator to its starting values for all wires.

set_clock(clock)
Sets the clock to the value determined in the input, where $clock = 0$ switches to the external clock, if one is available, and $clock = 1$ switches to the internal clock (cf. 2.3).

set_mode(mode)
Selects the output mode for the wires, where $mode = 0$ selects the random pattern mode and $mode = 1$ selects the fix pattern mode.

set_cable(cable, value)
Writes to the wire select memory. The *cable* parameter selects the wire and the *value* parameter determines whether its pulse output will be enabled or not, where $value = 1$ enables the pulse output and $value = 0$ disables it. As the wire select memory does only take 32 bit data words as input, an array was required for the initialisation. The array-word restores the information of the 31 unchanged wires such that each wire can be (de-)activated individually.

set_modulus(value)
Sets the modulus parameter to the value determined by the input.

set_rate(value)
Sets r_p to the value determined by the input. Further information about which effective rates r_p entails is given in the section 3.2 discussing the measurement results.

set_trg(modulus, pattern_count_ena, pattern_max)
Enables the reaction to a trigger. The *modulus* value determines the memory modulus (cf. section 2.3). The *pattern_count_ena* value enables (1) or disables (0) the output of a specific number of patterns, determined by the *pattern_max* value, after a trigger is received.

get_cable_out(cable)

Returns the output value of the wire determined by the input. Caution is advised when using this function due to its inherent lack of precision.

set_phshift(direction, phsmax)

Shifts the output clock with respect to the input clock of the PLL by the amount of steps and the direction determined by the input. The *direction* variable takes the values "up" or "down", including the quotation marks. The *phsmax* parameter determines the amount of steps for the phase to be shifted, where each step shifts the phase by $2.08\overline{3}$ ns.

write_mem(textfile)

Writes the pattern information stored in a text file to the pattern memory. The parsing was designed to work with text files of the following form:

cable xx:
pattern: 1001011001
cable xy:
pattern: 0100101010
all:
pattern: 1011010101

In the example above, a modulus value of ten was assumed. If "*all*" is included in the file, the following pattern will be written to the pattern memory for all wires ignoring additional patterns designed for particular wires. A text file should therefore only contain patterns for specific wires or one pattern for all wires. A simple Python program to write such files is contained in a github repository which can be accessed using the link given in the appendix.
The maximum pattern length which will actually be written into the memory is limited by the maximum memory depth and by the *max_modulus* parameter specified in the initialisation.

When writing to the RAM memory, care must be taken regarding the timing. The data to be written as well as the address of its destination must be asserted before the *write enable* signal is asserted and deasserted only after the *write enable* signal was deasserted. Since there is only one command in the FrontPanel class to update the wires, which updates all wires at once,

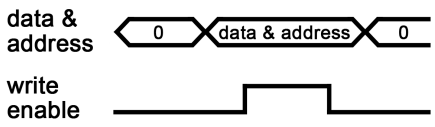

Figure 6: The ep_wires must be updated such to fulfil the depicted timing requirements. The final deassertion of the data and address is optional.

the updating must be done in at least three steps as illustrated in figure 6.

The memory management in the FPGA class is done via two–dimensional arrays, which are created during the initialisation. The dimension of the pattern memory array is determined by the *max_modulus* parameter specified in the initialisation, which also determines the part of the actual pattern memory to be reset. As a result, the initialisation speed does depend on the *max_modulus* parameter. Any changes to the memory are written into the arrays first. This facilitates the programming of the actual memory and on the other hand saves the current state allowing to rewrite the memory several times.

2.5 Digital Test Board

The purpose of the DTB is to manage all signals transferred between a host and a ROC or a module. The DTB possesses two programmable LEMO outputs. In connection with the EHR setup, one of those was used to output the DTB clock (40 MHz) and feed it to the EHR setup. The other connection was used to send a so-called synchronising signal from to DTB to the EHR setup whenever a read out sequence was sent. This synchronising signal was used to trigger the pattern output of the EHR setup when needed.

The read out sequence sent by the DTB is arbitrarily programmable. Yet, for all presented measurements it contained only four signals if not mentioned otherwise. The sequence is illustrated in figure 7.
The synchronisation signal was fed to the EHR setup as trigger to start the pattern output. The calibration signal induces a voltage, called V_{Cal}, in the pixel to simulate a hit, if the pixel is programmed to accept it.
The trigger signal determines the clock cycle to be read out along with an 8 bit parameter called WBC, which determines the distance between the targeted clock cycle and the trigger signal in clock cycles. The token signal

eventually starts the read out process. All the data collected belonging to one trigger is often referred to as *event*.
To set the chip, including for instance its data buffer, to a well defined state, a reset signal can be sent. The reset signal can be programmed to be part of the read out sequence, yet it was always sent separately in all measurements presented in the following. It was sent at the start of each measurement only, to test the ROC under realistic circumstances.

The *WBC* value determines the clock cycle to be read out while the *tct* value determines the distance between the calibrate and the trigger signal as depicted in figure 7. As the V_{Cal} is injected after the calibrate signal, the WBC value must be smaller than the *tct* value. By default WBC must be equal to $tct - 6$ to measure the V_{Cal}. Since both values are stored in 8 bits, this means that the maximum WBC value is, by default, equal to 249.

The read out process itself is organised in column-pairs called double columns. If a pixel gets a hit, it sends a signal to the periphery. Subsequently, a token is sent from the chip periphery through the first column of the pair and returns in the adjacent column checking the all the pixels for hits along the way. In many aspects, the double column is therefore more relevant than each single column. The process of transferring the hit information from the pixels to the data buffer is called *column drain*.
Each column drain is stored with a time stamp inside the data buffer. To start the read out process of the whole chip, a signal, in fact a different token, is sent to the chip to read out the data belonging to a certain time stamp determined by the trigger signal and the WBC value. The data buffer is cleared afterwards. Thus, one such token signal is sent after each trigger signal to read the desired data out.
In the following, the signal triggering the read out of the whole chip, as depicted in figure 7, is referred to as token. On the other hand, the signal traveling through the double column is considered a part of the column drain.

For each clock cycle to be read out a trigger must be sent to mark it before a token is sent to read the data out. The data buffer will collect data until a trigger is received or it is full. If the data of a column drain, i.e. the data of a time stamp, can not be completely written into the data buffer a reset signal will be sent to the chip, also clearing the data buffer. This process, leading to a loss of data, is called *data buffer overflow*. Therefore, the time between the marked clock cycle and the token signal as well as the hit rate determine whether the desired time stamp can be read out correctly

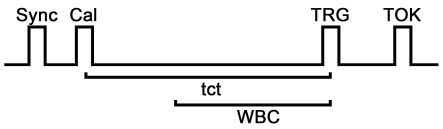

Figure 7: The sequence the DTB sent to the ROC during the measurements. A brief description of all depicted signals and values is given in the text.

or if no hits will be read out as the desired time stamp was already deleted. Since for all presented measurements only one trigger was sent per sequence and the time between the trigger and the token was kept constant, only the WBC value determined the distance between the marked clock cycle and the token.

3 Results

While in the previous sections several different trigger signals were described, the term trigger will henceforth be used for ROC triggers only (cf. section 2.5), unless mentioned otherwise.

3.1 Wire Location

To learn which pixels are hit by a particular wire, a test was performed that collects the address information of the hit pixels and prints it on a ROC map. Only one wire was active per run and the pulse pattern was pseudo–random such that various read outs were needed per wire. For the wire location map shown in figure 8, r_p was set to $26 \cdot 10^6$ and $5 \cdot 10^3$ triggers were sent.
A pulse sent through a wire may cause hits in more than one clock cycle. Thus, the run mode was chosen to be pseudo–random to gather the hits of all such possibilities.
Additionally, the pulse height, recorded in Analogue-to-Digital Converter (ADC) values, was measured. Separate measurements were performed measuring the pulse height of the calibration signal for different Digital-to-Analogue Converter (DAC) values. Comparing the results yielded a pulse height for the EHR setup corresponding to a DAC value of the calibration signal of roughly 30, which in turn corresponds to the charge of approximately 1500 electrons.

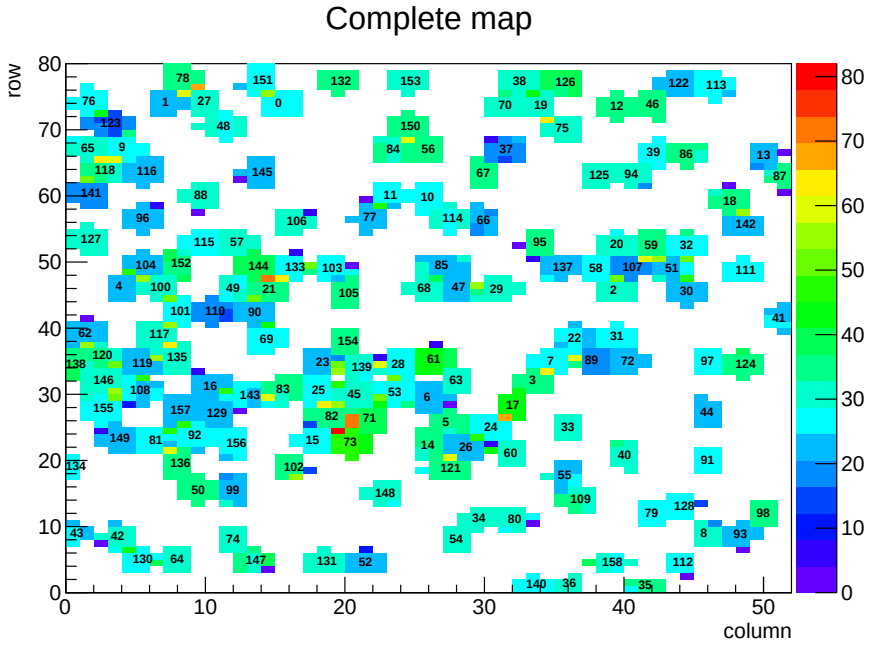

Figure 8: All clusters generated by the wires. The number of hits is shown as a function of the pixel address. Additionally, the wire number is printed on the corresponding cluster. The wire diameter of roughly 380 μm caused an average cluster size of approximately 9.8 pixels.

3.2 Rate Measurements

The end-user can only influence the hit rate by varying the rate parameter (cf. section 2.3). Yet, effective rate can be calculated, as a function of this rate parameter, considering the FPGA design. This section covers the calculation as well as the measurements performed to create a link between the effective hit rate and the rate parameter.

The measured rate depends on the number of pulses sent by the FPGA as well as on the effect the pulse has on the ROC, i.e. how many pixel-hits are induced by the specific wire.

The number of pulses sent by the FPGA is determined by its comparator and the state machine which forms the pulse (cf. section 2.3). As the rate parameter r_p is stored in 32 bit, the probability P_c for the comparator to

send a trigger in an arbitrary clock cycle is given by:

$$P_c = \frac{r_p}{2^{32} - 1} \tag{1}$$

After the state machine receives a trigger it is blocked for the subsequent six clock cycles. Since the state machine can only accept a trigger if it is idle, there is an additional clock cycle without a pulse. As a result, the maximum rate, i.e. the comparator is permanently at signal level one, is one pulse in seven clock cycles.
To calculate the rate one can calculate the mean spacing between two pulses. It is determined by adding all possible distances multiplied by their probability to occur.

$$\begin{aligned} < T > &= 7 \cdot P_c \ + \ 8 \cdot P_c \cdot (1 - P_c) \ + \ 9 \cdot P_c \cdot (1 - P_c)^2 + \ldots \\ &= \sum_{n=0}^{\infty} (n + 7) \cdot P_c \cdot (1 - P_c)^n \end{aligned} \tag{2}$$

After a bit of calculation, the rate is obtained by inverting $< T >$:

$$E_p := < T >^{-1} = \ \frac{1}{6 + \frac{1}{P_c}} \tag{3}$$

To verify equation 3, a universal counter was used to count the pulses sent through a wire within a determined time interval. Ten runs were averaged where the time interval per run was chosen to be 1 s to reduce the error to less than 100 counts. The pulse counts and E_p are shown in figure 9, where E_p was multiplied by $40 \cdot 10^6$ to compare it to the number of pulses measured in one second as the 40 MHz clock of the DTB was used for the measurement.

The contribution of each wire, i.e. how many hits it causes, was measured by sending 10^5 triggers while only one wire was activated and collecting the number of hits per event (cf. section 2.5). The measurement was performed once with an r_p value of $2 \cdot 10^6$ and twice with an r_p value of $20 \cdot 10^6$. For the second measurement at a high rate, the phase of the PLL was shifted by 10.4 ns. Typical hits per event distributions for the wires 0 and 52 are shown in the figures 10 and 11, respectively. The files containing all these measurements can be found in the appendix.
Two phenomenons can be observed accompanying the expected sharp peak. First, the chip inefficiency which yields a tail to the peak as it can be seen in figure 10. This effect gains in strength with increasing rate. Second, a

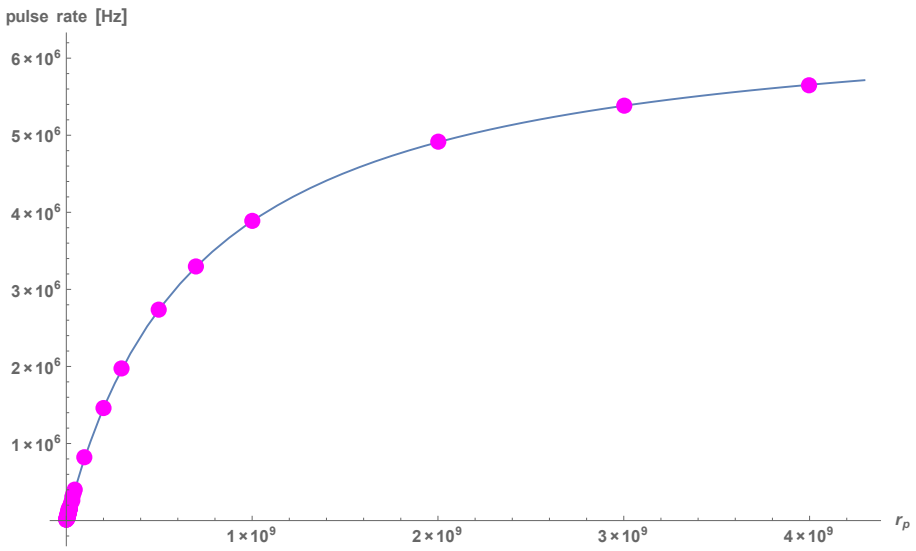

Figure 9: The pulse rate as output by the FPGA as a function of r_p as predicted by equation 3 (blue) and measured (pink). The measurement was performed using an universal frequency counter attached to a wire.

splitting of the hits into two clock cycles as it happens if the pulse edge crosses the boundary of the two. This yields a second peak as can be seen in figure 11. Moreover, figure 11 also shows that the splitting can be repaired by applying a phase shift to the PLL output clock. However, during the measurements only two phase shift settings, namely 0 ns and 10.4 ns, were tested. Neither of those removed all pulse edge splittings. It is, therefore, unclear if a setting exists eliminating all of them.

On the other hand, the splittings do not interfere with rate measurements. Halving the hits per pulse while simultaneously create two clock cycles containing those halves still yields the same rate as if no pulse edges were split.

The measurement performed with $r_p = 2 \cdot 10^6$ revealed an average of 8.1 hits per event for all wires while only considering the expected peaks yielded an average of 9.8 hits per event. These averages were used for the following estimate.

To obtain a rate prediction for one or more wires, E_p must be multiplied

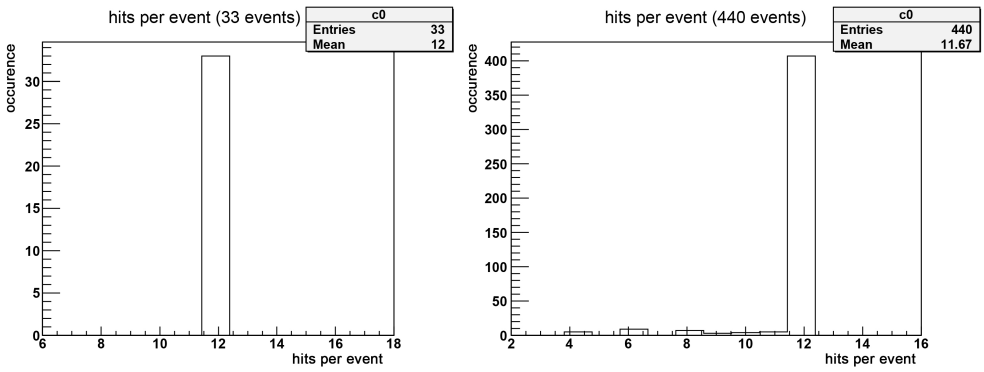

Figure 10: Two distributions of hits per event for wire 0 omitting events with no hits.
Left: $r_p = 2 \cdot 10^6$ Right: $r_p = 20 \cdot 10^6$
The phase shift of 10.4 ns did not have an effect on this wire. The tail to the peak in the right figure is attributed to the ROC inefficiency.

by the average number of hits caused by one pulse. Furthermore, the rate was converted from $[\frac{MHz}{40 \cdot ROC\ area}]$ to $[\frac{MHz}{cm^2}]$ as it was done for all rate measurements. Similarly, P_c, from equation 1, was multiplied by the average number of hits per pulse and also converted to $[\frac{MHz}{cm^2}]$. In the following, E_c denotes the converted P_c.
Figure 12 and 13 show the rate predicted by equation 3 and the results of a rate measurement with only wire 0 active. The estimate was plotted with an uncertainty of 0.5 hits per event to indicate the extent of the variation. Increasing r_p, the predicted rate exceeds the measured rate more and more. The difference between the two curves is attributed to the inefficiency of the ROC.

Figure 14 and 15 show the estimates and a measurement performed with all wires. The E_p estimate was plotted twice with hits per event averages of 8.1 and 9.8 found in measurement mentioned earlier.
As the curve plotted with an hit per event average of 9.8 fits the measurement far better, this result supports the claim that pulse edge splittings do not influence rate measurements.

As previously mentioned, the deviation between the measured values and the estimate is attributed to the ROC inefficiency. Therefore, the ROC inefficiency can be obtained by dividing the measured values by the estimate.

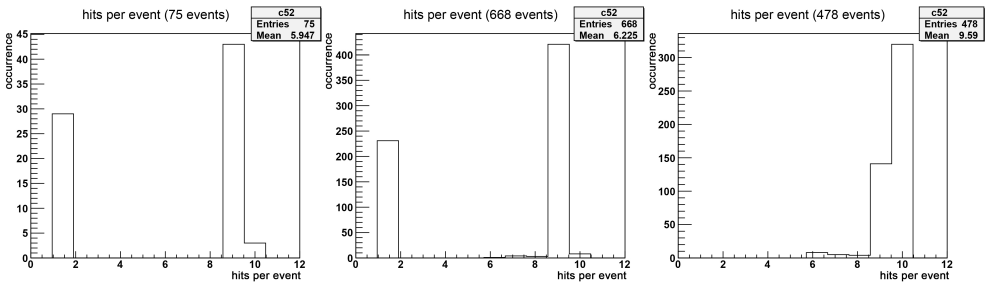

Figure 11: Two distributions of hits per event for wire 52 omitting events with no hits.
Left: $r_p = 2 \cdot 10^6$ Middle: $r_p = 20 \cdot 10^6$ Right: $r_p = 20 \cdot 10^6$ (phase shifted by 10.4 ns)
The two-peak structure in the left and middle figure is attributed to pulse edge splittings, whereas the tail to the peak in the right figure is attributed to the ROC inefficiency.

However, the precision of such a calculation depends chiefly on the precision of the hit per event average, as the FPGA dependent contribution was found to be very accurate (cf. figure 9). Even though the pulse edge splittings do not influence the rate measurements, they do influence the hit per event measurements. As a result, the values for the efficiency are difficult to obtain using only rate measurements as long as pulse edge splittings can be observed.

Each measurement for one r_p value shown in figure 14 was repeated 100 times to estimate the error of the measurement. Figure 16 shows the standard deviation plotted against the mean rate. The distribution width saturates along with the measured rate. The values outside the saturation region, i.e. all measurement points below a rate of 600 $\frac{MHz}{cm^2}$, were fitted linearly merely as a guide to the eye. To include the data in the saturation region, a linear approach does clearly not suffice.
The obtained linear function was used as error estimate for the efficiency measurements in section 3.4, albeit with a somewhat higher offset to account for the uncertainty using a linear function.

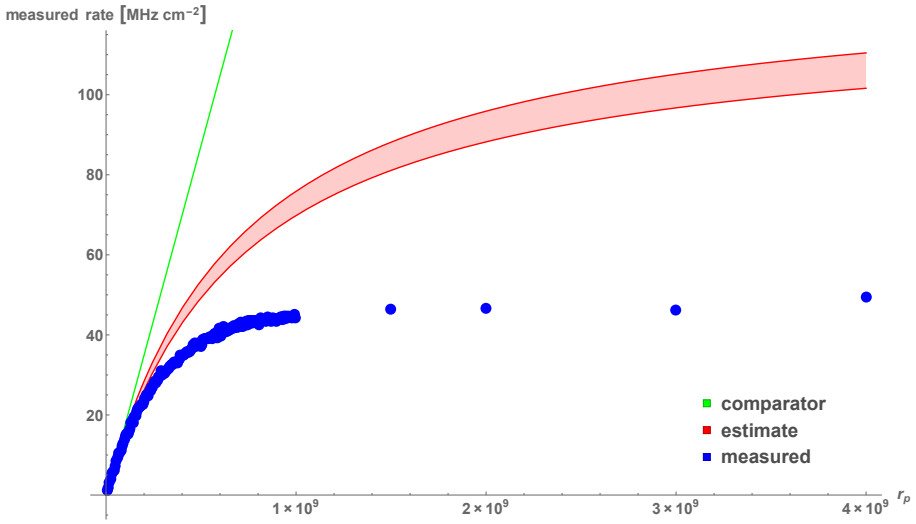

Figure 12: The two estimates E_c (green), E_p (red) and the measured rate (blue) as a function of the rate parameter r_p. The measurements were performed with one active wire only at a WBC value of 75 and a trigger rate of 10 kHz. The difference between the estimate (red) and the measurement is attributed to the ROC inefficiency. The estimate was plotted with an uncertainty regarding the average cluster size of 0.5 as a guide to the eye.

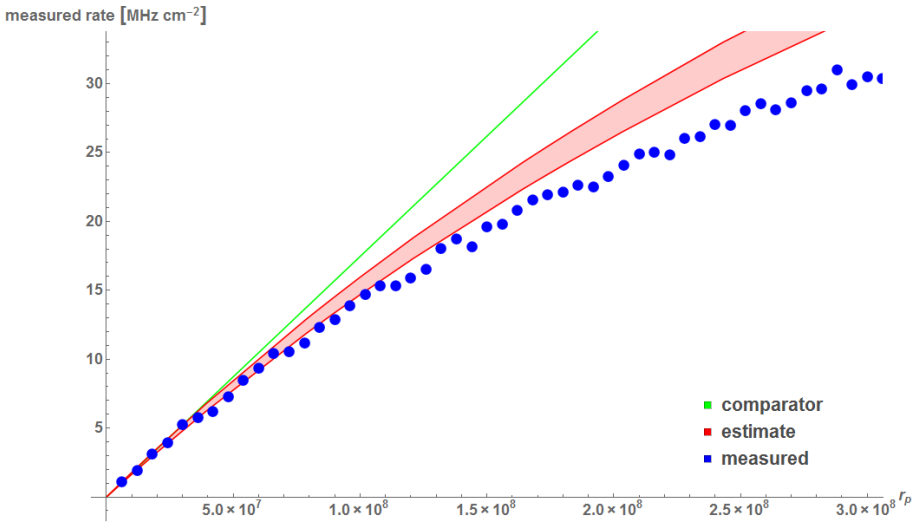

Figure 13: A zoom-in of the plot shown in figure 12.

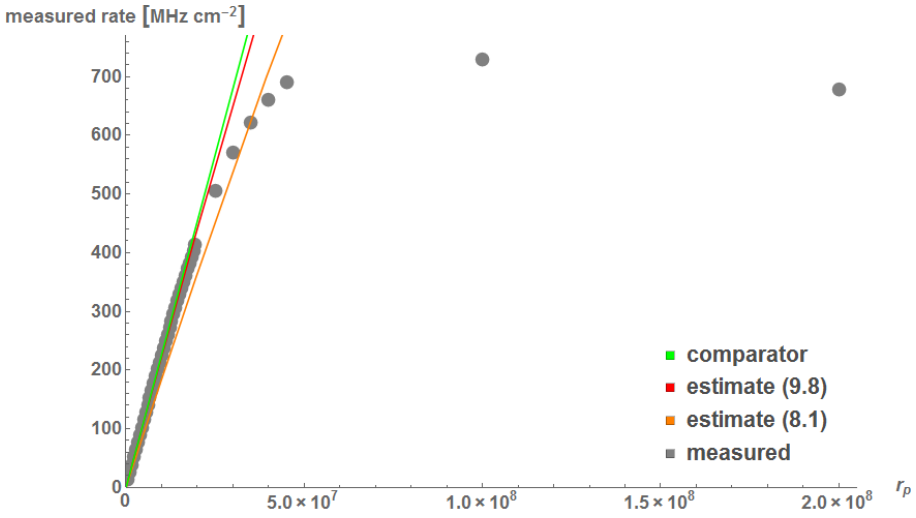

Figure 14: The two estimates E_c (green), E_p (red and orange) and the measured rate (grey) as a function of the rate parameter r_p. The measurements were performed with all wires active at a WBC value of 100 and a trigger rate of 10 kHz. The difference between the estimates (red and orange) and the measurement is attributed to the ROC inefficiency. For the two estimates for E_p different cluster size averages were used. As the red curve fits the measurements far better than the orange curve, it follows that pulse edge splittings have no effect on rate measurements.

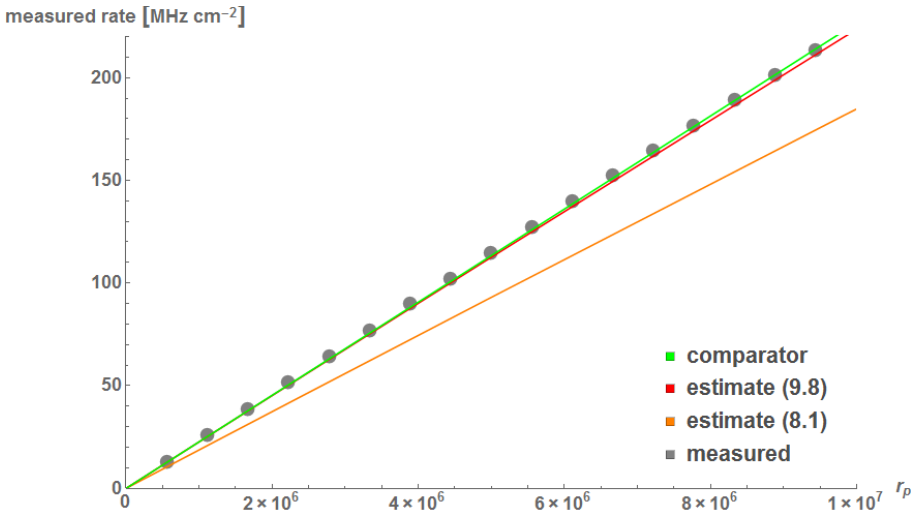

Figure 15: A zoom-in of the plot shown in figure 14.

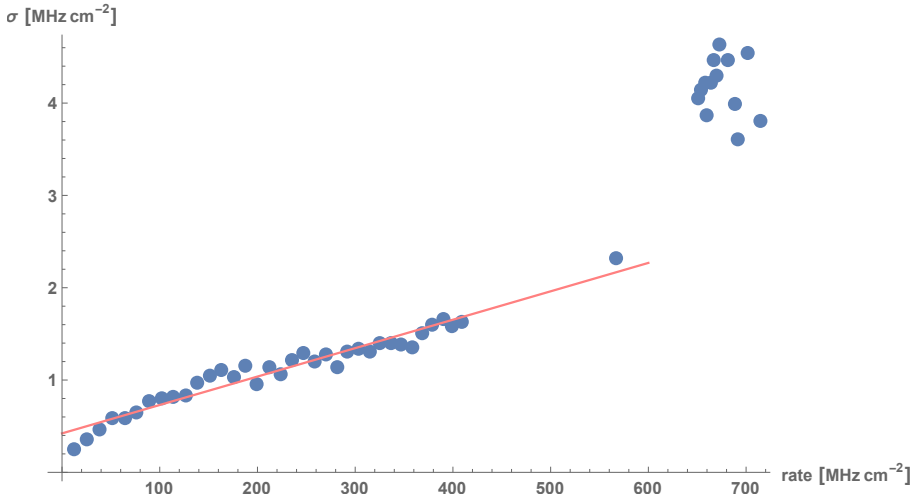

Figure 16: The standard deviation (σ) as a function of the measured rate. The linear fit includes only the data below 600 $\frac{MHz}{cm^2}$ and acts as a guide to the eye. The points in the top right correspond to rates inside the saturation region.

3.3 ROC Power Consumption

Performing numerous rate measurements also allowed to study the power consumption of the read out process. In particular, the digital current (Idig) was of high interest as it depends on the amount of data being transferred inside the ROC and, therefore, also on the hit rate. The actual EHR-design offered new data with respect to X-ray measurements as the cluster size does have an impact on the power consumption.
The reason is that for a constant hit rate less column drains are executed with an increasing cluster size. The amount of column drains as well as the amount of pixel hits during a column drain contribute to the current. As soon as every double column contains a hit, the contribution of the column drain to the digital current will saturate. However, the contribution arising from the pixel transfer will only saturate when every pixel contains a hit. Therefore, a change in slope is expected when measuring the digital current as a function of the rate, due to the saturation of the column drain contribution. A saturation due to the pixel hit contribution is expected to lie beyond the data buffer capacity of the ROC.

Figure 17 shows two data sets taken with X-rays [9], one data set taken with the EHR setup and two estimates. One estimate assumes that the frequency of the column drain, which is equivalent to the frequency of the time stamp generation (fTS), is equal to the frequency of pixel hits (fPIX). This situation corresponds well to the manner of hit generation with X-rays, as they only induce a hit in one pixel. The other estimate assumes a very low time stamp frequency. This would correspond to the situation where all hits are induced in the same double column until it is filled before inducing hits in another double column.
The power consumption for rates below 600 $\frac{MHz}{cm^2}$ is always lower for the EHR setup than with X-rays. However, there is no clear change in slope visible in the EHR data. A linear fit through all points revealed a slope of approximately 0.083 $\frac{mA \cdot cm^2}{MHz}$.

As the time stamp frequency is lower than with X-rays at a fix rate, also the change in slope would be expected to occur at a higher rate than it does with X-rays. However, the rate measurements suggest that it is not possible to measure rates above 700 $\frac{MHz}{cm^2}$ with the current setup.

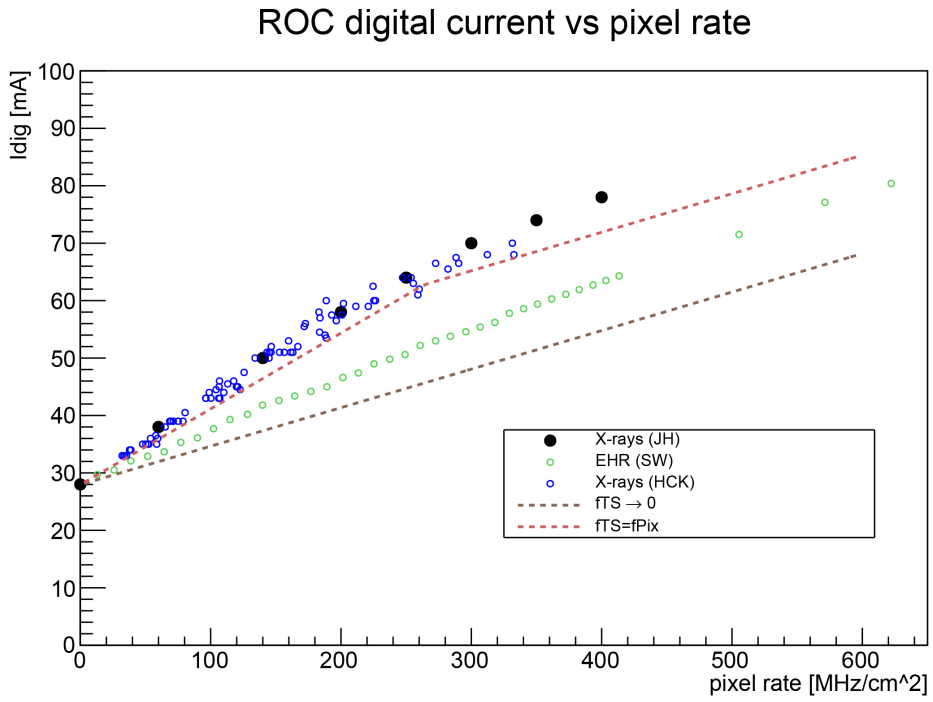

Figure 17: The digital current as a function of the pixel rate. The dotted lines correspond to estimates for the cases of a time stamp frequency going towards zero (brown) or being equal to the pixel rate (red). The black and blue circles correspond to measurements performed by Jan Hoss (ETH) and Hans-Christian Käestli (PSI), respectively. The green circles correspond to measurements performed using the EHR setup.

3.4 ROC Efficiency

To measure the ROC efficiency, a calibration signal was injected into a pixel and read out while the EHR setup was running at a previously measured rate. With increasing rate, the read out of the calibration signals is expected to decrease. Possible reasons are that the pixel is blocked due to an ongoing readout or that data is lost due to a data buffer overflow.
In all of the following measurements, 10^5 calibration signals were sent to each pixel. The efficiency was measured for ten different rate values, containing and even exceeding the range of real hit rates within the LHC, namely (in $\frac{MHz}{cm^2}$):
25.9, 51.7, 77.1, 102.2, 127.3, 152.4, 176.9, 201.5, 249.2, 295.8

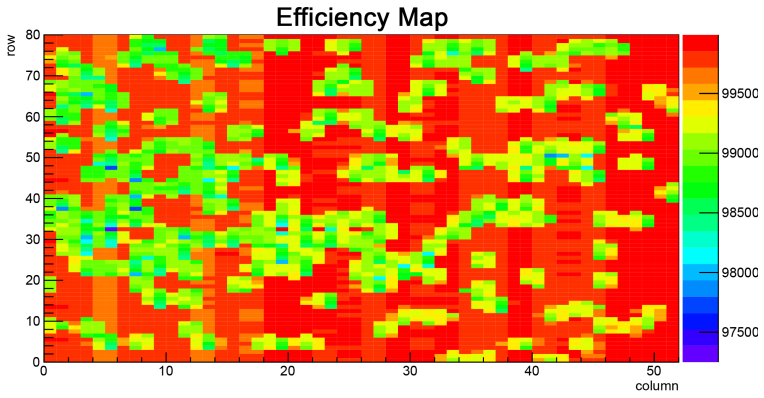

Figure 18: The pixel efficiency as a function of the pixel address. The measurement was performed at a rate of 77.1 $\frac{MHz}{cm^2}$, a trigger rate of 10 kHz and with a WBC value of 100. The ROC efficiency is decreased at the position of the clusters and in a few double columns.

These particular rate values were the result of some of the rate measurements presented in the previous section. Therefore, the r_p value and the errors, which can be read out of figure 16, were already known. The efficiency tests were performed for different WBC values (cf. section 2.5) as more data will be collected, after the calibration signal injection, with increasing WBC, possibly leading to a data buffer overflow. As a result, the efficiency was expected to decrease significantly with increasing WBC. Additionally, the tests were performed for trigger rates of 10 kHz, 30 kHz, 60 kHz and 100 kHz, where the last value is the closest to real values encountered in the CMS experiment.

Figure 18 shows a resulting ROC map for one efficiency measurement. The clusters can easily be detected by the low efficiency at their position. Moreover, a few double columns show a decreased efficiency. Such an effect is caused by a high number of pixel hits in the same double column exceeding the capacity of the data buffer. All the efficiency maps acquired can be found in the appendix.

The average efficiency of all pixels, considered the chip efficiency, was collected for the rates mentioned above to produce a plot as shown in figure 19. These were then grouped and summarised in the figures 20 and 21.

Figure 20 shows all efficiency measurements performed with a WBC value of 100 and indicates that there is no difference in the chip efficiency when

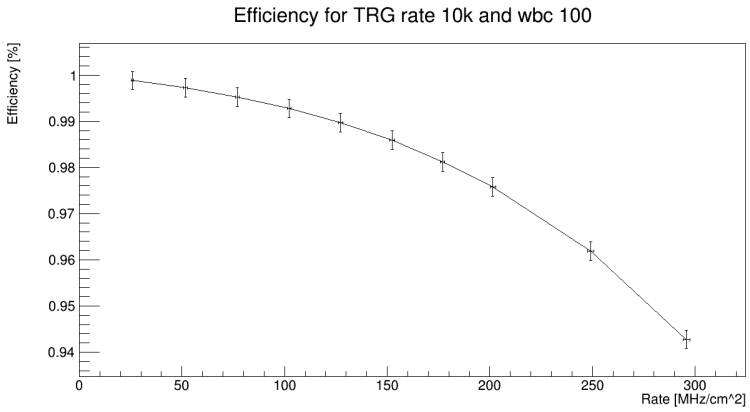

Figure 19: The ROC efficiency as a function of the hit rate. This and similar measurements were used to produce the plots depicted in the figures 20 and 21.

varying the trigger rate. Therefore, it suffices to show the result of one trigger rate varying the WBC value as it is shown in figure 21 for a trigger rate of 10 kHz.
Only one exception remains remarkable. When setting the WBC to 249 and the trigger rate to 100 kHz there may be not enough time for the readout, leading to corrupted data.
Figure 21 shows that the efficiency does not depend on the WBC value in the design region between 100 and 150. Due to buffer overflows, the efficiency decreases with increasing WBC. Unexpectedly, the efficiency also decreases when decreasing the WBC from 100 to 50. A possible reason is the time for the data transfer before the trigger is received being to short. However, this assumption could not be confirmed.

A comparison with results obtained performing measurements using X-rays reveals a difference in the efficiency curves. The data for rates below 100 $\frac{MHz}{cm^2}$ are very similar. However, the curves start to deviate afterwards. The efficiency using the X-ray setup decreases stronger than with the EHR setup. At a rate of 300 $\frac{MHz}{cm^2}$, the X-ray efficiency decreased to roughly 86.5 %, whereas the EHR data reached roughly 94 %. Figure 22 shows the results of efficiency tests performed with X-rays for twelve different ROCs [10] and one measurement of the EHR setup for a better comparison. The X-ray data was taken using a WBC value of 99, whereas the EHR data included in figure 22 was taken using a WBC value of 100.

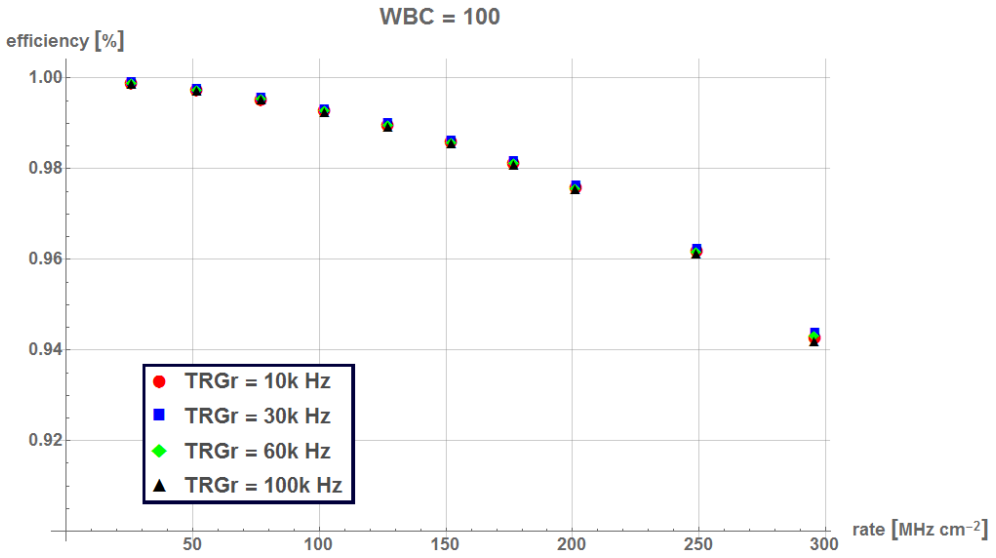

Figure 20: The ROC efficiency as a function of the hit rate for a WBC value of 100 and four different trigger rates. The ROC efficiency can be seen not to depend on the trigger rate.

As mentioned earlier, X-rays only induce single pixel hits instead of clusters. For high rates it is assumed that all columns are in action. As the X-ray hits are distributed uniformly, all double columns drop in efficiency leading to a strong decrease in the total efficiency. At the same rate, the EHR setup may not stress all double columns as the hits are concentrated in clusters. As can be seen in efficiency maps, there are also double columns at nearly 100 % efficiency at high rates. As an example, figure 23 shows an efficiency map resulting from a measurement performed at 295.8 $\frac{MHz}{cm^2}$.
On the other hand, the EHR prototype had far less active wires available, leading to less double columns being effectively stressed. Therefore, the efficiency did not decrease with higher rates as much as with the current setup, let alone with X-rays. As the prototype induced multiple, typically around eight, clusters per pulse, which did generally not share a double column, the measured efficiency at low rates was roughly at 99 %. This indicates what can also be seen in efficiency maps for low rates, e.g. in figure 18, that the efficiency is dominated by pixel effects rather than double column effects.
Figure 24 shows the efficiency measurements obtained with the EHR prototype [3]. The data for 300 and 350 $\frac{MHz}{cm^2}$ already suggest that the maximum inefficiency, for this setup, is reached. And indeed, the highest measurable

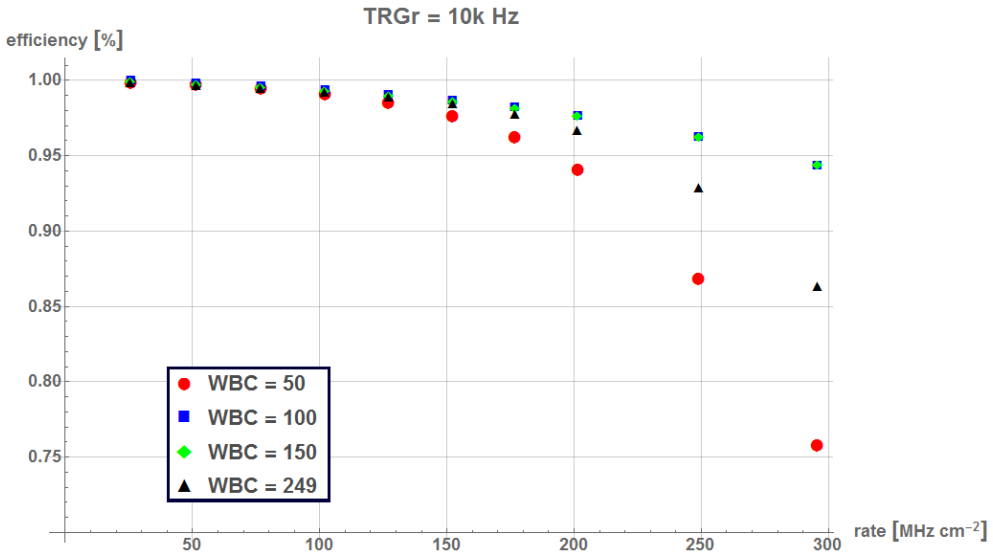

Figure 21: The ROC efficiency as a function of the hit rate for a trigger rate of 10 kHz and four different WBC values. The efficiency is maximal at the WBC values 100 and 150, which are typically encountered in the CMS experiment. The decreased efficiency for a WBC value of 249 is attributed to buffer overflows, whereas the decreased efficiency for a WBC value of 50 is attributed to the trigger signal arriving before the data can be processed properly.

rate using the prototype was found to be in between these rate values.
The smoother shape of the data set shown in figure 19 compared to the data taken with the prototype, shown in figure 24, is attributed to the higher degree of randomness of the EHR setup. While the EHR setup features a pseudo random number generator for each wire, the prototype had a fixed pattern with the randomness being introduced in the read out process. Yet, the latter approach suffered from the lack of speed the ROC could be programmed at.

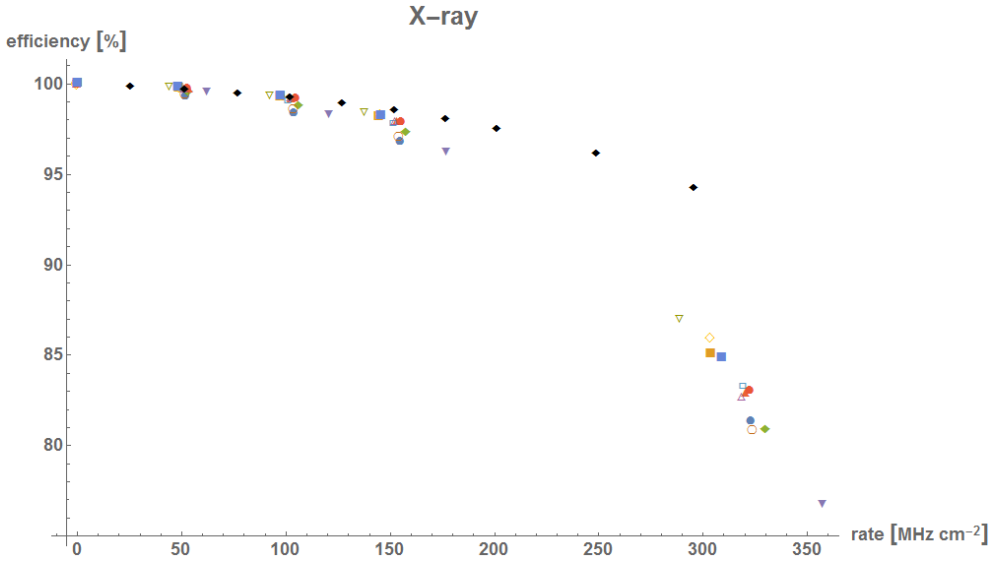

Figure 22: The ROC efficiency as a function of the hit rate for twelve different ROCs measured with X-rays at a WBC value of 99 [10]. The black diamonds correspond to a measurement performed using the EHR setup for a WBC value of 100.

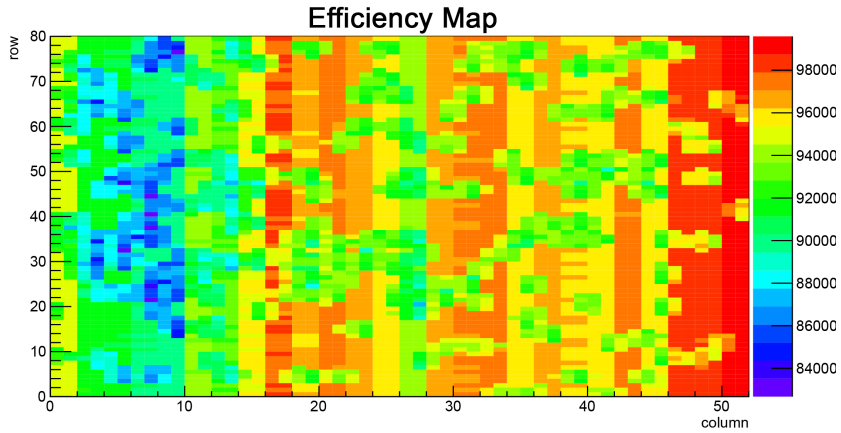

Figure 23: The pixel efficiency as a function of the pixel address. The measurement was performed at a rate of 295.8 $\frac{MHz}{cm^2}$, a trigger rate of 10 kHz and with a WBC value of 100. Despite the high hit rate some double columns have a high efficiency due to the lack of clusters located inside them.

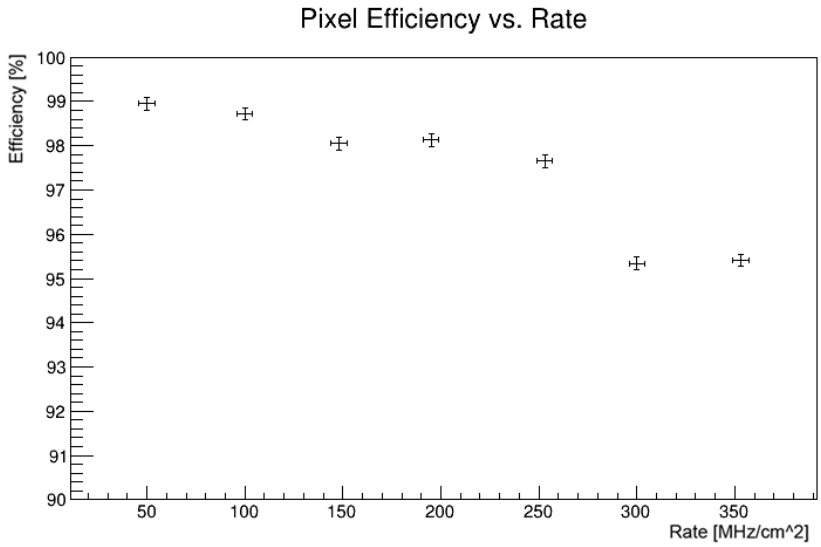

Figure 24: The ROC efficiency as a function of the hit rate measured with the EHR prototype. It induced roughly 8 clusters per pulse while the average cluster size was roughly 10 pixels. The highest measurable hit rate using the prototype was in between 300 and 350 $\frac{MHz}{cm^2}$ [3].

4 Conclusion

The presented EHR setup could demonstrate the control over the individual wires. However, when the precise timing of pulses becomes important, difficulties, related to the exact position of the pulse edges with respect to the clock, arise the more wires are involved. The possibility to shift the phase of the internal clock of the setup with respect to the clock of the DTB was shown to influence the issue. However, there were not enough measurements performed to find a phase shift fully resolving the problem.

The control over the pulse rate was found to be very accurate. Figure 9 indicates the possibility to predict the pulse rate using the expression for E_p in equation 3 in the entire range of the rate parameter r_p. In combination with hit per event measurements, the hit rate can also be predicted reliably. Sending 10^5 triggers, the error was never found to exceed 3 $\frac{MHz}{cm^2}$ outside the saturation region. Yet, the ROC inefficiency influences the result, depending on the number of active wires, starting at roughly 15 $\frac{MHz}{cm^2}$, indicated by figure 13.

In terms of hits per pulse the EHR setup emerges as useful tool, as the found 9.8 hits per pulse average provides a different point of view than X-rays do. This was verified in the power consumption measurement, shown in figure 17, where the slope of roughly 0.083 $\frac{mA \cdot cm^2}{MHz}$ lies in between the two extreme assumptions considering the time stamp frequency.
Also the efficiency measurements delivered the expected results, as for all tested WBC values the efficiency was found to be in between 99 and 100 % for rates lower than 100 $\frac{MHz}{cm^2}$ and thus very similar to the X-ray data. Further increasing the rate, the efficiency was found to depend on the WBC value. The decrease in efficiency when increasing the WBC above 150 is understood as an overflow of the data buffer, whereas the decrease occurring at a WBC value of 50 was attributed to the trigger being sent while the data may still be processed. The efficiency was found to be maximal for the WBC values 100 and 150. The WBC in the CMS experiment is typically between these two values. In this range of WBC values, the efficiency never fell below 94 % for rates up to 300 $\frac{MHz}{cm^2}$.
As the current EHR setup does not have the same number of active wires in every double column, the order in which they start to be inefficient is predetermined. Moreover, it is assumed that some double columns do not have enough active wires pointing on them to be ever stressed enough to show double column effects. Therefore, the inefficiency in the current setup

is assumed to saturate well before it does with other setups.

The high degree of programmability of the presented setup facilitated its use, but it is the increased number of active wires, individually addressable, plus the improved randomness which stand out in comparison with the EHR prototype.

4.1 Recommendations for Further Studies

During the design and the measurements performed with the presented EHR setup, a few shortcomings of it were revealed which may be addressed in future studies. The most striking ones are listed below.

The ready-made FPGA board was used to shorten development time. However, its number of user programmable I/O connections does not suffice to cover the whole ROC with clusters. Therefore, the FPGA board and the connection PCB should be merged and designed such to route all wires pointing on the active area of the ROC to user programmable I/Os on the FPGA.
Merging the PCBs would also allow to send pulses of varying voltage through the wires as some FPGA architectures are split into banks, each providing individual I/O voltages, causing different pulse heights on the ROC. The FPGA board architecture for the presented setup does only allow to set two voltages, one for each Samtec connector.

Especially if there is no phase shift value eliminating all pulse splittings, the tracks on the PCB should be designed with the timing constraints in mind.

Other ways to guide the pulses to the ROC should also be considered, as the metal block focusing the wires is a very sensitive part of the setup and does not offer any service options. Swapping wires can become somewhat fiddly, depending on their position on the PCB. A different approach may transfer the issue from the PCB to the other end of the wires.

5 Acknowledgments

I would like to thank Prof. Dr. R. Horisberger and his group for offering me the opportunity to do this project and the very pleasant atmosphere I have encountered.
In particular, I would like to thank Dr. W. Erdmann for his guidance throughout the whole project and B. Meier for his assistance and answers related to any kind of question involving electronics.

6 Appendix

Programs and data files can be found at:
https://github.com/wistepha/EHR-1.0
The README.md file contains a brief explanation of all files in the repository.

To exchange the ROC, the following steps must be executed where the mentioned numbers refer to the numbers in the figures 2 and 3:

1. The four screws, marked as 1, connecting the PCB to the base must be removed.

2. Now, the chip board can be pulled out of the adapter. It is recommended to hold on to the chip board and the PCB during the process.

3. The two screws, marked as 2, fixing the chip board to the metal block must be removed. It is recommended to slightly pull the chip board away from the metal block while removing the screws to avoid damage to the wire bonds linking the ROC to the chip board.

4. Fix the new ROC to a chip board.
   Care must be taken of the wire bonds. If the ROC is not glued close enough to the tracks on the chip board the bonds might be destroyed during the process.

5. Screw the chip board to the metal block, push the chip board into the adapter and fix the PCB to the base again.

Table 1: A list of all connections to the FPGA supported by the PCB. The wire label denotes the wire enumerator as it is used in the Verilog code. The net label refers to the connection on the Samtech connectors and is printed on the PCB above the corresponding wire. Finally, the pin label indicates the used FPGA pin.

wire	net	pin	wire	net	pin	wire	net	pin
0	J2_62	U13	53	J2_141	F20	106	J1_100	Y7
1	J2_63	W15	54	J2_142	U17	107	J1_103	P4
2	J2_64	AB18	55	J2_145	D21	108	J1_104	AA7
3	J2_67	J19	56	J2_146	T14	109	J1_105	P3

Continued on next page

Table 1 – continued from previous page

4	J2_68	AA13	57	J2_147	D22	110	J1_106	AB7
5	J2_69	J20	58	J2_148	T15	111	J1_109	U2
6	J2_70	AB13	59	J2_151	C21	112	J1_110	AA8
7	J2_73	H17	60	J2_152	N19	113	J1_111	U1
8	J2_74	AA14	61	J2_153	C22	114	J1_112	AB8
9	J2_75	G18	62	J2_154	N20	115	J1_115	R19
10	J2_76	AB14	63	J2_157	B21	116	J1_116	AA1
11	J2_79	K18	64	J2_158	N18	117	J1_117	R18
12	J2_80	W13	65	J2_159	B22	118	J1_118	Y8
13	J2_81	K17	66	J2_160	N17	119	J1_121	V2
14	J2_82	Y13	67	J2_163	D20	120	J1_122	T10
15	J2_85	M19	68	J2_164	R14	121	J1_123	V1
16	J2_86	AA15	69	J2_165	C20	122	J1_124	T11
17	J2_87	M20	70	J2_166	R15	123	J1_127	R4
18	J2_88	AB15	71	J2_169	G17	124	J1_128	AA9
19	J2_91	N21	72	J2_170	U15	125	J1_129	R3
20	J2_92	AA16	73	J2_171	F17	126	J1_130	AB9
21	J2_93	N22	74	J2_172	V15	127	J1_133	W2
22	J2_94	AB16	75	J2_175	K21	128	J1_134	W10
23	J2_97	H19	76	J2_176	J18	129	J1_135	W1
24	J2_98	AA17	77	J2_177	K22	130	J1_136	Y10
25	J2_99	H20	78	J2_178	J17	131	J1_139	Y2
26	J2_100	AB17	79	J1_61	U9	132	J1_140	U11
27	J2_103	M21	80	J1_62	V8	133	J1_141	Y1
28	J2_104	AA20	81	J1_63	Y3	134	J1_142	V11
29	J2_105	M22	82	J1_64	Y6	135	J1_145	N7
30	J2_106	AB20	83	J1_67	L6	136	J1_146	AA10
31	J2_109	J21	84	J1_68	V6	137	J1_147	P7
32	J2_110	Y14	85	J1_69	M6	138	J1_148	AB10
33	J2_111	J22	86	J1_70	V5	139	J1_151	P6
34	J2_112	Y15	87	J1_73	M2	140	J1_152	P17
35	J2_115	R21	88	J1_74	U7	141	J1_153	R5
36	J2_116	L21	89	J1_75	M1	142	J1_154	R17
37	J2_117	R22	90	J1_76	U8	143	J1_157	U19
38	J2_118	L22	91	J1_79	M4	144	J1_158	T19
39	J2_121	F21	92	J1_80	W6	145	J1_159	U20
40	J2_122	V14	93	J1_81	M3	146	J1_160	T20

Continued on next page

Table 1 – continued from previous page

41	J2_123	F22	94	J1_82	V7	147	J1_163	Y21
42	J2_124	U14	95	J1_85	N2	148	J1_164	V21
43	J2_127	H21	96	J1_86	AA5	149	J1_165	Y22
44	J2_128	W17	97	J1_87	N1	150	J1_166	V22
45	J2_129	H22	98	J1_88	AA6	151	J1_169	W21
46	J2_130	Y17	99	J1_91	P2	152	J1_170	U21
47	J2_133	E21	100	J1_92	AB6	153	J1_171	W22
48	J2_134	P21	101	J1_93	P1	154	J1_172	U22
49	J2_135	E22	102	J1_94	AB5	155	J1_175	T5
50	J2_136	P22	103	J1_97	R2	156	J1_176	N5
51	J2_139	E19	104	J1_98	W7	157	J1_177	R6
52	J2_140	U16	105	J1_99	R1	158	J1_178	P5

As mentioned in section 2.1, the I/Os of the FrontPanel interface are organised in endpoints. For the following list of inputs and outputs of the FPGA these are denoted as: epXXtype[ms:ls], where *XX* is the address and *type* is one of the available endpoint types. The information about the assignment of bits, in case not all 32 bits are used, is contained in the square brackets, where *ms* is the most significant bit and *ls* the least significant bit. Outputs of the FrontPanel interface are generally provided for debugging only, as the USB communication is far slower than the internal clock of the FPGA and can, therefore, not reproduce the FPGA internals accurately. I/Os which are not assigned to an endpoint are directly linked to the FPGA. The length of the signal is given in brackets unless it is one bit.

Input: trigger_in
: Accepts a trigger signal. If the FPGA is set to react to it, the pattern output starts after the trigger is deasserted.

Input: clock_in
: Accepts a clock signal. If the FPGA can process it, switching to this clock becomes possible.

Output: activeclock
: Indicates the used clock, where 1 signals the internal clock and 0 the external clock. The inverse activeclock signal is sent to the LED 1 on the FPGA board. Note, that the LEDs are inverted again, i.e. the LED glows when activeclock is asserted.

Output: cable(159)

Outputs the pulses where each bit is assigned to one wire.

Input: ep00wire[9:0]:
pattern_max(7), pattern_count_ena, patternena

The *pattern_max* bus contains the maximum number of patterns output after a trigger signal is received.

The *pattern_count_ena* signal enables the output of a specific number of patterns if set to 1.

The *patternena* signal chooses the module responsible for the pulse output. 1 activates the fix pattern module and 0 the random pattern module.

Input: ep01wire: rate

Sets the rate parameter r_p, which is sent to all random pattern modules.

Input: ep02wire[5:0]: sel_data_addr(4), wena_sel, read_sel

The *sel_add_addr* bus contains an address of the wire selection memory.

The *wena_sel* signal enables the writing process of *sel_data* to *sel_data_addr*.

The *read_sel* signal determines which of the two memory output words are read to the determine the active wires.

Input: ep03wire: sel_data

The data to be written into the wire selection memory.

Input: ep04wire: mem_data_addr(15), wena_mem

The *mem_data_addr* bus contains an address of the pattern memory.

The *wena_mem* signal enables the writing process of *mem_data* to *mem_data_addr*.

Input: ep05wire: mem_data

The data to be written into the pattern memory.

Input: ep06wire[11:0]: count_mod

Defines the value at which the pattern memory restarts.

Input: ep07wire: phsmax(7), ran_reset, somereset,
phsphase(2), phscounter(3), pllreset, clockswitch

The *phsmax* bus determines the number of steps the phase shift module will perform.

The *ran_reset* signal resets the random pattern module such that the pseudo random number generator restarts with its initial values.

The *somereset* signal was foreseen to reset the phase shift module without resetting the PLL, but it remained unused.

The *phsphase* bus determines whether the phase will be shifted up- or downwards.
The *phscounter* bus selects the post-scale counter of the PLL to be modified, leading to a phase shift.
The *pllreset* signal resets the PLL and the phase shift module.
The *clockswitch* signal enables the switching of a clock. If it is asserted while an external clock is provided, the clock is switched. To switch the clock again, the signal has to be deasserted and reasserted afresh.

Output: ep20wire, ep21wire, ep22wire, ep23wire, ep24wire: activeclock, cable

The *activeclock* signal is output in the most significant bit of *ep20wire* while all other bits and wire endpoints contain the pulse information.

Output: ep25wire[1:0]: phasedone, locked

The *phasedone* signal is output by the PLL. It is asserted whenever the phase of the output clock is constant with respect to the input clock.
The *locked* signal is also output by the PLL. It is asserted whenever the output clock frequency is at or close enough to the designed output frequency.

Glossary

ADC Analogue-to-Digital Converter.

ALICE A Large Ion Collider Experiment. "Detects quark-gluon plasma, a state of matter thought to have formed just after the big bang." [11]

ASIC Application Specific Integrated Circuit.

ATLAS A general purpose detector designed to probe for fundamental particles. [11]

CERN - Conseil Européen pour la Recherche Nucléaire European Organization for Nuclear Research.

CMS Compact Muon Solenoid.

DAC Digital-to-Analogue Converter.

EHR Electrical High Rate.

FPGA Field Programmable Gate Array.

JTAG Joint Test Action Group.

LHC Large Hadron Collider.

LHCb A detector designed to study the "beauty" quark [11].

LHCf A detector designed to study forward scattered particles [11].

LS Long Shutdown.

MoEDAL An experiment designed to find the magnetic monopole [11].

PCB Printed Circuit Board.

PSI Paul Scherrer Institute.

PUC Pixel Unit Cell.

ROC Read Out Chip.

tct "time calibrate trigger". Determines the time between the calibrate signal and the trigger signal.

TOTEM Total, elastic and diffractive cross-section measurement. It is designed to study forward scattered particles [11].

WBC "write bunch crossing counter". The parameter determining the clock cycle to be read out when sending a trigger to the read out chip.

Bibliography

[1] The information used was partially taken from
CMS TECHNICAL DESIGN REPORT FOR THE PIXEL DETECTOR UPGRADE,
ISBN 978-92-9083-380-2
and from the CERN homepage.
http://home.web.cern.ch/about (online 10.02.15)
and its subpages.

[2] *CMS TECHNICAL DESIGN REPORT FOR THE PIXEL DETECTOR UPGRADE*, p.16
ISBN 978-92-9083-380-2

[3] Stephan Wiederkehr, *Electrical High Rate Test*, 2014, ETH Zürich

[4] The information used was partially taken from the Xilinx homepage.
http://www.xilinx.com/fpga (online 10.02.15).

[5] Opal Kelly Incorporated, 13500 SW 72nd Ave, STE 205 Portland, OR 97223, www.opalkelly.com

[6] Beat Meier, Electronics and measuring systems, PSI

[7] Opal Kelly, ZEM User's Manual, 2014, p. 11-12 (flash memory), p. 21 (HSMC connection voltage)

[8] Opal Kelly, FrontPanel, 2014

[9] The data sets were taken and kindly provided by Jan Hoss (ETH) and Hans-Christian Käestli (PSI).

[10] The data sets were taken and kindly provided by Jan Hoss (ETH).

[11] The information was taken from the CERN homepage.
http://home.web.cern.ch/about/experiments (online 10.02.15)
and its subpages.