Internship-Report

Characterization of a PLL circuit used on a 65 nm analog Neuromorphic Hardware System

Aron Leibfried

May 14, 2018

Contents

1 Introduction

The HICANN-DLS 3 chip (High Input Count Analog Neural Network with Digital Learning System) is a neuromorphic chip. The goal of a neuromophic chip is to emulate neural networks as found in the human brain. The aim of the HICANN-DLS 3 is to implement this on analog hardware. It consists of 32 neurons with a corresponding Array of 32x32 synapses. The synapses have individual 6 bit weights, which can be changed. These plasticity processes are the foundation of learning models.

Learning models can be realized with the PPU (Plasticity Processing Unit), which allows implementing flexible learning rules by accessing all of the on-chip memory. It is based on the PowerPC architecture and has a vector unit. It represents a co-processor to the analog circuits.

To clock the PPU with an adjustable frequency a PLL (Phase Locked Loop) is used. The PLL provides the main clock of the digital system components and can be configured via JTAG.

This internship is about the PLL. One goal of this internship is to write a PLL-Config to easily configure the PLL via python. Another goal is to research the characteristics of the PLL. This includes classifing the occuring jitter.

For the experimental part the v3-Baseboard "Jack London" was used together with "Chip 8: Green Bamboo".

2 Phase Locked Loop (PLL)

2.1 General Information

A PLL (Phase Locked Loop) is an electronic circuit, which is used to get an adjustable clock signal. It compares the incoming frequency f_{in} with the frequency of an internal oscillator. This is realized by a control system. The aim is to get an output signal f_{out}, whose phase is related to the input phase. A simple PLL can be seen in figure 1 and consists of four different parts.

The *Phase Comparator* or *Phase Frequency Detector* (PFD) compares the phase of f_{in} with the phase of the DCO (Digitally Controlled Oscillator) and outputs an error signal, which is proportional to the phase difference. The *Loop Filter* delivers the control signal for the DCO, to keep the phase difference on a small level. This can be done by a *PID controller*. The DCO (Digitally Controlled Oscillator) generates the output signal f_{out} according to the settings from the *Loop Filter*. The *Divider* is connected between the DCO and the *Phase Comparator*, to divide the frequency of the DCO by a factor $N \in \mathbb{N}$. So it is ideally: $f_{\text{out}} = N \cdot f_{\text{in}}$.

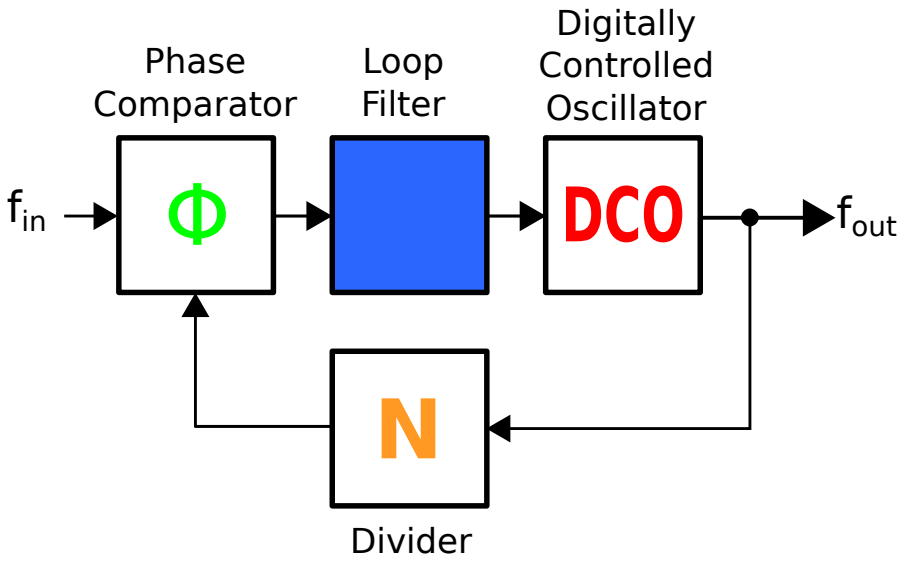

Figure 1: A simple PLL circuit with a *Phase Comparator* to compare the two incoming frequencies. It is connected to the *Loop Filter*, which controls the *Digitally Controlled Oscillator*. It outputs a constant frequency. A *Divider* lowers the frequency, which is compared to the reference clock.

2.2 The PLL on DLS 3

The used clock generator on HICANN-DLS 3 is called hs_clockgen and was designed by the *Technische Universität Dresden* [2]. A so called ADPLL (All-Digital Phase-Locked Loop) is used as PLL. It can be seen in figure 2. It contains two independent ADPLL's with 3 different output frequencies. It has also a total of four independent clock outputs. Another feature is the BIST (Frequency built-in self test) to test the different frequencies.

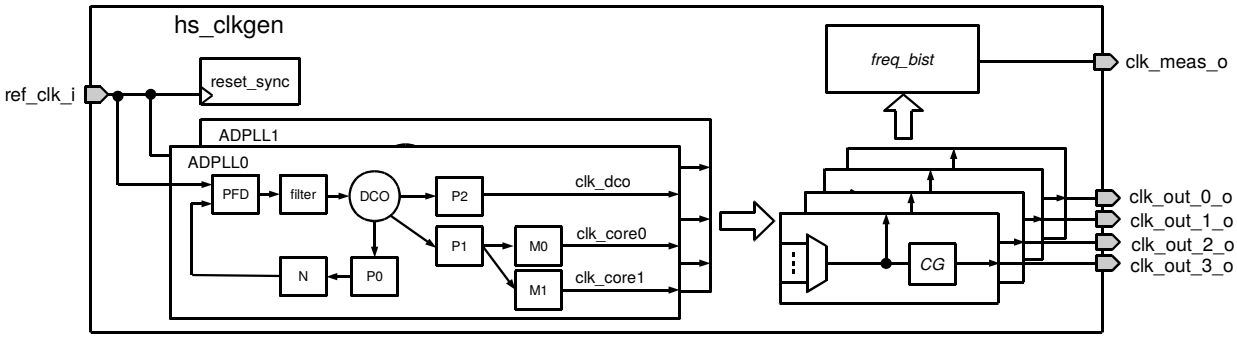

Figure 2: Schematic of the hs_clockgen used in the HICANN-DLS 3 Chip. It contains two ADPLL's and a total of four configurable output pins. Also a Frequency built-in self test is implemented. Figure from [2].

To the ADPLL's is a reference clock with frequency f_{ref} from the FPGA connected. There are several dividers in each ADPLL (see figure 2) to configure the different frequencies, which can be calculated by

$$f_{\text{dco}} = P0 \cdot N \cdot f_{\text{ref}}, \tag{1}$$

$$f_{\text{clk\_dco}} = f_{\text{dco}}/P2, \tag{2}$$

$$f_{\text{clk\_core0}} = f_{\text{dco}}/(P1 \cdot M0), \tag{3}$$

$$f_{\text{clk\_core1}} = f_{\text{dco}}/(P1 \cdot M1). \tag{4}$$

The different possible settings are collected in table 1.

value	N	$P0$	$P1$	$P2$	$M0$	$M1$
max	31	4	4	4	31	31
min	1	2	2	2	1	1

Table 1: Possible settings for the ADPLL used on HICANN-DLS 3.

As reference frequency it is used: $f_{\text{ref}} = 50\,\text{MHz}$

According to [2] it is recommended to keep f_{dco} between 1000 MHz and 2000 MHz, so it is best to set $f_{\text{dco}} = 1500\,\text{MHz}$, which corresponds to $N \cdot P0 = 30$. This was also verified in section 4.7.

Each of the four output pins can be enabled and connected to the different outputs of the different ADPPL's. Also a bypass is possible to get f_{ref} at the output. This will create a stable environment for digital tests, because one uses the $f_{\text{ref}} = 50\,\text{MHz}$ from the FPGA for the whole chip. This mode should not be used for experiments which use the analog part of the chip. Some digital test results (e.g. SRAM) might not be transferable to higher clock frequencies.

As seen in section 4, the digital support circuitry of the chip is driven by $f_{\text{clk\_out\_0}}$. The so called CapMem-Ramp is created with a capacitor and a counter, which is also clocked with the PLL. A current starts flowing to the capacitor while the counter starts. One can measure the actual voltage over the capacitor. When the counter value is reached, the capacitor gets discharged and the counter resetted. When the counter reaches his counter value a second time, the capacitor will be charged again. So the CapMem-Ramp frequency is proportional to the frequency at this output. Also the PPU is clocked with this frequency.

As mentioned above, there is also a built-in self test (BIST) contained in the hs_clockgen. This allows testing the clock generator by counting the cycles of the selected output clock $f_{\text{clk\_out}}$ within a specified number of reference clock cycles f_{ref}. The specified number is set by a selectable pre-scaler value p as 2^{p+2}. This leads to the expected counter value

$$\text{counter\_value} = \frac{f_{\text{clk\_out}}}{f_{\text{ref}}} \cdot 2^{p+2}. \tag{5}$$

With configuring the PLL with the expected counter_value, the test starts and the cycles are counted. Then both values are compared within a configurable tolerance range (check range). The included pass/fail checking unit outputs whether the test failed or was a success.

The PLL can be configured via JTAG. There are 10 configuration registers, each with 32 bits. The instruction register width is 4 bits. It's important to mention that the register numbers and the according JTAG instruction numbers are shifted by a factor of 3. I.e. register 0 can be configured with the JTAG instruction register 3 [1].

In the default hardware settings $f_{\text{clk\_core1}}$ from the ADPLL0 is connected to the clk_out_0 pin (See figure 2), which drives the digital circuitry. This means after every chip reset, the chip will run with this frequency.

This default setting causes problems, as seen in section 4.2. The problem can be solved by using the PLL-config container as described in section 3.

Because most of the experiments, which were made on this chip, ran with the ADPLL0 and their clk_core1 output, in section 4 just this configuration will be studied. Other possible configurations are not covered by this internship-report.

3 The PLL-Config Container

To change the different parameters of the PLL, a python-based PLL-config container is used to easily configure the PLL. After the creation of an instance of this class, the different parameters can be changed and exported to the PLL.

To configure the PLL, the `export_data` command have to be called after the parameters inside the class have been changed. To get the hardware configuration a `import_data` command is possible. By printing the class, one will get the actual configuration of the PLL. By using the `frequencies` function, one will get information about the different frequencies of the different ADPLL's.

As mentioned above, the configuration is written to the PLL via JTAG. Until now it is just possible to write on the JTAG via Impact (See `ImpactJTAGDriver`). In the future it will be possible to contact this with a FPGA-driver. The Driver can be changed by setting the `driver` value to the preferred driver. By default it is set to the `ImpactJTAGDriver`.

Name	min	max	default hardware value	ADPLL0-config value
`loop_filter_int`	1	31	2	2
`loop_filter_prop`	1	31	8	8
`loop_div_N`	1	31	10	10
`core_div_M0`	1	31	4	2
`core_div_M1`	1	31	2	1
`pre_div_P0`	2	4	2	2
`pre_div_P1`	2	4	3	3
`pre_div_P2`	2	4	2	2
`tune`	0	4095	512	512
`dco_power_switch`	0	63	63	63
`open_loop`	0	1	0	0
`enforce_lock`	0	1	0	1
`pfd_select`	0	1	0	0
`lock_window`	0	1	0	0
`filter_shift`	0	3	0	3
`disable_lock`	0	1	0	0

Table 2: The tunable parameters of the ADPLL in the hs_clockgen with min/max possible values and the standard settings on hardware and in the container.

Table 2 includes the parameters for the ADPLL. In the PLL-config container one have to add `_pll0` or `_pll1` to change the ADPLL0 or ADPLL1 configuration. The table also contains the different standard values of the ADPLL's (hardware and class settings). For the ADPLL1 the container holds the same configuration as the hardware, but for the ADPLL0 they are different. This is because of a problem with the standard values on hardware, explained in section 4.2. So if an instance of the PLL-config container is created and they data gets exported to the PLL, the settings on the ADPLL0 will

change to the default settings held by by the PLL-config container! This will cause a fix, because the ADPLL0 is connected to the clk_out_0 output by default.

Name	Description
`enable_clock_clk`	Enables the output of the pin: 0 for disable, 1 for enable
`enable_bypass_clk`	Sets pin to bypass mode (FPGA-Clock) by setting it to 1
`select_adpll_clk`	Select which ADPLL should be conntected
`select_clock_clk`	Selects the ADPLL output: 0 for clk_core0, 1 for clk_core1 and 2 or 3 for clk_dco

Table 3: Configuration parameters of the hs_clockgen output pins.

It is also possible to change the configuration of the different output pins clk_out_k with k in [0:3]. Each pin has four parameters, collected in table 3. To change the according pin, one have to add `_k`, with k as the pin you want to change. The standard settings of the output pins can be found in table 4.

Output-Pin	Enabled	Bypass	ADPLL	Clock
0	yes (1)	no (0)	0	clk_core1
1	yes (1)	no (0)	0	clk_core0
2	yes (1)	no (0)	0	clk_dco
3	yes (1)	no (0)	1	clk_core1

Table 4: Hardware settings of the output pins.

To execute the built-in self test the function `self_test` can be used. It uses the values collected in table 5. It is not recommended to change the `check_value` parameter, as the function calculates the expected value according to equation 5.

Name	Std	Min	Max	Task
`pre_scaler_p`	8	0	15	pre-scaler p, explained in 2.2
`select_source`	0	0	3	Choose the output pin which should be tested
`check_range`	2	0	15	Tolerance range to accept the results
`check_value`	-	0	$2^{20} - 1$	Expected Counter Value

Table 5: BIST-Function parameters.

It's important to note that `self_test` uses the `export_data` function at the beginning. So it is important to note that previously changed parameters on the PLL are changed according to the configuration in the PLL-config class.

If the test failed the function will return `False`, otherwise `True`. The function will print the used ADPLL and the according output with its frequency if `print_info` is set to `True`. The counter values are also compared and the result is also printed when `print_info` is set to `True`.

4 PLL-Measurements

Now different measurements are performed, to get more information about the functionality of the PLL. If not other specified, the standard PLL-config values from table 2 are used.

4.1 Measurements of the Capacitive Memory Ramp

We measure the frequency of the CapMem-Rampout (f_{CAP}) (see section 2.2) for different settings of the PLL. f_{CAP} is directly related to $f_{\mathrm{clk\_core1}}$. f_{dco} gets observed to find a good frequency range. For this measurement $M0 = M1 = 31$ and $P1 = P2 = 4$ are fixed values.

We measure for different values of N for a given $P0 = 2$:

N	1	2	3	4	5	6	7	8	9	10
f_{CAP} [Hz]	212.5	10.7	17.5	23.3	29.1	35.0	40.8	46.6	52.4	58.3

N	20	21	30	31
f_{CAP} [Hz]	116.5	122.3	174.8	180.6

Table 6: Measurement of f_{CAP} for $P0 = 2$ and different N.

If the value of N get doubled, the according f_{CAP} should also be doubled. As we can see in table 6, this happens for $3 \leq N \leq 31$. For $N = 1$ we get the maximum CapMem-Rampout frequency 212.5 Hz (See table 7). The value for $N = 2$ also doesn't fit into the expectations.

We measure for different values of N for a given $P0 = 4$:

N	2	10	15	16	18	19	20	25	30
f_{CAP} [Hz]	23.3	116.5	174.8	186.4	209.7	212.5	212.5	212.5	212.5

Table 7: Measurement of f_{CAP} for $P0 = 4$ and different N.

We can see in table 7, that the CapMem-Rampout frequency has its maximum at 212.5 Hz. For $N \leq 18$ we get the results we expected. But for $19 \leq N$ we get a maximum value of f_{CAP}.

As seen above, f_{dco} works stable for $4 \leq P0 \cdot N \leq 72$. So we get the frequency range of the PLL

$$4 \cdot f_{\mathrm{ref}} = 200\,\mathrm{MHz} \leq f_{\mathrm{dco}} \leq 3600\,\mathrm{MHz} = 72 \cdot f_{\mathrm{ref}}. \tag{6}$$

The same results can be measured on different chips. Two additional chips were tested: "Chip 7: Green Cheese" and "Chip 3: Indigo Hammer".

4.2 Measurements with the PPU

To get better results and to automate the measurement the PPU is used. Most of the instructions executed by the PPU will take one clock cycle. We used a PPU application to toggle one of the Input/Output (GPIO) pins by setting the output pin to high and low for a specified period of time. Including the time to execute a jump instruction, the PPU can toggle the pin with a period of three clock cycles. Inserting a configurable number of NOP's we can scale the toggle frequency f_{PPU} by

$$m_{\mathrm{PPU}} = 3 + 2 \cdot N_{\mathrm{NOP}}, \tag{7}$$

$$f_{\mathrm{clk\_core1}} = m_{\mathrm{PPU}} \cdot f_{\mathrm{PPU}}. \tag{8}$$

With $m_{\mathrm{PPU}} = 15$ the trace of the PPU, with an unconfigured PLL connected, is measured with an oscilloscope. A frequency of $f_{\mathrm{PPU}} = 11.11\,\mathrm{MHz}$ is expected. The signal looks gated with a clock signal of around 50 KHz as seen in figure 3. During a "clock-high-signal" the expected frequency can be investigated as seen in figure 4 on the left side. Between the "clock-high-signal" can happen different things. It can be a "clock-low-signal" or it stays high with some peaks to the ground, as seen in figure 4 on the right.

This can be easily fixed by keeping all settings as they are and setting `enforce_lock` to 1. With this setting the PLL never stops and a frequency of 11.11 MHz can be measured. The difference can be clearly seen by comparing figure 3 with figure 5 and figure 4 with figure 6.

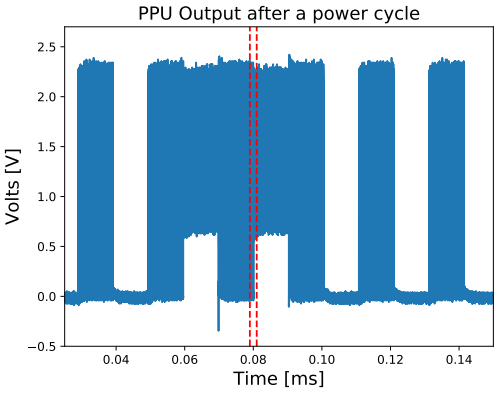

Figure 3: PLL hardware settings.

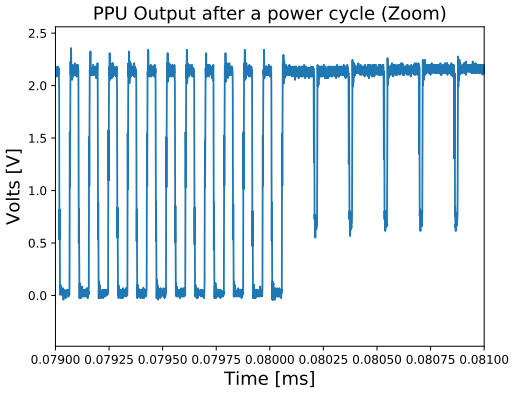

Figure 4: PLL hardware settings.

Because of this expectation, you should never set `enforce_lock` to 0. By default the PLL-container sets this parameter to 1, to fix the problems explained above.

4.3 Determine the "hang up" frequency

With an value of $m_{\mathrm{PPU}} = 7$ some tests with different PLL-settings are done. The PLL-config values of table 2 are used and the parameter N gets varied to make a conclusion about the PLL. The data is collected in table 8.

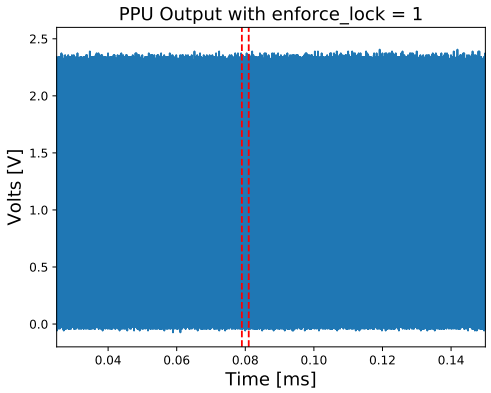

Figure 5: PLL hardware settings with enforce_lock = 1.

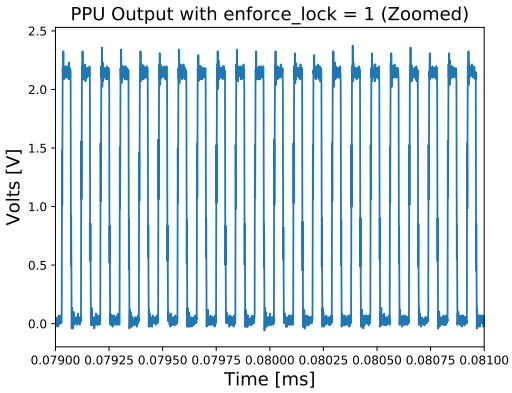

Figure 6: PLL hardware settings with enforce_lock = 1.

N	1	2	3	4	6	8
f_{PPU} [MHz]	-	7 - 11	12 - 15	18 - 20	27 - 29	38.0 - 38.1
f_{theo} [MHz]	4.76	9.52	14.29	19.05	28.57	38.10
N	9	10	12	14	15	
f_{PPU} [MHz]	42.6 - 43.1	47.5 - 47.9	57.1 - 57.3	66 - 67	-	
f_{theo} [MHz]	42.86	47.62	57.14	66.67	71.43	

Table 8: Measurement of f_{PPU} with $m_{\text{PPU}} = 7$ for different values of N. If there is no entry for f_{PPU} the chip "hanged up".

As seen in section 4.1 the PLL is't stable for $N = 1$. That's the reason why the chip "hang up" with this settings. The other values fit with the expactation, but for low N the error is pretty high. For $N \geq 15$ the chip also crashes. This corresponds to $f_{\text{clk\_core1}} = 500\,\text{MHz}$. We can conclude that the chip will "hang up" if $f_{\text{clk\_core1}} \geq 500\,\text{MHz}$.

4.4 Problems with the PPU

With the "High" and "Low" output of the PPU one would expect a rectangle signal as seen in figure 6. But for some settings we get a different signal, compare to figure 7 and figure 8. This measurement is done with $m_{\text{PPU}} = 3$, but some things also happen with a higher value of m_{PPU}.

With a high frequency, for example $f_{\text{clk\_core1}} = 250\,\text{MHz}$, the signal doesn't look like a rectangle signal how it should be (see figure 7). The signal looks more like a random signal. Maybe the frequency is to high for the PPU or the measurment technique with the oscilloscope isn't the best way to do this with such high frequencies. This can be fixed with a higher m_{PPU} value.

For low frequencies, for example $f_{\text{clk\_core1}} = 4.2\,\text{MHz}$, the signal looks like a rectangle signal (see figure 8). The problem is a "bad" peak in the middle of the "High"-Signal.

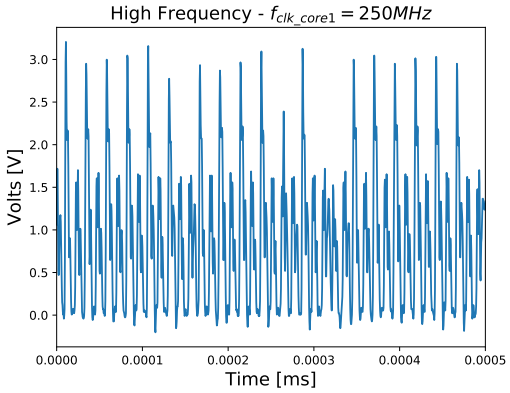

Figure 7: PPU-Output with $m_{PPU} = 3$: $N = 15$, $M1 = 2$, $P0 = 2$ and $P1 = 3$.

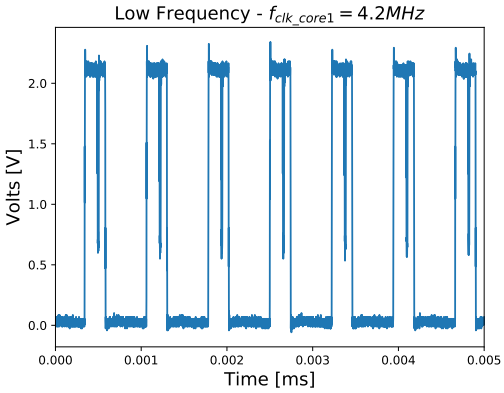

Figure 8: PPU-Output with $m_{PPU} = 3$: $N = 5$, $M1 = 30$, $P0 = 2$ and $P1 = 4$.

This peak doesn't make sense and with even lower frequencies more 'bad" peaks appear. This problem cannot be fixed with a higher m_{PPU} value, so it can be a problem with power supply or a problem with the PPU itself. Maybe this "bad" peaks and the "bad" peaks from figure 4 are related to each other.

4.5 Automated measurement series

To automate the measurement and to classify the jitter of the PLL, a measurement series was done. As seen in section 4.3, it is possible to "hang up" the chip. A power cycle would be necessary to run it again. To be sure that the chip is running on a safe operating point, the PLL-values were restricted. Because of the previous measurements the area was chosen with $4 \leq P0 \cdot N \leq 72$, $1 \leq \frac{P0 \cdot N}{P1 \cdot M1} \leq 9$ and $N \geq 2$. In total 2528 single measurements were made.

To classify the jitter of the PLL it would be best to measure with no NOP's. That's because the deviation of the timing gets lower with more operations, because the mean is taken. But to classify the jitter with $m_{\mathrm{PPU}} = 3$ is also a problem, see section 4.4. As a compromise $m_{\mathrm{PPU}} = 15$ is used.

An oscilloscope can be accessed via ethernet connection, to collect the trace data. It would be possible to store every signal and to evaluate them all after the measurement. But every trace takes more than 100 MB, so more than 200 GB would be needed. Also the evaluation would take a long time. It is more efficient to evaluate every signal directly in the measurement series.

For every trace are the times of the rising slopes determined and stored. They can be used to determine the frequency and with this information also the jitter can be classified by statistical methods.

4.6 Frequency and the corresponding error

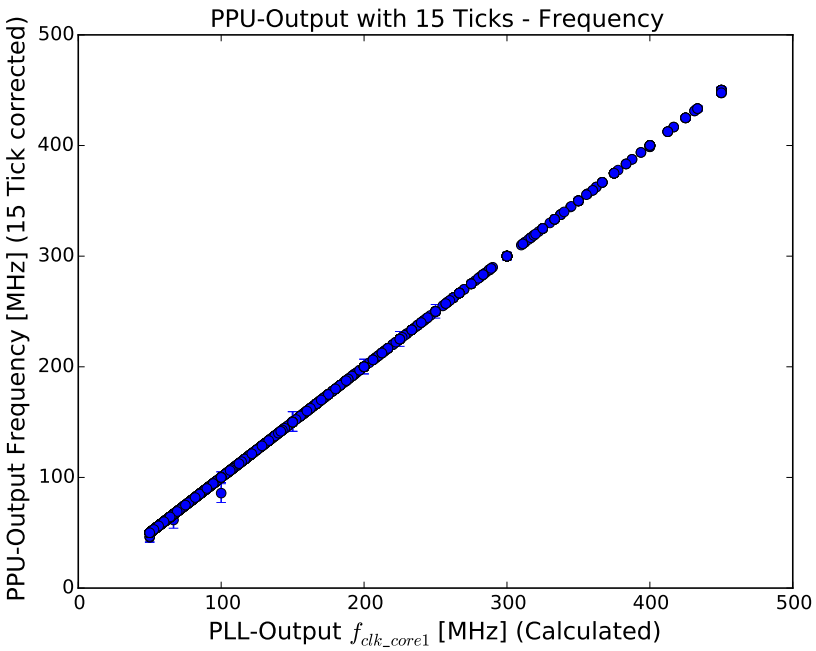

Figure 9: $f_{\text{clk\_core1}}$ against the corrected f_{PPU} gives a slope of one.

With the data of the measurement series the PPU-Frequency f_{PPU} can be determined by dividing one by the measured times and taking the mean. With this information also the standard deviation for one measurement can be calculated with statistical methods. By correcting f_{PPU} with a factor of $m_{\text{PPU}} = 15$ this should give a line when plotted against $f_{\text{clk\_core1}}$. The results can be seen in figure 9.

By calculating the coefficient of variation $\frac{\sigma}{\mu}$ of the frequency f_{PPU} and plotting it, which is done in figure 10, you can see that many points have a very small coefficient $\frac{\sigma}{\mu}$. But there are also points with an error higher by two orders of magnitude. If you plot some characteristic PLL-Settings, you can see that small values of $M1$ are causing a high jitter. That's because the $M1$-divider cuts slopes to lower the frequency by its amount. By cutting many slopes, the jitter of a single peak doesn't matter to much and so the total jitter lowers. But with $M1 = 1$ no slopes are cutted. So we can measure the "whole" jitter of f_{dco} (We also have to take $P1$ into account). A better research in this is done in section 4.7.

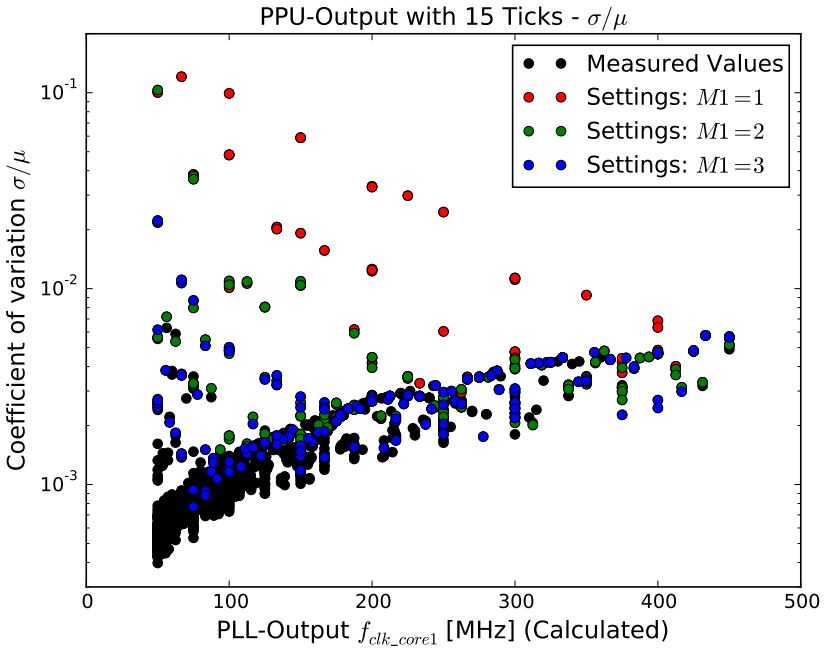

Figure 10: Coefficient of variation $\frac{\sigma}{\mu}$ of the frequency f_{PPU}.

4.7 DCO-Frequency

Now we want to classify the jitter for different values of f_{dco}. The dividers $P1$ and $M1$ reduce this jitter, because they cut many slopes (Compare to section 4.6). Because of the chosen PLL-values it is also not possible to search fixed $P1$ and $M1$ values and vary $P0$ and N for f_{dco}. For every measurement the standard deviation $\bar{t}$ of the period time t is determined. The period t is just the mean of the measured values (section 4.5). The real error Δt can be calculated by error propagation. We can calculate Δt by

$$\Delta t = \sqrt{P1} \cdot \sqrt{M1} \cdot \bar{t}. \tag{9}$$

The data can be seen in figure 11. The marked area is the originally recommended range from [2]. You can see that the jitter is pretty low in this area how it should be. For lower frequencies than 800 MHz the jitter rises and gets pretty big. This is especially when $N = 2$ ($N = 1$ wasn't measured, see section 4.3). For higher frequencies the jitter also rises, but it isn't too high. It should be possible to use the PLL with an f_{dco} till 3500 MHz.

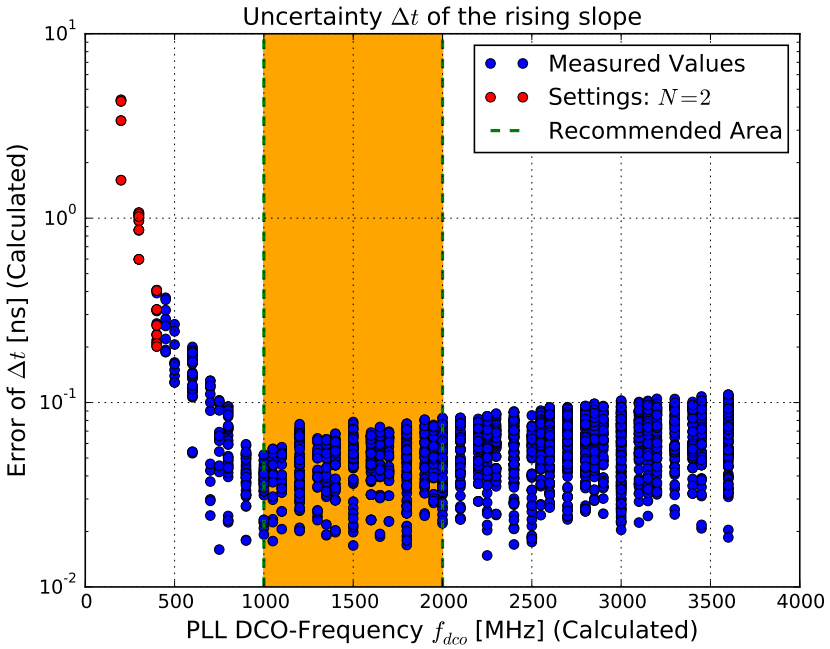

Figure 11: The jitter Δt for different values of f_{dco}.

5 Discussion

The PLL-config container works fine as expected. The only issue is that you have to use `Impact` as driver. So the chip must be connected via a "proprietary programming cable" to the Server. Sometimes this method is locking the cable and you have to fix this problem with the Impact-Shell.

The PLL however works fine with the standard settings of the container. The only issues are the reset parameters when you restart the chip. This should change for the next generation of HICANN. It's recommended to configure the PLL with the PLL-Config when you restart the chip.

By using the recommended area of f_{dco} (1000 MHz - 2000 MHz), the jitter can be lowered. Also the dividers $P1$ and $M1$ shouldn't be too high. In case of power consumption a low value of f_{dco} would be preferable. So it should be best to set $f_{\mathrm{dco}} = 1000\,\mathrm{MHz}$, which is also the default PLL-config container value for the ADPLL0.

References

[1] Andreas Hartel and Johannes Schemmel. *Specification of the HICANN-DLS ASIC*. 2018.

[2] Sebastian Höppner and Stefan Scholze. *TUD HPSN Clock Generator Specification for HICANN DLS*. 2016.