Documentation

Digital FPGA-based Phase Locked Loop

by Lucas Ludwig

lucasludwig97@gmail.com

Version 1.3

09.12.2023

Table of Contents

1. Structure

The purpose of a Phase-Locked Loop (PLL) is to synchronize the phase and frequency of an input signal with a reference signal, allowing for precise information about the input signal.

1.1 Digital Phase Locked Loop

Figure 1 visualizes the conceptual structure of a digital phase locked loop. The input signal V_i is multiplied with a sine and cosine of the reference signal, which oscillate with an approximation f_{PLL} of the 'real' frequency f_i of V_i. The mixed signal is filtered using a low pass, which will only keep the difference frequency of f_i and f_{PLL}. Using atan2, the phase relation $\Delta\varphi$ between input and reference can be computed. This value is used as error measure of a PI-controller (Proportional Integral), which tries to minimize $\Delta\varphi$ by adjusting the reference frequency f_{PLL}. After several iterations, f_{PLL} will be equal to f_i. The detected frequency can be applied to a Lock-In, which extracts the signal amplitude (V_{sig} contains a modulated signal with frequency f_i).

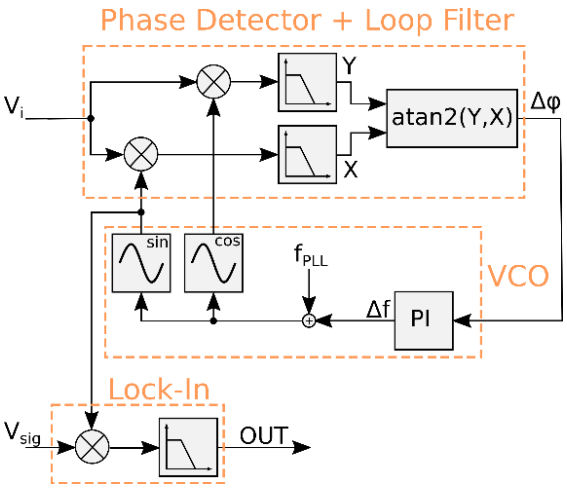

Figure 1: Schematic structure of a digital PLL (Lock-In not included in real software).

1.2 LabVIEW Implementation

Figure 2 illustrates the VI structure of the PLL implementation in LabVIEW. The FPGA board (NI myRIO) performs the actual signal processing, as depicted in Figure 1, using the 'FPGA Main' VI. An integrated microcontroller manages programming, I/O interfaces, and data management. It also runs the 'RT Main' VI which is responsible for data synchronization of specific variables. Connecting the device to a PC via USB or Ethernet enables bidirectional data transmission. The PC manages the user interface in 'PC Main,' sending updated parameters and tasks to 'RT Main.' Simultaneously, 'RT Main' transmits measured and processed data to 'PC Main,' which can be utilized for data visualization and storage.

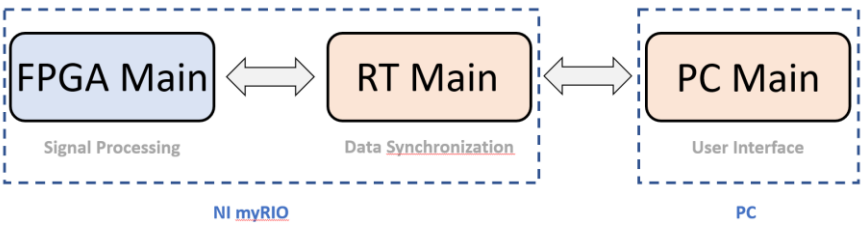

Figure 2: Fundamental VI-structure of the LabVIEW software.

FPGA-Main

As implied by its name, the FPGA-Main operates on the FPGA myRIO, implementing the core functionality of the PLL. This software segment is optimized for memory and timing considerations, accommodating hardware limitations. After any modifications in the block diagram, a new compilation into a bit file is required (with a compilation time of approximately 10-15 minutes).

Figure 3 depicts the LabVIEW VI 'FPGA Main', structured into distinct functional sections. Notably, the VI is designed to manage up to two PLLs simultaneously (three channels for fixed filter variant).

In **Section (1)**, the input voltage V_i undergoes digitization and is converted to fixed-point numbers (FXP). To mitigate discretization issues with small-amplitude signals, a preamplification factor, referred to as 'gain,' is applied through multiplication. The objective is to ensure that the resulting FXP signal falls within the range of [-1, 1].

Section (2) implements the core PLL functionality. Initially, the input signal is multiplied by the sine and cosine of the reference signal and then filtered using multiple digital low-pass filters, with the option to include a bandstop filter. The low-pass filter's cutoff frequency (and order) impacts the PLL response time, where a lower cutoff frequency leads to a slower response. Conversely, an increasing cutoff frequency enhances noise sensitivity and may introduce crosstalk with other modulation frequencies. Consequently, the cutoff frequency should be substantially smaller than any difference between modulation frequencies while still large enough to maintain a fast response, with typical values in the order of 4 kHz. A higher order for the low-pass filter is desirable to minimize its impact on the signal of interest.

Additionally, a bandstop filter can be used to eliminate specific frequencies components (e.g., modulation frequencies 20 kHz and 32 kHz lead to 12 kHz noise, so the bandstop center frequency should be set to 12 Hz). The bandstop 3-dB-width depends on the typical temporal variation of modulation frequencies, with a higher fluctuation requiring a broader bandwidth.

Subsequently, the phase difference ($\Delta\varphi$) between the input and reference oscillations is computed using atan2. Mixing and phase detection are executed in parallel. To optimize memory consumption, filtering is sequentially applied to all four components (2×sine and cosine). Shift registers inserted between the filters split the total computation into multiple loops for timing considerations.

The detected phase error $\Delta\varphi$ serves as input to a Proportional-Integral (PI) controller in **section (3)**, incorporating Proportional (P) and Integral (I) gain factors. The objective of the PI controller is minimizing $\Delta\varphi$. The output of the PI controller provides a correction for the internal reference frequency, ideally synchronizing it with the actual input frequency for an optimal phase match.

In **section (4)**, the internal reference phase is updated using the corrected reference frequency. The resulting reference phase is then utilized in **section (5)** to compute the reference sines and cosines, which are crucial components in **section (2)**.

Furthermore, the reference sine is employed in **section (6)** to generate trigger output pulses. The goal is to emit a voltage pulse precisely once per oscillation. Zero-crossings of the reference sine are detected, generating an index in the range of ±[1, 80]. This index approximates the actual time of zero crossing, considering the modulo operation with respect to the PLL loop time. This approach maximizes the time precision of the output

pulses, given that the PLL operates at 500 kHz, while the FPGA can handle up to 40 MHz (40 MHz/500 kHz = 80 ticks per PLL loop). The sign of the pulse index corresponds to a rising or falling zero crossing of the reference sine. The resulting PLL trigger output signal is rectangular shaped with zero-crossing corresponding to the input signal.

In **section (7)**, synchronization with external 40 MHz trigger-pulse loops is managed using handshakes.

Section (8) is responsible for handling the output of trigger pulses for each PLL channel.

In **section (9)**, communication between FPGA-Main (on the FPGA) and RT-Main (on the internal microcontroller) is implemented, facilitating data transfer to the FPGA-USB interface. To fit the data in an array with a consistent data type, values are normalized accordingly before being placed into a FIFO.

Realtime-Main

RT-Main implements the interface between the PC and FPGA, synchronizing variables such as PLL parameters and reading data from the FIFO. This VI must run while using the PLL to perform any communication with the FPGA.

PC-Main

PC-Main serves as the User Interface, allowing real-time adjustment of various PLL parameters. It also facilitates the plotting of different types of data and analyzes the behavior of the PLL. A detailed description of the User Interface is provided in chapter 3.

1.3 PLL Versions

The dominant limitations of the FPGA are memory and time constraints.

To enable best performance, all PLL sections are optimized for those two aspects. Depending on the requirements of the applications, there are two versions of the PLL available.

Version 1 (foldername: *PLL_3channels_fixed-filtering*) provides three PLL channels but has a fixed implementation of filter parameters. Properties such as cutoff frequency can only be modified in the FPGA Main VI, which requires a recompilation of the entire VI (~15 minutes).

Version 2 (foldername: *PLL_2channels_variable-filtering*) implements only two PLL channels but enables runtime adjustments of filter parameters. This might be beneficial for optimization purposes (e. g., finding optimal filter parameters for a given input noise and response time requirement. Filter properties are accessible in PC Main, as any other PLL parameter.

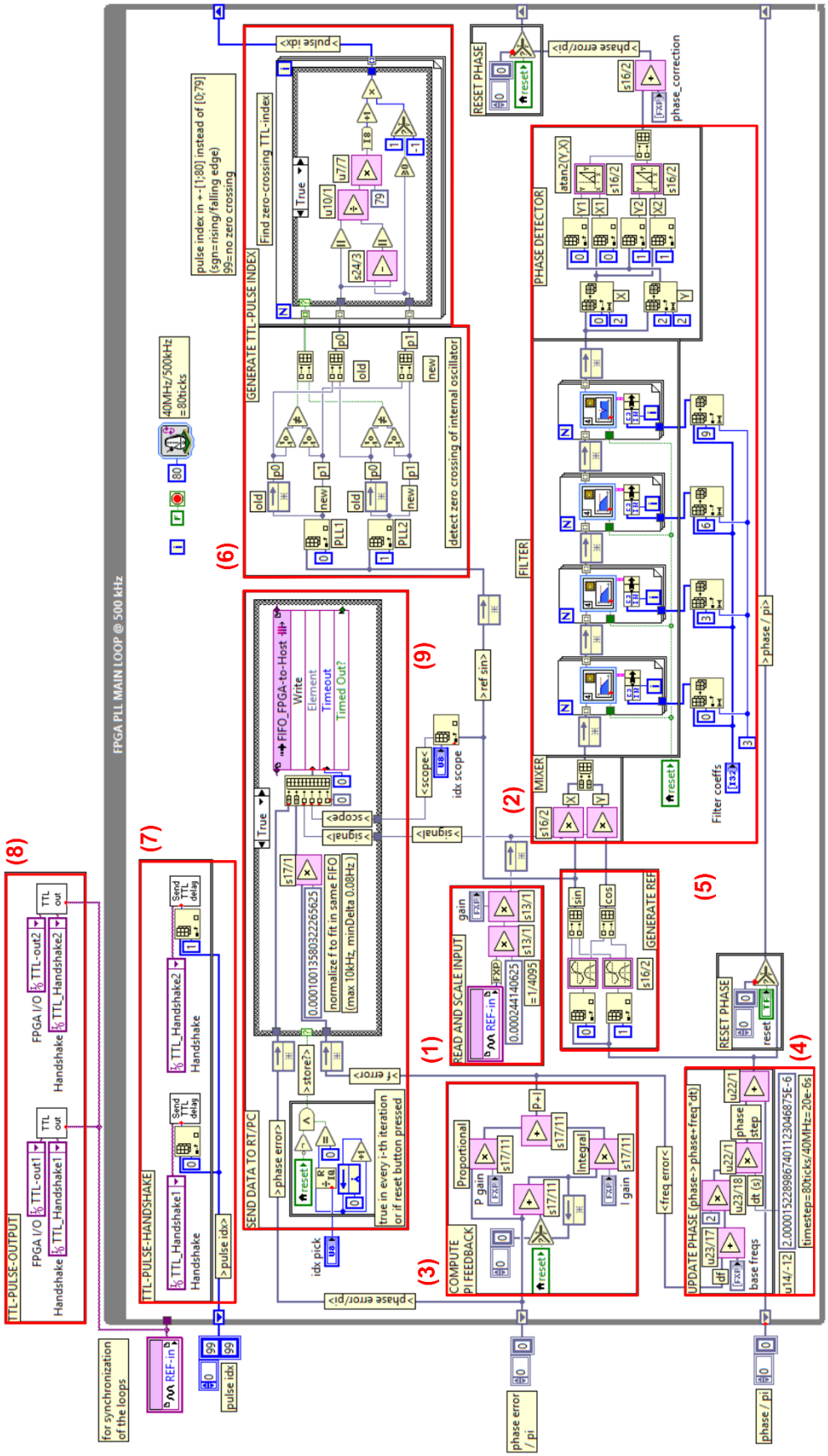

Figure 3: FPGA Main VI (version 2 with variable filters and 2 PLL channels).

2. Start-up

Operation of one FPGA-device

1. Connect the FPGA-board (NI myRIO) to a PC (using USB) and to a power supply.
2. Connection oft input/ output signals:

Input signal:	*Port AI0+/AI0-*
Output trigger signal PLL1:	*Port DIO0*
Output trigger signal PLL2:	*Port DIO3*
Output trigger signal PLL3:	*Port DIO7*

3. Open the PLL-project in a **32bit-version** of LabVIEW.
4. In case of first use with this PC: Compilation of FPGA Main necessary.
   In project tree: 'My Computer / NI-myRIO-1900-0307b121 / Chassis / FPGA Target / Build Specifications' → Right mouse button on FPGA_Main → Build (or alternatively Rebuild) → 'Use local Compile Server', OK
   (typical compilation time: 15 minutes)
5. Open the VI 'PC_Main' (in project tree in 'My Computer/') and the VI 'RT_Main' (in project tree in 'My Computer / NI-myRIO-1900-0307b121/')
6. Run the VI 'RT_Main' (normal run, not infinite loop)
7. Run the VI 'PC_Main' (normal run, not infinite loop)
8. Configure the PLL-parameters for your application (see section 3)
   (note: you can save your configuration (when the VI is NOT running) by
   Right mouse button on the parameter → 'Data Operations' → 'Make current value default')

Operation of two FPGA-devices simultaneously with one PC

(e. g. this PLL implementation in combination with an Überscan-PLL)

The Überscan-FPGA must be connected FIRST with the PC via USB. Next, connect the FPGA for this PLL. (Reason: auto-IP-selection of the FPGAs occurs in order of connection, IP configuration more complicated in Überscan).

Next, the IP-address of the second PLL (this PLL) must be determined using the software 'NI Max'. This Address must now be applied to the LabVIEW-PLL-Project (Right mouse button on 'NI-myRIO-1900-0307b121' in the project tree → Properties → General → IP Address / DNS Name).

The FPGA that was connected first has typically the IP-address 172.22.11.2, the second FPGA 172.22.11.10.

3. User Interface

Figure 4 shows the user interface of the VI ‘PC Main’.

(a) 3 PLL channels, fixed filters. (b) 2 PLL channels, var. filters

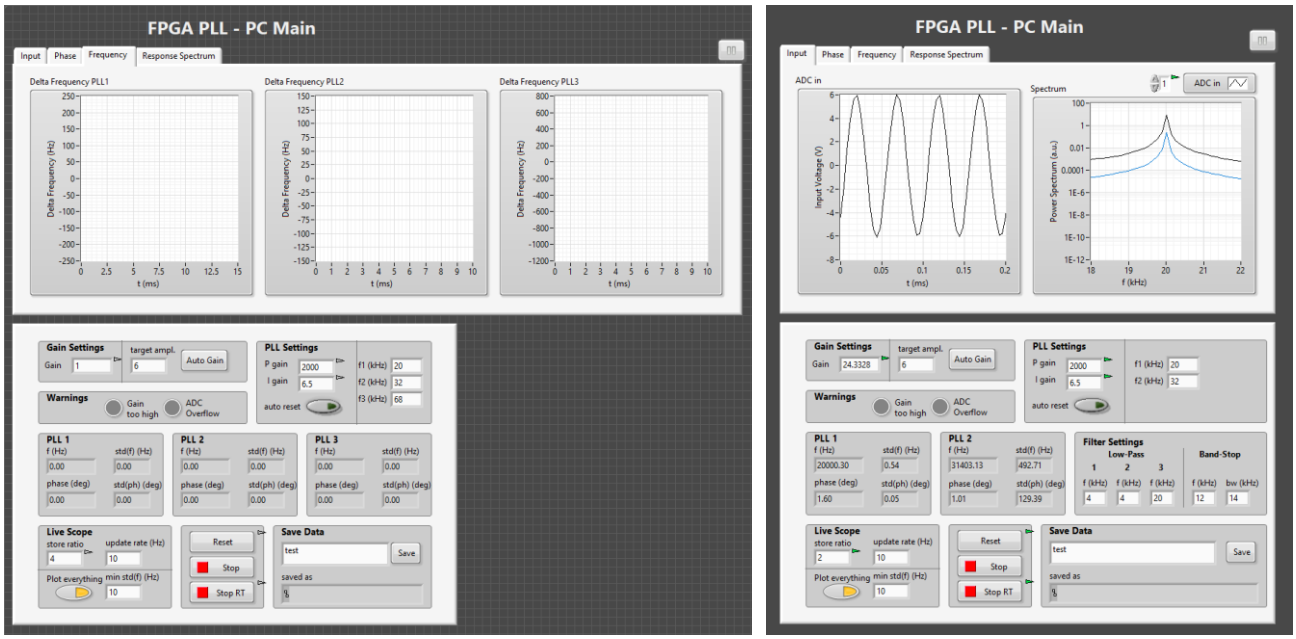

Figure 4: User interface of ‘PC Main’.
(a) Version 1 with 3 PLL channels and fixed filter parameters.
(b) Version 2 with 2 PLL channels and runtime-variable filter parameters.

3.1 Tabs in the upper region of the UI

a) Tab ‘Input’

Plot on the left: time trace of the input voltage

x-axis: time in milliseconds

y-axis: Voltage in volts (corresponds to the input voltage, multiplied with the parameter ‘Gain’ (see section ‘Gain Settings’)

Note: The time trace has a length of 2048 data points, the time step is determined by the parameter ‘store ratio’ in the UI-section ‘Live Scope’. With store ratio=1, for every iteration of the 500 kHz sampling loop, one data point will be stored. With store ratio=2, only in every second iteration (etc.).

The **time step** can be calculated with

$$\Delta t = \frac{\text{store ratio}}{500\ \text{kHz}}$$

and the **total length** of the time signal with

$$T = 2047 \cdot \frac{\text{store ratio}}{500\ \text{kHz}} \quad .$$

Plot on the right: power spectrum of the input voltage

x-axis: frequency in kHz

y-axis: power spectrum in a. u.

Note: The properties of the spectrum depend on the discretization of the input signal (properties of fourier transform).

Max. detectable frequency: $f_{\max} = \frac{1}{2\Delta t} = \frac{250\ \text{kHz}}{\text{store ratio}}$

Frequency resolution $\Delta f = \frac{1}{T} = \frac{500\ \text{kHz}}{2047 \cdot \text{store ratio}}$

b) Tab ‘Phase’

Plots of the detected phase deviation between input signal and the local oscillator (should be ideally 0 degrees).

x-axis: time in milliseconds

y-axis: phase deviation in degrees

c) Tab ‘Frequency’

Plots of the detected frequency deviation (internal frequency correction relative to the base-frequency of each PLL).

x-axis: time in milliseconds

y-axis: frequency deviation in in Hz

Note: The visible frequency deviation does not correspond to the ‘real’ frequency deviation for fast frequency variations, due to the limited response time of the PLL (depending on the PLL parameters, see section 4).

d) Tab ‘Response Spectrum’

Plots of the spectra of the detected frequency deviation (relative to the selected base frequency). This plot is useful to detect periodic perturbances in the PLL-response (for example components of other modulation frequencies, which were not suppressed entirely by the PLL-filters.

x-axis: frequency in kHz

y-axis: spectrum of detected frequency deviation in a. u.

Note: The properties of the spectrum depend on the discretization of the input signal (see section 3.1.1).

e) Additional plot features

Pause-Feature

The refresh process of the plots can be paused using the button on the top right corner of the UI. This can be useful to compare frequency- and phase response of the same time section.

Refresh-Rate

The update rate of all properties in the UI can be adjusted in the UI-section ‘Live Scope’ via the parameter ‘update rate in Hz’. A value of 10 Hz is recommended.

3.2 Bottom region of the UI (PLL-parameters and status)

a) Gain Settings

Parameter ‘Gain’

Factor which is used to minimize internal discretization errors (applied to ADC input before any other computation). The plot ‘ADC in’ already includes this factor. The value should be chosen such that the maximum amplitude of the input signal is in between 5-8 V. Ideally, the input signal should be pre-amplified analogously to prevent discretization losses (12bit ADC). Figure 5 visualizes the effects of ‘Gain’.

An appropriate value for ‘gain’ can be found using the auto-gain-feature (see corresponding section).

Parameter ‘target ampl.’

Amplitude of the input signal after digital gain (used for auto-gain feature). A typical value is 6 V.

Button ‘Auto Gain’

Automatically sets the gain-parameter in such a way, that the digitally amplified input amplitude reaches the value chosen via ‘target ampl.’. Sometimes it is necessary to press this button multiple times until the value saturates, so always check the signal in the tab ‘Input’.

(a) (b)

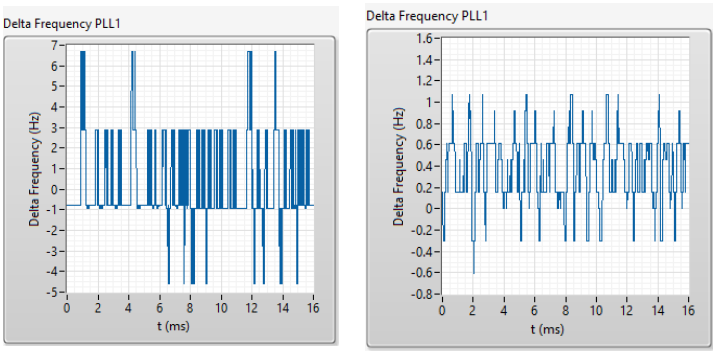

Figure 5: Time trace of the PLL frequency response to visualization the importance of the ‘Gain’ parameter (digital amplification).
(a) Value of gain too small.
(b) Value of gain correct.

b) Warnings

‘Gain too high’:

Digital amplification (parameter gain) is too high. Reduce the gain-parameter to prevent saturation effects. The auto-gain feature can help in this situation.

‘ADC-Overflow’:

Amplitude of the input signal is too high (independently of digital amplification). Please reduce the signal amplitude before feeding it into the FPGA.

c) PLL Settings

‘P gain’:

Value of the proportional gain of the PI-controller. See section 4 for further details.

‘I gain’:

Value of the integral gain of the PI-controller. See section 4 for further details.

‘base freq 1/2':

Central frequency of each PLL in Hz. The value should be equal to the intended modulation frequency. The maximum value is about 128 kHz.

‘delay’:

Compensation value of internal PLL delay due to computation time. Default value is 8.44 µs for this implementation.

d) PLL 1/2/3

General status information of each PLL.

‘f (Hz)’: averaged detected frequency deviation (see tab “frequency”)

‘phase (deg)’: averaged detected phase difference (see tab “phase”)

‘std(f)’: standard deviation of the detected frequency deviation (see tab ‘frequency’)

‘std(phase)’: standard deviation of the detected phase difference (see tab ‘phase’)

e) Live Scope

Settings for the plots in the upper region of the UI.

‘store ratio’:

Determines, in what iteration of the 500 kHz-Loop the data is stored in the FIFO. The parameter doesn’t have any effect on the PLL-behaviour, but on the plots in the UI.

Example: store ratio=4 means that in every fourth iteration data is stored in the FIFO. This will lead to a visible sampling rate of 500 kHz / 4 = 125 kHz.

The parameter can be used to change the maximum duration in the time traces or the frequency resolution in the spectra, since the FIFO can store up to 2048 data points.

Parameter-dependent plot-properties:

Time step: $\Delta t = \frac{\text{store ratio}}{500\ \text{kHz}}$

Signal duration $T = 2048 \cdot \frac{\text{store ratio}}{500\ \text{kHz}}$

Max. detectable frequency: $f_{\max} = \frac{1}{2\Delta t} = \frac{250\ \text{kHz}}{\text{store ratio}}$

Frequency resolution: $\Delta f = \frac{1}{T} = \frac{500\ \text{kHz}}{2048 \cdot \text{store ratio}}$

'update rate':

Refresh rate of the plots and status information in the UI.

Button 'Plot everything':

If deactivated: update the plot only, if the standard deviation of the detected frequency difference is above the value in the field 'min std(f)'. This can be useful, e. g. to only plot the settling behaviour of the PLL.

'min std(f)':

See button 'Plot everything'.

f) VI-Control buttons

'Reset':

Overall reset of the internal PLL state (phase, PI-controller, filter, …)

'Stop':

Stops the PC-Main VI

'Stop RT':

Stops the RT-Main VI.

g) Save Data

Save current PLL state (data of all currently plotted graphs). The file will be saved in the folder "data" in the PLLs project directory.

h) Filter Settings (only in version with 2 PLL channels, variable filtering)

Parameters of the PLL filters. See section 4 for further information. Adjustments will update in runtime.

4. Optimization methods

Please note: the 'optimal' parameters of filters and PI-controller impact each other, so the optimization might have to be done iteratively.

4.1 Optimization of the PI-Controller

Introduction:

The PI-Controller is used to adjust the frequency of the internal oscillator in such a way, that the phase difference between external signal and internal oscillator converges 0°. To do so, the phase difference is used to compute a frequency correction relative to the centre frequency of the PLL (e.g. 16 kHz). The calculation consists out of two components: a proportional part (phase deviation multiplied by a constant factor P), and an integral part (accumulated phase difference multiplied with a constant factor I). For more details, there are lots of information on PID-controller-optimization in literature.

The parameters P and I must be optimized in such a way, that the phase difference settles as fast as possible. Depending on the certain application, it might be better to optimize for a fast phase adjustment or to make the detected frequency as representative as possible. The optimization steps are in both cases very similar.

Setup:

Connect a function generator with the PLL-input. Generate an AC-signal with e.g. 16 kHz (value does not matter as long as it is in the range of ~10-70 kHz). The amplitude could be for example 1 Vpp. Apply a frequency-modulation to the signal, e.g. with a modulation frequency of 100 Hz and a frequency deviation of 100 Hz. The FM-type should be rectangular. Lock the PLL on the base frequency 16 kHz and open the Phase or Frequency tab.

Please note that you can pause the live scope through the bottom on the top right corner of the PC-Main VI. The settling time τ of the phase or frequency (depending on what you want to optimize for) is a measure for the reaction time of the PLL to frequency deviations of the input signal.

Optimization steps:

1) Initial state: P small (e. g. $P = 100$), $I = 0$
2) Look at the plots in the Phase or Frequency tab in PC-Main (the one you want to optimize) and increase the P-value, until there occurs some kind of oscillation in the PLL-response. Reduce the value again until there is no oscillation anymore.
3) Increase the I -parameter slowly (e. g. in steps of 0.1) and look at the PLL-response. Please note, that I is much more sensitive compared to P. If I is too small, there will always be a remaining constant phase difference, so even if you optimize the frequency response, you might also have to look on the phase response.
4) Repeat steps 2) and 3) until the settling time is as small as possible without oscillations.

There are lots of different methods for PID-optimization in literature. The steps above represent an intuitive and direct approach, while other methods rely less on testing but on semi-analytic considerations. However, the presented method enables a fast optimization without deeper knowledge of the topic.

Figure 6 illustrates the phase and frequency response of the PLL for several PI-combinations. The response time τ of the phase relaxation is also visible. Depending on the parameters, the PLL-response might include oscillating (b) or creeping (c) behaviour, both leading to a slower relaxation time. Ideally, there should be no overshooting of the quantity to optimize and a fast convergence to zero. The phase response should also not include a constant offset. Figure 6 (a) depicts the (almost) ideal phase response, for the given PLL. Please note, that higher filter cutoff-frequencies enable a faster possible relaxation time. It is also important to note, that the plotted phase response does NOT include the PLL delay of approximately 8.4 µs, which is always present due to the computation time.

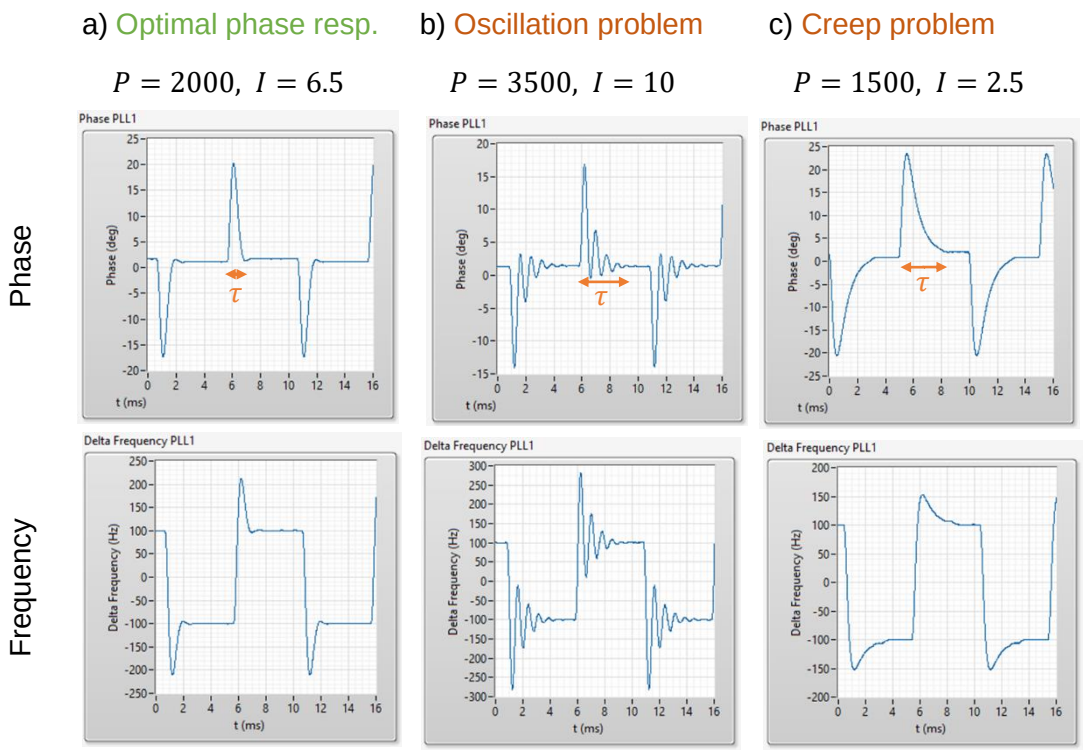

Figure 6: Phase and frequency response for a frequency-modulated input signal (rect. modulation with frequency $f_{\mathrm{m}} = 100$ Hz, frequency deviation $\Delta f_{\mathrm{m}} = 100$ Hz and central frequency $f_0 = 20$ kHz).

4.2 Optimization of the filter-parameters

Introduction:

The filter consists out of multiple fundamental filters in series. For filter optimization it is recommended to use the PLL-version with runtime-variable filter parameters, since the fixed-filter version needs to be recompiled after adjustment of filter parameters.

Setup:

Connect two function generator outputs (FG) via a $50\ \Omega$ splitter with each other and plug the output in the PLL-input. If you only have one modulation frequency at once, one function generator is sufficient.

Set the output signals of the FGs in such a way, that they represent the modulated signals in your application. Frequency modulation can be applied if necessary (leads to sidebands).

Example: FG1: $f_0 = 20$ kHz, $V_{PP} = 500$ mV; FG2: $f_0 = 32$ kHz , $V_{PP} = 500$ mV

Filter types:

Butterworth: low-pass filter with a 3-dB-cutoff frequency.

→used to eliminate sum and difference frequencies (> 0 Hz) in the mixer output and high-frequency noise.

Please note: a low cutoff-frequency leads to less noise and other distortions BUT also filters higher frequency components in the PLL-response, thus leads to a slower PLL-reaction!

Notch: band-stop filter with a central-frequency and a 3-dB-bandwidth.

→used to eliminate certain sum or difference frequencies of nearby modulation frequencies without slowing down the reaction time too much.

Example: modulation at 20 kHz, 32 kHz →difference at 12 kHz → small Butterworth frequency would be necessary →better: Notch filter at 12 kHz).

Please note: Notch filter might lead to additional higher harmonics which must also be considered.

To find an optimal filter combination for your application, please look at the PLL-response *spectra* in PC-Main and adjust the filters in such a way that the distortion fits your needs. To check the reaction time with your filter, please test again with a FM-signal with rectangular modulation as shown in the beginning of the section.

Important: changing the filter parameters might make it necessary to also configure the PI-values.

Save the new parameter settings

To save the new parameters, press right mouse button on the parameter → 'Data Operations' → 'Make current value default' (while the VI is NOT running).

5. Further specifications

Amplitude of trigger pulses: $U_t = 3.3$ V
→see figure 7 (a)

Delay of the PLL (in settled state): $\tau_{\text{delay}} = 8.4$ µs
→see figure 7 (a)
→plots and displayed values of the phase difference $\Delta\varphi$ do NOT include τ_{delay}
→Compensation of phase delay using the parameter 'delay' in PC-Main

Jitter of the trigger pulses: $\tau_{\text{jitter}} = 80$ ns
→see figure 7 (b)

Typ. relaxation time after frequency step: $\tau_{\text{relax},f} = 500$ µs (frequency response)
(exact value depends on parameters) $\tau_{\text{relax},\varphi} = 1000$ µs (phase response)
→see figure 6 (a)

Voltage interval of input signal: $[-10\text{ V}; +10\text{ V}]$, resolution 12 bit

Maximum detectable frequency: $f_{0,\max} \approx 128$ kHz

Maximum frequency deviation: $\Delta f_{\max} \approx 1$ kHz

(a) (b)

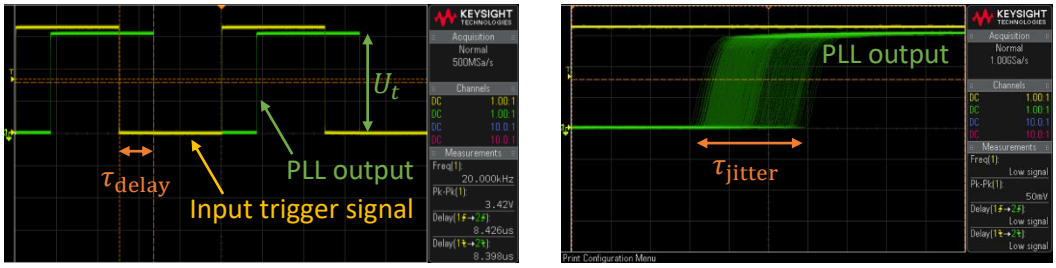

Figure 7:
Oscilloscope time trace of input trigger signal (trigger output of function generator) in yellow and PLL output in green.
(a) Visualization of PLL-delay τ_{delay} and PLL-trigger amplitude U_t.
(b) Visualization of PLL-trigger jitter τ_{jitter}.

6. Frequently Asked Questions

6.1) RT-Main cannot be executed

Is FPGA-Main compiled? (necessary when first use on a PC)

→ see section 2, step 4.

6.2) The PLL won't lock anymore, e. g. stuck at maximum frequency deviation

Reset the PLL by pressing the "Reset"-button in PC-Main.

6.3) Signals in the warning-section in PC-Main light up

Reduce the gain-parameter and/or the maximum input voltage.

→ see sections 3.3.1 and 3.3.2.

6.4) PLL response is very discretized (large steps relative to the signal)

Increase the gain-parameter in PC-Main in such a way, that the maximum amplitude (left plot in tab 'Input') is approximately 5-8 V.

→ see section 3.3.1

6.5) How can I change the filter properties in the fixed filter version?

Open the FPGA-Main block diagram and search the filter section. Click on the filter-VI and change the parameters. Repeat for all filters. Recompile the project (section 2 step 3).

For testing purposes, it might be beneficial to use the PLL variant with adjustable filter parameters in PC-Main.

6.6) After closing the LabVIEW-project, all parameter adjustments are gone. Can I save them?

While the 'PC Main' VI is NOT running:
Right mouse click on the parameter → 'Data Operations' → 'Make current value default'.

6.7) There are oscillations or noise in my PLL frequency response.

Look at the PLL response spectrum in PC-main. Are there any dominating distortion frequencies? These effects can be caused if multiple modulations frequencies are involved. Reconfigure the filter properties according to section 5.2 (smaller low-pass cutoff frequency, set the band-stop frequency at position of your distortion frequency). Optimize the PI-parameters according to section 5.1.

6.8) There is some constant phase shift in the PLL response.

Your Integral value of the PI-controller might be too small. Try to optimize the PI-parameters according to section 4.1.

The PLL has also some intrinsic delay due to the computation time, which is approximately $8.4\ \mu s$.

6.9) The detected frequency deviation does not match the expectation.

Does your expected frequency deviation change over time (e. g. frequency modulation)?

Detected frequency modulation > real frequency deviation:

Can be caused by overshooting. Your PI parameters are probably not optimized. Too big P or I values can cause overshooting and oscillating behaviour of the PLL response. Reconfigure your PI settings according to section 4.1.

Detected frequency deviation < real frequency deviation:

Your real frequency deviation might change too fast relative to the PLLs response time. Try to use less filtering (higher cut-off frequencies of the low-pass filters). Also check your PI-parameter configuration according to section 4.1. Too small values of P and I can cause creeping behaviour and thus a slower PLL response. If those aspects did not solve the problem, try to reduce the frequency modulation frequency.

For any further questions please contact our **customer support**:

lucasludwig97@gmail.com

(Response not guaranteed)