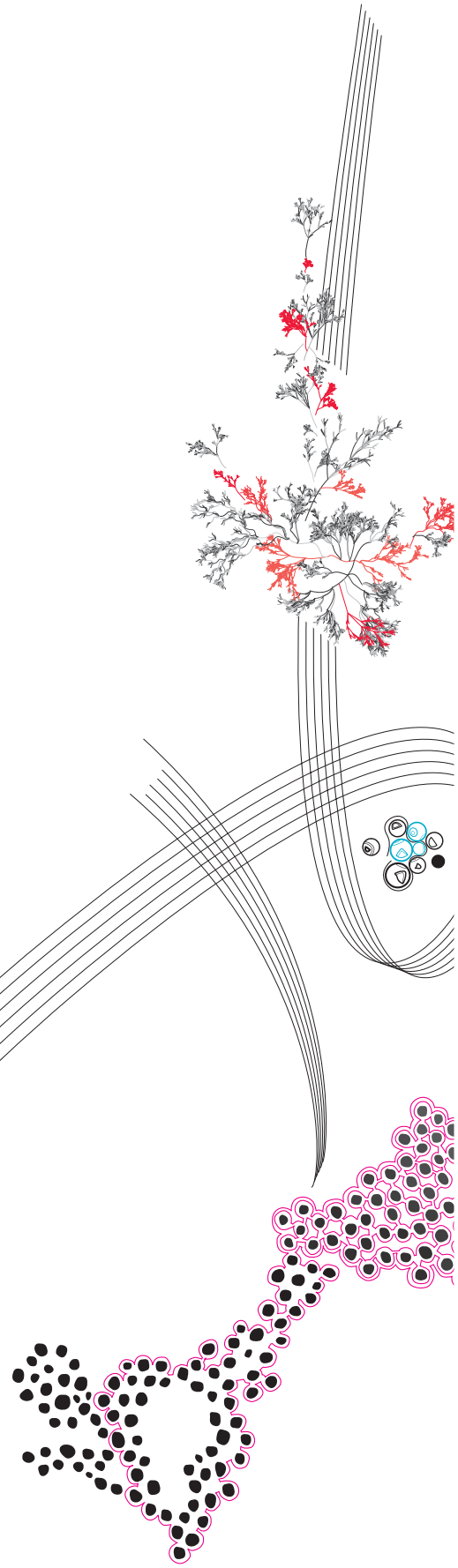

BSc Thesis Electrical Engineering

# Design of an FPGA based PLL system

Niels Wisselink

Supervisors:
Dr. R.A. Vogt-Ardatjew
A. Matthee

External Committee Member:
D. Alveringh

June, 2022

Department of Electrical Engineering
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

## Preface

First of all, I would like to thank my supervisor Alex for giving a big amount of freedom when designing this system and allowing me to flexibly work both from home and on-campus. This definitely helped to create a more relaxed work environment for myself. I would also like to thank Alex for being supportive throughout the project and always delivering positive feedback.
Furthermore, I would like to thank Bram for his cooperation when working together during the project. As a fellow student doing his bachelor thesis, it was nice to be able to work with a fellow bachelor student for part of the project.
Lastly, I would like to thank my family and my girlfriend for being both supportive and allowing me to work on the project from home, mostly uninterrupted. Especially my girlfriend, who always is always supportive and knows the struggles of doing a bachelor thesis herself.

# Design Report of a Fast-Acting Grid-Tied Phase-Locked Loop system, Implemented on an FPGA Using a Model-Based Design Approach

Niels Wisselink*

August, 2022

## Abstract

In this report a fast-acting grid-tied Phase-locked loop system (PLL), implemented on a field programmable gate array (FPGA), developed using a model-based approach in Simulink is discussed. The system is part of an inrush current mitigation project, which aims to mitigate inrush currents and other transients using a gallium nitride based inverter. A phase-locked loop is required for synchronization to the grid and error-calculation. This system was designed, simulations were performed and hardware tests were conducted. After which was concluded that the system was operating correctly. The PLL system was integrated with the error controller, after verification the system was confirmed to operate correctly.

*Email: n.wisselink@student.utwente.nl

# Contents

# 1 Introduction

With the accelerating adoption of switching power electronics converters (SPECs) and other non-linear loads, inrush current and transient events are becoming a major concern for low-inertia or weak microgrids [11]. Power grids, especially islanded micro-grids, may suffer from these increasing non-linear loads, as they typically produce significant short term pulsed currents. These currents can overload a micro-grid and cause it to go into protection mode or fail and possible damage components. An example of inrush current induced failure is shown in Fig. 1. This event resulted in over-current protection mode activation of an inverter with complete power outage.

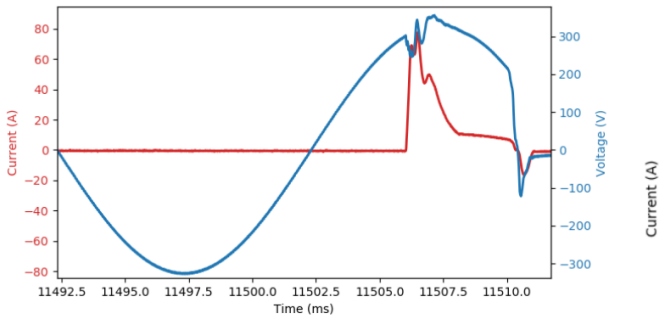


FIGURE 1: Example of an inrush-current event[11]

By adding an additional, small footprint and fast-reacting power source, the high frequency current components can be compensated for. Essentially a low energy, high power active filter which protects the main power source from disruptive or damaging loads. Monitoring of grid voltage is essential for this concept to operate. To obtain the error from the grid signal, a reference signal needs to be constructed, which is suitable for this error calculation. It thus needs to be both in-phase and in-frequency to the grid signal. A phase-locked loop (PLL) can achieve this.

A phase-locked loop is a control system which aims to relate the phase of a generated signal, to the phase of an input signal[12]. This relation could for instance be made to equalize their phases. Commonly, the output signal phase is controlled using a variable frequency oscillator. A phase comparator can then be used to establish the phase difference between the input and output signals. Fig. 2 shows a high-level generalised block diagram of a PLL. A common application of PLLs are signal synchronization. An example of this can be seen in Fig. 3.

PLLs are widely used in micro-grid applications, as usually these will contain energy sources, such as solar panels, which generate DC power. An inverter is utilized to convert the DC power to AC. A PLL can then be used to synchronize the generated AC voltage to the grid, such that the inverter is able to deliver power back to the grid.
This is one example of a PLL use case, as they prove useful for all kinds of synchronization tasks, such as FM demodulation or clock multiplication in microprocessors[12].

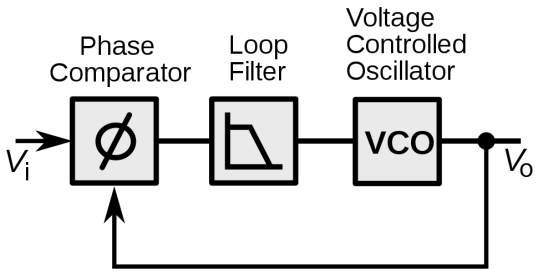


FIGURE 2: Block diagram of a generalized PLL[2]

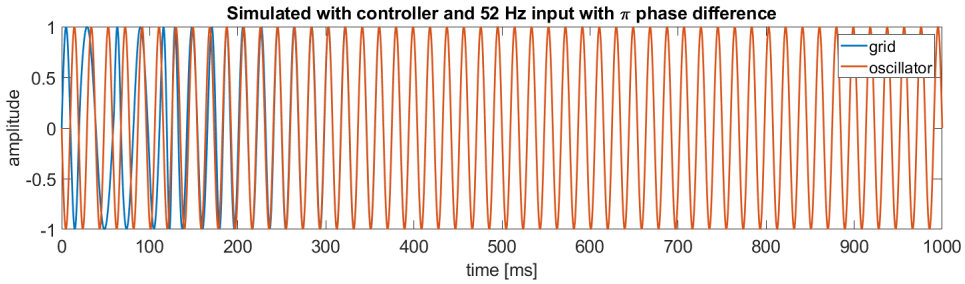


FIGURE 3: Example of a PLL synchronizing to another signal

The PLL will be needed for voltage-error calculations, as it will allow synchronization of the grid and reference signal. The need of a fast-reacting system suggests the use of a hardware implementation. To prototype a digital hardware implemented system, it will be chosen to use a field-programmable programmable gate array (FPGA).

The complete system consists of sub-modules which all will be integrated in a Simulink environment. The basic devisions of the sub-modules are: (1) grid voltage sensing and PLL, (2) control system and error system generation and (3) SVPWM engine which provides switching commands to the output hardware in order to control the power output. The PLL and supporting sensing hardware is required to monitor the grid voltage at a high sample rate, react as quickly as possible to provide the control system sub-module with information. The project design utilises Simulink with a Vivado VHDL environment for sysnthesis of FPGA devices. An FPGA is chosen as the appropriate hardware due to it's rapid development and fast processing capabilities.

# 2 Methodology

To accomplish the goal of creating a grid-following PLL system on an FPGA, it is necessary to understand the general concept of what a PLL does. A Phase-locked loop is a controlled loop, which uses phase difference to control an input oscillator in order to manipulate this phase difference as needed. Fig. 2 shows a generalized block

diagram of a PLL. In it is clearly visible how the phases of theoscillator signal and reference signal $V_i$ are compared, generate a control signal for the Voltage-controlled oscillator (VCO).

The aim of this system is to do exactly what is described above, using a high sample-rate input, and use the generated high sample-rate synchronized reference signal as an input for a controller which measures grid voltage error. The following subsections will elaborate on how the different subsystems of the PLL are designed.

## 2.1 System Overview

Fig. 4 shows a high-level block diagram of the system. It has a small part in the analog domain, which is just the input circuit. Most of the processing is performed by the FPGA.

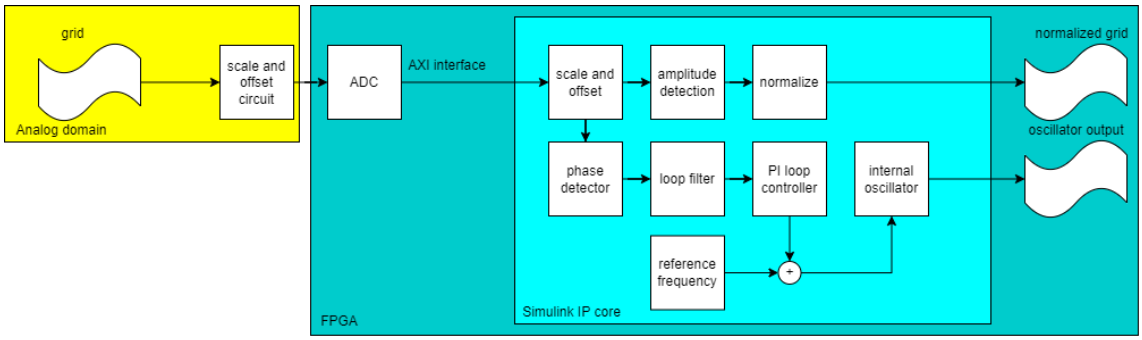


FIGURE 4: High-level block diagram of the system

## 2.2 Scaling and Offsetting of Input

The FPGA board used in this project is the Digilent CMOD A7-35T[3]. The FPGA on this board contains an ADC which has an input range of 0-1V. The ADC pins are in turn connected to some on-board circuitry that increases this range to 0-3.3V, which can be seen in Fig. 5. The grid voltage (in the Netherlands) is an AC signal with 230 Vrms. It is thus necessary to both scale down and offset the grid voltage to convert it down from $-230\sqrt{2} \leq x \leq 230\sqrt{2}$ V to $0 \leq x \leq 3.3$ V.

A circuit with an operational amplifier as its basis can be used to both scale down and offset the voltage. As the circuit will mostly be used for testing purposes and needs not to be robust, the grid voltage can simply be scaled down using a resistive voltage divider. An external DC signal of 1.65 V, potentially from the FPGA, can then be used to add an offset such that the AC signal will have its centre in the middle of the ADC capture range. Fig. 6 shows a circuit which achieves this result, as can be seen from the simulation in Fig. 7, with an input voltage of $320\sqrt{2}$V. The choice for an op-amp based buffering circuit was made to make sure the ADC input would not be current limited, regardless of current draw.

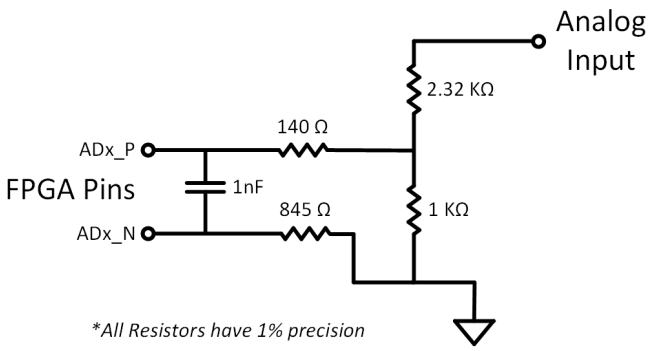


FIGURE 5: onboard ADC circuitry[3]

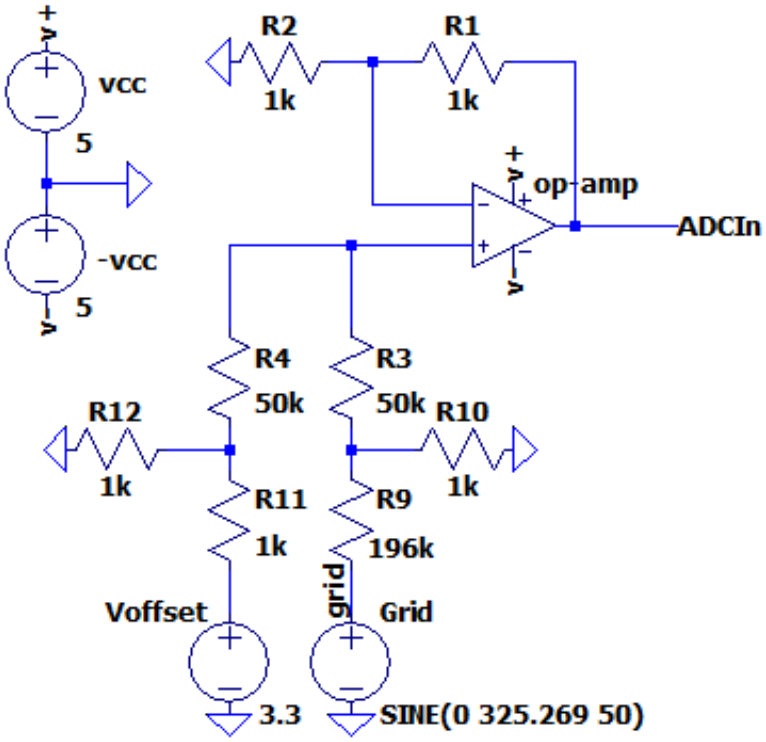


FIGURE 6: Circuit diagram of input circuit

After capture through the ADC, the modified grid signal needs to be restored to its original scale and the offset needs to be removed. This is achievable to implement inside of simulink using addition and multiplication functions.

## 2.3 ADC and Internal FPGA Dataflow Using AXI

To capture samples from the grid, the built-in ADC functionality (called XADC in Vivado) of the CMOD A7 can be used. It supports 2 board inputs for analog capture. both of these channels can capture 12 bit samples at a maximum rate of 1 MSPS, in the range of 0-3.3 V. The Artix-7 FPGA on this board actually has an input range of 0-1 V, but this is altered with some board circuitry. Fig. 5 shows this

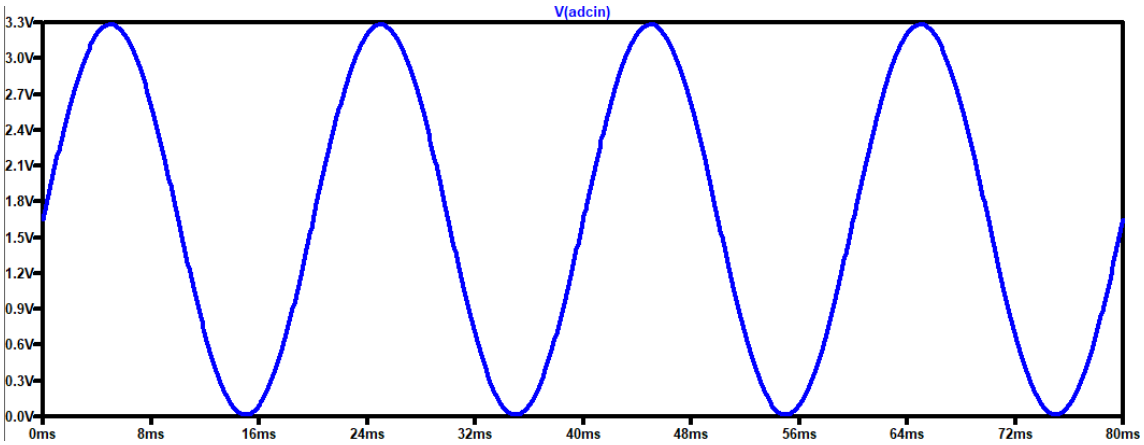


FIGURE 7: Simulation of input circuit

input board circuitry.

The ADC can be configured with the XADC-wizard block inside the block designer of Xilinx Vivado. For the Simulink system to obtain ADC samples from the XADC block, the system needs a method of communication, supported by both the XADC block and Simulink. For this, the AXI4 interface will be chosen, as this is straightforward to implement, in part due to the documentation MathWorks has available on it for Simulink[10].

AXI4 is a high speed communication bus protocol used for on-chip communication. The interface is both high-bandwidth and low latency, without being complicated to implement[1]. Most of the specific details of AXI4 are not relevant to this project, but important ones are that AXI4 uses an address and data-bus to let a master communicate with slaves. The master can either request to execute a read or write operation on a slave. The slave is addressed through the address bus and each slave has an address range attached to it.

To read the ADC samples from Simulink, the model will need to have an AXI4-master read interface, as the XADC block only contains a slave interface[14] and the system only needs to read from that. This interface can modelled in Simulink, after which it can be synthesized into a valid IP core block[10].

The implementation of an AXI4-master read interface in simulink should be straightforward and has enough documentation[10]. The interface should have three connections, a data line, an input bus and an output bus. The input bus should at minimum contain a "ready" port, which indicates if the bus is ready for data transfer; and a "valid" port, to indicate whether the data on the bus is valid. The output bus should at minimum contain an "address" port; a "length" port, to indicate data length; and a "valid" port, to indicate when the read request is valid.

## 2.4 Amplitude Detection

Grid voltage can have variations in amplitude, which needs to be taken into account. An easy way to measure the maximum amplitude of the grid signal is to measure at peak amplitude. It is hard to detect when the amplitude is maximum from just the sinusoidal wave, so the derivative can be used, as this is always $\frac{\pi}{2}$ radians out of phase from the original as shown in eq. equation (1), which means that it crosses through zero when the signal is at peak amplitude.

$$\frac{d}{dt}sin(\omega t) = \omega sin(\omega t + \frac{\pi}{2}) = \omega cos(\omega t) \tag{1}$$

This method should be quite straightforward to implement, which is the main reason why it was chosen. The only trouble is the implementation of the derivative. Due to the use of an FPGA, the derivative can only be approximated using discrete methods. a common approximation is the difference, which subtracts the last sample from the current one.

## 2.5 Phase Difference Detection

It is necessary for the internal oscillator to synchronize with the measured grid signal in both phase and frequency for error calculation. This synchronization is traditionally done by looking at the phase difference of both signals and adjusting the frequency of the controllable signal accordingly. Both frequency and phase difference information can be retrieved by looking at the observed phase difference of two signals. This can be shown in equation (2), where $(\omega_\Delta t + \phi_\Delta)$ is what can be observed and what needs to be compensated for.

$$f_1(t) = sin(\omega_1 t + \phi_1) \tag{2a}$$
$$f_2(t) = sin(\omega_2 t + \phi_2) \tag{2b}$$
$$= sin(\omega_1 t + \phi_1 + (\omega_\Delta t + \phi_\Delta)) \tag{2c}$$

A simple method for detecting frequency and phase differences is to use a phase frequency detector (PFD). there typically are two types and both types can be constructed with a small amount of logic circuitry[6].This is ideal for implementation on an FPGA.

The first of the two types of PFD produces a pulse between the falling edge of both signals[6]. Fig. 8 demonstrates this quite well. The phase difference is either early or late depending on which falling edge comes first. The second type of PFD stays produces a pulse when an edge of the first signal comes and stops this pulse when an edge of the second signal comes.

As the phase detector will be used for sinusoids (the example shown a square wave) and will be implemented on an FPGA, the PFD concept can be modified a little bit to be a little more refined. Instead of producing a single pulse with a variable length, the detector will count the cycles between both positive and negative zero-crossings. The FPGA will start a counter when a zero-crossing happens on the first signal and sample the counter when the same zero-crossing happens on the other

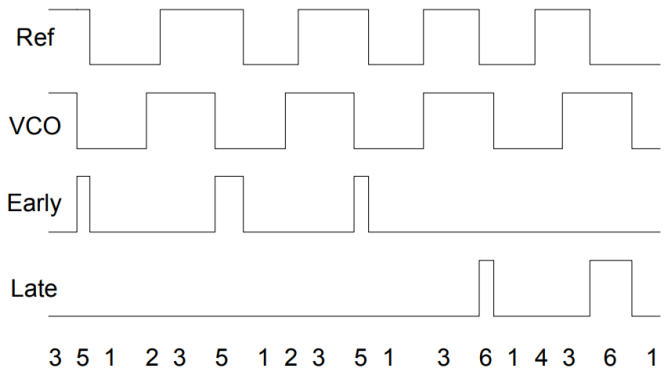


FIGURE 8: The first type of PFD[6]

signal. This can produce a time related phase difference. Since there are two different zero-crossings in a signal, the detection can be done twice in one signal period.

Theoretically, if the signal noise is small enough, differentiation can be used to add a $\frac{\pi}{2}$ rad phase shift, so two more detections can be done in a signal period. This will unfortunatly not be possible in this project, due to the limited resolution of the ADC. it can represent 12 bits in a range of 0-1V. so the smallest observable value is $2^{-12}$. To fit in the range of the ADC the input signal will have expression equation (3) with its derivative also shown. The amount of change of this signal is $50\pi$. using these facts it can be calculated that the smallest amount of time it takes for a single bit to change is $1.55 \cdot 10^{-6}$ s or a maximum frequency of $6.44 \cdot 10^{5}$ Hz as shown in equation (4). This means that it almost always will take more than one sample for a value to change, so either the change is 0, or the change is a small value.

$$f(t) = 0.5sin(100\pi t + \phi) + 0.5 \tag{3a}$$
$$f'(t) = 50\pi sin(100\pi t + \phi) \tag{3b}$$

$$\frac{2^{-12}}{50\pi} = 1.55 \cdot 10^{-6} \tag{4}$$

The only issue now is that this time related value is always positive, which makes it unusable for control purposes. The solution to this is to count the amount of cycles for a signal period, and subtract this from the measured phase difference if it exceeds half of the counted period. The phase detector can now make a distinction between lagging and leading signals.

The two phase detectors need to be combined into one signal again. There are two methods for this: by using an average of the two measurements, or by continuously switching to the most recent measurement. If there is a difference between falling and rising edge measurements, the averaging method will smooth them out, but the output might differ a bit from the actual phase difference. The switching method might be noisier, as the averaging also acts as a filter, but always has the

most recent measurement.

The main reason why this method will be chosen is that a PFD derived system is very simple at its core. Actual PFDs can be constructed using only a couple of logic gates, which means there is almost no resource usage on an FPGA. Some counting and subtraction needs done to find an actual phase difference, but this should not complicate the system too much.

## 2.6 Internal Oscillator

The PLL needs to generate a reference sinusoidal signal. This signal will need to have a controllable frequency, so it can catch up with the grid signal in phase and frequency. The requirement of this is that a change in frequency should not lead to a jump in phase, but rather only to a higher frequency oscillation. The oscillator should not be modelled as $sin(\omega t)$, but rather it should be modelled as equation (5), with $\Delta x$ being the variable element. equation (5b) is thus the input to the sine function and can be seen as a counter with variable step size, or even a discrete integrator.

$$f(n) = sin(x(n)) \tag{5a}$$
$$x(n) = x(n-1) + \Delta x \tag{5b}$$

But FPGAs are not good at traditional trigonometric calculations, as this typically uses a taylor series expansion of these functions. Another way of handling these trigonometric functions is by using a CORDIC algorithm that is specialized for implementation in hardware. CORDIC is suited for FPGAs as it requires no multiplication, rather just bit shift and additional operations[13]. A method of implementation is not relevant to this project, as an implementation of CORDIC is already built into Simulink and can be synthesized into HDL code.
A restriction of the CORDIC algorithm is that the input has a limited range of $-2\pi \leq x \leq 2\pi$, which will be taken into account by limiting the range of the variable-step counter from 0 to $2\pi$.

The use of a CORDIC algorithm was obvious. The other solution would have been to use a look-up table, which either limits the precision, or will need to use a great amount of memory. CORDIC does add the downside of more delay based on the requested precision, as it needs to iterate to get more precision. But this is not an issue in this system, as this introduced delay can be compensated for by the controller.

## 2.7 Loop Filter and Controller

The oscillator as described before will likely run at 1 MHz sampling frequency, as this is the limit of the FPGA ADC[3]. The phase detector however, produces a value twice (or four times using a differentiator if possible) every signal period, so for a 50 Hz signal this means that it will run at 100 Hz. Because of this, the oscillator

feedback will have sudden jumps in phase difference. To smooth this out somewhat, a low-pass filter can be used to reduce these jumps. As the filter should not add much delay to the signal path, it should not be of low order, as a higher order will add more delay. The simplest low-pass filter that can be constructed is the RC filter. A discrete transfer function of this filter can be shown in equation (6) using the Z-transform, with $\omega_c$ being the radial cut-off frequency and $Ts$ being the sampling period of the system.

$$H(z) = \frac{A}{z + (1 - A)} \tag{6a}$$

$$A = \omega_c Ts \tag{6b}$$

As stated in equation (2), the grid signal might not only have a difference in phase, but also in frequency, which can just be seen as time dependent phase. The original feedback signal will thus be unusable for control purposes, as it is a measurement of phase difference, not frequency difference directly. The frequency component in this phase measurement can be retrieved by integrating it. The system will thus need at least proportional and integral control, to account for both phase and frequency differences. The controller will sit in the feedback loop, as the reference input is non-zero (50 Hz), which will otherwise get effected by the controller.

### 2.8 Integration with Inverter Project

The PLL system in this project is part of a bigger project with a gallium nitride based inverter. Both the measured grid signal and the generated reference signal will serve as an input to an error control block that tries to compensate for potential voltage sags. These two signals will be the main output of this system because of that.

## 3 Implementation

The following subsections go into detail on how the different subsystems are implemented and how they are different from the methods proposed in the previous sections. Fig. 9 shows an overview of the system block diagram in Simulink.

A remark: Simulink does not support the use of floating point numbers when generating HDL code for an FPGA. All signals that use a decimal point notation thus need to use a fixed-point type for this. This can limit the range of certain signals somewhat and sometimes poses a general design challenge.

### 3.1 Tools Used

To get an idea of the workflow that will be used, some details are given on the hardware and software used for the project.

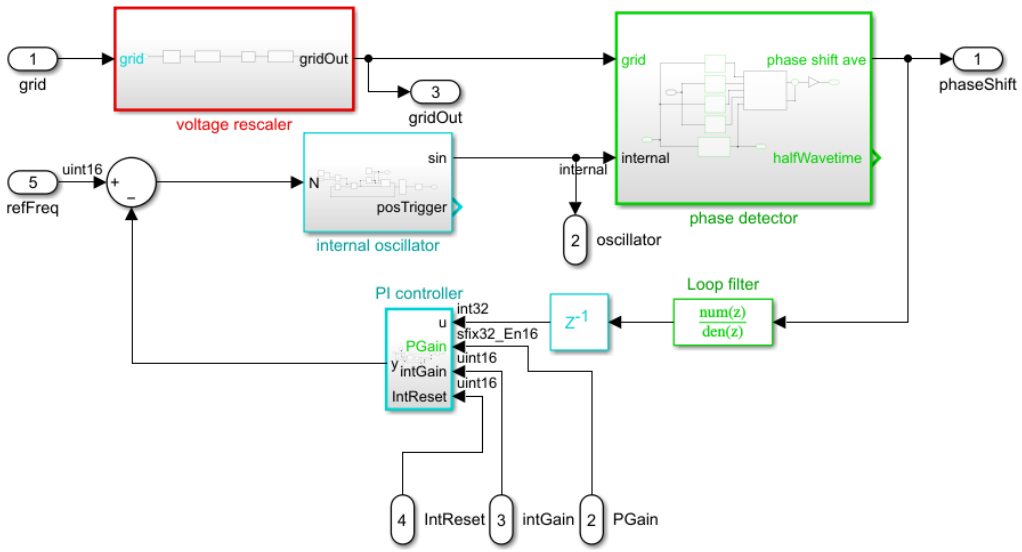


FIGURE 9: System block diagram as implemented in Simulink

### 3.1.1 Hardware

The Digilent CMOD A7-35T is used as the FPGA board in this project. It is based on the Xilinx Artix 7 platform and has built-in 12 bits 1 MSPS ADC support[3]. This FPGA is easily breadboardable, which simplifies prototyping somewhat. For the input circuitry the RC4558p dual op-amp is used to act as a signal buffer[4]. To provide power to the negative rail of this op-amp, the TL7660 is used[5]. These chips were not chosen with any specific reason other than their easy availability at the time. To generate and observe signals, a Picoscope 4824a is used. It is portable USB oscilloscope and function generator. This Picoscope was primarily used because it was available in the lab.

### 3.1.2 Software

Mostly Mathworks Simulink and Matlab will be used to design and simulate using HDL coder[8]. HDL verifier will be used to communicate with the deployed system using an AXI interfacing over JTAG[9]. Simulink was chosen, because it allows for a model-based design approach, which in turn enables fast turnaround of an entire system without the need for coding by hand. Furthermore, it allows for easy integration into the already existing workflow of the bigger inverter project. After HDL code is generated with HDL coder, Xilinx Vivado is used to deploy the system to the FPGA, which is a design suite for Xilinx FPGAs. the Picoscope software is then used in combination with the Picoscope to observe and generate signals from and to the FPGA. Finally LTspice is used for any analog circuit design.

## 3.2 Scaling and Offsetting of Input

The implemented input circuit was slightly altered to account for the use of low-voltage test signals, so tests of the system become easier to perform. Fig. 10 shows

this modified input circuit, where the voltage divider at the input was changed slightly to account for an input signal with a peak voltage of 2 V.

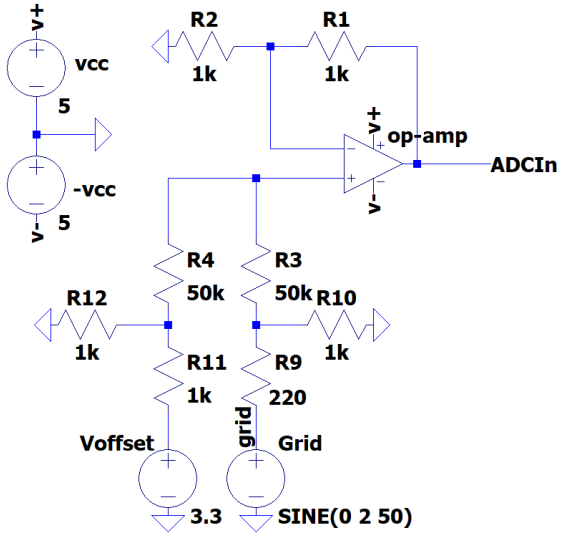


FIGURE 10: Modified version of the input circuit

The RC4558p dual op-amp was not able to amplify the input signal using only a positive rail, even though the input signal is purely a positive signal. As stated before in the methodology the TL7660 was used to generate the negative voltage rail. 5 V from the USB input was used to drive both the positive and negative rail of the op-amp, since the input signal is well within range of this voltage.

The signal measured by the ADC is now purely positive and needs to be shifted back to also have a negative halve. This is done by the subsystem block shown in Fig. 11. The FPGA sends data in chunks of 32 bits, so the signal coming in from the ADC is 32 bits wide. The ADC measurement lies on the 16 least significant bits and the first step of this subsystem is to strip the 4 least significant bits, as these contain mostly noise. The ADC is rated to be 12 bits in resolution[3], which explains this stripping of bits. The next step is to convert the datatype to include a sign bit so $2^{-11}$ can be subtracted to recover a pure AC signal again. Finally a type conversion is done again, this time without keeping the real world value of the signal, to reduce the width to 12 bits and normalize the signal to have a maximum amplitude of 1.

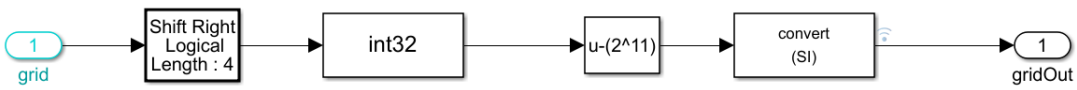


FIGURE 11: Input rescaler subsystem

## 3.3 ADC and Internal FPGA Dataflow Using AXI

The ADC of the FPGA was configured to have one channel open and to run at the maximum sample rate of 1 MSPS. In order for this sample rate to be achieved, the FPGA clock had to be raised to 104 MHz from 100 MHz.
For the PLL system to communicate with the ADC, an AXI master read port needed to be introduced. An implementation was created based on [10]. Fig. 12 shows the implementation, where the 'systemIn' path is connected to the grid input of Fig. 9. The ADCCounter has a limit of 103, which when hit, sets the count_hit signal high and a read request is sent to the ADC if the bus is ready for traffic. The rd_dvalid signal is set high when there is valid data on the gridIn line, which is then sampled by the sample and hold block to avoid possibly reading invalid data. the RT block transitions the operating frequency from 104 MHz to 1 MHz.

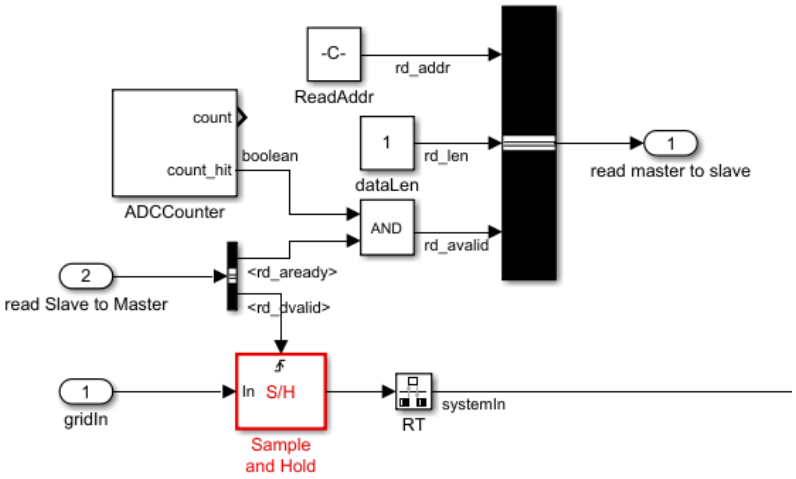


FIGURE 12: AXI master read interface

## 3.4 Amplitude Detection

An initial implementation of the amplitude detector was made, but it was eventually decided that the amplitude data was not crucial for the total inverter system to work. The initial implementation can be shown in Fig. 13, which works in the way described in the previous section. As discussed earlier, the differentiator can not be used at the sample rate of 1 MHz, so the amplitude detector was designed to work at about 50 KHz, where the differentiator was simulated to work correctly.

## 3.5 Phase Difference Detection

The chosen implementation of the phase detector only has 2 detections per signal period. This was done as differentiation is not possible at a sampling rate of 1 MHz.

Fig. 14 shows the top level of the phase detector. It has 2 inputs for both the internal oscillator and the grid signal. These signals are passed through negative and positive edge zero-crossing detectors. The big subsystem block is responsible

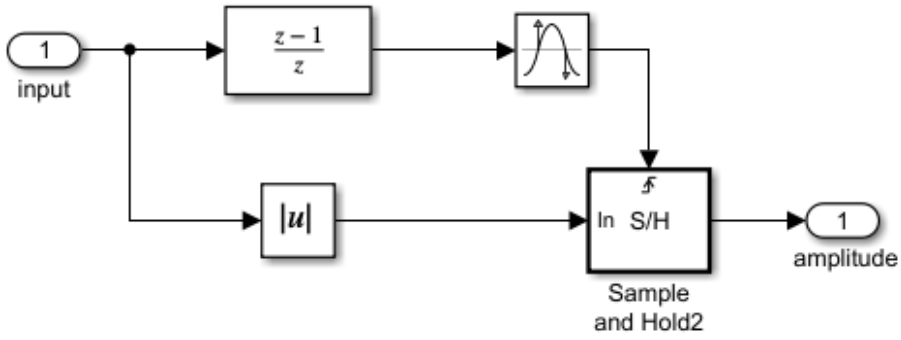


FIGURE 13: Initial amplitude detector

for the phase difference calculations. The 2 outputs of this are averaged. The bottom subsystem block is responsible for calculating the half wave time, needed for offsetting the counted phase value to a negative number.

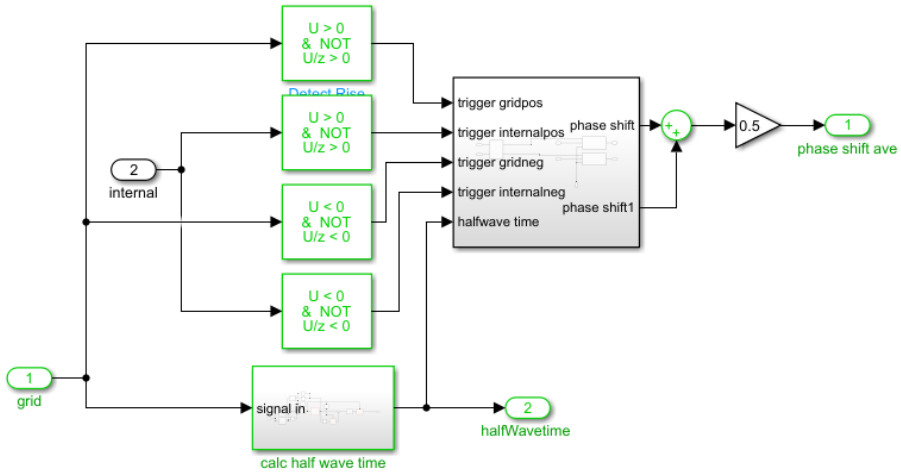


FIGURE 14: Top level of phase detector

Fig. 15 shows the second level of the phase detector. This contains three subsystem blocks. The left most block processes the triggers for the internal oscillator. The other 2 blocks both have identical functionality and serve to calculate the phase difference between the signals. the only difference is that one has falling-edge zero-crossing detections as an input, while the other has rising-edge zero-crossing detections.

Fig. 16 Shows the trigger processor from Fig. 15. This system was implemented after finding a bug in the oscillator that caused both zero-crossing detectors to trigger in one zero-crossing. Fig. 17 shows this glitch in the oscillator. It clearly shows how when the signal crosses zero, it crosses zero twice more. the trigger processor thus makes sure these extra triggers are not used in the phase difference calculation. It does this by switching both outputs to a constant low for 10 cycles after the first trigger was passed.

Fig. 18 shows one of the phase difference calculator blocks. This block counts the amount of cycles between the initial grid zero-crossing trigger and the internal

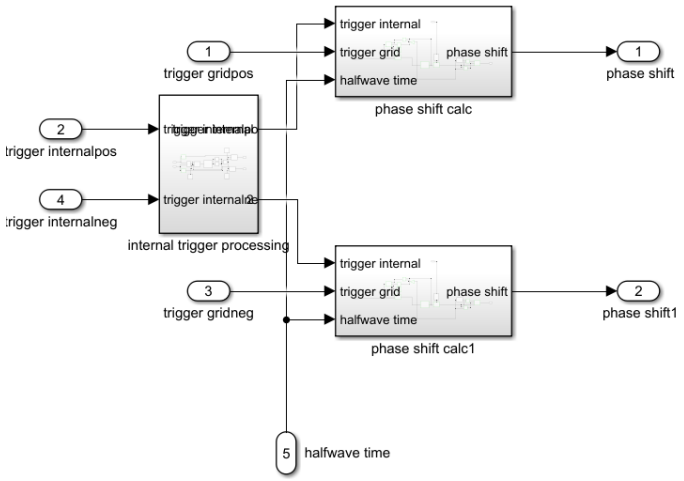


FIGURE 15: second level of phase detector

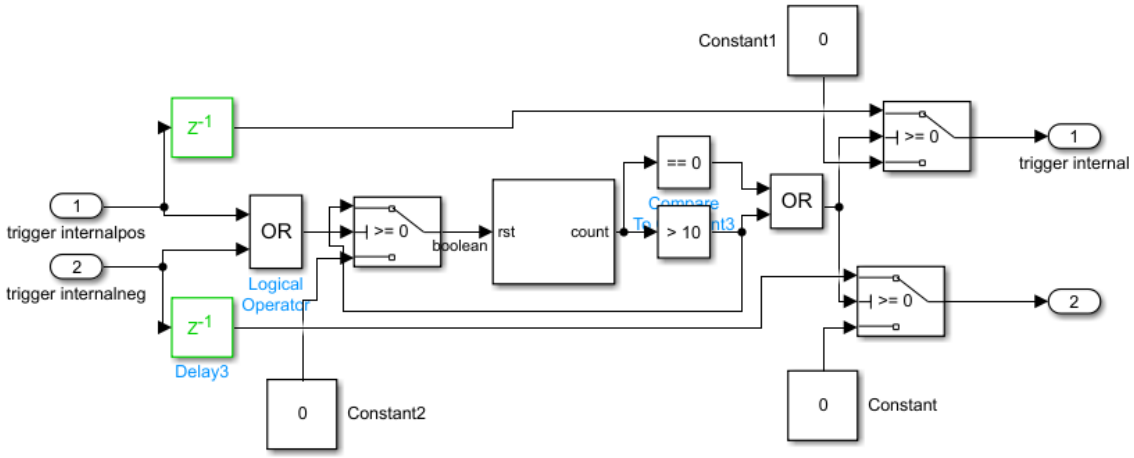


FIGURE 16: trigger processor of phase detector

oscillator trigger. The left side of the block has some trigger processing for the grid trigger signal. It switches this signal to low a certain amount of time, to make sure no other triggers are passed that could reset the counter.

If an initial zero-crossing is detected from the grid signal, the counter is reset. The counter value is then sampled when the trigger from the internal oscillator comes. The blocks on the right subtract the length of one signal period if the detected phase difference is more than half a signal period.

To figure out what the length of a signal period is, the half wave time detector shown in Fig. 19 is used. It uses both zero-crossings of the grid signal to determine the length of half a period. This is done by sampling, then resetting a counter that is used to count the cycles between each zero-crossing. Again, on the left side, there is some trigger processing done to make sure no premature triggers pass through. Furthermore the left side has 2 sample and hold blocks. This is so that the average of the last two measurements can be used, for noise reduction.

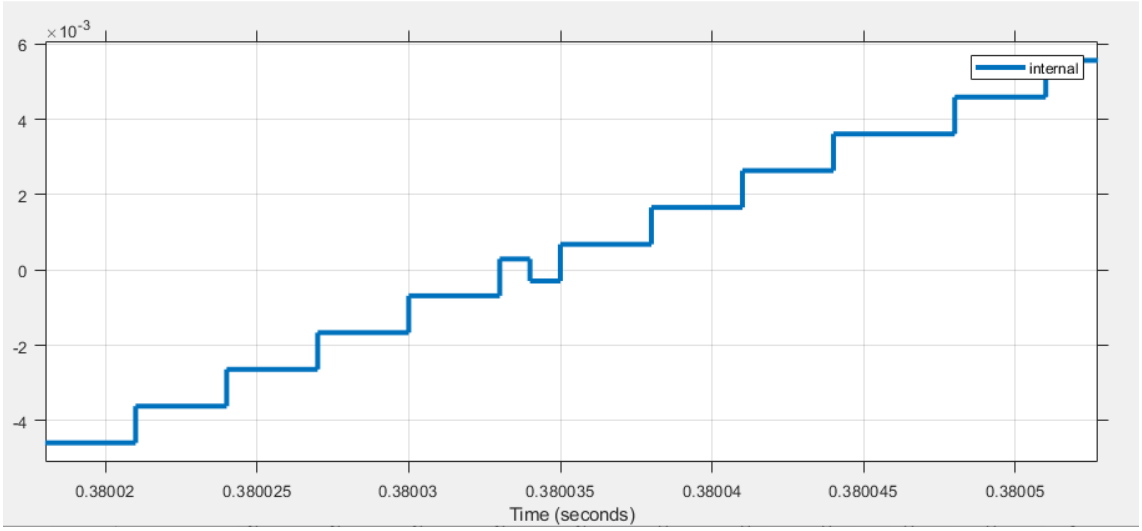


FIGURE 17: Glitch in the oscillator

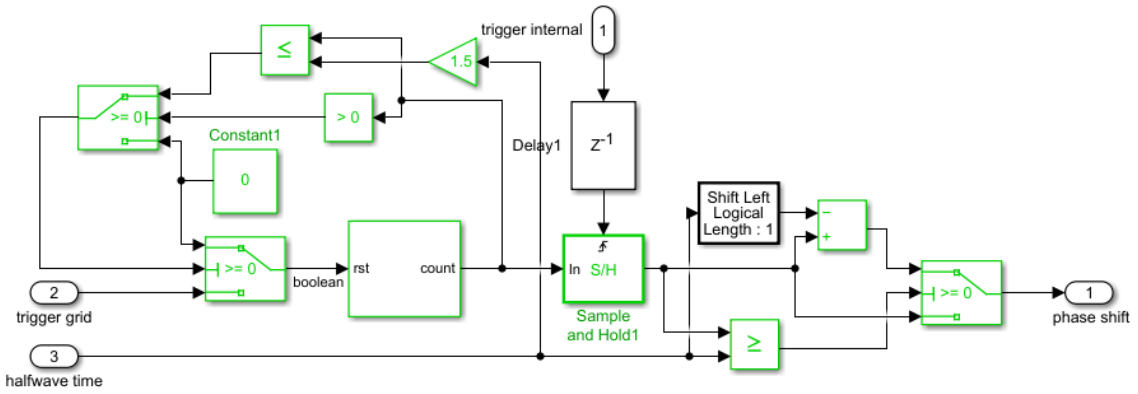


FIGURE 18: Phase difference calculator

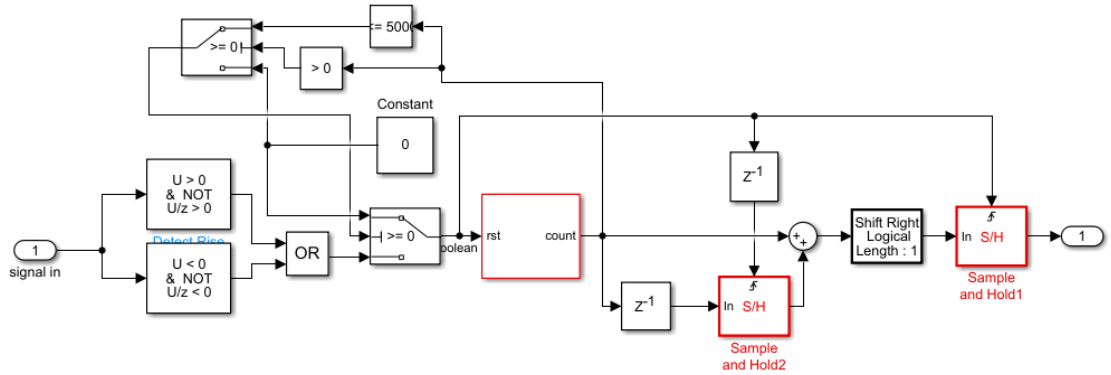


FIGURE 19: Half wave time detector

## 3.6 Internal Oscillator

The oscillator as implemented can be shown in Fig. 20. The input N of this subsystem is the amount of cycles for one oscillator period. Equation (7) shows how N can be calculated using the oscillator frequency $f_o$ and the sampling frequency $f_s$.

$$N = \frac{f_s}{f_o} = \frac{10^6}{50} = 2 \cdot 10^4 \tag{7}$$

$2\pi$ is divided by divided by N to get the step-size per cycle of the oscillator counter. This custom counter implementation consists of a memory element that works according to equation (5). As the CORDIC implementation of the sine function only has a range of $-2\pi \geq x \geq 2\pi$, the counter is reset to zero before it can exceed $2\pi$. The counter value is then passed through the sine block, which is configured for 16 bits of precision. This causes some delay on the output, but the control loop in the PLL compensates for it.

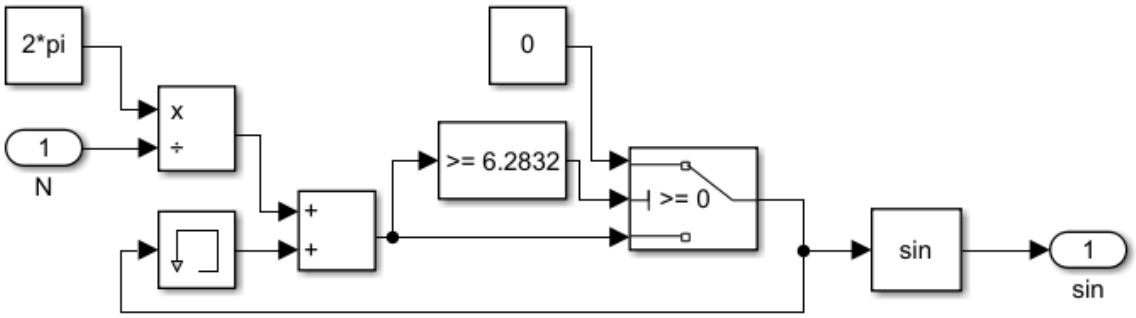


FIGURE 20: Oscillator implementation

## 3.7 Loop Filter and Controller

For the loop filter, the cut-off frequency was chosen to be 100 Hz, as the phase detector runs at this rate, for a grid frequency of 50 Hz. The final loop controller implemented is a PI controller. Fig. 21 shows the block diagram of this PI controller. Input "u" is the filtered phase detector input of the system. this signal splits into the proportional and integral controller. the proportional controller can be found by following u horizontally and consists just of a product block, with PGain being an externally configurable parameter, which can be configured through the AXI4 slave interface.

The integral controller is somewhat more complicated. The discrete-time integrator of Simulink could not be used due to algebraic loop errors, so a custom integrator was implemented. this integrator is highly similar to the oscillator counter shown in Fig. 20. The input of the oscillator is multiplied by $\frac{1}{f_s}$ with $f_s$ being the system sampling frequency of 1 MHz. This value is then added to the previous output of the memory block to create the new value. intReset indicates the absolute reset value of the integrator, where it resets back to 0. This is done to prevent the integrator from "wandering off too far" from zero and causing potentially long recovery times. The output of the integrator is then multiplied with intGain to produce the control output. both intGain and intReset are tunable parameters, again usin the AXI slave interface. The proportional and integral components are then added back together and limited to make sure the oscillator will either never oscillate too fast or stop oscillating all together.

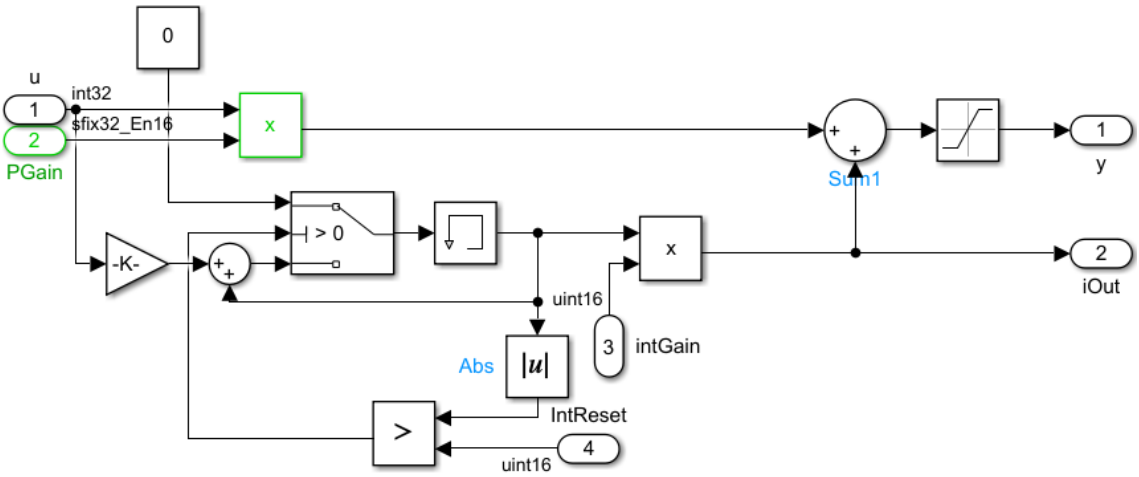


FIGURE 21: implemented PI controller

The controller was first tuned using simulation. Once stable values for P and I were found they were tested on the FPGA. Once the controller values were verified to work on the FPGA, the system was fine-tuned. Final tuning values are: proportional gain of 5 and integral gain of 20, with an integral limit of 400.

## 3.8 Integration with Inverter Project

After verifying that the PLL worked on its own, the system needed to be integrated into the bigger inverter project. The outputs of this system serve as an input for an error controller. The only modification that was needed to get the systems connected was data type conversion. the inputs of the error controller are a signed fixed-point type with a word length of 18 bits and contained 17 fractional bits, while the oscillator used a word length of 32 bits and the grid 12 bits.

# 4 Results and Discussion

To demonstrate the system, some test were conducted on both the implemented hardware and the simulated model. The simulation and the hardware implementation will be compared and discussed in this section.

## 4.1 Simulation

To get an initial idea of the system validity before it is implemented on the FPGA, simulations are run to test the different major components of the system. These components include the oscillator, the phase detector and the whole PLL with control enabled.

### 4.1.1 Oscillator

The oscillator was tested with three different signal frequencies to validate its behaviour. The frequencies chosen for this test were 48, 50, and 52 Hz. Apart from 50

Hz, the other 2 are considered extreme cases, which the actual electricity grid never reaches[7]. Fig. 22 shows all three cases in one plot. It clearly shows how the 50 Hz signal is able to finish two periods in the time-span of 40 ms, while the 48 Hz signal does not finish two periods and the 52 Hz signal leads over the other two. Fig. 23 shows the phases of these three signals, which gives a different visualization of their frequency differences.

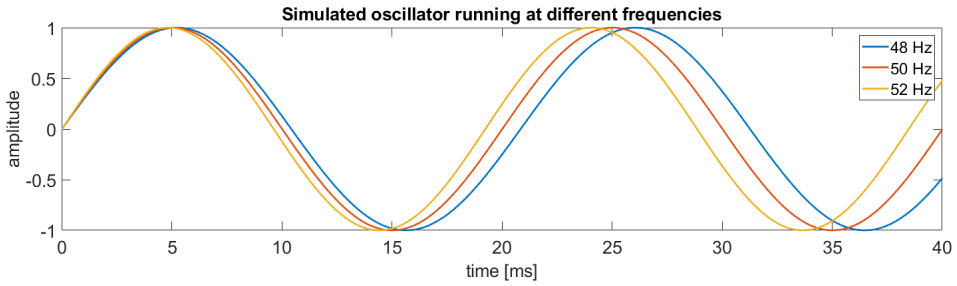


FIGURE 22: Simulated oscillator ran with different frequencies

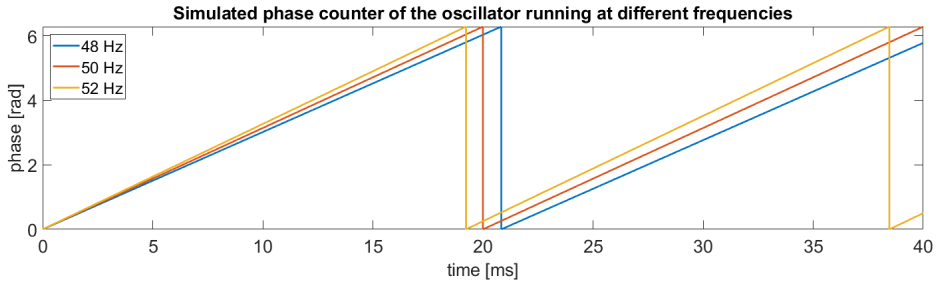


FIGURE 23: Simulated oscillator phase counter ran with different frequencies

### 4.1.2 Phase Detection

To test the simulated phase detector, different cases must be considered.First of all, the detector should be able to detect no phase difference if the actual phase difference is zero. Constant phase differences should also remain constant. And finally, a frequency difference should result in a linear detected phase wrapping to negative values if this phase difference exceeds $\pi$ rad.
The first case of constant phase difference can be shown in the top graph of Fig. 24. This case was tested with phase differences of 0, $\frac{\pi}{2}$ and $\pi$ rad all with the oscillator running at 50 Hz. At 50 Hz, a phase difference of $\frac{\pi}{2}$ rad corresponds to 5000 samples and $\pi$ rad to 10000 samples, which wraps back to become -10000 samples, for control purposes.
The bottom graph of Fig. 24, shows the phase detector when exposed to different grid frequencies, and a reference oscillator frequency of 50 Hz. Again, the same frequencies of 48 Hz, 50 Hz, and 52 Hz were chosen. The plot clearly shows how the 52 Hz signal leads, as the phase difference has an increasing slope, and the 48 Hz signal lags, shown by the by decreasing slope. Their phase difference are both the same, which can be seen by the fact that they cross zero at the same points.

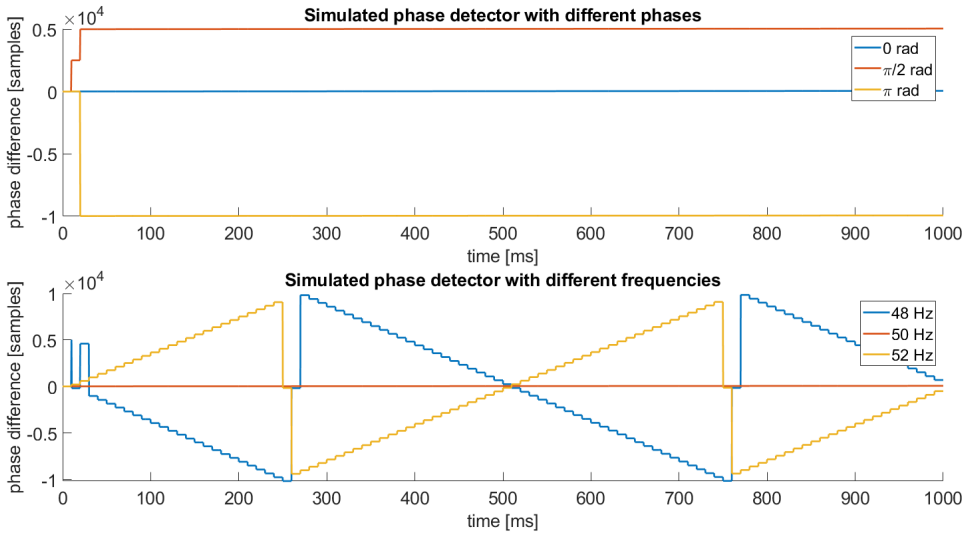


FIGURE 24: Simulated phase detector with both different different frequency and phase differences

Something to be noted in these plots is that the phase difference initially only shows half of the actual value. This is due to averaging of both phase measurements, as explained in the implementation section. Furthermore, the 48 Hz signal shows some odd initial measurements, which at the time of writing could not be explained.

### 4.1.3 Introducing Feedback

To test the simulated controller, the same test was done as for the phase detector test, but this time with the PI controller enabled. Proportional control was set to 2, integral control to 50, and the integral reset threshold to 400.
Fig. 25 shows the results of these tests, with the top graph again having different initial phases and the bottom graph having different initial frequencies and an initial phase difference of *pi* rad. In the top graph it can be seen that the controller takes no more than 300 ms to lock the phases of the two signals in all cases. For the bottom graph, it takes slightly longer to stabilize the oscillator, as there is a frequency difference in the cases of 48 Hz and 52 Hz. But, a "lock" is achieved in 400 ms.

To validate that the oscillator correctly responds to the control signal, the oscillator in the 52 Hz case is plotted together with the references signal in Fig. 26. This shows how the oscillator smoothly changes frequency and almost replicates the grid signals frequency and phase after only 200 ms.

## 4.2 Hardware

After validating the simulation, the hardware implementation of the PLL was tested. Not all of the simulation tests could be replicated. For the ability to capture analog data, a PWM block in combination with a software low-pass filter at 10 KHz was

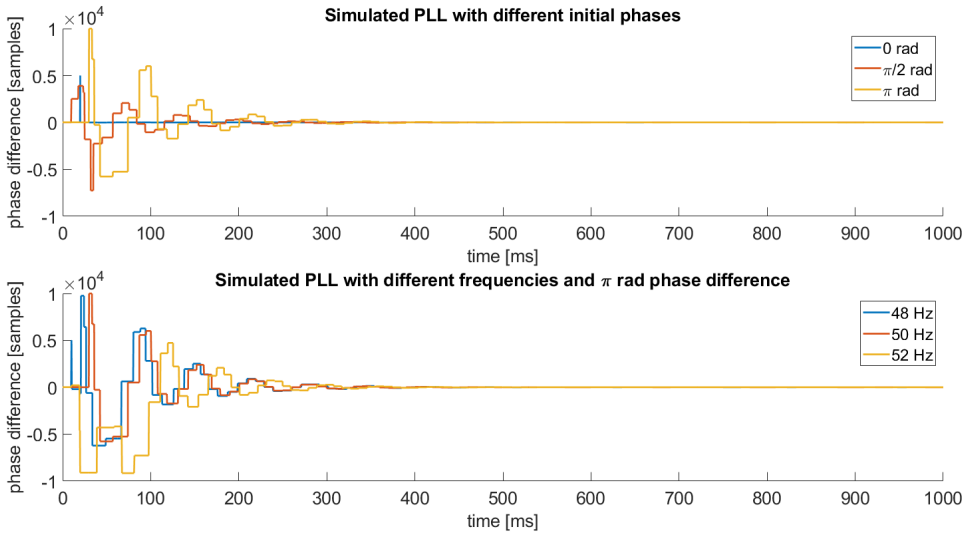


FIGURE 25: Simulated controller with both different initial frequency and phase differences

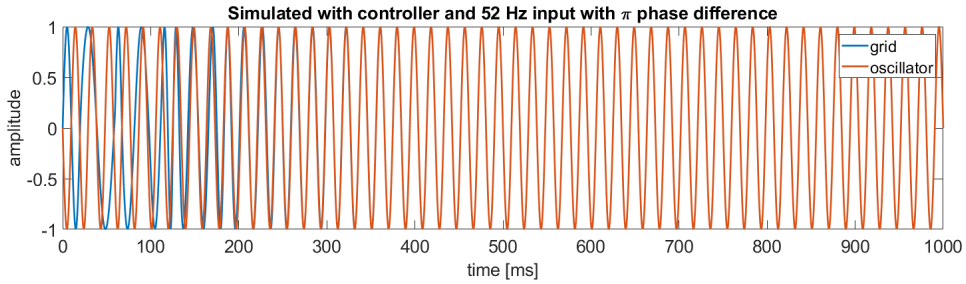


FIGURE 26: Simulated with controller and 52 Hz input with $\pi$ phase difference

used to serve as a DAC. A side effect of using this PWM block was that it could represent values in a range of $0 \leq x < 1$, which does not include the value 1. This means that top and bottom of the oscillator measurements have slight clipping issues. This problem is purely caused by the method of measurement, not by the oscillator itself.
One last note: the FPGA has a logic level of 3.3 v. This is why the measurements have a range of $0 \leq x \leq 3.3$.
All measurements were done using a Picoscope 4824a.

### 4.2.1 Oscillator

The hardware oscillator was tested using the same method as the simulated oscillator. The results of this test can be seen in Fig. 27. This result looks visually identical to the one shown in Fig. 22, with the only difference being the different offset and scale, which are adjusted for in the plot.

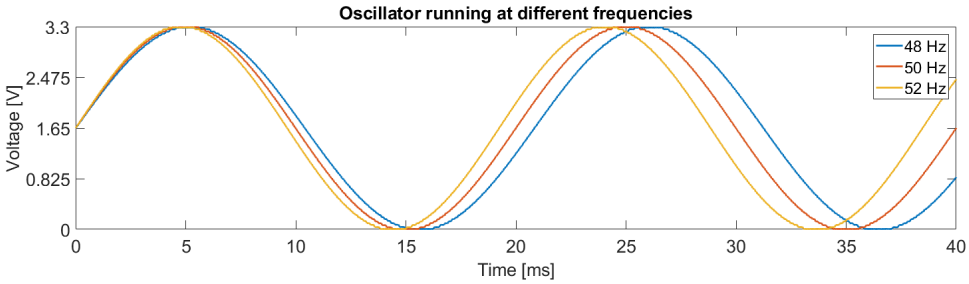


FIGURE 27: Oscillator ran with different frequencies

### 4.2.2 Phase Detection

No easy method could be found to get a consistent measurement of the phase detector. Using AXI communication over JTAG the value could be read out and plotted, but this could not be done with a known and consistent sample time.

### 4.2.3 Introducing Feedback

For testing the hardware controller, it was only possible to control the initial frequency difference. The initial response from the system could also not be plotted, as this required an external trigger when the frequency was changed, which the Picoscope could not provide. The measurements that were taken are of the stabilized system after achieving "lock". This was tested for the different grid frequencies of 48, 50, and 52 Hz. This grid signal was a signal contaminated with "voltage sag". The controller values used for this test are: P = 5, I = 20, Ireset = 400.

The 48 Hz measurement can be seen in Fig. 28, the 50 Hz measurement in Fig. 29, and the 52 Hz measurement in Fig. 30. All of these figures show the same result; the internal oscillator is able to closely follow the captured grid signal, even for different grid frequencies, and contamination.
Not only were these measurements done to test the oscillator and controller, but also to verify the functionality of the ADC. The "grid" signal shown in these figures, was after being captured with the ADC and converted back to analog using the PWM block.

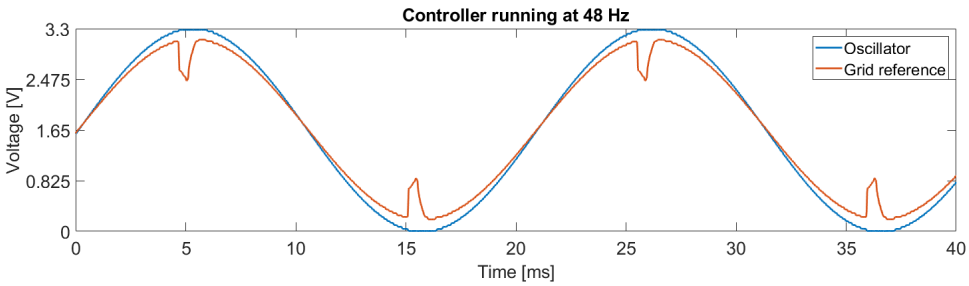


FIGURE 28: Controller running with "contaminated" 48 Hz reference signal

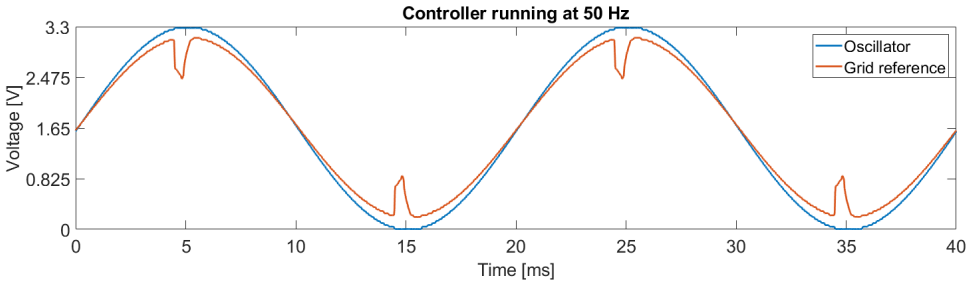


FIGURE 29: Controller running with "contaminated" 50 Hz reference signal

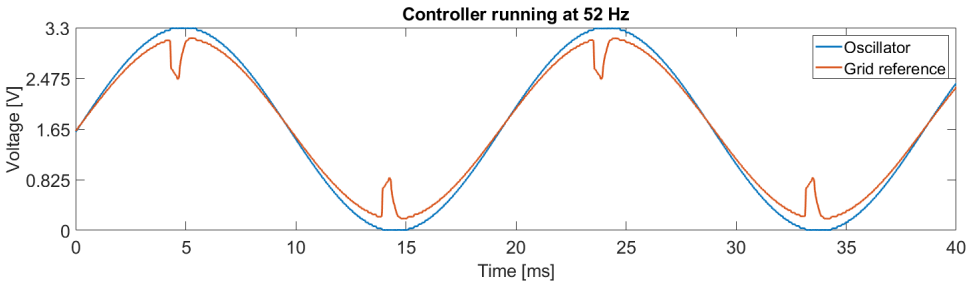


FIGURE 30: Controller running with "contaminated" 52 Hz reference signal

### 4.2.4 Integration with error controller

After integrating the PLL and the error controller systems, it was tested using the "contaminated" grid signal, which has a simulated voltage sag added. Fig. 31 shows the results of the tested integrated setup. It again shows how the PLL closely follows the input grid, but also shows how the error is correctly estimated.

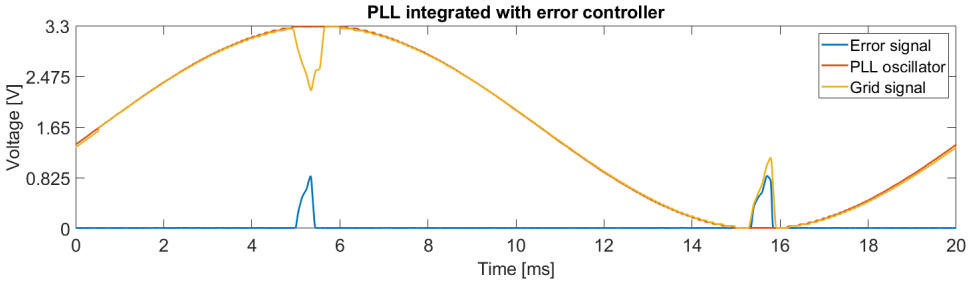


FIGURE 31: PLL integrated with error controller

## 4.3 Further discussion

For the phase detection system to operate correctly, there is a block that measures the grid period. This period could theoretically be used instead of the integral control in the system, as it also provides the frequency information needed for the oscillator. This would only leave the need for a proportional controller, which now only needs to compensate for a constant phase difference.

It was argued in the methodology that it was not possible to use a differentiator in the phase detector, since the resolution of the ADC was too little, as found in equation (4). Instead of using a differentiator, the same result could theoretically be obtained using an integrator. This also shifts the signal by 90 degrees (albeit, to the other side). The integrator would not suffer the same problems as a differentiator, due to its different method of operation. The downside of is that an integrator could introduce an offset to its output, due to different kinds of non-idealities, such as noise. It was thus chosen not to try this method.

Instead of always using the grid signal as the start of the phase detector count, both signals could have been used to start the count. This potentially could make it easier to detect when the internal oscillator is either leading or lagging, without having to measure the length of a grid period.

By finding a way to create an external trigger when the grid oscillator changes frequency, the initial response to this frequency change can be observed. Due to time constraints it was not possible to realize this.

It was not possible to find an reliable way of capturing the phase detector measurements of the FPGA, at least not in time. It could be done by normalizing the phase measurements and using the PWM block, so the data can be captured, and finally rescaling the measurements back. But this was not possible due to time constraints.

# 5 Conclusion

This section will be divided into two subsections. First, a summary of the implemented system will be given, after which conclusions will be drawn regarding the functionality of the system.

## 5.1 System summarized

The system implemented as shown in Fig. 9 is an FPGA based phase-locked loop. This PLL system contains a frequency controllable oscillator, an input signal rescaler, a phase difference detector, a loop filter and a controller.

The frequency controllable oscillator, as shown in Fig. 20 works by increasing a custom variable-step counter, which wraps back to zero after hitting $2\pi$. This counter value is passed through a CORDIC approximation for the sine function, which results in an oscillation.

The input rescaler shifts and scales back the signal captured by the ADC. The ADC has a limited range of $0 \leq x \leq 1$, which causes the need for the grid signal to be scaled down and shifted to be purely positive, which is all done by an op-amp based buffer circuit, which is shown in Fig. 6. The captured input is then shifted

and scaled back to have normalized values, by the rescaler shown in Fig. 11.

The phase difference detector, as shown in Fig. 14-19 consists of 2 detectors, which detect the phase difference between the grid signal and internal oscillator on both the falling and rising edge zero-crossings. They do this using a system inspired by the phase-frequency detector. The detectors count the amount of cycles between the zero-crossing of the grid signal and internal oscillator, after which the detected phase difference, which is only positive, is processed to allow for a negative phase shift, or phase lead.

A digital RC filter is used as loop-filter, to smooth out transitions between phase detections, after which the signal is put through the PI controller, shown in Fig. 21, which uses proportional and integral control, to obtain both phase and frequency control.

## 5.2 conclusion

The oscillator, phase detector, and the controlled system were all tested in simulation, of which the first and the last were also tested on physical hardware. The oscillator simulations, shown in Fig. 22 and 23, show a functional oscillator with the ability to run at different frequencies. The phased detector was confirmed to work in simulation, as shown in Fig. 24, where signals with different phase and frequency differences were tested. Simulation of the controller, as shown in Fig. 25 and 26, showed promising results, with the system stabilizing within 300 ms when exposed to a difference in initial phase and stabilizing in about the same time when exposed to a difference in initial phase and frequency.

On actual hardware, The oscillator performs very similar as shown in Fig. 27. The controlled system also performs well and is able to "lock" onto "contaminated" grid signals running with different frequencies as shown in Fig. 28-30. The PLL system was also integrated with the error controller. The result of this can be shown in Fig. 31.

In conclusion an FPGA based fast-acting grid-tied Phase-locked loop has been constructed, using a model-based design approach in Simulink. The simulated model was evaluated and deployed on an FPGA, where it was tested and confirmed to be operational. The deployed system was able to lock to "contaminated" reference signals, with varying frequencies.

# References


[1] ARM. *AMBA AXI and ACE Protocol Specification Version E.* URL: `https://developer.arm.com/documentation/ihi0022/e`. (accessed: 14.06.2022).

[2] Chetvorno. *Phase locked loop.* URL: `https://commons.wikimedia.org/wiki/File:Phase_locked_loop.svg`. (accessed: 13.06.2022).

[3] Digilent. *CMOD A7-35T reference*. URL: https://digilent.com/reference/programmable-logic/cmod-a7/start. (accessed: 13.06.2022).

[4] Texas Instruments. *RC4558P datasheet*. URL: https://www.ti.com/lit/ds/symlink/rc4558.pdf. (accessed: 17.06.2022).

[5] Texas Instruments. *TL7660 datasheet*. URL: https://www.ti.com/lit/ds/symlink/tl7660.pdf. (accessed: 17.06.2022).

[6] Cornelis J. Kikkert. "Two Novel Phase-Frequency Detectors". In: *APCCAS 2006 - 2006 IEEE Asia Pacific Conference on Circuits and Systems*. 2006, pp. 712–715. DOI: 10.1109/APCCAS.2006.342106.

[7] Mainsfrequency. *Mains frequency*. URL: https://www.mainsfrequency.com/. (accessed: 21.06.2022).

[8] Mathworks. *HDL coder*. URL: https://www.mathworks.com/help/hdlcoder/. (accessed: 17.06.2022).

[9] Mathworks. *HDL verifier*. URL: https://www.mathworks.com/help/hdlverifier/. (accessed: 17.06.2022).

[10] Mathworks. *Model Design for AXI4 Master Interface Generation*. URL: https://www.mathworks.com/help/hdlcoder/ug/model-design-for-axi4-master-interface-generation.html. (accessed: 14.06.2022).

[11] Alexander Matthee, Niek Moonen, and Frank Leferink. "Micro-Grid Inrush Current Stability Analysis". In: *2021 IEEE International Joint EMC/SI/PI and EMC Europe Symposium*. 2021, pp. 440–444. DOI: 10.1109/EMC/SI/PI/EMCEurope52599.2021.9559165.

[12] Wikipedia. *CORDIC*. URL: https://en.wikipedia.org/wiki/Phase-locked_loop. (accessed: 21.06.2022).

[13] Wikipedia. *CORDIC*. URL: https://en.wikipedia.org/wiki/CORDIC. (accessed: 17.06.2022).

[14] Xilinx. *XADC-wizard product guide*. URL: https://docs.xilinx.com/v/u/en-US/pg091-xadc-wiz. (accessed: 14.06.2022).