MASTER'S THESIS

All-Digital Control and Natural Frequency Tracking of Resonant, Electrostatic Comb Driven, MEMS Scanning Mirrors

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Electrical Engineering and Economics

submitted to

Graz University of Technology

by

Nikola Blazevic, BSc (01130672)

Supervisor

Ass.Prof. Dipl.-Ing. Dr.techn. Peter Söser

Department of Electronics

Inffeldgasse 12/I, A - 8010 Graz

in coorperation with

Infineon Technologies Austria AG

Alberto Garcia Izquierdo, BEng MSc

Graz, December 17, 2019

AFFIDAVIT

Ethischer Kodex der TU Graz:

Wir sind eine Gemeinschaft der Forschenden, Lehrenden, Studierenden, Mitarbeiterinnen und Mitarbeiter und Alumnae und Alumni in einer Atmosphäre der intellektuellen Freiheit und Verantwortung. Wir bekennen uns zur Verbindung von Forschung und Lehre auf höchstem Niveau im weltweiten Wettbewerb vergleichbarer Einrichtungen. Auf den Grundlagen der Vision der TU Graz und dem Wissen, dass TechnikerInnen und WissenschafterInnen mit ihrem Denken und Handeln einen überaus starken Einfluss auf Gesellschaft und Umwelt ausüben, ist es der TU Graz wichtig, schon zu Beginn des Studiums bzw. eines Dienstverhältnisses auf die damit verbundene Verantwortung hinzuweisen und ihre Angehörigen zu einem ethisch einwandfreien Handeln zu verpflichten. Als StudentIn und AngehörigeR der TU Graz habe ich die Richtlinie des Rektorates der Technischen Universität Graz zur Sicherung guter wissenschaftlicher Praxis und zur Vermeidung von Fehlverhalten in der Wissenschaft gelesen und verstanden, ich unterstütze und anerkenne diese Richtlinie vorbehaltlos.

Ort, Datum *Unterschrift*

Abstract

Microscanner or MEMS Scanning Mirrors get used at LIDAR as well in other optical reflecting systems. Depending upon the type, the movement can be either translational or rotational. The aim of this Master-Thesis is to bring and keep such a mirror at resonance frequency and this with the aid of an all digital closed control loop. For this task, a resource-saving FM-Demodulator should demodulate the FM signal by the mirror and a DPLL should generate a voltage signal which moves and controls the mirror electro statically. Firstly, a Matlab Simulink Model should be created and afterwards a VHDL code should be written. Additionally, several FM-Demodulation methods should get tested to find the most resource-saving demodulation method. However, the most resource-saving method shows a worse SNR than the method which needs the most resources. In summary, an all digital and resource-saving control loop is possible but due to error proneness not wise to implement. On the other hand, a not resource-saving method would not be economic.

Keywords: LIDAR, MEMS Scanning Mirror, All Digital PLL, FM-Demodulation, MATLAB, VHDL

Kurzfassung

Mikroscanner oder auch MEMS Scanning Mirrors werden sowohl für LIDAR als auch für andere optisch reflektierende Systeme verwendet. Ziel dieser Diplomarbeit ist es einen solchen Mikroscanner durch eine vollkommen digitale geschlossene Kontrollschleife auf seine Resonanzfrequenz zu schwingen und diese beizubehalten. Dabei soll eine ressourcenschonende FM-Demodulation des vom Spiegel FM modulierten Signals stattfinden und mit Hilfe einer DPLL ein Spannungssignal erzeugt werden, welches den Spiegel durch elektrostatische Kräfte in Bewegung hält. Zuerst soll ein Matlab Simulink Modell angefertigt und anschließend ein VHDL Code erzeugt werden. Zudem werden mehrere FM-Demodulationsmethoden erprobt um die am Ressourcen schonendste Methode herauszufinden. Die Methode, die die wenigsten Ressourcen benötigt weist jedoch eine deutlich schlechtere SNR auf als die Methode, die die meisten Ressourcen benötigt. Hierdurch entsteht das Resümee, dass eine komplett digitale und ressourcenschonende Kontrollschleife möglich, jedoch aufgrund von Fehleranfälligkeit nicht sinnvoll ist. Andererseits wäre eine Methode, die mehr Ressourcen benötigt, wiederum nicht wirtschaftlich.

Schlagwörter: LIDAR, MEMS Scanner, Digitale PLL, FM-Demodulation, MATLAB, VHDL

Contents

List of Figures

List of Tables

Abbreviations

ASIC Application-Specific Integrated Circuit

DPLL Digital Phased-Lock Loop

FPGA Field Programmable Gate Array

IEEE Institute of Electrical and Electronics Engineers

LF Loop Filter

LIDAR Light Detection and Ranging

LSB/MSB Least/Most Significant Bit

MEMS Micro-Electro-Mechanical System

NCO/VCO Numerically/Voltage Controlled Oscillator

OPA Optical Phased Array

PFD Phase Frequency Detector

PLL Phase-Locked Loop

SNR Signal-To-Noise Ratio

VHDL V for VHSIC, Very High Speed Integrated Circuit, and HDL, Hardware Description Language

1 Introduction

1.1 Motivation

Self-driving cars are an upcoming topic of future. With the aid of LIDAR systems it is possible to make 3D environment images which are important for the car's calculations[1]. To implement such a LIDAR not directly visible in a car body, it is necessary to create it in a mini size. A bundled laser beam gets reflected by a rotating mirror to reach and scan a certain angle of the environment. Usually, MEMS, Micro-Electro-Mechanical System, Scanning Mirrors are used for this kind of challenges[2]. Now it is about to control it through electrostatic and to swing the mirror to its resonance frequency. This thesis should bring an overview if it is possible to do this in an all digital way and how satisfying this method is.

1.2 Problem statement

The initial problem is to find a mathematical model for the MEMS mirror. Generally, it is a non linear mass-spring-damper system. For that reason the oscillation build-up is another problem beside the right moment of the rectangular pulse voltage at the input. Since the mirror has a varying capacitance, depending on its orientation, an FM modulated signal should be the output of the mirror Matlab Simulink Model. To demodulate this signal, a proper solution should be found which is resource-saving to have the possibility of producing an ASIC of the whole control loop. An additional problem is the signal extraction of the demodulated signal. Since all signals should be quantized to simulate an environment like on an ASIC, the demodulated signal will not be as smooth as to extract a falling and rising edge of the mirror that easily. This extracted signal should be the reference input of an All Digital PLL which should be created and adjusted.

1.3 Aim of the work

The aim of the work is the creation of an All Digital PLL and closed control loop to swing and keep the MEMS Scanning Mirror at its resonance frequency. First of all it should get created in Matlab Simulink and afterwards, if a quantized version of this model works, in VHDL. The control loop should be checked for a low SNR by overlapping a white noise over the FM signal. The finished work should include a proper working quantized Matlab Simulink Model and a synthesis possible VHDL code.

1.4 Structure of the work

First of all, a Matlab Simulink Model of the MEMS Scanning Mirror has to be created. Then a DPLL has to get designed and with the aid of it the mirror has to get swung to its resonance frequency and kept at it. When these parts both work properly an FM-Modulation and FM-Demodulation part, with an additional signal extraction, have to get placed between the mirror and the DPLL. Afterwards, if the closed control loop works, all signals have to get quantized and a shaper has to shape the input voltage to make the simulation more realistic. If the Matlab Simulink Model works properly a VHDL code of it has to get written.
The closed control loop has to be split in these parts:

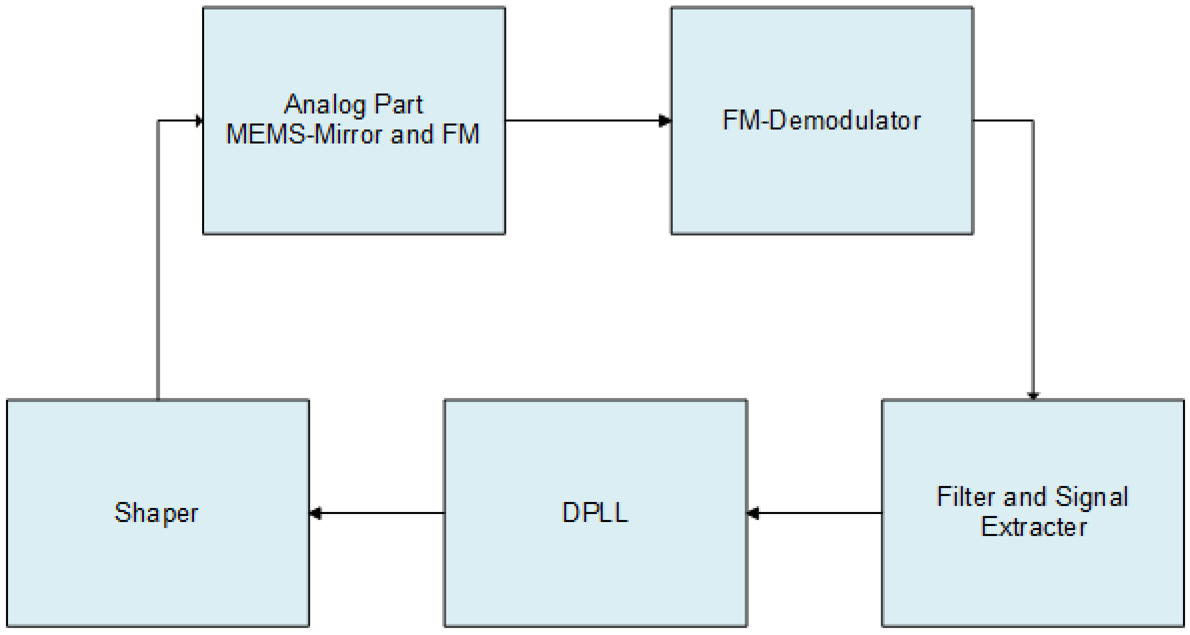


Figure 1.1: Block diagram of the whole control model

To control the mirror movement a control model like in Figure 1.1 will be implemented.

Analog Part MEMS-Mirror and FM is the part of the mirror which includes the MEMS mirror model and the FM-Modulation. The model of the mirror is described in Chapter 2.2. The mirror's output signal is the corresponding capacitance which depends on the angle. With the changing capacitance an LC-Oscillator generates a frequency modulated signal.

FM-Demodulator is the part which demodulates the FM signal. Several methods of FM-Demodulation will be discussed at Chapter 2.4.

Filter and Signal Extracter is the part which filters the demodulated FM signal and generates a corresponding square wave at 0°-angle-crossing of the mirror.

DPLL is the part which gets an input square wave and tries to reproduce this signal at the same frequency. To compensate the determined delay from the parts before, a delay will be placed into the DPLL. Full particulars are at Chapter 2.5.

Shaper is the part which shapes the output square wave of the DPLL to input a rounded voltage signal to the mirror.

2 Background

2.1 LIDAR

LIDAR, Light Detection And Ranging, is an optical measuring method for detection and distance measuring of objects in an area. It is similar to the RADAR method but instead of using microwaves LIDAR is using laser light. The common used method at LIDAR systems is Time of Flight. In that case the time from the outgoing until receiving the from the object reflected laser light gets measured and this is proportional to the distance of the object.[3] In practice, LIDAR systems get used at traffic supervision, wind energy market to measure wind speed and find the right spot for building a wind turbine and at the automotive industry[4]. As break assistances LIDAR systems detect objects in front of the driving car to send warnings or to initiate a full breaking[5]. As well LIDAR systems already get used to make 3D environment pictures. For that reason, a rotating mirror reflects the laser waves and through the differences at several points of the objects it is possible to calculate a 3D copy of the environment like several car companies do for tests on self-driving cars[6]. In the future, Solid-State LIDAR Systems will lead the automotive LIDAR market. This variation is more resistant in relation to vibrations and more compact by not using mechanically moving mirrors. Basically, if we talk about Solid-State LIDAR Systems two methods are common today. An OPA, Optical Phased Array, which channels the laser beam by a two-dimensional surface using adjustable surface elements[7]. The second method is a MEMS Scanning Mirror, Chapter 2.2, which has a high potential to be the future LIDAR solution. The benefits are light-weight, compact and low power consumption[8].

Figure 2.1 demonstrates the principle MEMS Mirror scanning method. A laser steers its beam to a rotating mirror which allows a vertical scanning of an object. If the mirror reflects a laser beam which is channelled through a, by more than one laser, illuminated lens, a horizontal scan occurs too. The result is a 3D scan of an object.

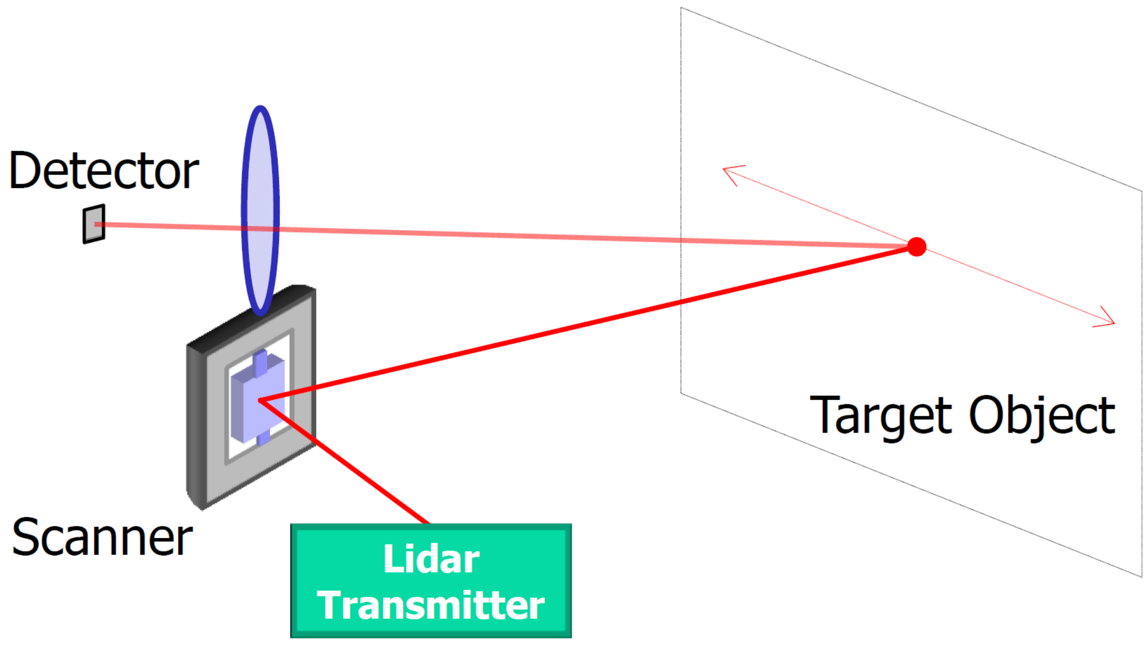


Figure 2.1: LIDAR model of 3D-detection

2.2 MEMS Scanning Mirror

Nowadays, MEMS, Micro Electromechanical Systems, are common in parts like position detection at digital cameras or as microphones at smartphones and headsets. But also at automotive sector MEMS take place as activators for air bags or as tire pressure sensors. MEMS combine semiconductor technology with miniature mechanics in the micron range. Low pressure changes or small electrical impulses are necessary to move the thin silicon channels.[9] Thank to its very small size and low production costs MEMS should innovate the automotive LIDAR market in future. The most common method to rotate the MEMS mirror is electrostatic comb drive. At this method a voltage gets applied and this produces an electrostatic force which pulls the mirror to its centre line. Also the MEMS Scanning Mirror can be categorized into resonant and non-resonant, according to their operating frequency, mirrors. The benefits of resonant MEMS mirrors are a large scan angle at a simple control design at which non-resonant MEMS mirrors have a large degree of freedom in the trajectory design. But all the more complex actuation controller is needed to keep a constant scan speed at large scan ranges. Non-resonant, also called quasi-static, MEMS mirrors have a smaller scanning angle compared to the resonant MEMS mirrors.[8] This thesis deals with a resonant controlled MEMS Scanning Mirror therefore such this type gets described in detail. Figure 2.2 shows the mirror to control. The mirror has a size of 2.7 mm x 2.3 mm and the smallest parts are just 1 µm big.

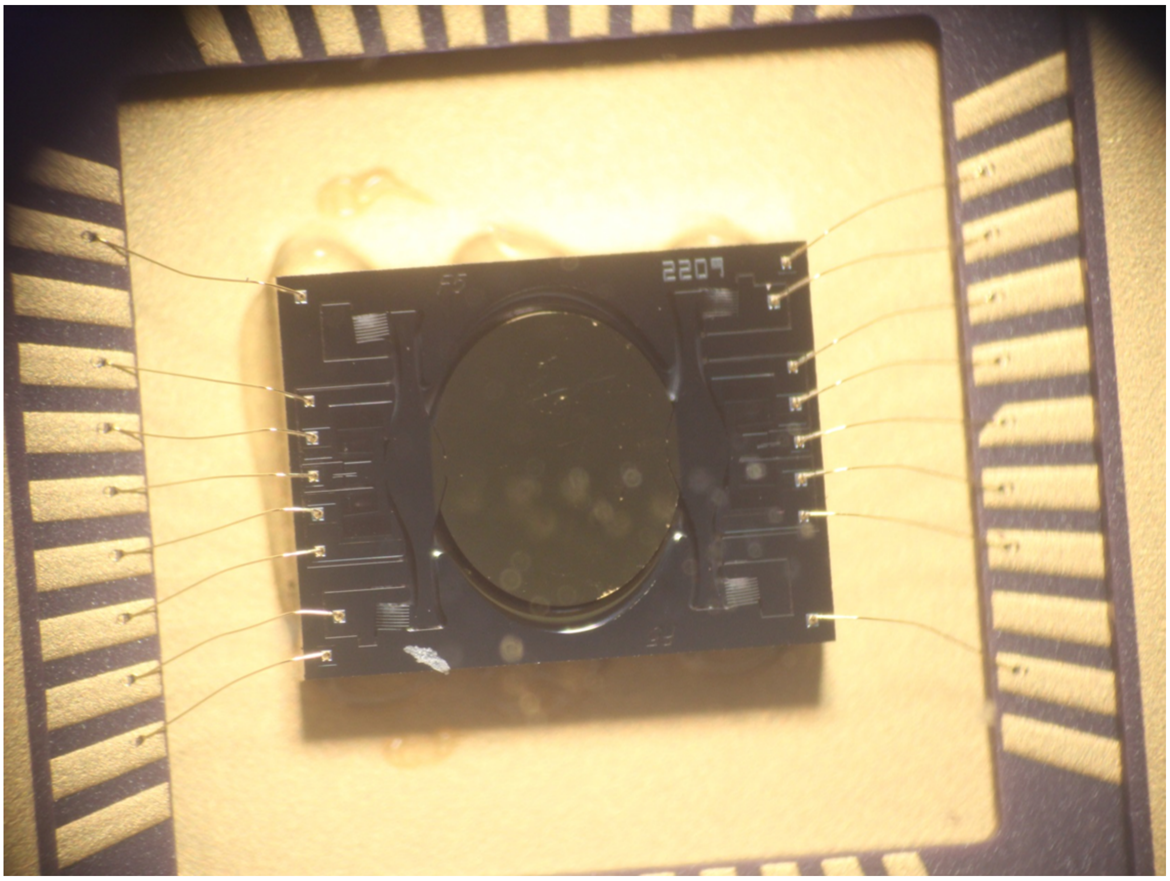


Figure 2.2: MEMS-mirror

Figure 2.3 shows a 3D model of the used mirror. It is clearly to see that the mirror is stiffly connected to two bars at each side. These bars are connected over two spring combs at each corner to the package. Rotation of the mirror causes a geometrical changing capacitance like shown at Figure 2.4.

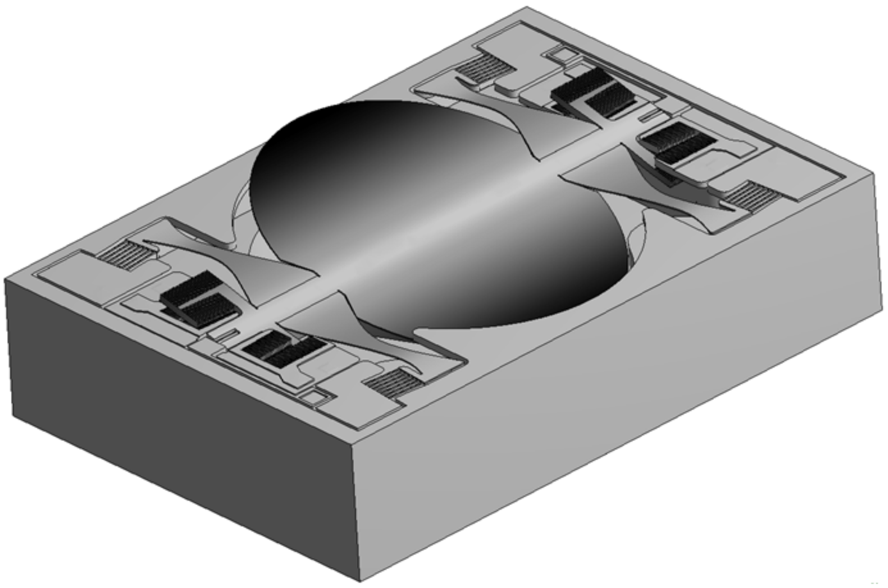


Figure 2.3: Model of the MEMS-mirror

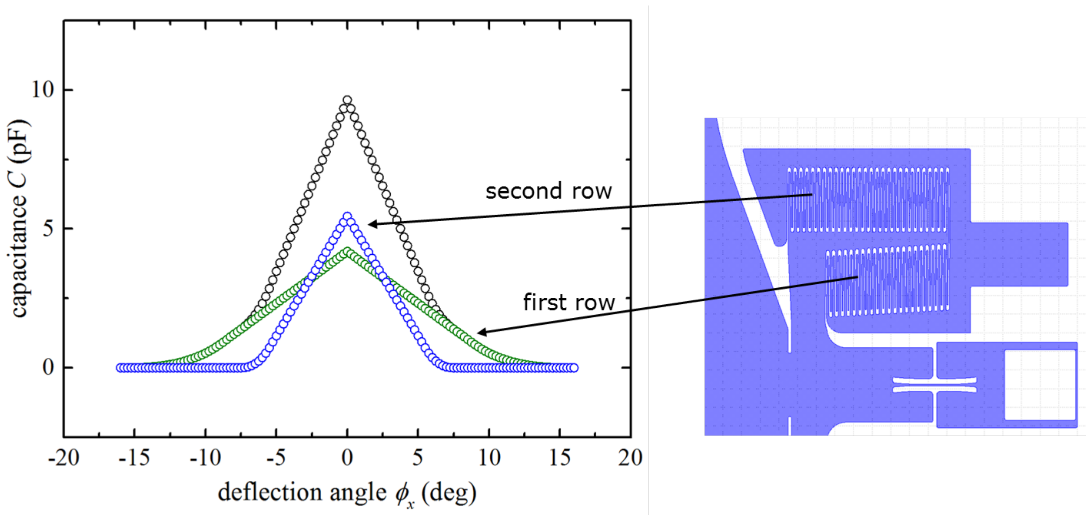


Figure 2.4: Geometrical changing capacitance and overlap approximation

These of two different springs caused capacitances have different but symmetrical waves and form a main capacitance. This resulting capacitance creates at connection to an inductor an LC-Oscillator which, under simplified assumptions, has a frequency of [10]

$$f = \frac{1}{2\pi\sqrt{LC_{var}}} \tag{2.1}$$

With the aid of a varying capacitance and a constant inductance the LC-Oscillator generates a frequency modulated signal. Details about frequency modulation at Chapter 2.3.
The mirror has a moving range of -16° to +16° and due to the mirror's electrostatic comb driven actuation its initial position has to be unequal 0°. The electrostatic comb driven actuation works as follows: Since the capacitive combs are attached to a pivot the changing capacitance and the changing angle produce a torque

$$\tau = \frac{U^2 dC}{2d\phi} \tag{2.2}$$

if a voltage is applied at the capacitive building plates.[11] This produced torque pulls the mirror to its centre line of 0° if the mirror is tilted. Therefore the mirror's initial position has to be unequal 0° which is solved by its geometrical form. This is the method how to control the mirror. The voltage has to be applied until the mirror reaches 0° and after turning off the voltage the mirror swings through its kinetic energy but at the same time it gets damped by the spring. After the mirror gets damped and reaches its maximum amplitude it changes its direction and the voltage has to get applied again to create a torque and hence to gain momentum. This method brings the mirror to resonance. Figure 2.5 illustrates the method with the angle of the mirror φ, the applied voltage U and the produced torque T from the centre line.

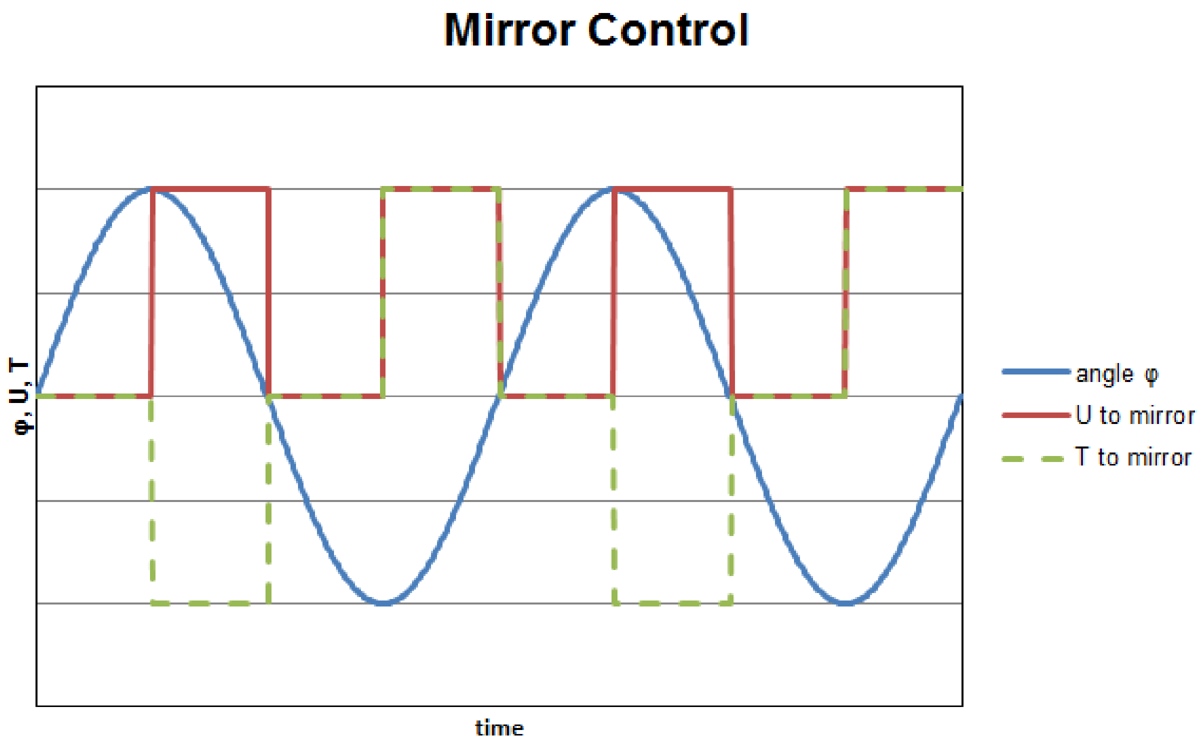


Figure 2.5: Capacitance dependence in overlap approximation

At positive angle the torque becomes negative and vice verse at negative angle the torque is positive. For that reason the mirror gets always pulled to 0° angle. This brings four different sectors of the mirror movement depending on its angle direction of velocity for controlling the mirror. Figure 2.6 illustrates this assumption.

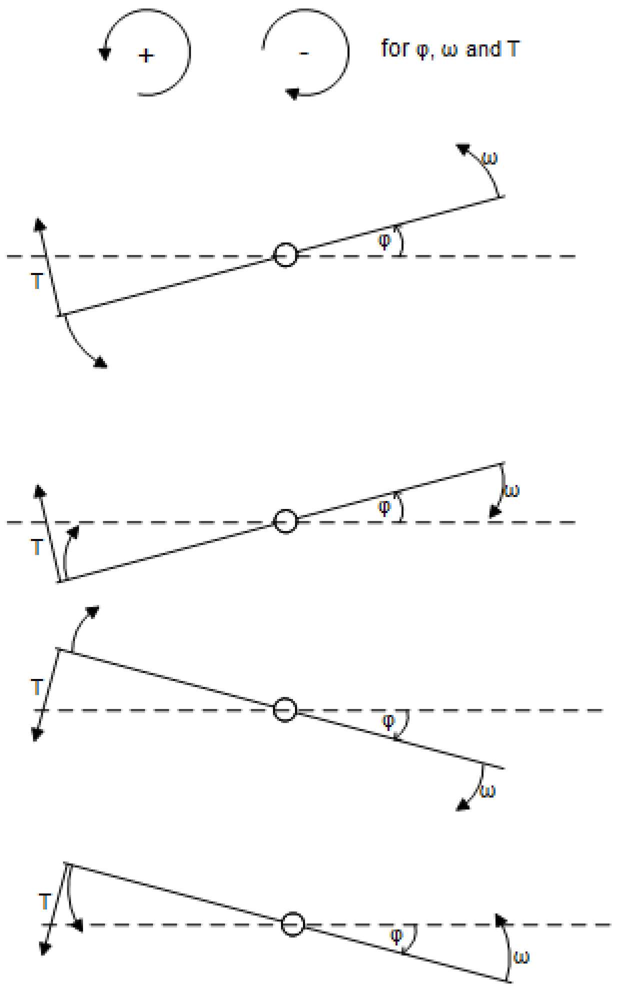


Figure 2.6: 4 Sectors of mirror movement

Sector 1:

Mirror angle is positive as well as the velocity.

Torque is negative and pulls the mirror.

Sector 2:

Mirror angle is still positive but velocity turned to other direction.

Torque is still negative because of the angle.

Sector 3:

Mirror angle is negative as well as velocity now.

Torque pulls from other side and is positive.

Sector 4:

Mirror angle is still negative but velocity turned to other direction again. Torque is still positive.

Table 2.1 summarizes the four sectors:

	angle φ	velocity ω	torque T
1	+	+	-
2	+	-	-
3	-	-	+
4	-	+	+

Table 2.1: 4 Sectors of mirror movement

Figure 2.7 illustrates the torque depending on the angle of the MEMS Scanning Mirror.

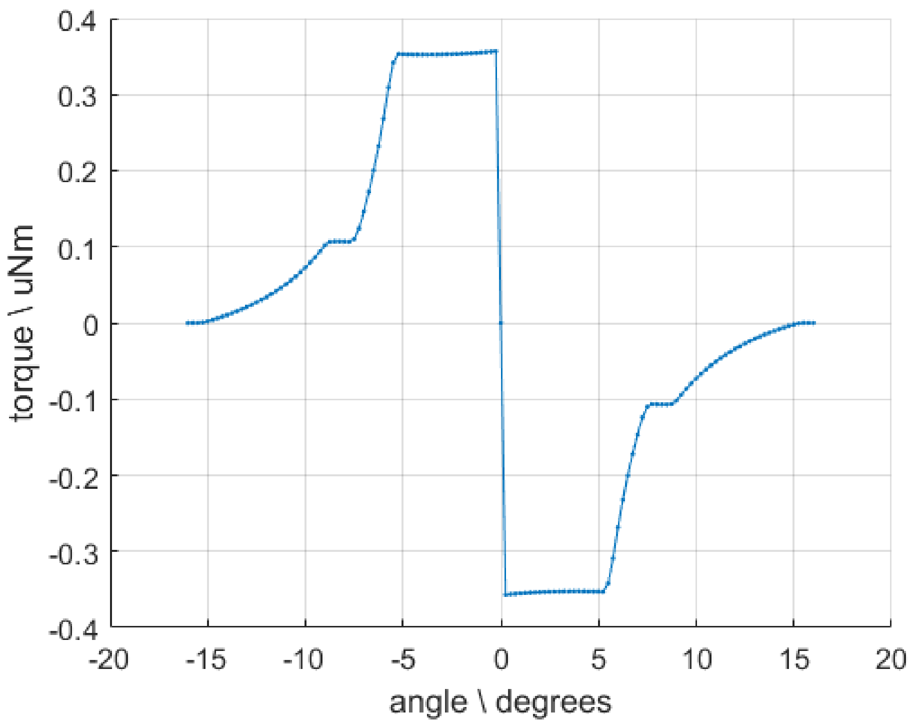


Figure 2.7: Torque depending on angle at 100V

The MEMS Scanning Mirror behaves like a Duffing oscillator with linear damping and can be described mathematically as follows:[12]

$$I\ddot{\varphi} + \gamma\dot{\varphi} + k_1\varphi + k_3\varphi^3 = \tau_{max}h(\varphi)f_{\Omega}(t) \tag{2.3}$$

φ : angle of the mirror

$\dot{\varphi}$: velocity ω of the mirror

$\ddot{\varphi}$: acceleration α of the mirror

I : moment of inertia

γ : damping term (γ=2ςIω_0)

ς : damping

ω_0 : angular frequency

k_1, k_3 : linear and cubic restoring force coefficients

τ_{max} : maximal actuation moment. Design- and voltage-dependent. It is reached (h(φ)=1) when all comb drives are engaged, but not perfectly overlapping.

$h(\varphi)$: "shape function" of actuating moment. Depends on position, which makes the mirror a "parametric" oscillator. Shown at Figure 2.7

$f_{\Omega}(t)$: supplied driving signal, square pulse voltage

The non-linearity of the Duffing oscillator effects the jump phenomenon[13] which produces a tilted oscillation amplitude as in Figure 2.8. This Figure would typify an externally forced MEMS mirror defined by the equation

$$I\ddot{\varphi} + \gamma\dot{\varphi} + k_1\varphi + k_3\varphi^3 = \tau_{max}sin(\Omega t) \tag{2.4}$$

Ω : excitation frequency

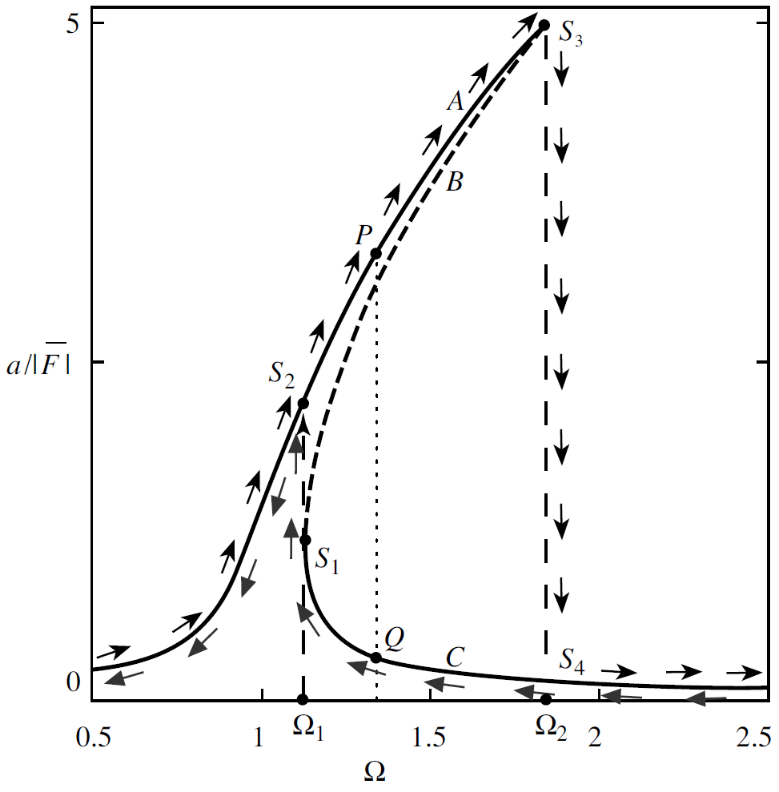


Figure 2.8: Response of the Duffing oscillator

Since the MEMS mirror is electrostatically driven this causes a non-linearity at the torque by changing the sign at 0° crossing. This additional non-linearity causes a changed, cut oscillation amplitude like in Figure 2.8. For that reason the MEMS mirror just can be animated from frequency higher than its resonance frequency.

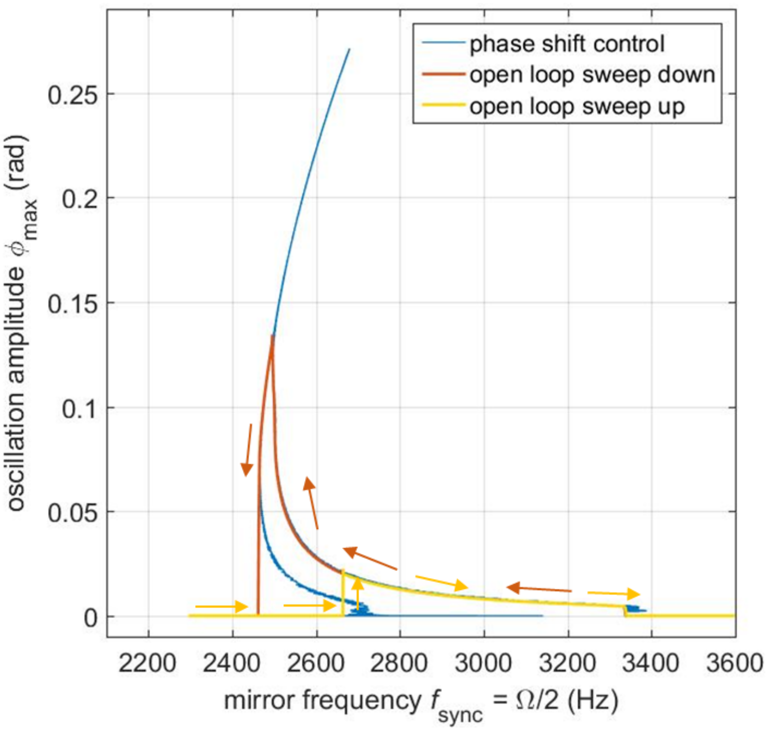


Figure 2.9: Response of the mirror

This produces the following stability Table 2.2 for the open loop driven MEMS mirror which is illustrated in Figure 2.10.

		initial position φ_0 (mrad)				
open loop driving frequency Ω (Hz)		100	10	1	0.1	0.01
	6400	5.8	5.8	5.8	5.8	5.8
	6000	8.1	8.1	8.1	8.1	8.1
	5600	12.6	12.6	12.6	12.6	12.6
	5200	27.5	27.5	0.0	0.0	0.0

Table 2.2: Stability of open loop driven MEMS mirror table

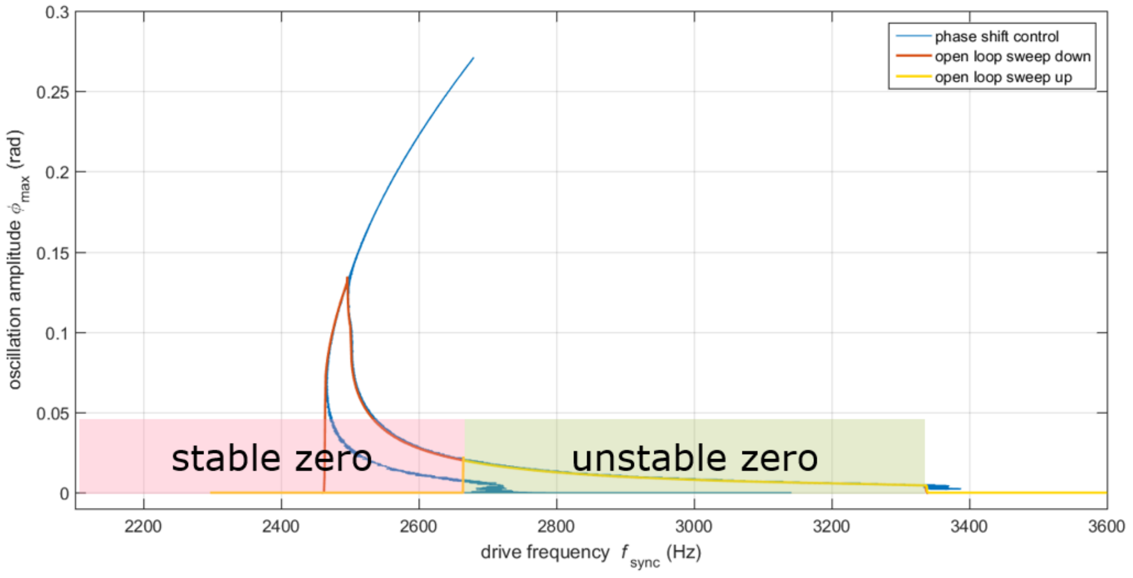


Figure 2.10: Stability of open loop driven MEMS mirror

2.3 Frequency Modulation - FM

At FM, Frequency Modulation, the incoming information changes the frequency of the carrier signal while the amplitude remains constant. This causes a frequency change like illustrated in Figure 2.11 in which the black signal is the incoming information and the blue signal is the frequency modulated signal.

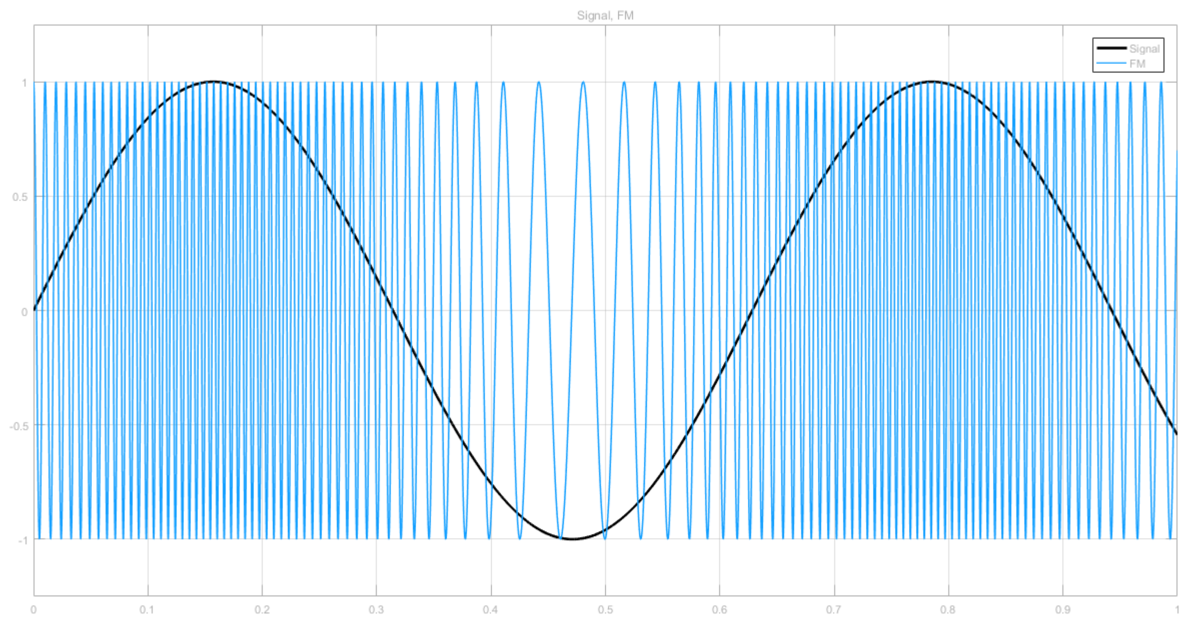


Figure 2.11: Frequency Modulated Signal

The most famous usage of this method is the FM radio.[14] The reason why a modulation gets used is because the signal gets less interference-prone especially at radio links. A frequency modulated signal has a carrier frequency f_T and a frequency swing Δf which can be calculated as follows:

$$\Delta f = \frac{f_{max} - f_{min}}{2} \tag{2.5}$$

$$f_{max} = f_T + \Delta f \tag{2.6}$$

$$f_{min} = f_T - \Delta f \tag{2.7}$$

At high frequencies an FM signal can be generated with the aid of oscillator circuits like the MEMS Scanning Mirror produces its FM signal. A changing capacitance causes a changing frequency like Equation 2.1 describes. But also a VCO/NCO, voltage/numerical controlled oscillator, can serve as an FM-Modulator.[15]

2.4 FM-Demodulation

There are a few reliable methods to demodulate an FM modulated signal. The following three methods got tested at the closed control loop of the MEMS Scanning Mirror at the Matlab Simulink Model.

2.4.1 PLL-Demodulator

Because the PLL wants to follow the incoming, FM modulated, signal the loop filter produces a signal which is corresponding to the FM demodulated signal.[16] The PLL's functionality is described at Chapter 2.5 as well as the loop filter's demodulating attitude. Figure 2.12 illustrates the PLL-Demodulator functionality.

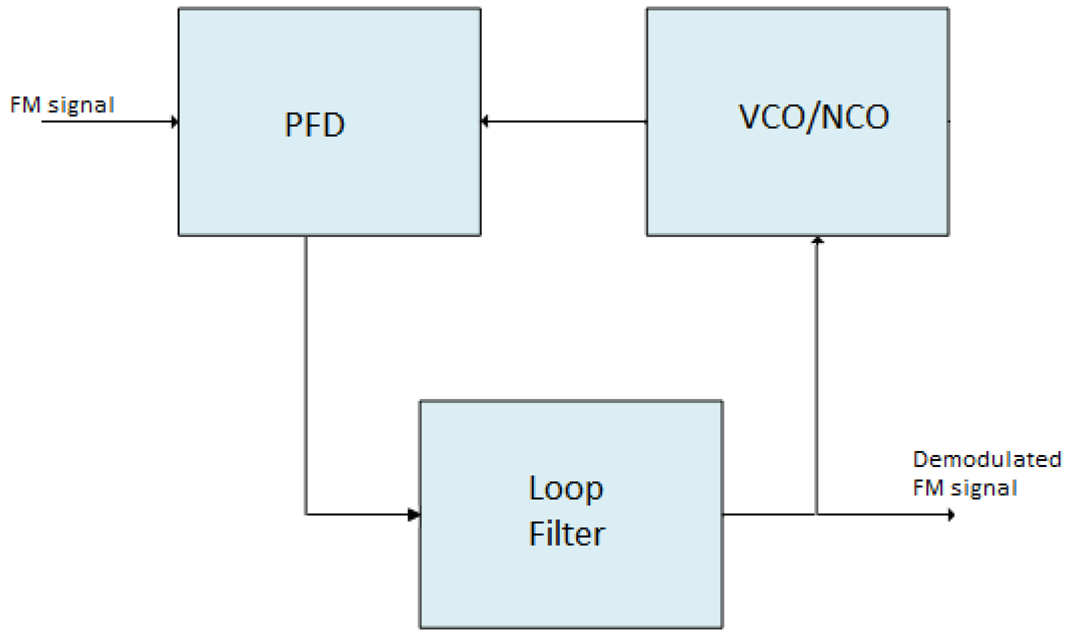


Figure 2.12: PLL-Demodulator

2.4.2 Slope Detection Demodulator

Figure 2.13 illustrates the functionality of this FM-Demodulator method.

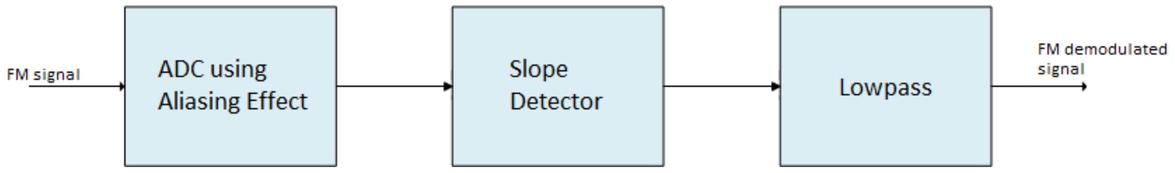


Figure 2.13: Slope Detection Demodulator

The Slope Detection Demodulator works as follows: The incoming analog signal goes to the ADC and gets sampled at a consciously chosen lower sample frequency than the actual frequency of the FM signal. Thus the high frequency of the FM signal gets, with the aid of the aliasing effect, transformed to a lower frequency so that the sampling rate of the digital circuit can be lower than with the former frequency range of the FM signal. This method is called Undersampling, band pass sampling, harmonic sampling or super-Nyquist sampling. The resulting aliasing frequency can be calculated as follows:[17]

$$f_a = f_s - f_{in} \tag{2.8}$$

f_a: appearing aliasing frequency

f_s: sampling frequency

f_{in}: incoming frequency

Figure 2.14 illustrates the functionality of undersampling.

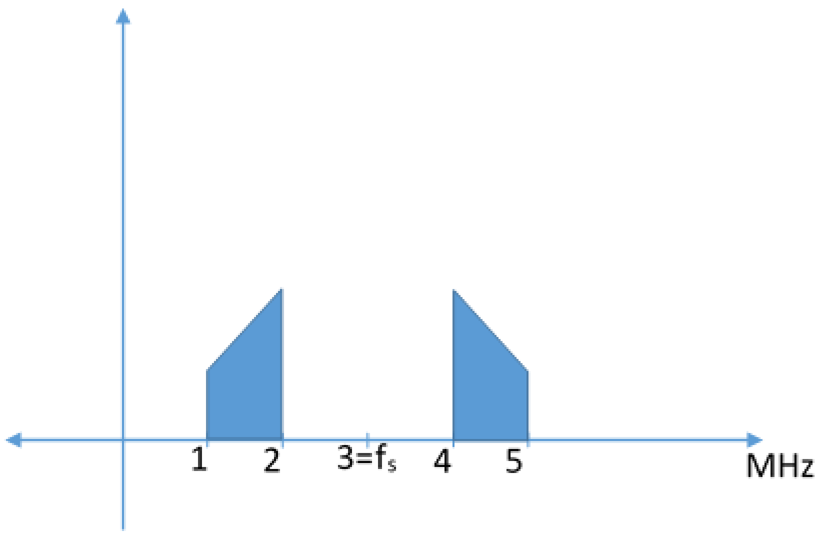


Figure 2.14: Undersampling

Afterwards, the digitalized FM signal goes to a slope detector which generates an impulse at each rising or falling edge. The following lowpass filter counts and integrates the incoming impulses and produces the FM demodulated signal[18]. Figure 2.15 illustrates the functionality of the lowpass integrating over the rectangular impulses.

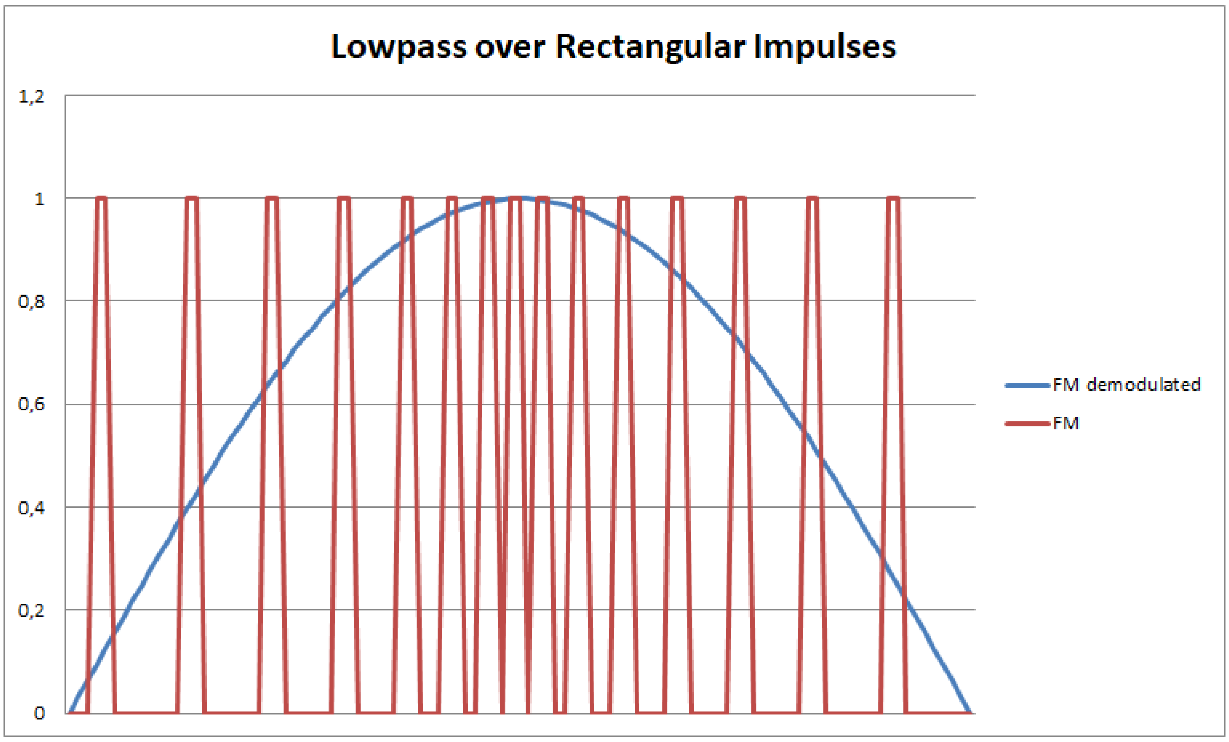


Figure 2.15: Rectangular Impulses and Integrating Lowpass

2.4.3 Coincidence-Demodulator

Figure 2.16 illustrates the functionality of this FM-Demodulator method.

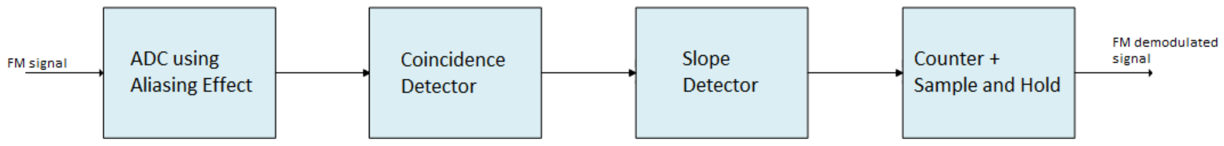


Figure 2.16: Coincidence-Demodulator

At the Coincidence-Demodulator the FM signal goes like in the previous FM-Demodulator to the ADC where it gets converted to digital and transformed in frequency. The next step is the coincidence detector which is an XOR logic gate and has two inputs. One is the digital FM signal and the second is the same signal just time delayed.[19] The time delay determines the working frequency range of the Coincidence-Demodulator.

$$f_{wr} = \frac{1}{2T_{delay}} \qquad (2.9)$$

f_{wr} : working frequency range of the Coincidence-Demodulator

T_{delay} : time delay of the second input

The Coincidence-Demodulator has more than one working range, to be specific the working range is repeating for each f_{wr}. For each second f_{wr} the output is exactly mirrored. Figure 2.17 illustrates this.

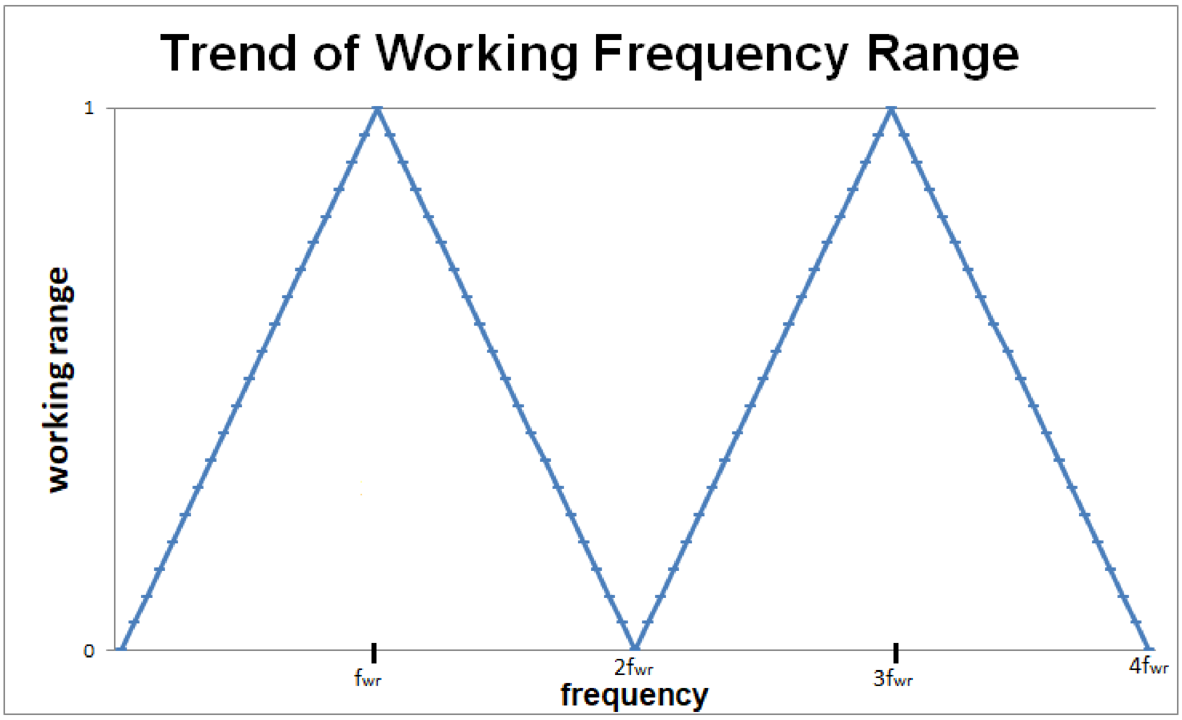


Figure 2.17: Working Frequency Range for Coincidence-Demodulator

So for the first working range the lowest possible input frequency can be 0 Hz and the maximum possible input frequency f_{wr}. For that reason the incoming frequency range has to be a part of the working frequency range. If the incoming frequency is at the lowest point of the working frequency range the Coincidence-Demodulator has a constant output of 0 like in Figure 2.18.

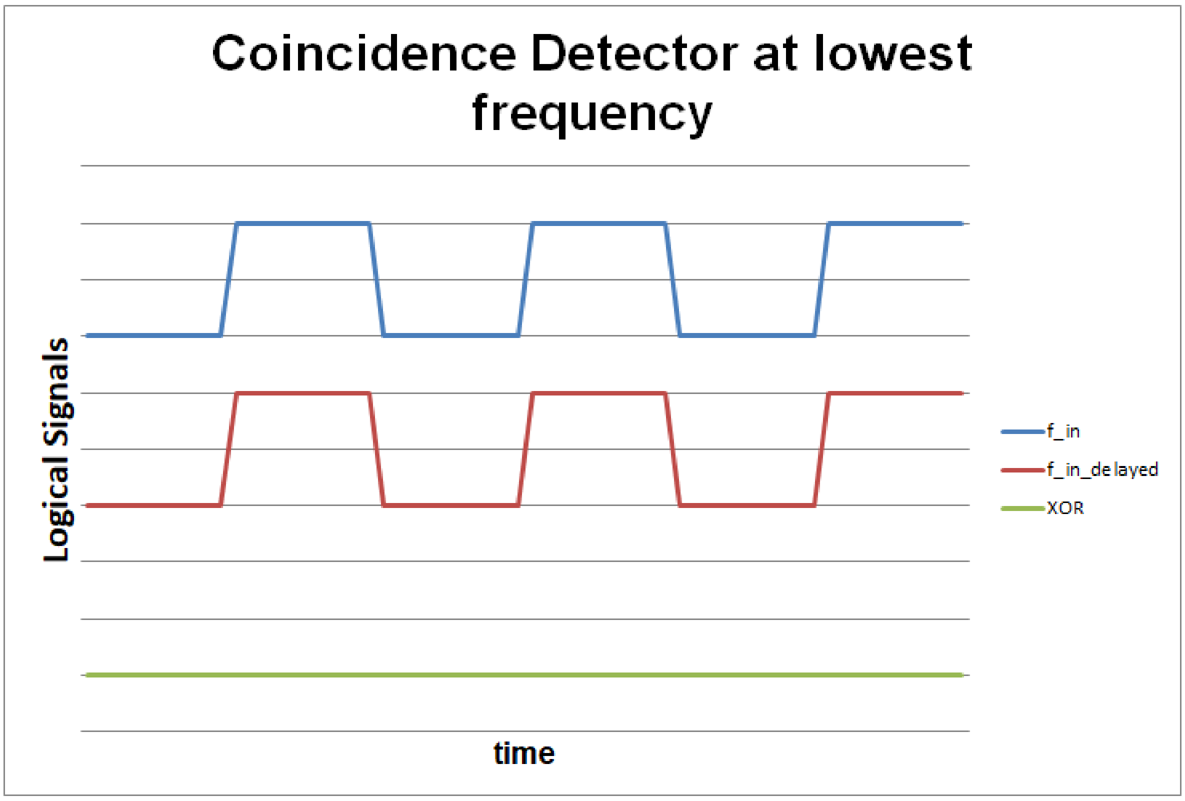


Figure 2.18: Coincidence-Demodulator at lowest frequency input

If the input frequency is at the middle of the working frequency range the output is a square signal with double frequency of the input signal like in Figure 2.19.

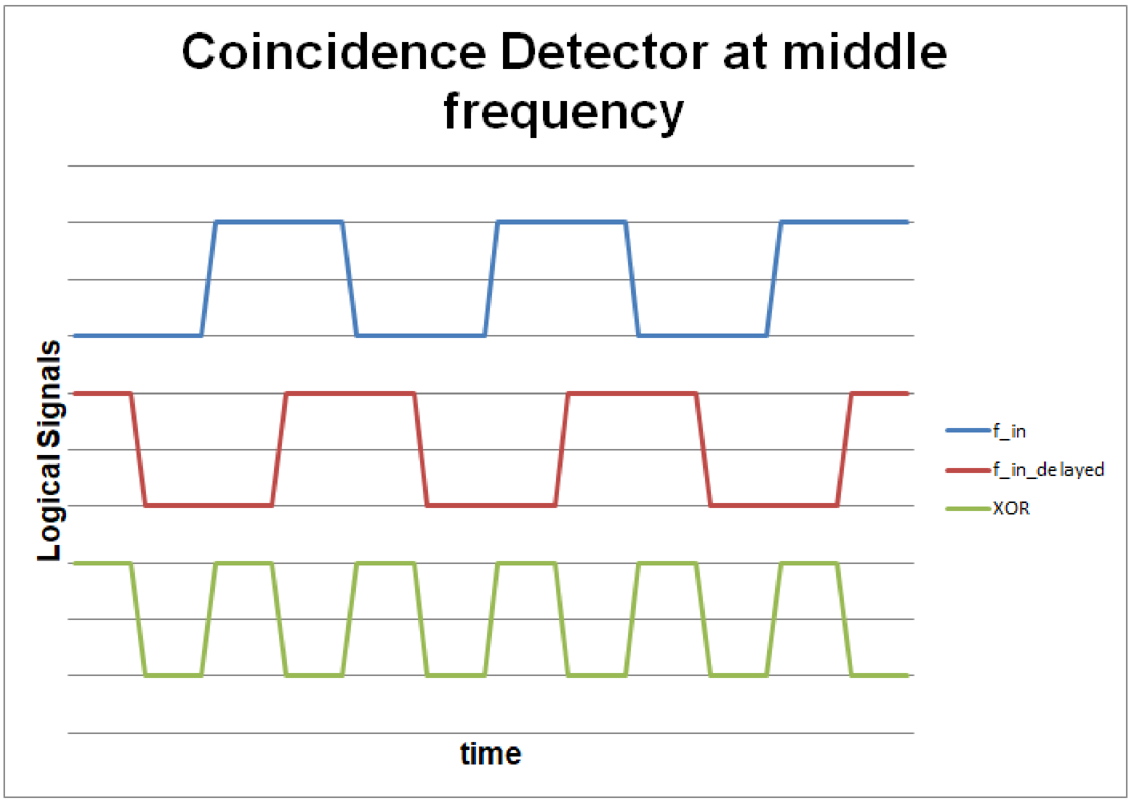


Figure 2.19: Coincidence-Demodulator at middle frequency input

For the highest possible input frequency the output of the Coincidence-Demodulator is a constant 1 like in Figure 2.20.

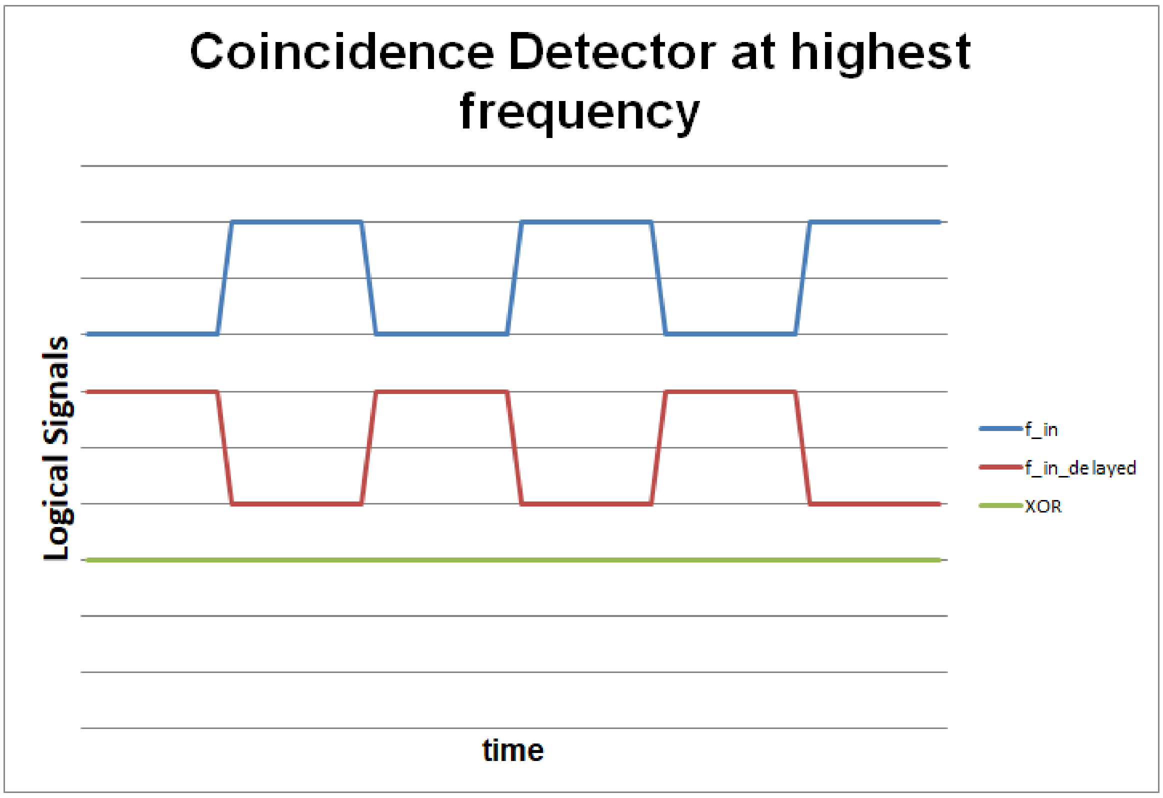


Figure 2.20: Coincidence-Demodulator at highest frequency input

To summarize this, the output of the coincidence detector is, apart of the both extreme cases, a pulse width modulated signal. This signal serves as an input to the previous described slope detector. A counter replaces the previous used lowpass integrator and counts the samples between each impulse and with the aid of the sample and hold block the counted number keeps at the output until the next impulse. This series of sequenced counter values is the digital demodulated FM signal.

2.5 All Digital PLL - DPLL

PLLs, Phase Locked Loops, are control loops which can synchronize with the incoming signal by the aid of a local oscillator. It can be built analog, analog and digital mixed or completely digital and gets used in many fields like as an FM-Demodulator, frequency multiplier, carrier frequency synthesis at radio and television and synchronisation control for engines.[20] Figure 2.21 illustrates the general structure of the PLL.

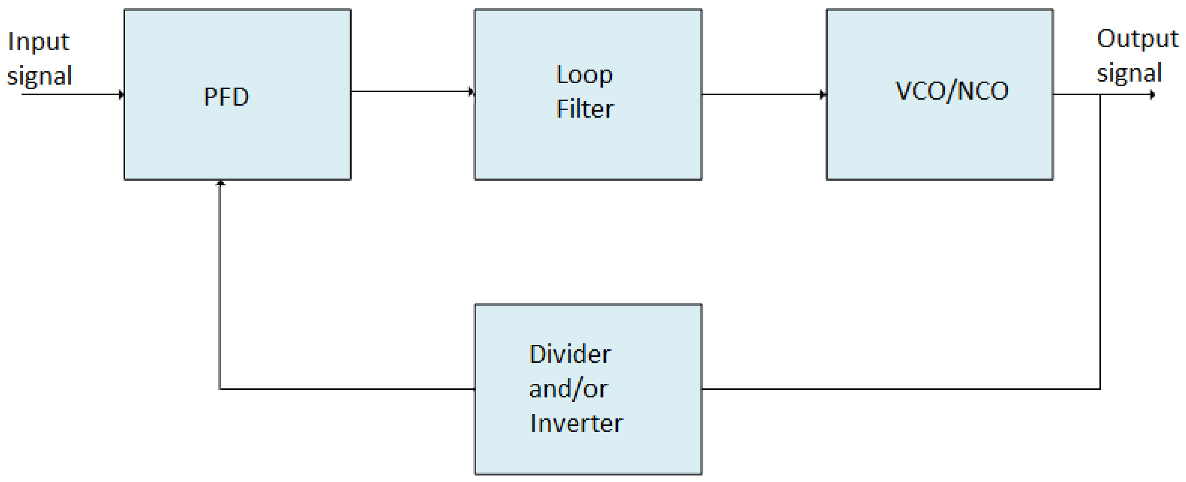


Figure 2.21: PLL Structure

PFD, Phase Frequency Detector, has two inputs, the incoming reference signal and the generated, optional modified, signal. The output is a signal which signals the phase or frequency shift between the both incoming signals. At control engineering it is typically called the control error signal.
LF, Loop Filter, is in general a lowpass filter and significant for the control loop stability. It characterises how the control loop reacts to changes in reference frequency or at start-up. The output signal is appointed to the VCO/NCO and signalises a raise drop in the output frequency.
VCO/NCO, Voltage/Numerical Controlled Oscillator, is the signal creating part of the control loop. A VCO gets used at analog PLLs and an NCO at digital PLLs and like the name already says the input voltage or number controls the output of the oscillator.
Divider and/or Inverter is an optional part in the feedback path of the control loop. If the Divider divides the frequency by two so the incoming

frequency to the PFD is half the reference frequency. The PFD issues an error signal so that the loop filter creates a signal that the VCO/NCO has to raise the frequency. This happens until the output frequency is double the input frequency. So if the double frequency of the output gets divided by two the PFD compares two signals with the same frequency again and the error is zero.

To exemplify this mathematically without neither a divider not an inverter:[21]
The PFD corresponds mathematically to

$$e(s) = (x(s) - y(s))K_d \tag{2.10}$$

s: signals frequency domain of Laplace transforms
$e(s)$: Laplace transform of the error signal after the PFD
$x(s)$: Laplace transform of input signal
$y(s)$: Laplace transform of output signal
K_d: coefficient of the PFD

$$G(s) = \frac{K_d K_o F(s)}{s} \tag{2.11}$$

$G(s)$: Laplace transform of PFD+LF+VCO/NCO
K_o: coefficient of the VCO/NCO
$F(s)$: Laplace transform of LF

$$\frac{y(s)}{x(s)} = \frac{G(s)}{1 + G(s)} \tag{2.12}$$

$$F(s) = K_p + \frac{K_I}{s} \tag{2.13}$$

K_p: proportional coefficient of the LF

K_I: integral coefficient of the LF

If these equations get combined it results

$$G(s) = \frac{K_d K_o K_p s + K_d K_o K_I}{s^2} \tag{2.14}$$

$$\frac{e(s)}{x(s)} = \frac{s^2}{s^2 + K_d K_o K_p s + K_d K_o K_I} == \frac{s^2}{s^2 + 2\xi\omega s + \omega^2} \tag{2.15}$$

ξ: damping of a 2nd order system

ω: natural angular frequency of a 2nd order system

$$\xi = \frac{K_p}{2}\sqrt{\frac{K_o K_d}{K_I}} \tag{2.16}$$

$$\omega = \sqrt{K_o K_d K_I} \tag{2.17}$$

Due to the PLL should be an All Digital PLL the equations have to be time discrete. A Bilinear Transformation has to get performed. The Euler method takes place for that because it is simpler and the sampling frequency will be much higher than the mirror and DPLL frequency.

$$s = (1 - z^{-1})\frac{1}{T_s} \tag{2.18}$$

z: signals frequency domain of Z transforms

T_s: sampling time

$$F(s)|_{s=(1-z^{-1})\frac{1}{T_s}} = \frac{K_p - K_p z^{-1} + K_I T_s}{1 - z^{-1}} = \frac{\alpha + \beta z^{-1}}{1 - z^{-1}} \tag{2.19}$$

Invention of two new variables for the LF.

$$\alpha = K_p + K_I T_s = \frac{\omega(\omega T_s - 2\xi)}{K_o K_d} \tag{2.20}$$

$$\beta = -K_p = \frac{\omega 2\xi}{K_o K_d} \tag{2.21}$$

To develop a DPLL the damping and the natural angular frequency have to get chosen and the coefficients of the PFD and NCO have to be set. At the MATLAB Chapter 3.4 the coefficients get chosen and the DPLL developed.

2.6 MATLAB and Simulink

MATLAB is a software product of the MathWorks company to solve and visualize mathematical problems. It works on several operating systems and is extendible with diverse toolboxes. The name MATLAB combines the two words MATrix and LABoratory because MATLAB's calculations base on matrices. The software combines numerical analyses, matrix calculation, signal processing and visualisation. It comprises an interactive programming language which works object-oriented. It is possible to call

MATLAB functions from other programming languages like Fortran and C and vice verse. The most famous and used toolbox of MATLAB is Simulink.[22]

The software **Simulink** is also a product of the MathWorks company and makes it possible to create graphical function blocks which can be simulated with or without an additional MATLAB script but needs the MATLAB program environment. It is possible to create technical, physical or theoretical mathematical continuous- or discrete-time systems and to simulate them. Simulink describes the created graphical function blocks by linear, non-linear and differential equations. The usage of Simulink is in several categories like signal processing or control loops possible and Simulink models can be exported to other languages like SystemC or VHDL.[23] These are the benefits for this master's thesis.

2.7 VHDL and Synthesis

The name VHDL is a combination of the letter V which stands for VHSIC, Very High Speed Integrated Circuit, and HDL, Hardware Description Language. Besides Verilog, VHDL is one of both mostly used HDLs. VHDL got developed by proxy of the US Department of Defense as a hardware documentation standard in the early 1980s and got standardised as IEEE 1076-87 in 1987.[24] The functional principle of VHDL is to describe the behaviour of a circuit and this in a higher level of abstraction. Hence it is possible to develop extensive circuits like microprocessors and to simulate, synthesis and verify them. Synthesis is a refinement process that realizes a description with components from the lower abstraction level. Also it is possible to produce net lists which can be directly loaded into FPGAs. The code can be separated between synthesis-able and functional code. The difference is that both codes can be simulated but just synthesis-able code can get translated to net lists. Functional code often gets used at testbenches to create a simulation environment for the proper VHDL code. To create synthesis-able code is in general harder and the developer has to abstain from big parts of the possible VHDL code. A synthesis-able VHDL code needs at least the two design units entity and architecture. The entity is the description of the ports and the architecture describes the behaviour of a VHDL unit.[25]

3 Matlab-Simulink Model

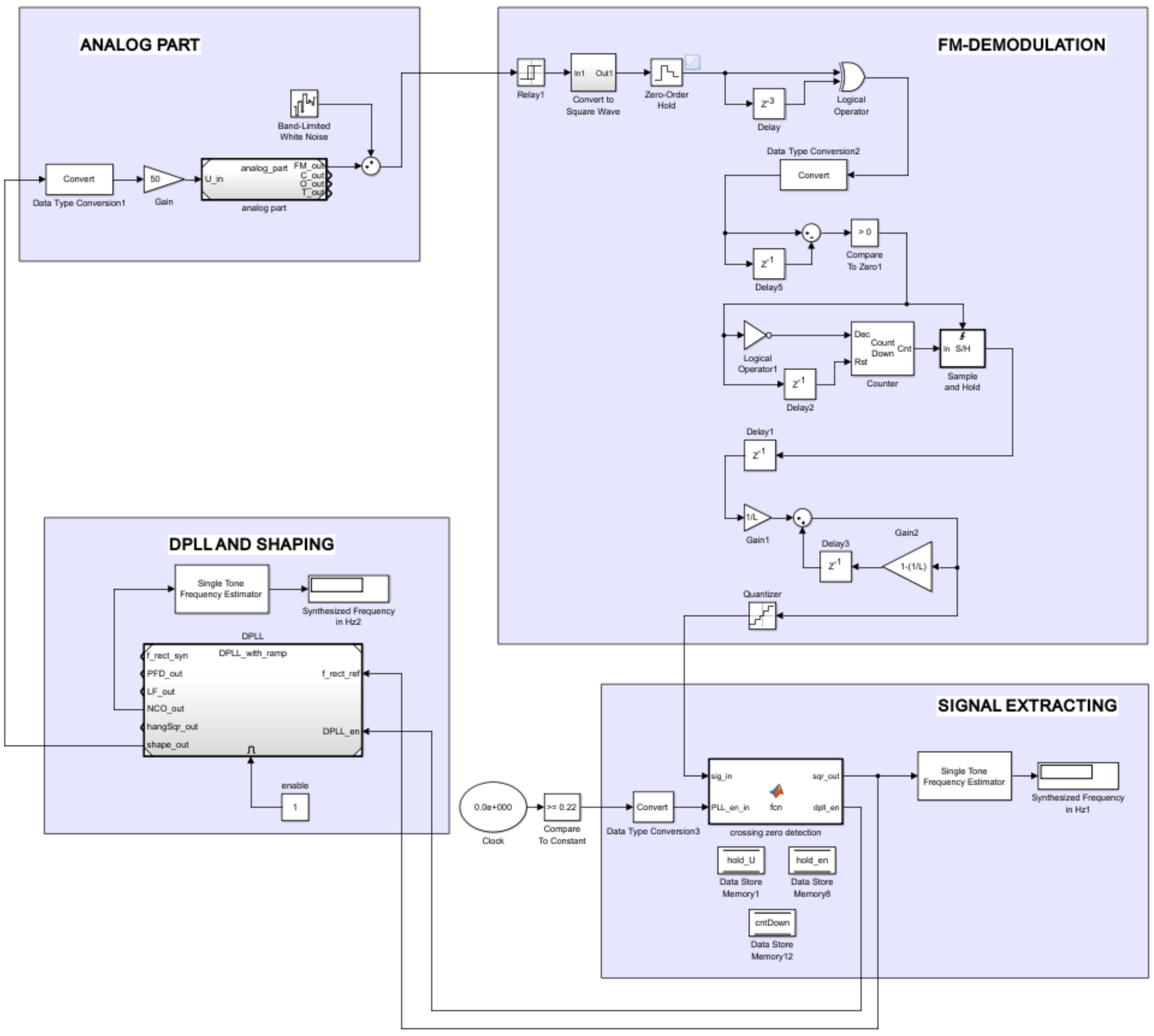


Figure 3.1: Simulink model of whole control model

```
fs_sim = 84e6;
Ts_sim = 1/fs_sim;
T_sim = 0.3;
```

Listing 3.1: MATLAB Parameter for Simulation

Figure 3.1 shows the final version of the MEMS-Mirror control loop. In this chapter some former versions of the analog part as well of the FM-Demodulator will be described and the reasons why they got changed will be listed.

The final version of the control loop got quantized to check functionality even with fixed bit length and to check SNR by adding white noise to the FM signal of the MEMS-mirror. This will be described in Chapter 3.6.

3.1 Analog Part

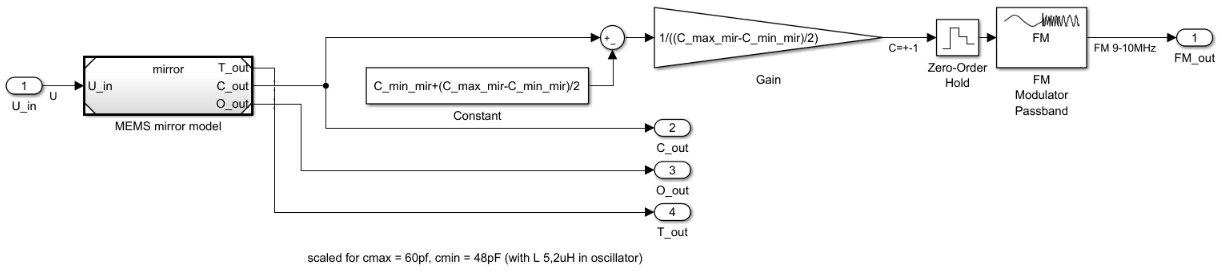


Figure 3.2: Simulink model of analog part 1st version

The Analog Part includes the MEMS Scanning Mirror, which is described at Chapter 3.1.1 and the FM-Modulation part. The mirror provides the changing capacitance which is representative for the current mirror angle. This capacitance gets scaled to a value between -1 and +1 by the minimum and maximum possible capacitance of the mirror. This by Equation 3.1 scaled capacitance value gets to an FM-Modulator block, Figure 3.4, and the output is an FM modulated signal between 9 MHz and 10 MHz.

$$C_{scaled} = \frac{C\_out - C\_min\_mir - \frac{C\_max\_mir - C\_min\_mir}{2}}{\frac{C\_max\_mir - C\_min\_mir}{2}} \tag{3.1}$$

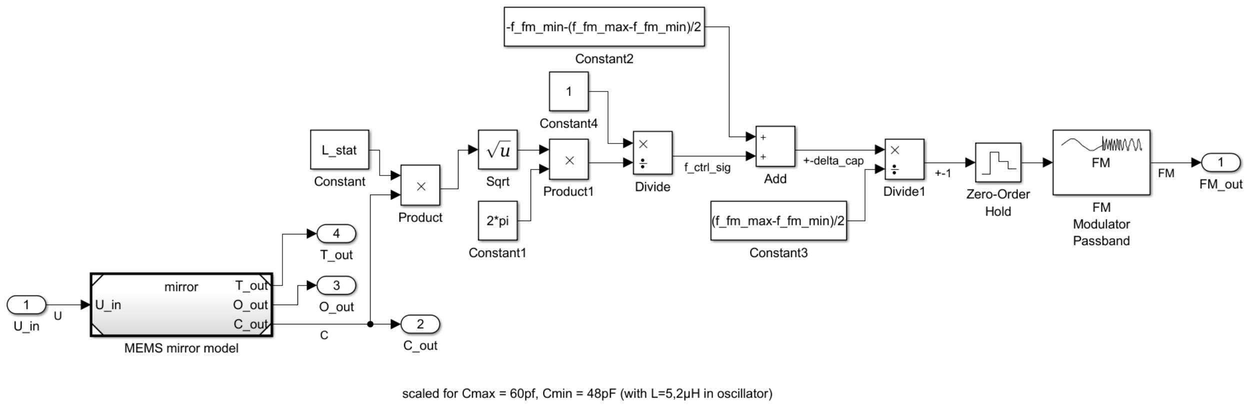
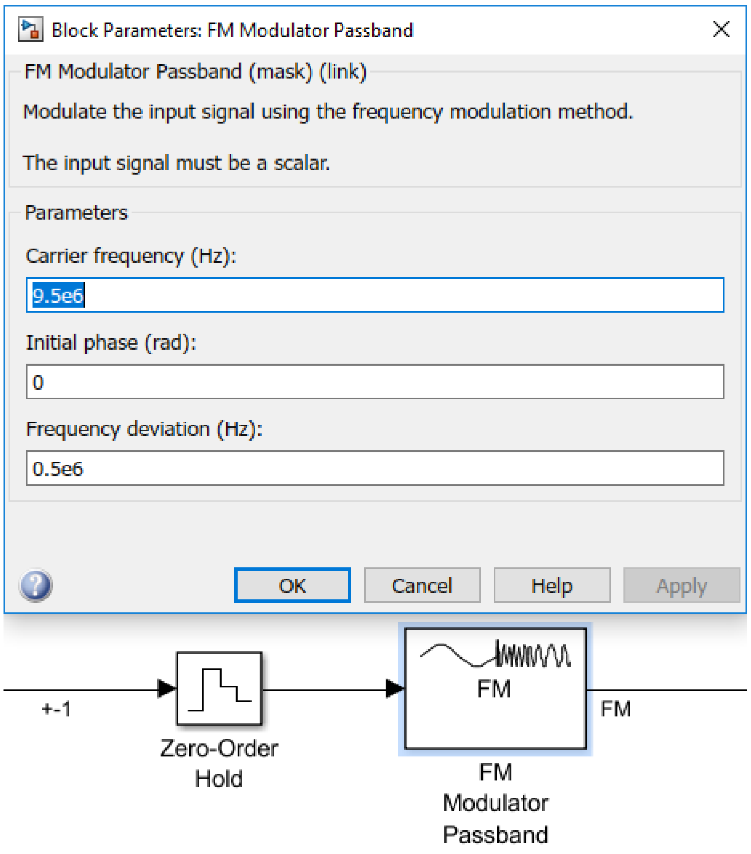


Figure 3.3: Simulink model of analog part 2nd version

In the second version of the analog part the output capacitance of the mirror gets more realistic FM modulated by an resonant circuit, Equation 3.2. For the sake of simplicity this FM modulated signal gets scaled again, Equation 3.3, and FM modulated to ensure a frequency range of 9 MHz to 10 MHz.

$$f\_ctrl\_sig = f_0 = \frac{1}{2\pi\sqrt{L\_statC\_out}} \tag{3.2}$$

$$f\_ctrl\_sig_{scaled} = \frac{f\_ctrl\_sig - f\_fm\_min - \frac{f\_fm\_max - f\_fm\_min}{2}}{\frac{f\_fm\_max - f\_fm\_min}{2}} \tag{3.3}$$



Block Parameters: FM Modulator Passband

FM Modulator Passband (mask) (link)

Modulate the input signal using the frequency modulation method.

The input signal must be a scalar.

Parameters

Carrier frequency (Hz): 9.5e6

Initial phase (rad): 0

Frequency deviation (Hz): 0.5e6

Figure 3.4: Simulink Block FM-Modulator Passband

```
% analog sampling - 84MHz because of FM
fs_mir = fs_sim; %[84MHz]
Ts_mir = 1/fs_mir;

% frequency modulation
L_stat = 10e-3; %[2µH]
f_fm_max = (2 * pi) / sqrt(L_stat * cap_min); % at 18°
f_fm_min = (2 * pi) / sqrt(L_stat * cap_max); % at 0°
```

Listing 3.2: MATLAB Code for Analog Part

Figure 3.5 shows the incoming voltage at 100 V and a frequency of 2 kHz. This produces the mirror movement and the following change of capacitance in Figure 3.6. These two figures compared illustrate that the incoming voltage is not exactly timed with the peak-reaching - voltage on - and 0° crossing - voltage off - mirror alignment like in Chapter 2.2 exactly described. This not exact timing is caused through the delay which the whole control loop creates. To compensate this delay the DPLL has an additional delay block which will be described in Chapter 3.4. The out coming FM modulated signal is illustrated at Figure 3.7 which is zoomed to a smaller part right before the capacitance reaches the peak or the mirror crosses 0°.

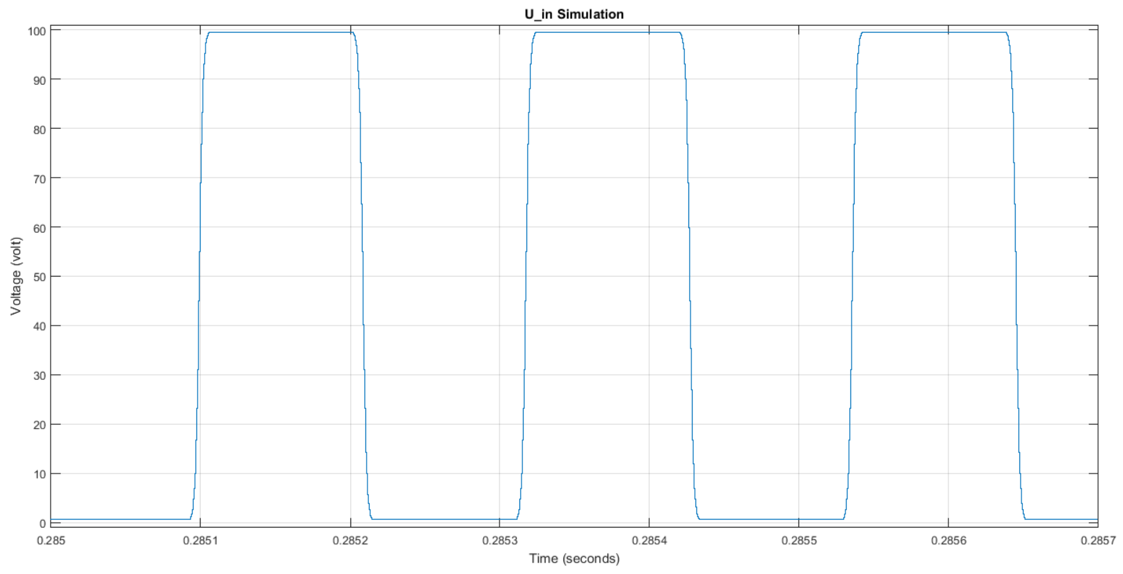


Figure 3.5: U_in Simulation of MEMS-Mirror

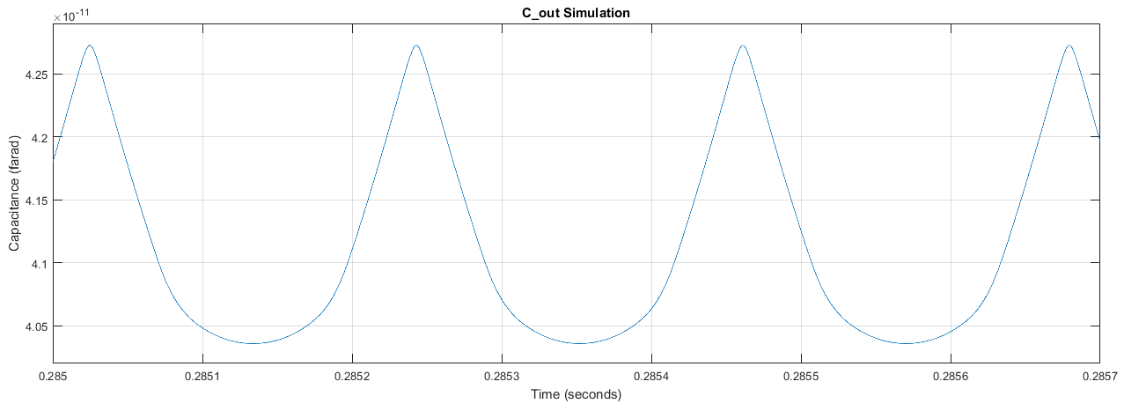


Figure 3.6: C_out Simulation of MEMS-Mirror

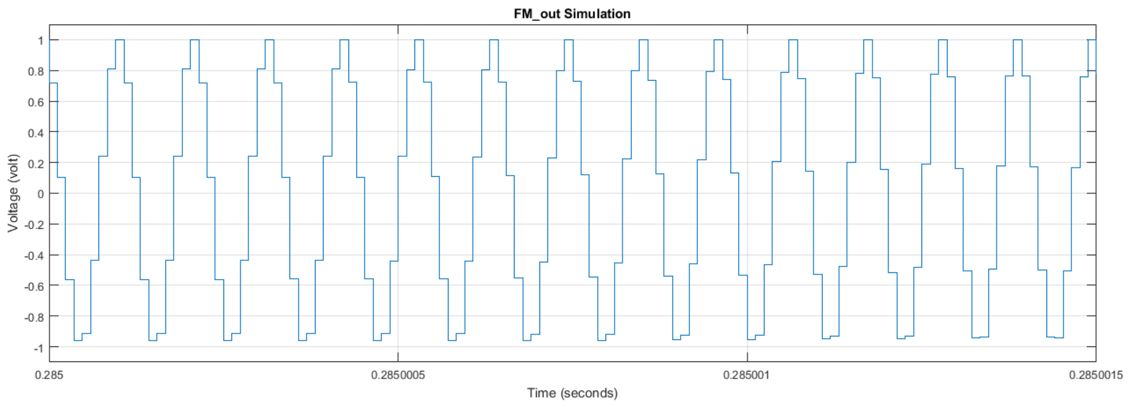


Figure 3.7: FM_out Simulation of MEMS-Mirror

3.1.1 Mirror

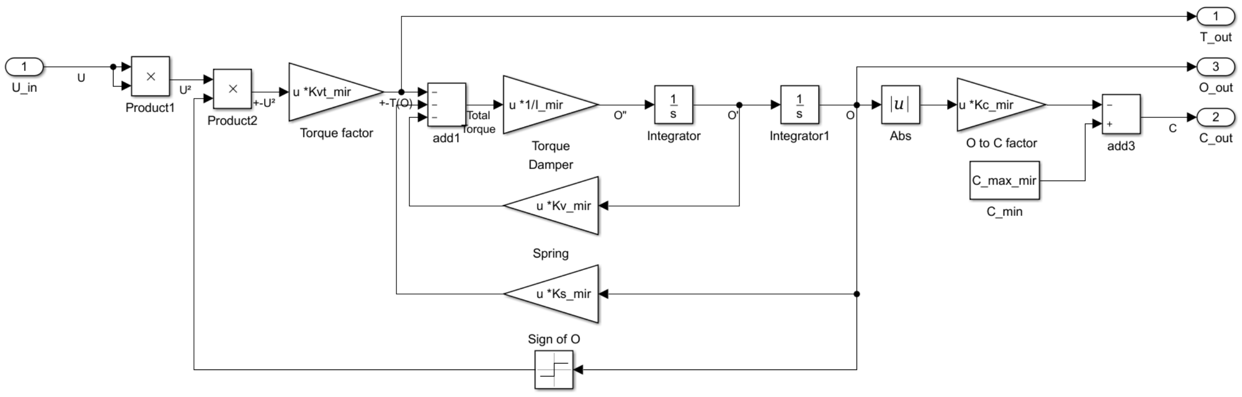


Figure 3.8: Simulink model of mirror 1st version

```
% mirror main
f_mir = 2e3; %[f = 2kHz]
wn_mir = 2*pi*f_mir;
D_mir = 0.004; %chosen

C_min_mir = 3*10^-12; %[3pF]
C_max_mir = 60*10^-12; %[60pF]
O_mir = 13.6; %[°]

Kc_mir = ((C_max_mir-C_min_mir)/O_mir); %[F/°]
Kvt_mir = 13/2500; %chosen for angle 13.6°
K_mir = 1;
```

```
I_mir = 1/(K_mir*wn_mir^2);
Kv_mir = 2*I_mir*D_mir*wn_mir;
Ks_mir = I_mir*wn_mir^2;
```

Listing 3.3: MATLAB Code for Mirror 1st version

Since at the beginning of the thesis the MEMS-mirror was unexplored a simple mass-spring-damper model got used as the first version of it, Figure 3.8. In Listing 3.3 the mirror's parameters get calculated by the given and chosen values of the resonant frequency and the damper constant. This calculation is described in Chapter 2.2. The value of the voltage-to-torque factor Kvt_mir got identified experimentally to achieve a maximum angle of 13.6° by an input voltage of 50 V. To guarantee that the mirror just gets pulled by the torque the squared input voltage gets multiplied by the sign of the angle. To calculate the resulting capacitance of the mirror the absolute value of the angle gets multiplied by the degree-to-farad factor Kc_mir and this value gets subtracted by the maximum value of the mirror capacitance C_max_mir. This is because the maximum capacitance is at 0° and it gets symmetrically less the more it moves in positive or negative alignment.

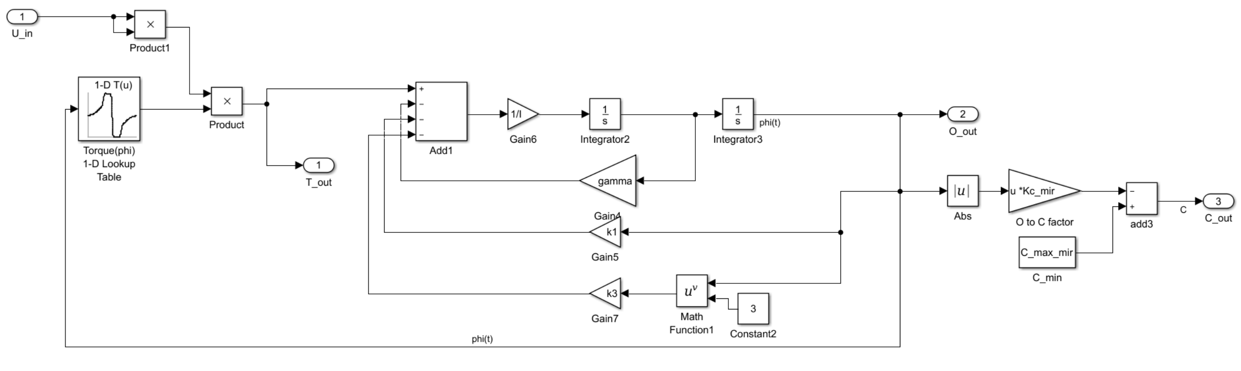


Figure 3.9: Simulink model of mirror 2nd version

```
% mirror main
k1 = 58e-6;
k3 = 360e-6;
I = 3.19e-13;
omega_0 = sqrt(k1/I);
Q = 150;
gamma = omega_0/Q*I;

% Initialize actuation lookup
% torque scaled for 1V input in Nm
data = xlsread('torque_lookup_1V_in_Nm','Sheet1');
% Allocate imported array to column variable names
angle_value = data(:,1);
torque_value = data(:,2);

% capacitance calculation
C_min_mir = 3*10^-12; %[3pF]
C_max_mir = 60*10^-12; %[60pF]
O_mir = 13.6; %[°]
Kc_mir = ((C_max_mir-C_min_mir)/O_mir); %[F/°]
```

Listing 3.4: MATLAB Code for Mirror 2nd version

After the mirror got exactly analysed by creating a finite element model of it and starting some simulations, reality true values for its parameter got discovered and got placed in the second version like in Figure 3.9. Additionally, to the new values for gamma, k1 and I, the mirror has the new parameter k3 to describe its non-linearity. Also this non-linearity and its consequences are exactly described at Chapter 2.2. The torque factor got replaced by a lookup table which describes the torque value by the mirror alignment in degree like in Figure 3.12. In this version of the mirror the capacitance calculation remained unchanged.

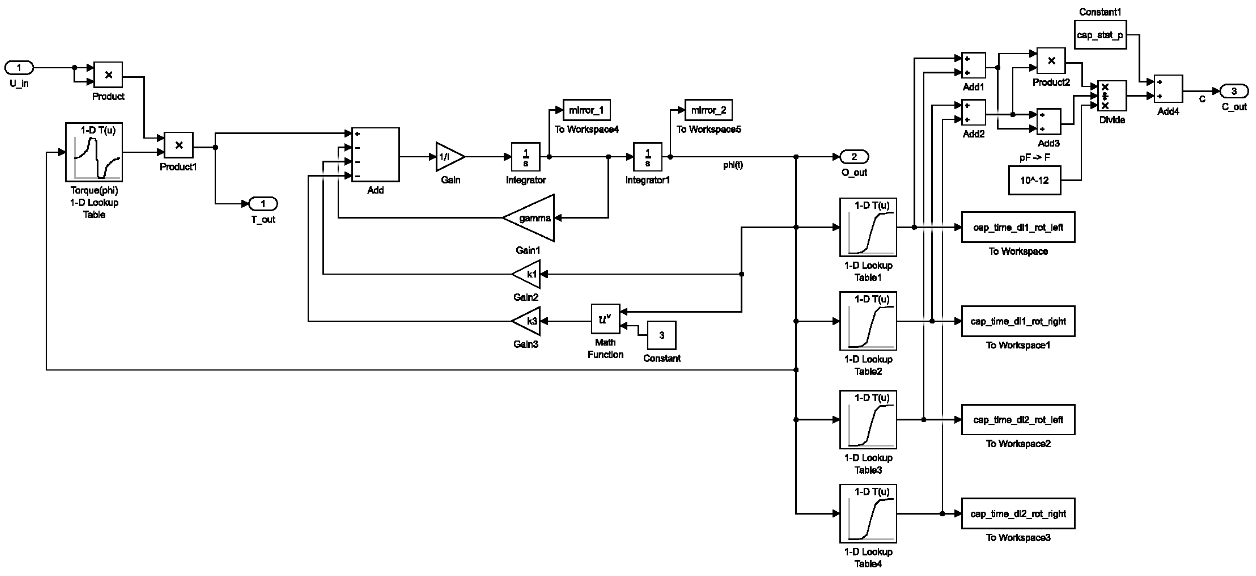


Figure 3.10: Simulink model of mirror 3rd version

```
% mirror main
f_mir = 2e3; %[mirror resonance frequency about 2kHz]
wn_mir = 2*pi*f_mir;

k1 = 58e-6;
k3 = 360e-6;
I = 3.19e-13;
omega_0 = sqrt(k1/I);
Q = 150;
gamma = omega_0/Q*I;

% Initialize actuation lookup
% torque scaled for 1V input in Nm
data = xlsread('torque_lookup_1V_in_Nm','Sheet1');
% Allocate imported array to column variable names
angle_value = data(:,1);
torque_value = data(:,2);
```

```
%Clear temporary variables
clearvars data raw;

% Initialize capacitance
delta_angle = 0.001; % degrees
cap_stat_p = 40e-12; %[40pF]
file_name_caps = 'caps_interp.mat';
if ~exist(file_name_caps,'file')
    data2 = xlsread('DL1-DL2-rotor-caps.xlsx','Sheet1');
    angle_deg = data2(:,1);
    angle_vec_long = -18:delta_angle:18;
    cap_stat_dl1_rot_left_tmp = data2(:,5);
    cap_stat_dl2_rot_left_tmp = data2(:,4);
    cap_stat_dl1_rot_left = ↩
          interp1(angle_deg,cap_stat_dl1_rot_left_tmp,angle_vec_long,'cubic');
    cap_stat_dl2_rot_left = ↩
          interp1(angle_deg,cap_stat_dl2_rot_left_tmp,angle_vec_long,'cubic');
    cap_stat_dl1_rot_right = fliplr(cap_stat_dl1_rot_left);
    cap_stat_dl2_rot_right = fliplr(cap_stat_dl2_rot_left);
    save(file_name_caps,'angle_vec_long','cap_stat_dl1_rot_left', ↩
          'cap_stat_dl1_rot_right','cap_stat_dl2_rot_left', ↩
          'cap_stat_dl2_rot_right');
else
    load(file_name_caps);
end
sum_cap = cap_stat_dl1_rot_left + cap_stat_dl1_rot_right + ↩
      cap_stat_dl2_rot_left + cap_stat_dl2_rot_right;
angle_vec_long_rad = angle_vec_long/180*pi;

c_l = cap_stat_dl1_rot_left + cap_stat_dl2_rot_left;
c_r = cap_stat_dl1_rot_right + cap_stat_dl2_rot_right;
cap_all = ((c_l.*c_r)./(c_l + c_r)).*(10^-12) + cap_stat_p;
cap_max = max(cap_all);
cap_min = min(cap_all);
```

Listing 3.5: MATLAB Code for Mirror 3rd version

In the third version of the mirror the capacitance calculation got more exactly like in Figure 3.10. Through the finite element simulation of the mirror four lookup tables got created to describe the capacitance created by the mirror movement. Like in Listing 3.5 an Excel Document with the values gets loaded and the variables for the lookup tables get created. Followed by the geometrical construction this calculation describes the parallel and serial capacitances of the mirror and is described in Chapter 2.2. This final version of the MEMS-mirror includes its non-linearity, the reality true torque depending on the angle and the five geometrical depending capacitances depending on the angle.

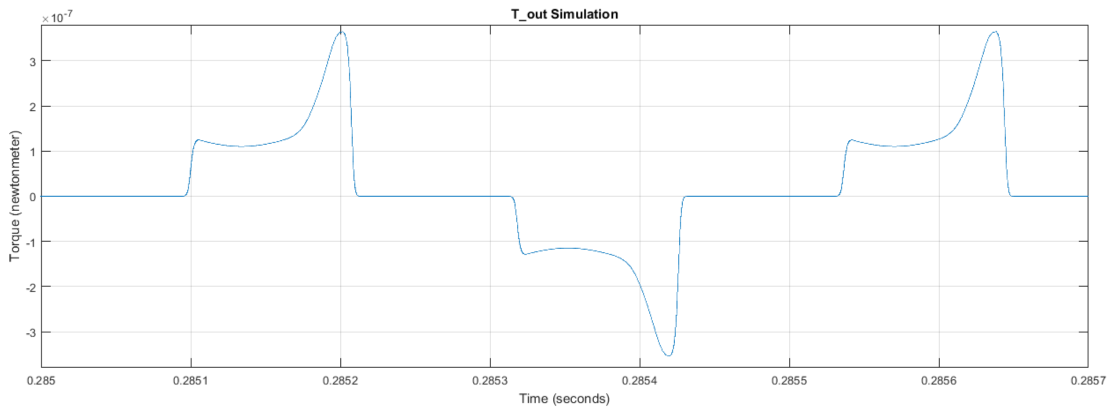


Figure 3.11: T_out Simulation of MEMS-Mirror

Figure 3.11 shows the torque produced by the input voltage of Figure 3.5 and the lookup table of Figure 3.12.

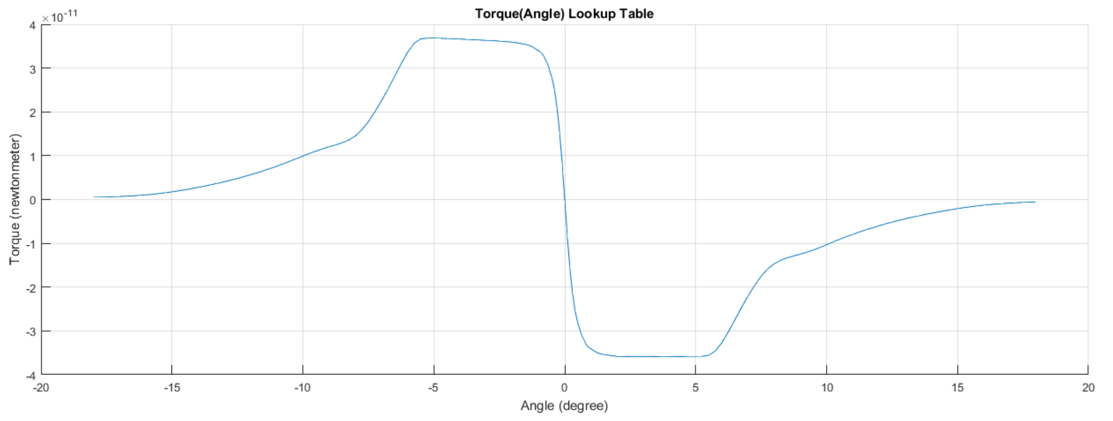


Figure 3.12: Torque(Angle) Lookup Table of MEMS-Mirror

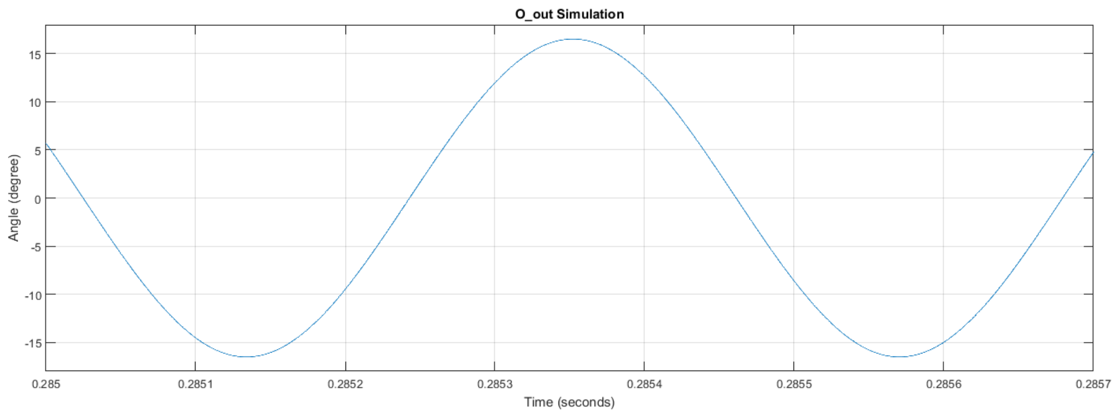


Figure 3.13: O_out Simulation of MEMS-Mirror

Through the applied torque and the second-order mirror model the mirror angle behaves like in Figure 3.13. It is clear to see that a positive applied torque pulls the mirror from negative angle to 0° and vice versa.

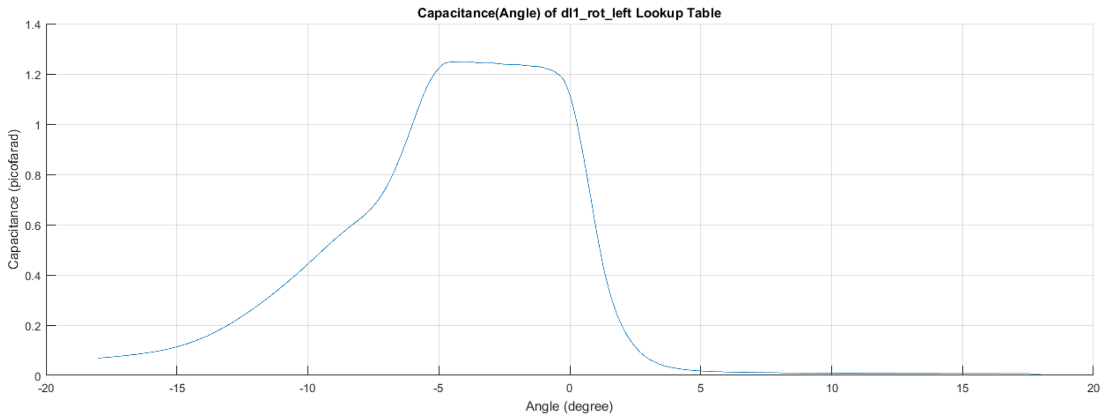


Figure 3.14: Capacitance(Angle) of dl1_rot_left Lookup Table of MEMS-Mirror

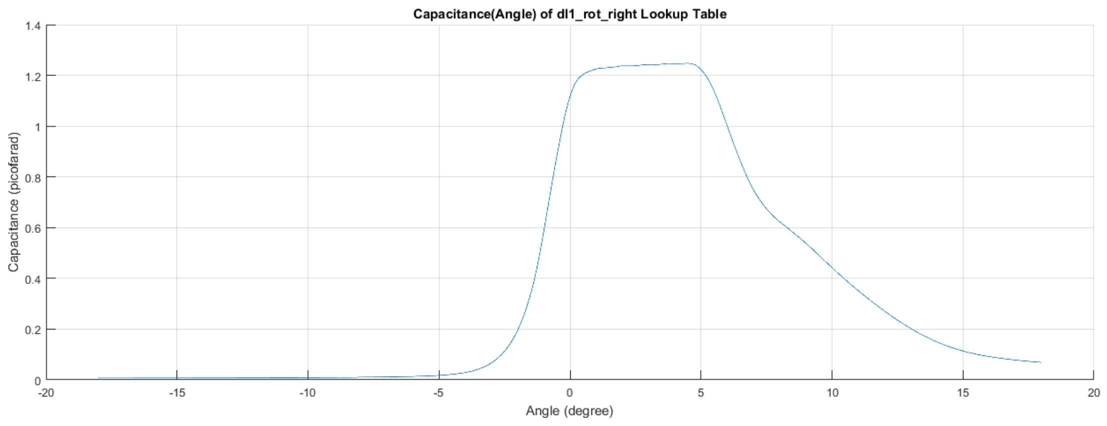


Figure 3.15: Capacitance(Angle) of dl1_rot_right Lookup Table of MEMS-Mirror

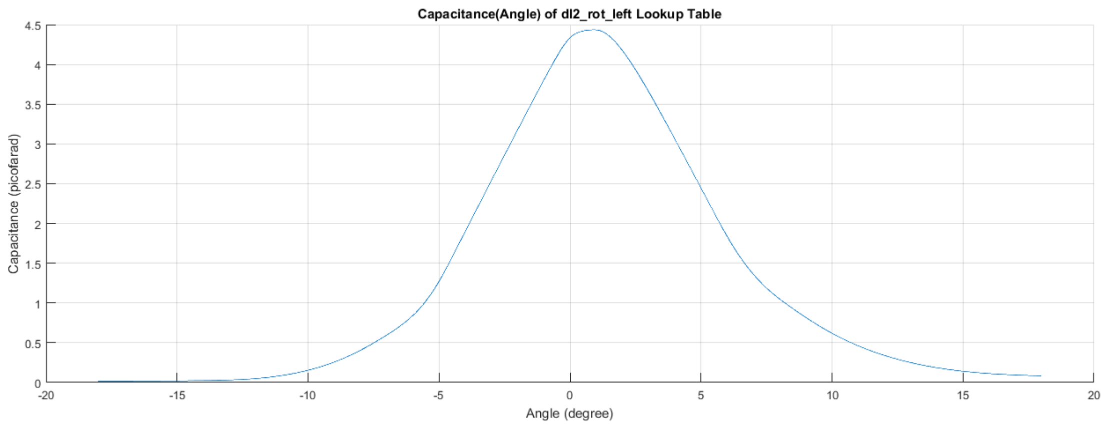


Figure 3.16: Capacitance(Angle) of dl2_rot_left Lookup Table of MEMS-Mirror

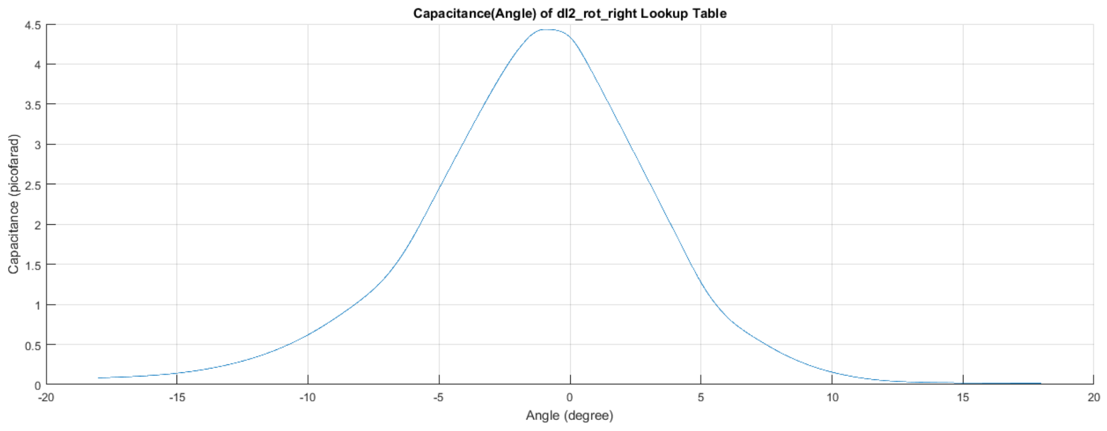


Figure 3.17: Capacitance(Angle) of dl2_rot_right Lookup Table of MEMS-Mirror

Figure 3.14 to Figure 3.17 show the lookup tables for the capacitances depending on the angle which produce summed to the static capacitance

cap_stat_p the resulting capacitance C_out in Figure 3.6.
This shows us that the capacitance signal is double the frequency, ~4 kHz, of the mirror, ~2 kHz.

3.2 FM-Demodulation

The FM-Demodulation is the most difficult part of the control loop because it has to be simple and realizable by using a low amount of fix bits. In this chapter all three used FM-Demodulation methods will be described and the reasons why the third and not the previous two methods got used to demodulate the signal.

3.2.1 FM-Demodulator 1st version - PLL-Demodulator

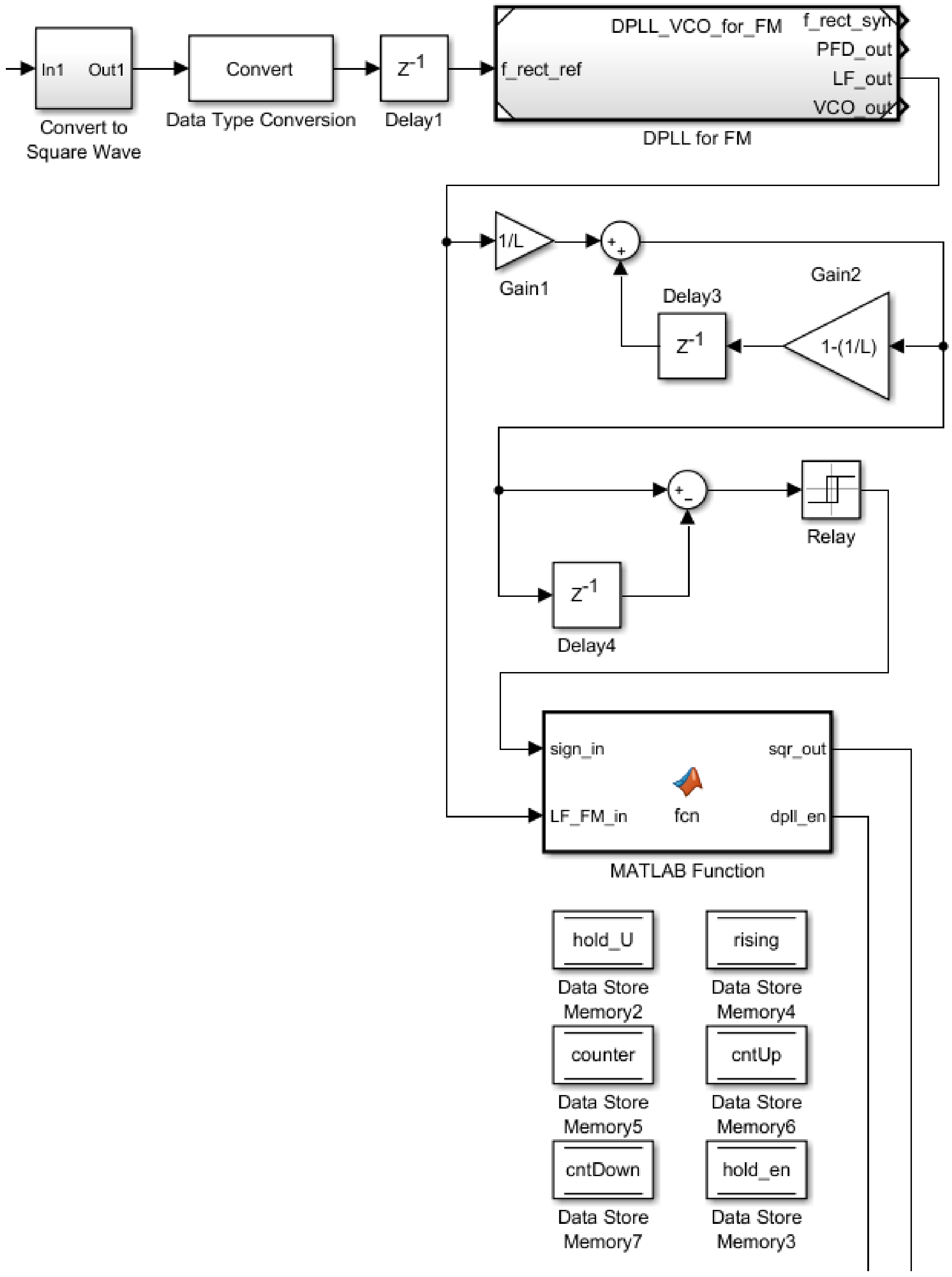


Figure 3.18: Simulink model of FM-Demodulator 1st version

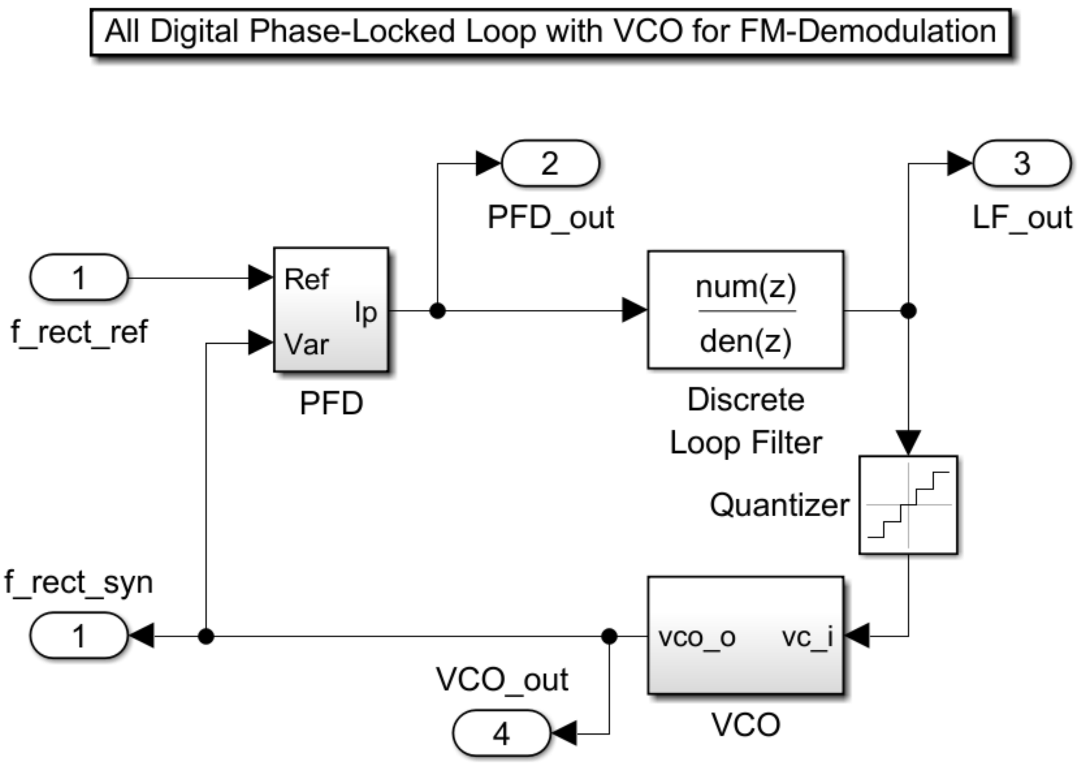
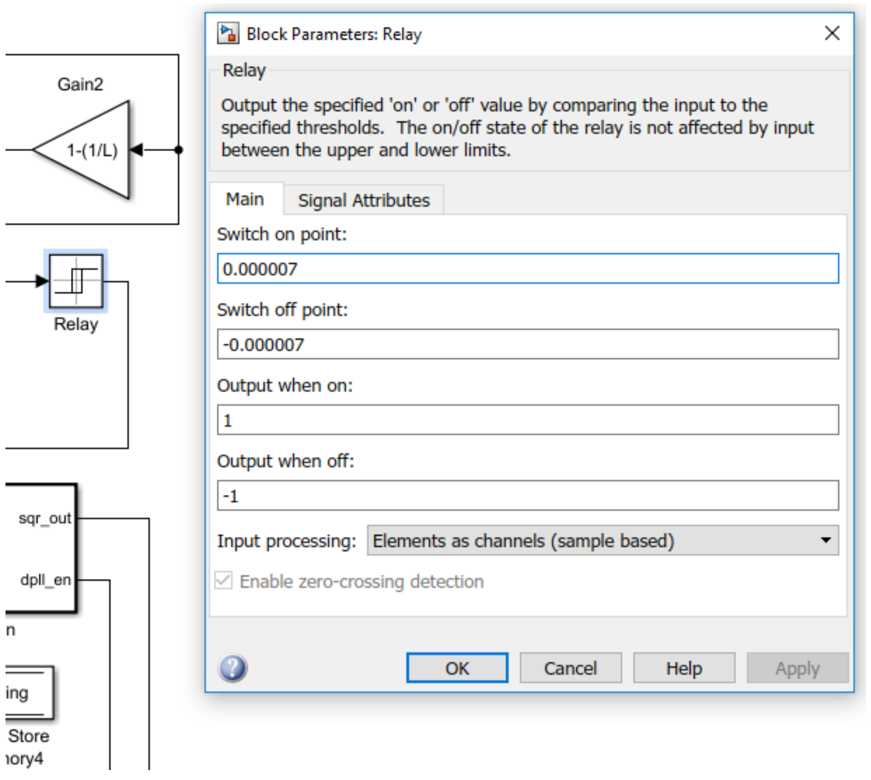


Figure 3.19: Simulink model of DPLL of FM-Demodulator 1st version



Block Parameters: Relay

Relay

Output the specified 'on' or 'off' value by comparing the input to the specified thresholds. The on/off state of the relay is not affected by input between the upper and lower limits.

Main | Signal Attributes

Switch on point: 0.000007

Switch off point: -0.000007

Output when on: 1

Output when off: -1

Input processing: Elements as channels (sample based)

[✓] Enable zero-crossing detection

OK Cancel Help Apply

Figure 3.20: Simulink Block Relay of FM-Demodulator 1st version

```
function [sqr_out, dpll_en]  = fcn(sign_in, LF_FM_in)
    global rising; % initial value rising = 1
    global counter; % initial value counter = 0
    global cntUp; % initial value cntUp = 0
    global cntDown; % initial value cntDown = 0
    global hold_U; % initial value hold_U = 0
    global hold_en; % initial value hold_en = 0

    if (rising == 1)
        counter = counter + 1;
        if (sign_in == 1)
            cntUp = cntUp + 1;
        else
            cntDown = cntDown + 1;
        end
        if (cntUp == 64)
            counter = 0;
            cntUp = 0;
            cntDown = 0;
        end
        if (counter == 512)
            rising = -1;
            counter = 0;
            cntUp = 0;
            cntDown = 0;
        end
    end
    if (rising == -1)
        counter = counter + 1;
        if (sign_in == -1)
            cntDown = cntDown + 1;
        else
            cntUp = cntUp + 1;
        end
        if (cntDown == 64)
            counter = 0;
            cntDown = 0;
            cntUp = 0;
        end
        if (counter == 512)
            rising = 1;
            counter = 0;
            cntDown = 0;
            cntUp = 0;
        end
    end
    if (rising == 1)
        hold_U = 1;
    else
        hold_U = 0;
    end
    if ( LF_FM_in <= -0.005)
        hold_en = 1;
    end

sqr_out = hold_U;
dpll_en = hold_en;
```

Listing 3.6: MATLAB Function for Signal Extracting of FM-Demodulator 1st version

After the DPLL was created and the functionality of it proved the idea was to use this knowledge to create an FM-Demodulator with help of the loop filter of the DPLL like in Figure 3.18. Like in Chapter 2.5 described the loop filter follows the frequency change of the incoming FM square signal, Figure 3.21 and this resulting signal is representative for the FM capacitance signal, thus the demodulated FM signal. Figure 3.19 shows the DPLL which got used to demodulate the FM signal. This DPLL is not all digital yet because a VCO got used instead of an NCO. The signal extracting function from the first version got replaced by Simulink blocks. Firstly, the demodulated signal LF_out, Figure 3.22, gets through a moving average and after all the rising and falling edge gets detected. The divisor of the moving average should be a number of

$$L = 2^n | n \epsilon \mathbb{N} \tag{3.4}$$

to guarantee a right shift of a bit vector instead of a complicated division. Figure 3.23 shows the loop filter signal after the moving average. The Simulink Relay block, Figure 3.20, serves to filter some numerical disseminations of the rising/falling signal and to output a signal of +1 or -1 like in Figure 3.24. This sign signal serves as the input for the Signal Extracting

Matlab Function which is described at Listing 3.6. Because of the disseminations of the sign signal after the Relay Block the Matlab Function has a counter to guarantee a continuous rising of the demodulated FM signal. This counter produces a delay which gets compensated in the DPLL. This time compensation is described in the DPLL Chapter 2.5. A rising signal, rising == 1, produces an 1 and a falling signal, rising == -1, produces a 0 at the generated square signal sqr_out of Figure 3.25. The code also includes the dpll_en signal which gets 1 after around 0.2 seconds when the loop filter signal gets under -0.005 and this enables the DPLL and closes the control loop. Details about the start-up of the mirror and the closing of the control loop are findable at Chapter 2.5.

The result of the demodulated signal was good but after quantizing it became clear that the loop filter would need a lot of resources because the amount of fix bits has to be really high to work properly. Besides that the clock rate of the Chip has to be around 100 MHz to realize this FM-Demodulation method. So it was about to find a low resource working FM-Demodulator.

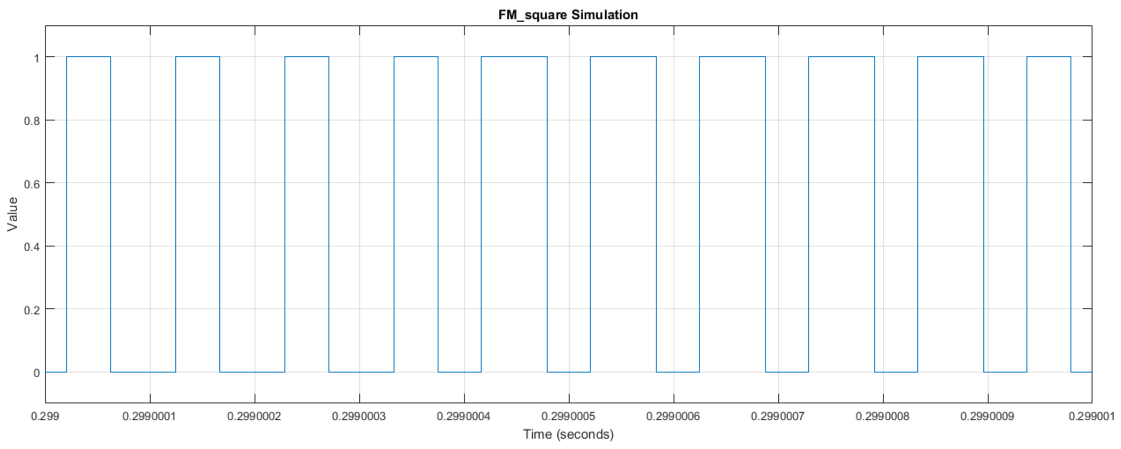


Figure 3.21: Incoming FM square Signal of FM-Demodulator 1st version

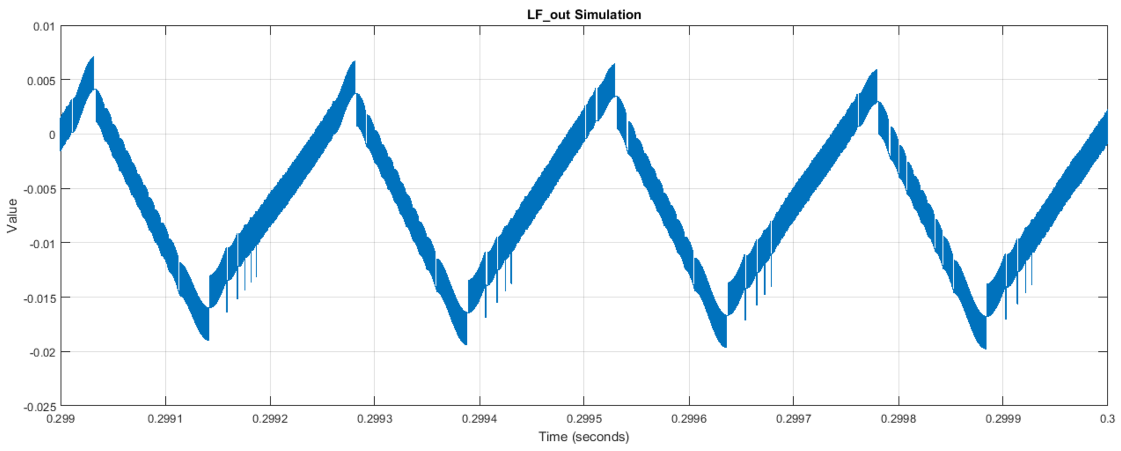


Figure 3.22: Loop Filter Signal of FM-Demodulator 1st version

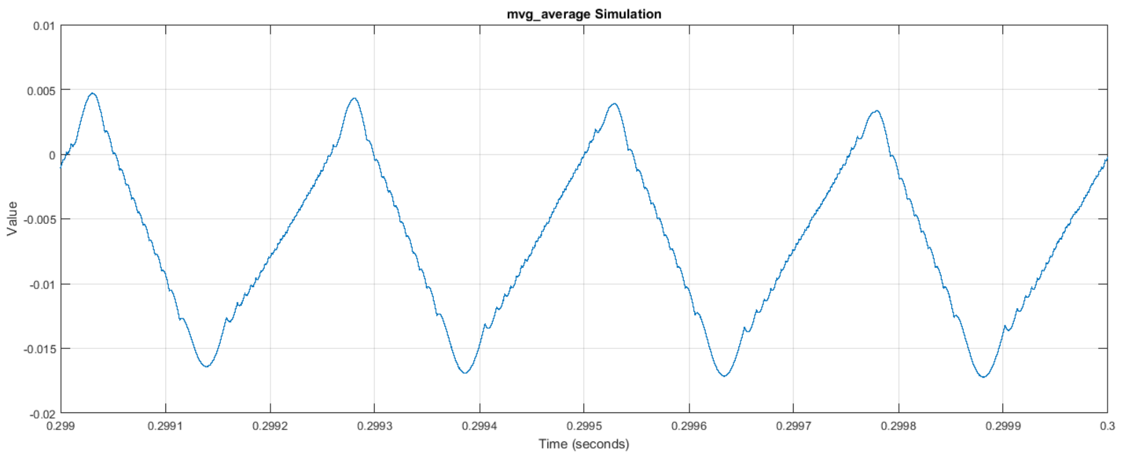


Figure 3.23: Moving Average Signal of FM-Demodulator 1st version

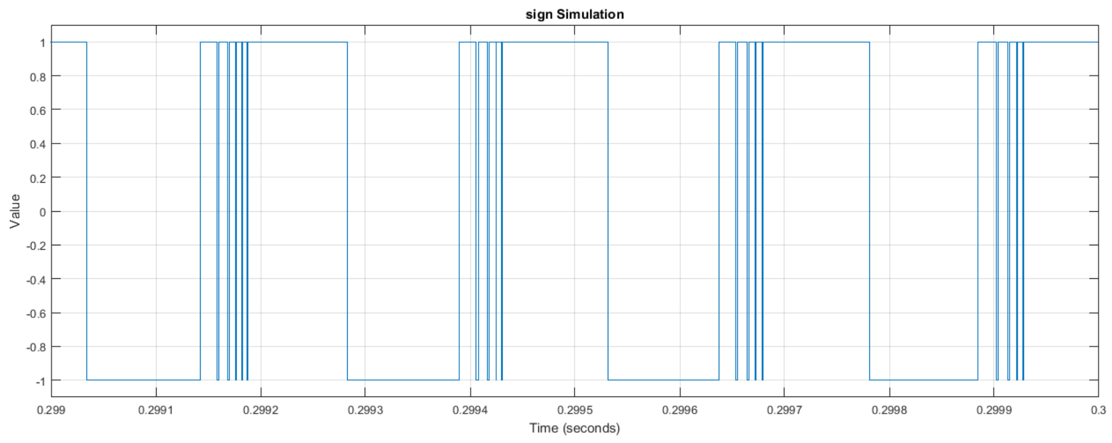


Figure 3.24: Sign Signal after Relay Block of FM-Demodulator 1st version

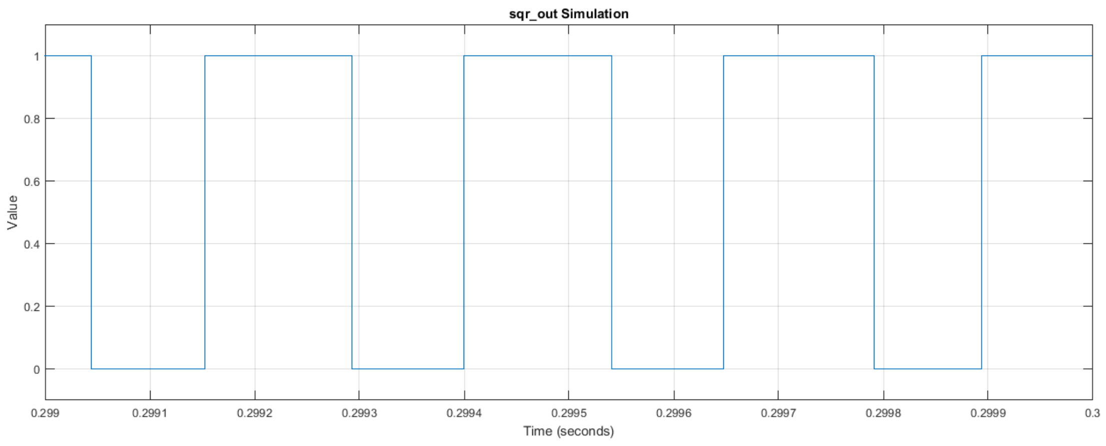


Figure 3.25: Generated Square Signal of Matlab Function of FM-Demodulator 1st version

3.2.2 FM-Demodulator 2nd version - Slope Detection Demodulator

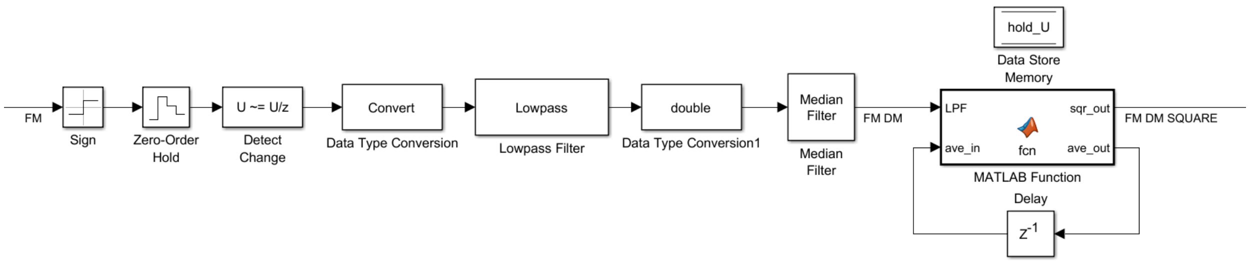
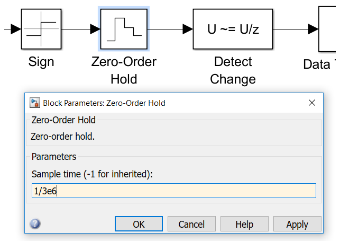


Figure 3.26: Simulink model of FM-Demodulator 2nd version



Block Parameters: Zero-Order Hold

Zero-Order Hold

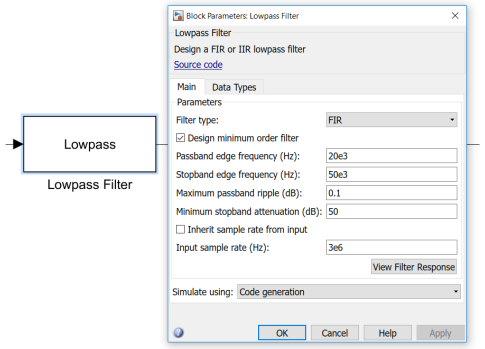
Zero-order hold.

Parameters

Sample time (-1 for inherited):

1/3e6

Figure 3.27: Simulink Block Zero-Order Hold of FM-Demodulator 2nd version



Lowpass Filter

Design a FIR or IIR lowpass filter

Source code

Parameters

Filter type: FIR

☑ Design minimum order filter

Passband edge frequency (Hz): 20e3

Stopband edge frequency (Hz): 50e3

Maximum passband ripple (dB): 0.1

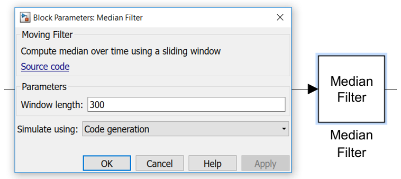
Minimum stopband attenuation (dB): 50

☐ Inherit sample rate from input

Input sample rate (Hz): 3e6

Simulate using: Code generation

Figure 3.28: Simulink Block Lowpass of FM-Demodulator 2nd version



Moving Filter

Compute median over time using a sliding window

Source code

Parameters

Window length: 300

Simulate using: Code generation

Figure 3.29: Simulink Block Median Filter of FM-Demodulator 2nd version

```matlab
function [sqr_out, ave_out] = fcn(LPF, ave_in)
    global hold_U; % initial value hold_U = 0
    T_tmp = hold_U;

    K = 1/16;
    tmp_ave = K*LPF + (1-K)*ave_in;

    if (tmp_ave < ave_in)
        T_tmp = 1;
    end
    if (tmp_ave > ave_in) || (LPF <= 0)
        T_tmp = 0;
    end
    hold_U = T_tmp;

sqr_out = T_tmp;
ave_out = tmp_ave;
```

Listing 3.7: MATLAB Function for Signal Extracting of FM-Demodulator 2nd version

The second version of the FM-Demodulator was a lowpass filter where the demodulated signal additionally got median filtered. For that the FM signal gets to a Sign Block at which the output is +1 for a positive, -1

for a negative and 0 for a 0 signal. Resumed the sinus FM signal becomes to a square signal from -1 to +1. This squared FM signal gets down sampled by the Zero-Order Hold Block at a frequency of 3 MHz, Figure 3.27 and these both blocks should represent an under-sampling ADC which transforms the signal from a higher to a lower frequency range like in Figure 3.30. The under-sampling principle is described at Chapter 2.4.2. The signal continues to the Detect Change Block where the output is a 1 when the value changes. So the under sampled square signal becomes to a rectangular pulse signal like in Figure 3.31. This pulse signal gets to the Lowpass Filter Block, Figure 3.28 and the resulting signal is a sinus similar signal which is finally the resulting FM demodulated signal like in Figure 3.32. To smooth the demodulated signal it gets filtered by a Median Filter Block, Figure 3.29 and this output signal, Figure 3.33, serves as an input of the Signal Extracting Matlab Function, Listing 3.7.
The Median Filter is a filter which replaces each value by the median of its surrounding values. The mathematical equation for a 2-dimensional array would be:

$$I'(u,v) \leftarrow median(I(u+i, v+j)|(i,j)\epsilon R) \tag{3.5}$$

The median of a sequence with $2n+1$ values a_i would be defined as

$$median(a_0, a_1, ..., a_n, ..., a_{2n}) = a_n \tag{3.6}$$

with a sorted sequence by $a_i \leqslant a_{i+1}$. For an even number of values the median of a sequence would be the arithmetic average:

$$median(a_0, ..., a_{n-1}, a_n, ..., a_{2n-1}) = \frac{a_{n-1} + a_n}{2} \tag{3.7}$$

This filter is a good method to remove disturbances.[26]

The Signal Extracting Matlab Function includes, apart of the signal extracting and output square signal producing, a moving average part to smooth the input signal a second time. The functional principle of this function is to weight the input signal with $\frac{1}{16}$ (6.25%) and the temporal average value with $1 - \frac{1}{16}$ (93.75%). If the function detects a falling moving average T_tmp, the output square signal, gets 1 unless the temporal average doesn't start to rise again. Figure 3.34 shows the generated square signal of the Matlab Function.

This first version was the easiest to find method for demodulating a signal and in the beginning of the thesis served to test the DPLL and the whole control loop. The reason why it didn't remain is because the used lowpass and median filter would cost too many resources at a final working digital chip.

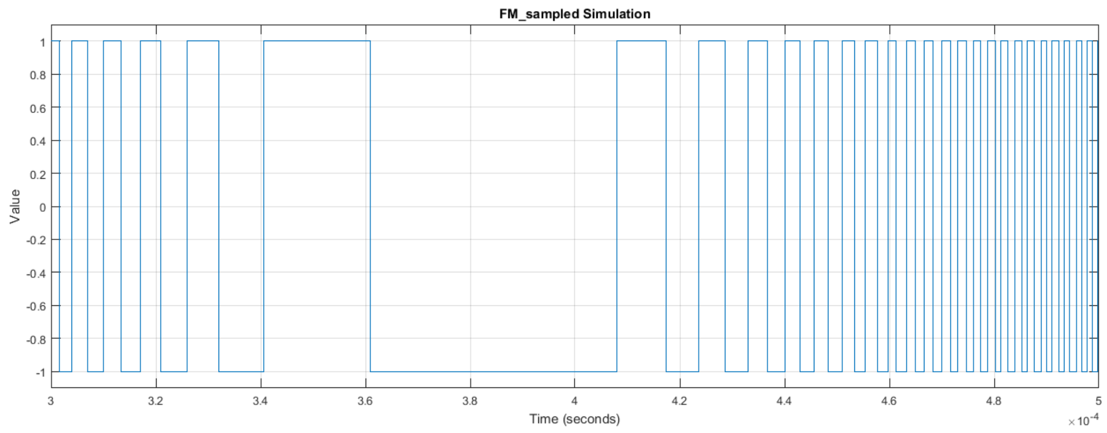


Figure 3.30: Sampled FM Signal of FM-Demodulator 2nd version

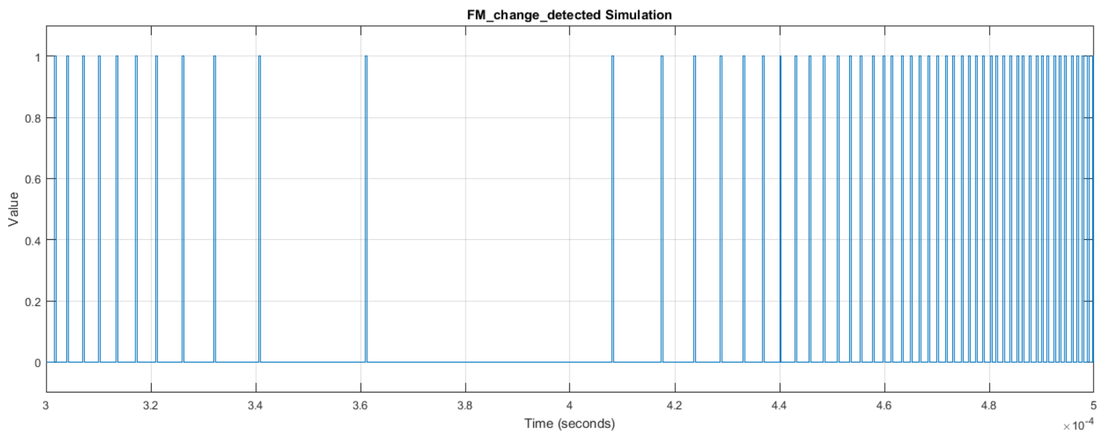


Figure 3.31: Rectangular Pulse Signal of FM-Demodulator 2nd version

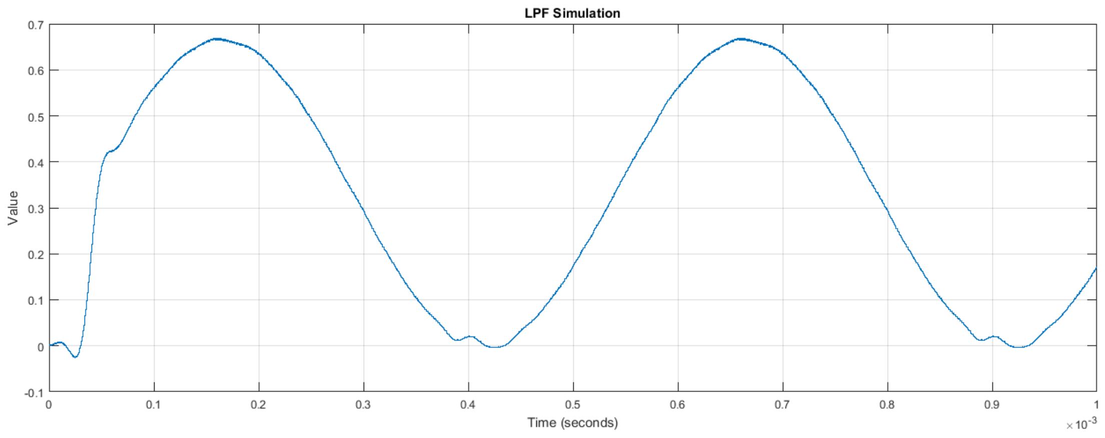


Figure 3.32: Signal after Lowpass Filter of FM-Demodulator 2nd version

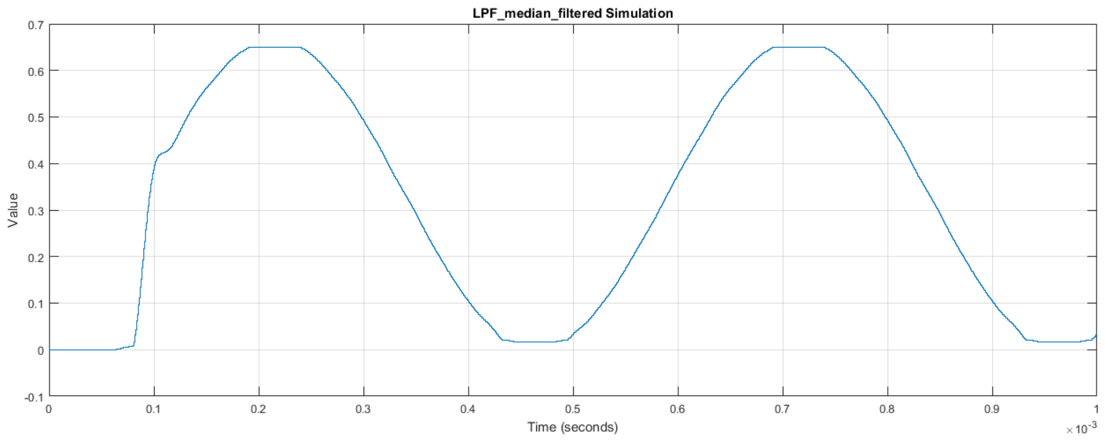


Figure 3.33: Median Filtered Signal after Lowpass Filter of FM-Demodulator 2nd version

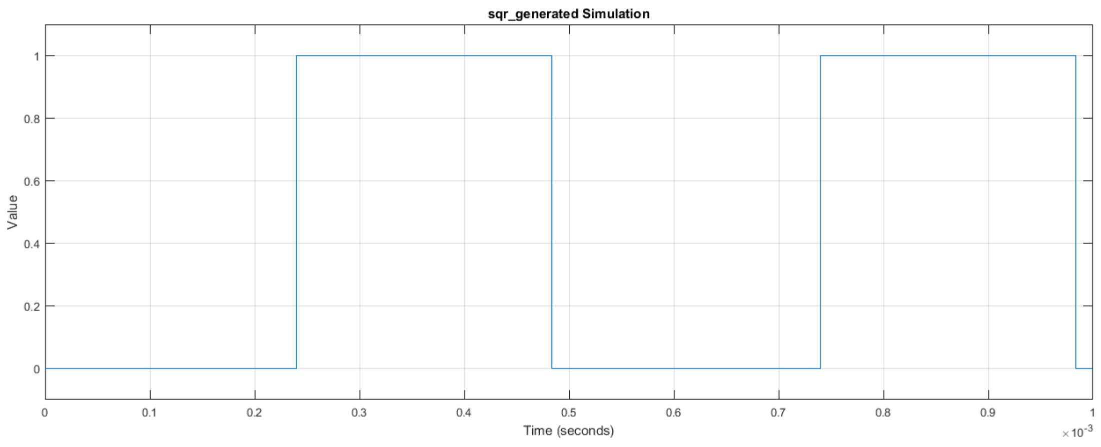


Figure 3.34: Generated Square Signal of FM-Demodulator 2nd version

3.2.3 FM-Demodulator 3rd version

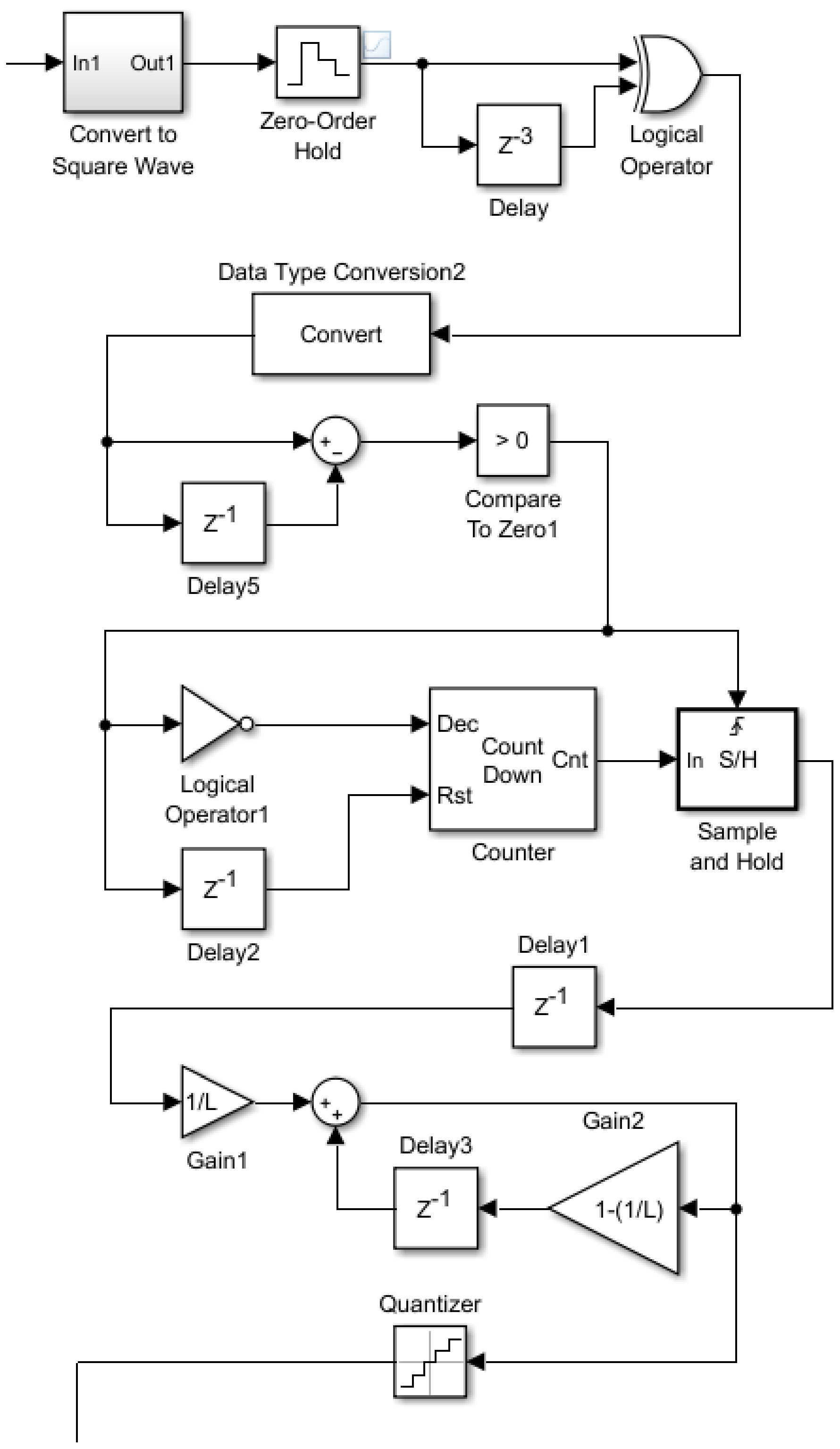


Figure 3.35: Simulink model of FM-Demodulator 3rd version

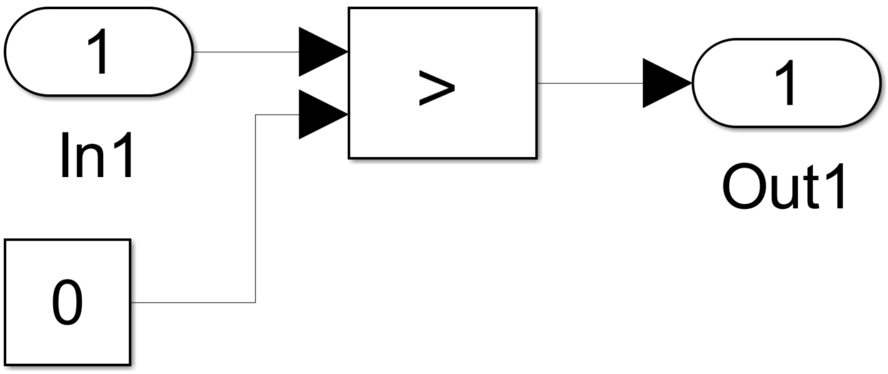
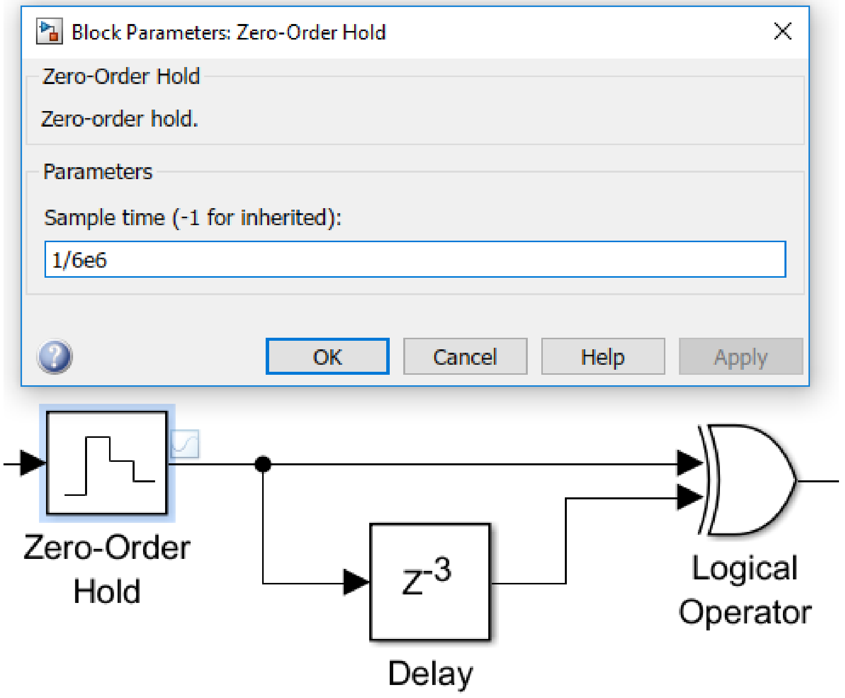


Figure 3.36: Simulink Block Convert to Square Wave


Block Parameters: Zero-Order Hold

Zero-Order Hold

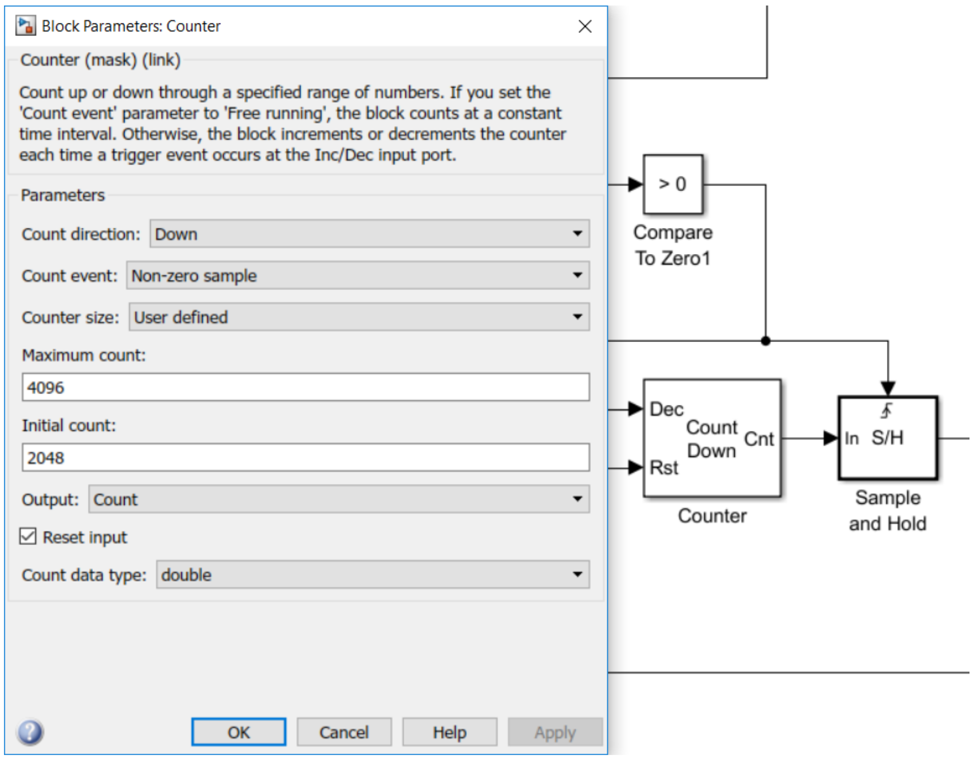
Zero-order hold.

Parameters

Sample time (-1 for inherited):

1/6e6

Figure 3.37: Simulink Block Zero-Order Hold



Block Parameters: Counter

Counter (mask) (link)

Count up or down through a specified range of numbers. If you set the 'Count event' parameter to 'Free running', the block counts at a constant time interval. Otherwise, the block increments or decrements the counter each time a trigger event occurs at the Inc/Dec input port.

Parameters

Count direction: Down

Count event: Non-zero sample

Counter size: User defined

Maximum count: 4096

Initial count: 2048

Output: Count

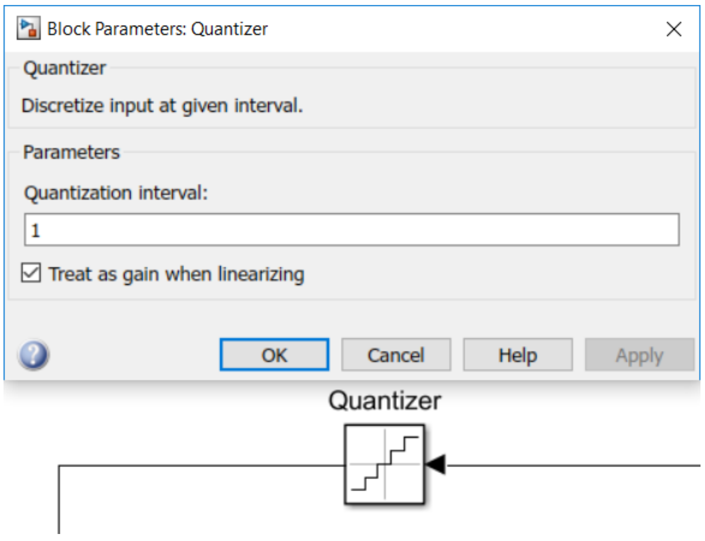
☑ Reset input

Count data type: double

Figure 3.38: Simulink Block Counter

```
%% FM-Demodulator 3rd version
% IIR moving average approximation
L = 1024;
```

Listing 3.8: MATLAB Code for Moving Average



Block Parameters: Quantizer

Quantizer

Discretize input at given interval.

Parameters

Quantization interval: 1

☑ Treat as gain when linearizing

Figure 3.39: Simulink Block Quantizer

The third and final version of the FM-Demodulator is the most simplest version of all three and it got used in the VHDL code for the control loop as well. A simple compare block, like in Figure 3.36, queries the incoming FM sinus signal if it is higher than 0 and produces an FM square signal from 0 to 1. This signal goes, like at the second version, to a Zero-Order Hold Block with an sample frequency of 6 MHz like shown in Figure 3.37. So the FM signal with a frequency range of 9 MHz to 10 MHz gets under-sampled to a lower frequency range of 2 MHz to 3 MHz. After this virtual, under-sampling ADC the real FM-Demodulator starts. This FM-Demodulator works for a determined frequency range depending on the delay before the XOR block. In this case the delay block has a delay length of 3 and a sample frequency of 6 MHz. This means in effect that the delay is 500 ns what is equal to 2 MHz. The FM-Demodulator works for a frequency range of the half of the result of the delay starting from 0 to infinity. So the working ranges are 0 MHz to 1 MHz, 1 MHz to 2 MHz, 2 MHz to 3 MHz and so on. For that reason it is essential to ensure that the FM signal has a frequency from 9 MHz to 10 MHz like in Chapter 3.1 mentioned. The functionality of the XOR part of the FM-Demodulator is shown in the following three Figures. Figure 3.40 shows the XOR working for an input signal of 2 MHz.

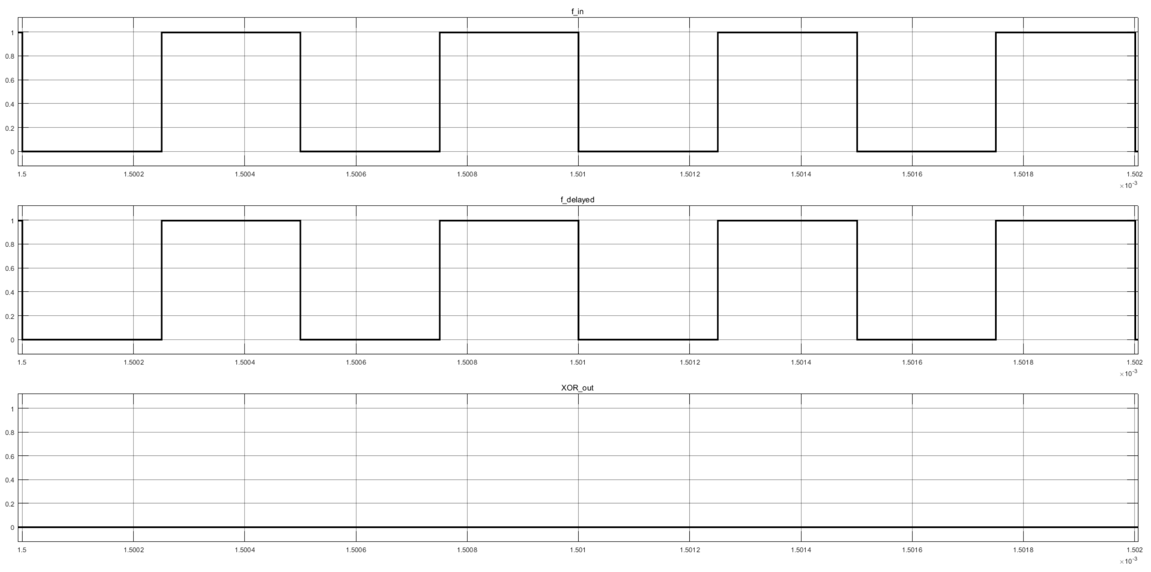


Figure 3.40: XOR of the FM-Demodulator 3rd version for an input of 2 MHz

Figure 3.40 shows the XOR working for an input signal of 2.5 MHz.

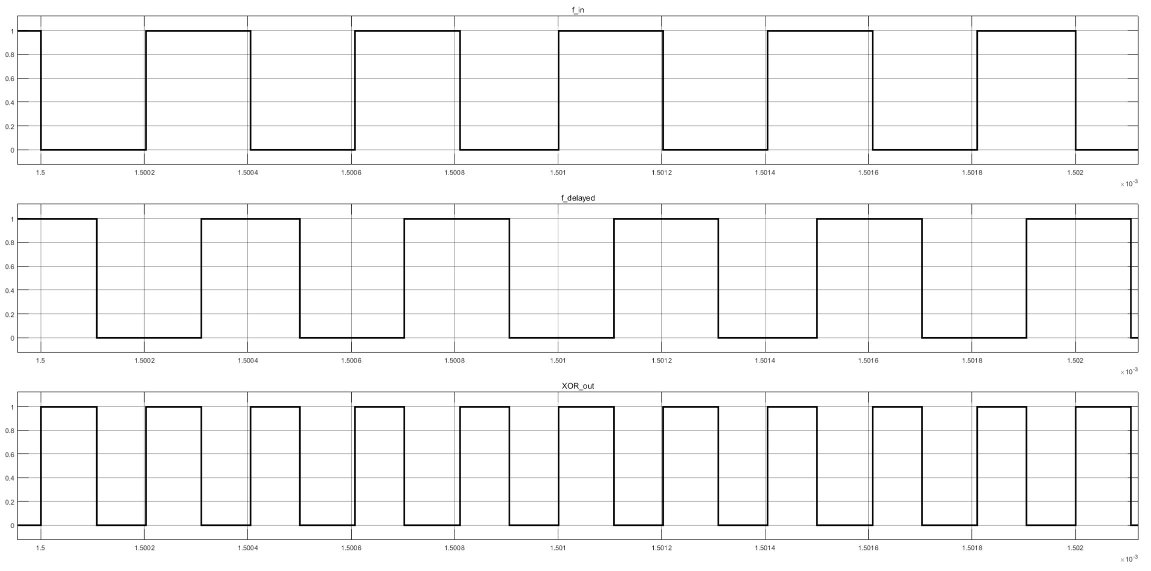


Figure 3.41: XOR of the FM-Demodulator 3rd version for an input of 2.5 MHz

Figure 3.40 shows the XOR working for an input signal of 3 MHz.

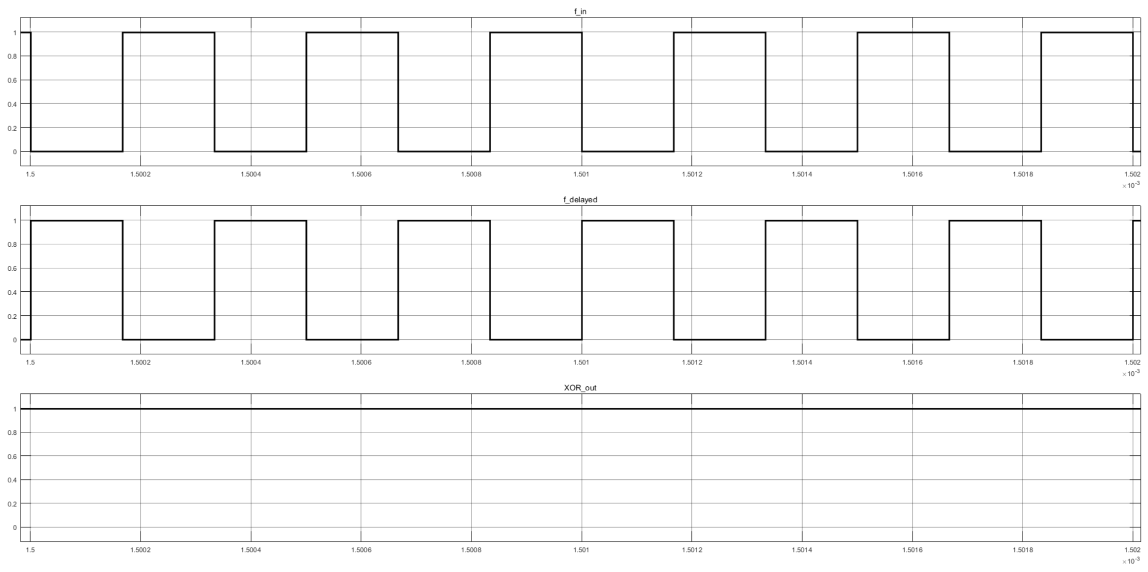


Figure 3.42: XOR of the FM-Demodulator 3rd version for an input of 3 MHz

The output of the XOR gets to the rising detection part where the XOR signal gets delayed and subtracted by the XOR signal. This resulting signal gets queried if it is higher than 0 and the result is a varying rectangular pulse signal like in Figure 3.43.

Now this rising signal serves as the input of the counter part to reproduce the frequency modulated signal. The Simulink Counter Block, Figure 3.38, works as follow: As long the Dec - Decrease - input has a value not equal to 0 the counter counts down from the initial value 2048. For this reason the rising signal gets inverted and connected to this input. The Rst - Reset - input resets the counter to the initial value when it gets a high and it is connected to the delayed rising signal. The counter counts down at each time step and gets reset after each rising pulse. This produces a signal like in Figure 3.44. To ensure a smooth and not triangular pulse signal a Sample and Hold Block takes place. The trigger input is the rising signal as well and this holds the value of the counter at rising impulse time as the output like in Figure 3.45. Like at the first FM-Demodulator a moving average gets used to smooth the signal. The representative FM demodulated signal, the output of the Sample and Hold Block, gets scaled and added to the scaled moving average signal. The result is the moving average signal like in Figure 3.46. To check the functionality of the FM-Demodulator at a chip similar behaviour the moving average signal gets quantized and produces a signal like in Figure 3.47. This signal serves as the input for the Signal Extracting Function which is described in the next Chapter 3.3.

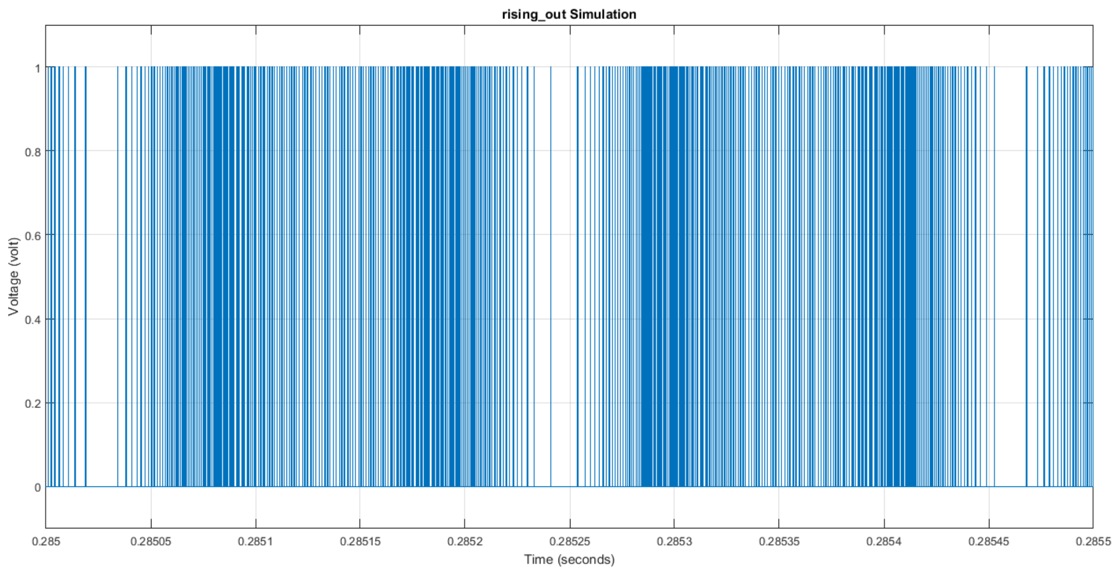


Figure 3.43: Detected Positive-Rising Signal of FM-Demodulator 3rd version

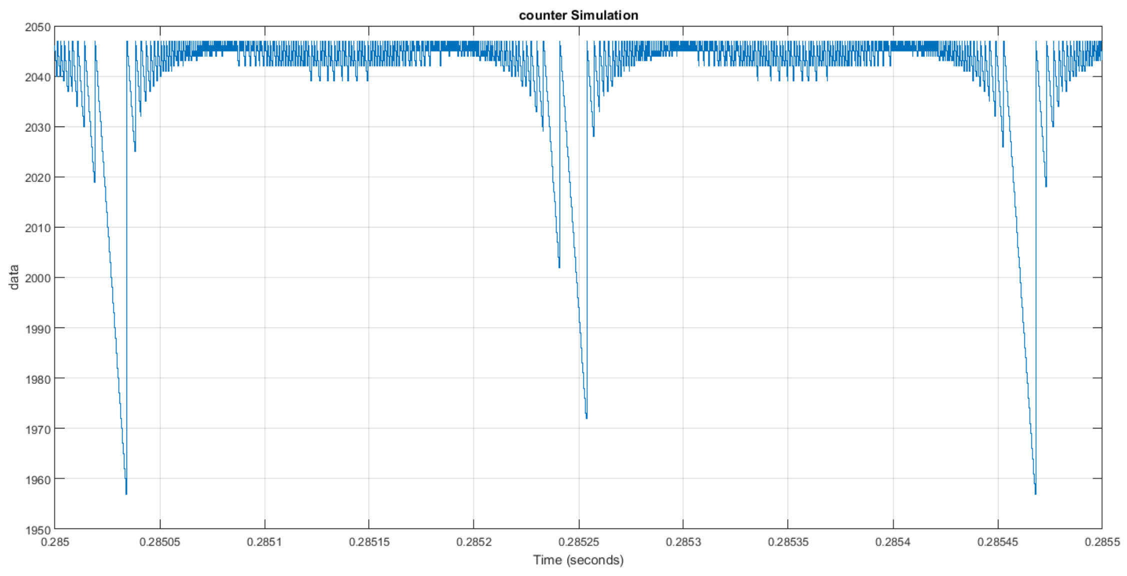


Figure 3.44: Counter Signal of FM-Demodulator 3rd version

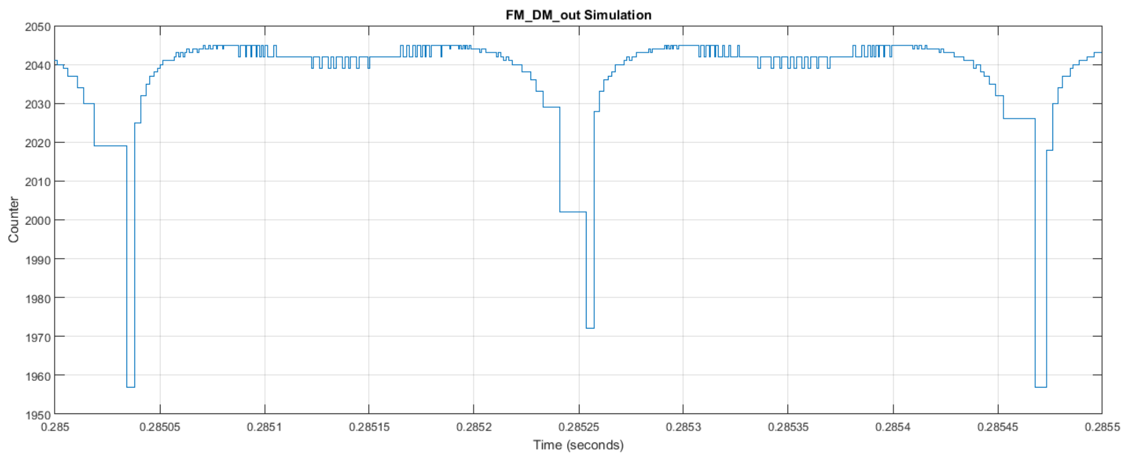


Figure 3.45: FM-Demodulated Signal of FM-Demodulator 3rd version

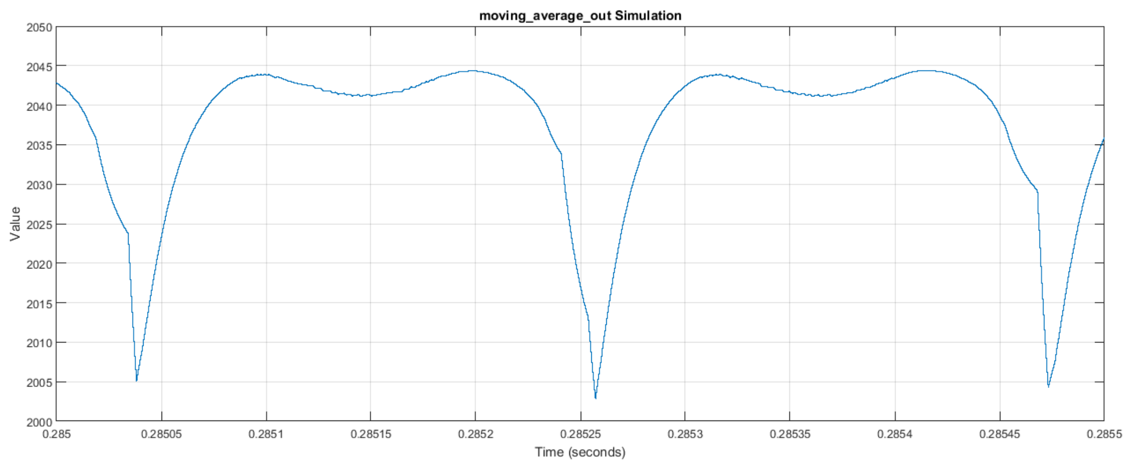


Figure 3.46: Moving-Average Signal of FM-Demodulator 3rd version

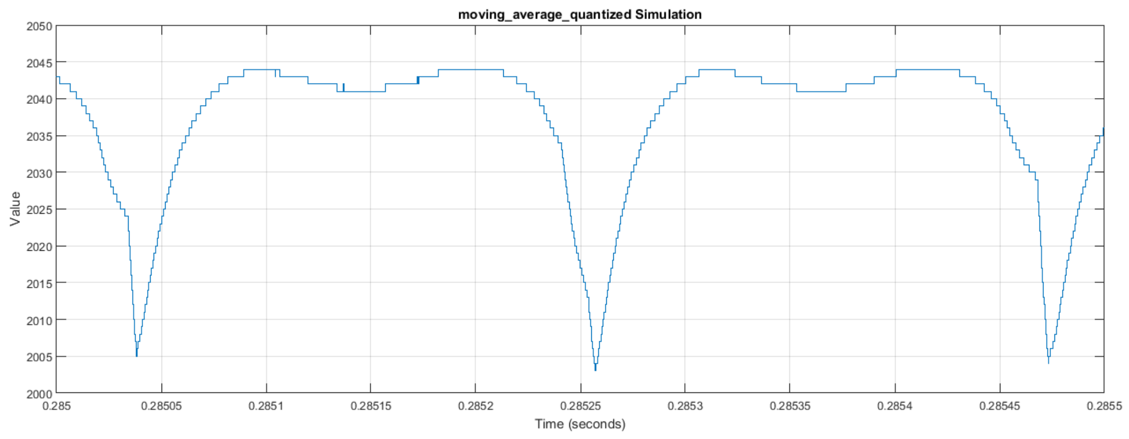


Figure 3.47: Quantized Moving-Average Signal of FM-Demodulator 3rd version

Figure 3.48 shows the functionality of the FM-Demodulator including its most important simulations and of the signal extracting part generated rectangular pulse signal sqr_out.

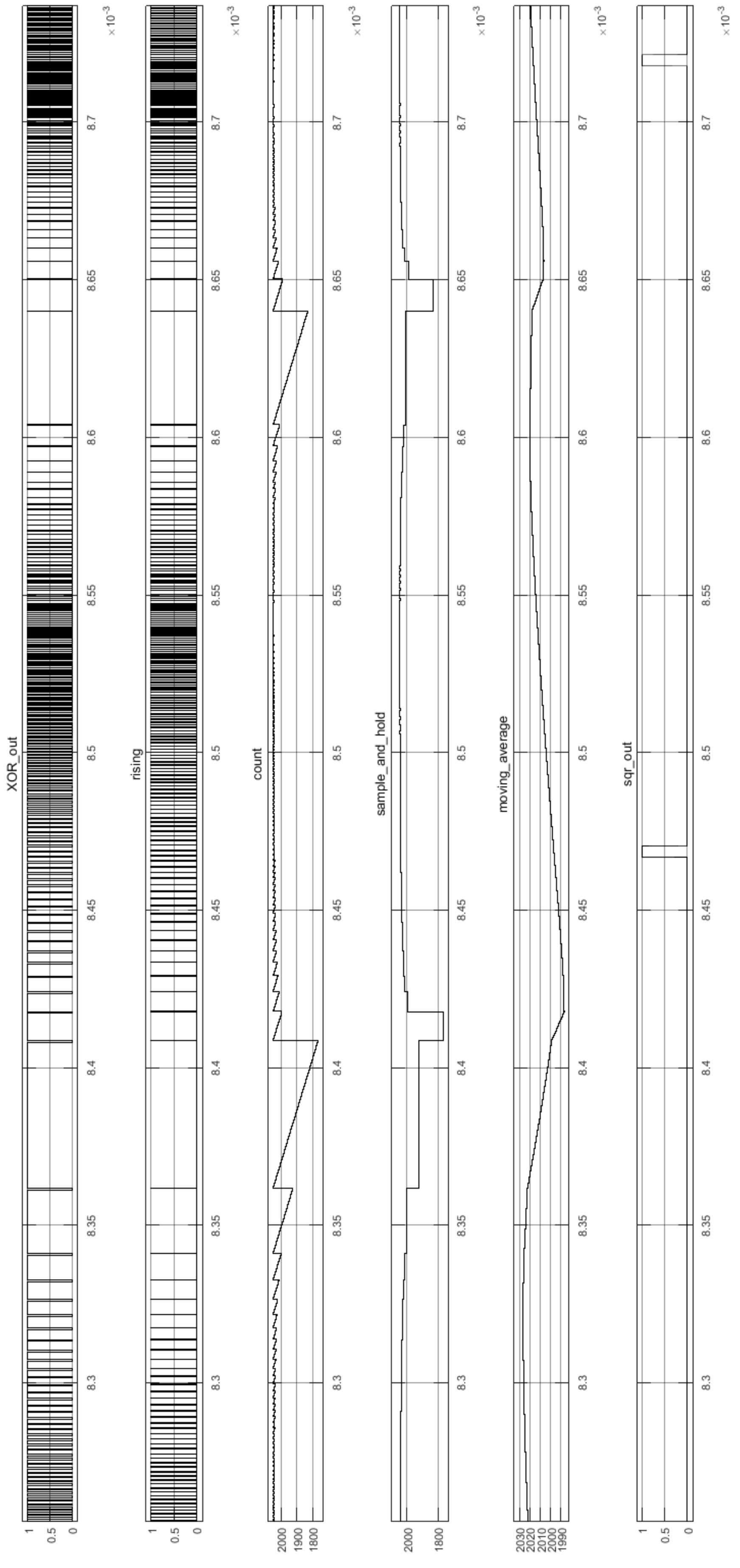


Figure 3.48: Functionality of the FM-Demodulator 3rd version

3.3 Signal Extracting

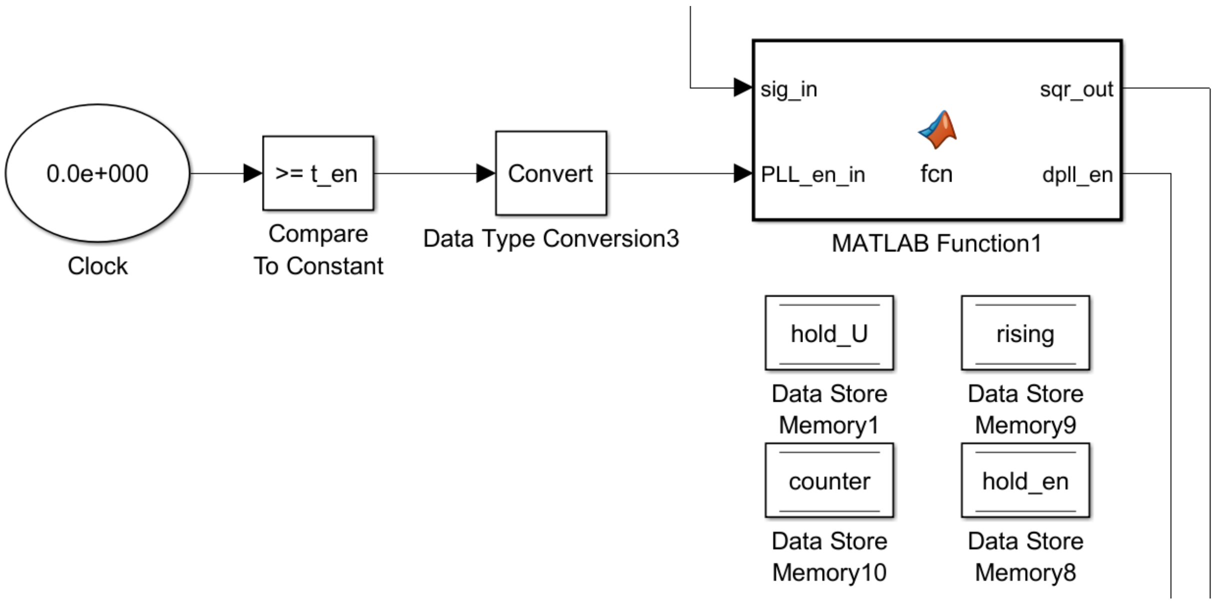


Figure 3.49: Simulink model of signal extracting

```
%% Signal Extracting
t_en = 0.22; % moment of closing the control loop

% initial values Data Store Memories
hold_U_initial = 0;
rising_initial = 1;
counter_initial = 0;
hold_en_initial = 0;
```

Listing 3.9: MATLAB Code for Signal Extracting

```
function [sqr_out, dpll_en] = fcn(sig_in, PLL_en_in)
    global counter;
    global hold_U;
    global hold_en;
    global rising;

    if ((hold_U > 0) && (hold_U < 300))
        hold_U = hold_U + 1;
    else
        hold_U = 0;
    end
```

```

        if ((sig_in > counter) && (rising == 1))
            counter = sig_in;
        end
        if (((counter-sig_in) >= 10) && (rising == 1))
            rising = 0;
            counter = sig_in;
        end
        if ((sig_in < counter) && (rising == 0))
            counter = sig_in;
        end
        if (((sig_in-counter) >= 10) && (rising == 0))
            rising = 1;
            counter = sig_in;
            hold_U = 1;
        end

        if ( PLL_en_in >= 0.6)
            hold_en = 1;
        end

    if (hold_U ~= 0)
        sqr_out = 1;
    else
        sqr_out = 0;
    end
    dpll_en = hold_en;
```

Listing 3.10: MATLAB Function Shaper and Inverter

The Signal Extracting part is the part of the control loop which extracts the incoming signal of Figure 3.47 and generates with the aid of the Matlab code of Listing 3.16 the rectangular pulse signal like in Figure 3.50. This signal serves as the reference signal for the DPLL like in Chapter 3.4 described.

The code works as follows: There are four variables to ensure the functionality of the code. Counter serves as a cache for the incoming quantized moving average signal. Hold_U serves as a counter to produce the rectangular pulse signal for a time amount of 300 samples which are equal to 3.5714 μs. Hold_en serves as the signal to enable the DPLL and close the control loop. Rising serves as a toggle to determine a rising or a falling of the incoming signal.

Lines 7 to 11 ensure the length of 300 samples of the rectangular pulse by counting up to 300 when the hold_U gets 1 at line 30. At the lines 13 to 15 the counter variables saves the highest value of the incoming signal sig_in at the rising mode. If the difference of the highest value of sig_in to the actual value of sig_in is higher or equal 10 then the falling mode starts by toggling the rising variable from 1 to 0 shown in lines 16 to 19. Also the counter variables saves the actual sig_in value. Vice versa works the falling mode. At the lines 20 to 22 the lowest value of sig_in gets saved at the counter. At the lines 23 to 27 the difference of the lowest point to the actual value gets checked to detect a rising mode. At positive detected rising state rising gets toggled from 0 to 1 again, counter caches the actual value of sig_in and hold_U gets set to 1 to start the rectangular pulse for the output. At the lines 29 to 31 the incoming signal PLL_en_in sets the variable hold_en to 1 when its value is higher or equal to 0.6. This happens after 0.22 s like in Figure 3.49 shown. At the line 38 the enable signal gets forwarded to the output of the Matlab Function Block dpll_en. At the lines 33 to 37 the output gets generated by querying the hold_U of not equality to 0.

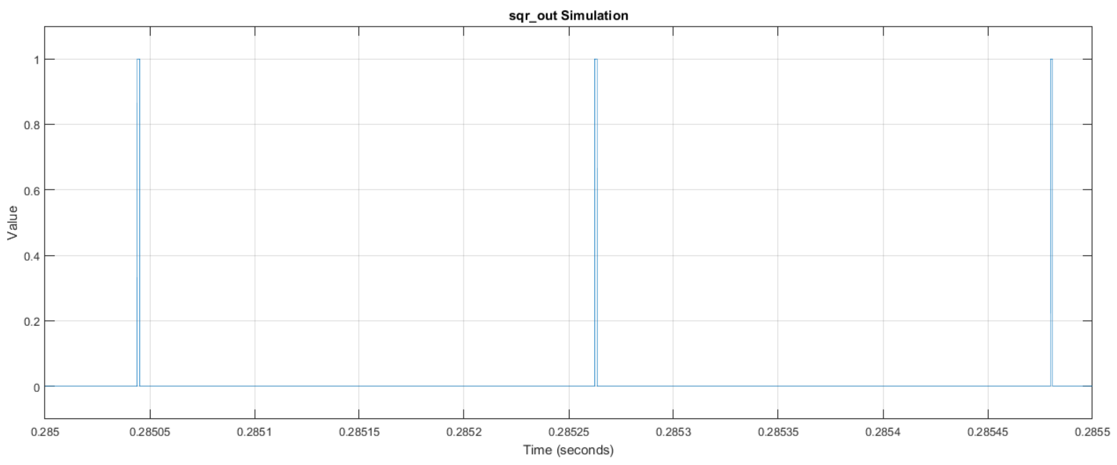


Figure 3.50: Signal-Extracted Signal sqr_out of Signal Extracting

3.4 DPLL

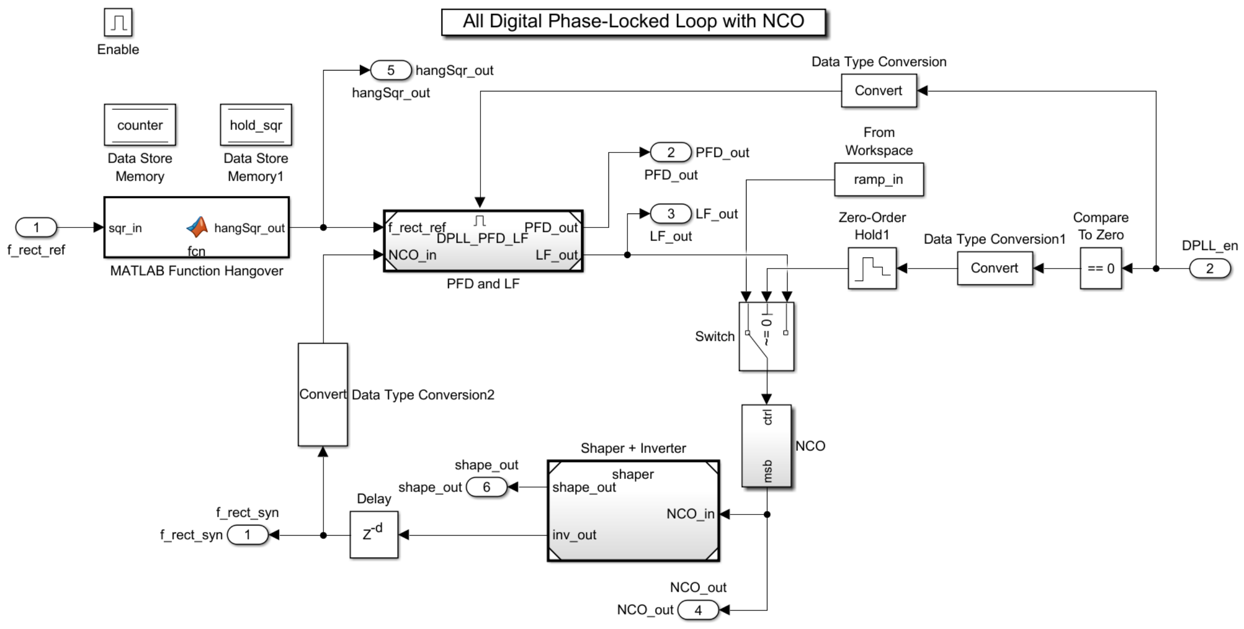


Figure 3.51: Simulink model of DPLL

```
%% parameter for DPLL
fs_dpll = 4e6;
Ts_dpll = 1/fs_dpll;

% natural frequency and damping
```

```
 6  wn_dpll = (2*pi*2*f_mir)/10;
 7  damp_dpll = 1;
 8
 9  % hangover function initial values
10  counter_initial = 0;
11  hold_sqr_initial = 0;
12
13  % PFD
14  Kd_dpll = 1;
15
16  % NCO
17  f_res_dpll = 0.5; %[0.5Hz]
18  Ko_dpll = fs_dpll*pi;
19  L_dpll = ceil(log(fs_dpll/f_res_dpll)/log(2));
20
21  % loop filter
22  alpha_dpll = (wn_dpll*(wn_dpll*Ts_dpll+2*damp_dpll))/(Ko_dpll*Kd_dpll);
23  betha_dpll = -(wn_dpll*2*damp_dpll)/(Ko_dpll*Kd_dpll);
24  F_dpll_z = tf([alpha_dpll betha_dpll],[1 -1],Ts_dpll);
25  [F_dpll_z_num, F_dpll_z_den] = tfdata(F_dpll_z);
26
27  % quantizing
28  B_lf = round(-log2((alpha_dpll + betha_dpll)/10)); % B_lf = 26
29
30  % shaper and inverter
31  % initial values
32  address_initial = 0;
33  inv_initial = 0;
34  % sigmoid shaper
35  sh_size = 51;
36  x_sh = 0:50;
37  y_sh = sigmf(x_sh,[0.2 25]);
38
39  % delay
40  delay_dpll = round(0.00006/Ts_dpll); % delay of 60µs
```

Listing 3.11: MATLAB Code for DPLL

This DPLL is an All Digital Phase-Locked Loop with a Hangover Function, a start-up ramp, a shaper and inverter and an additional delay to compensate the through the FM-Demodulation and signal extraction produced time delay. Listing 3.11 shows the used Matlab code to generate the DPLL. The sample frequency of 4 MHz, line 2, of the DPLL block is different to the whole loop with 84 MHz because this high sample frequency is not needed any more with an input signal of about 4 kHz. To calculate the loop filter two constants have to get chosen. For the natural frequency of the DPLL a tenth of the incoming frequency got chosen, line 6 and for the damping the value 1 like in line 7. The amount of delays gets calculated at line 40 with an delay of 60 μs.

3.4.1 Hangover Function

```
 1  function hangSqr_out  = fcn(sqr_in, f_mir, Ts_dpll)
 2      global counter; % initial value counter = 0
 3      global hold_sqr; % initial value hold_sqr = 0
 4
 5      maxCnt = round(((1/(2*f_mir))/4)/Ts_dpll);
 6      if (counter > 0 && counter < maxCnt)
 7          counter = counter + 1;
 8      end
 9      if (counter == maxCnt)
10          counter = 0;
11      end
12      if (hold_sqr ~= sqr_in && counter == 0)
13          counter = 1;
14          hold_sqr = sqr_in;
15      end
16
17  hangSqr_out = hold_sqr;
```

Listing 3.12: MATLAB Function Hangover

The Matlab Function Hangover of Listing 3.12 serves to ignore disseminations produced by signal extracting process when a white noise gets overlapped to the FM signal at the beginning. It works as follows: When the incoming reference signal of Figure 3.50 gets to the DPLL the output of the Hangover function is 1 for a period of 25% of 4 kHz like in line 5 calculated. After this time it can be changed to 0 again. Figure 3.52 shows the functionality of this function.

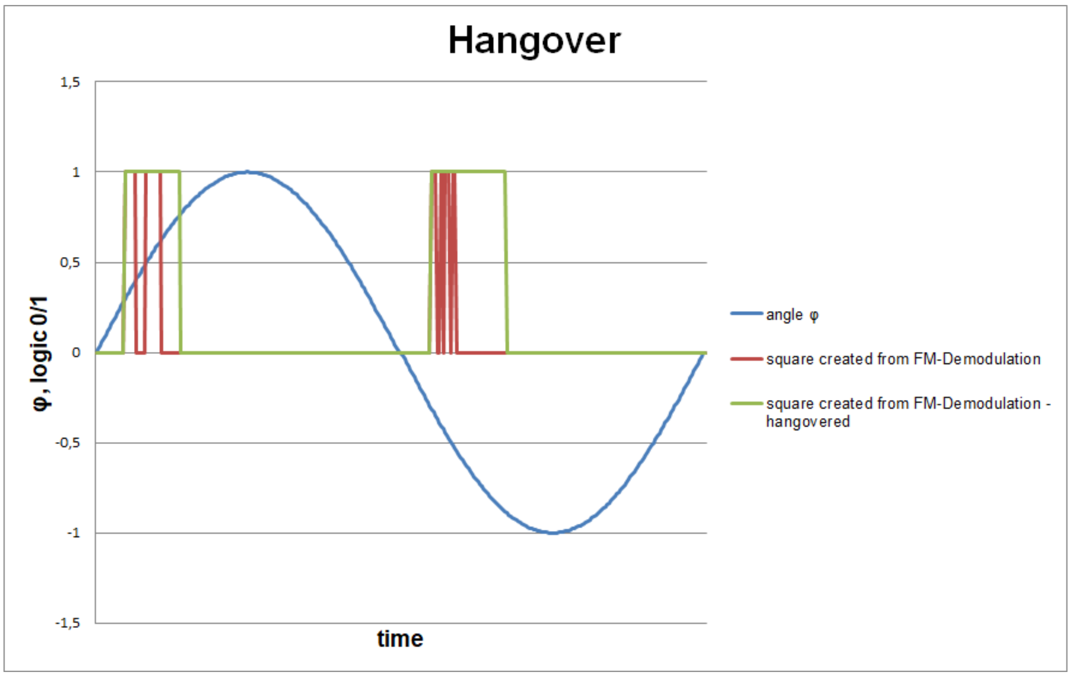


Figure 3.52: Functionality of the Hangover Function

Figure 3.53 is the output of the Hangover function after the input of Figure 3.50.

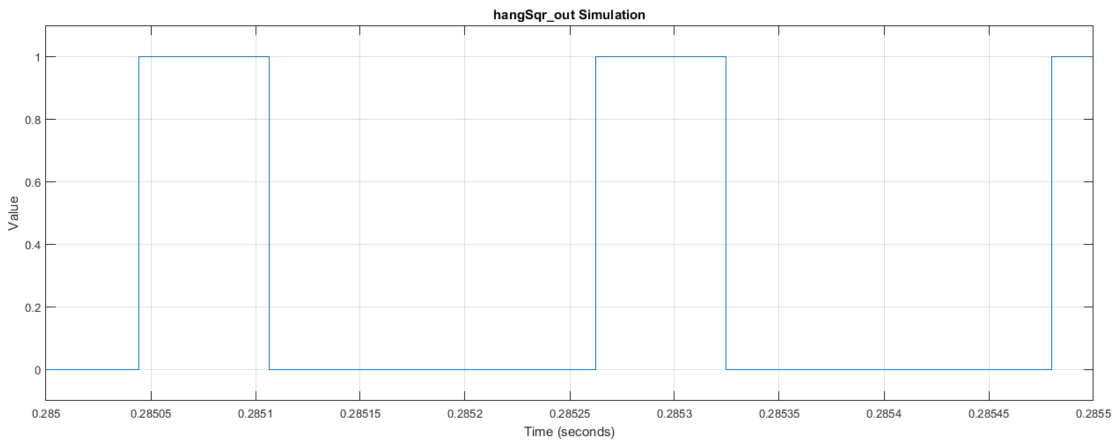


Figure 3.53: Hangovered Signal hangSqr_out of Hangover Function

3.4.2 Phase Frequency Detector and Loop Filter

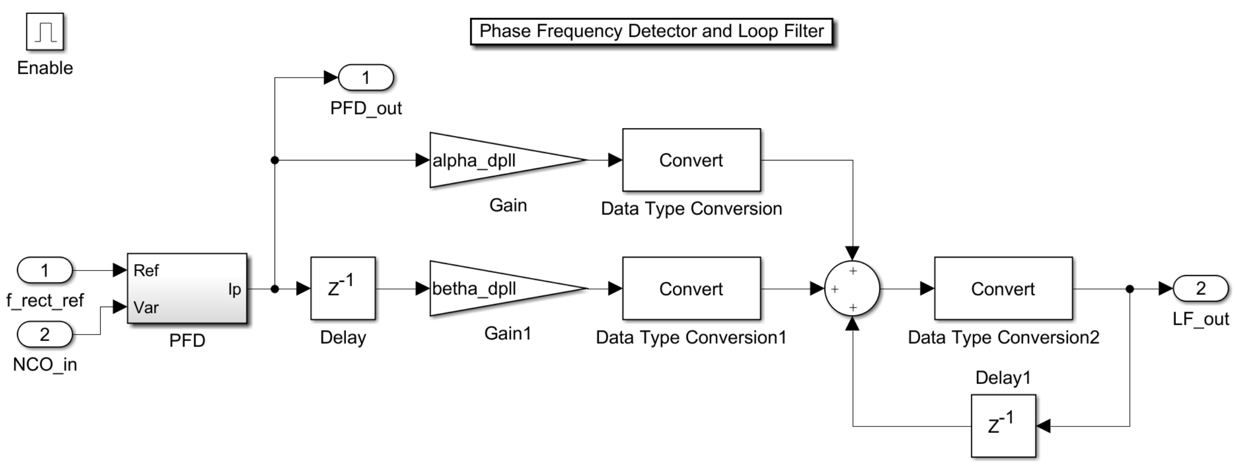


Figure 3.54: Simulink model of PFD and LF

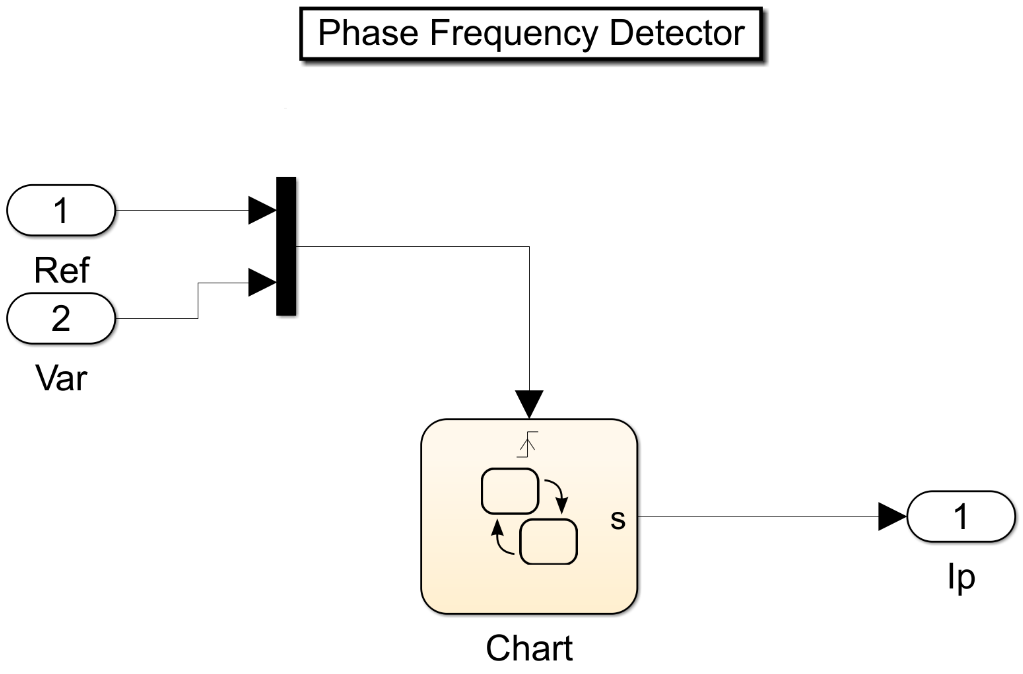


Figure 3.55: Simulink model of PFD

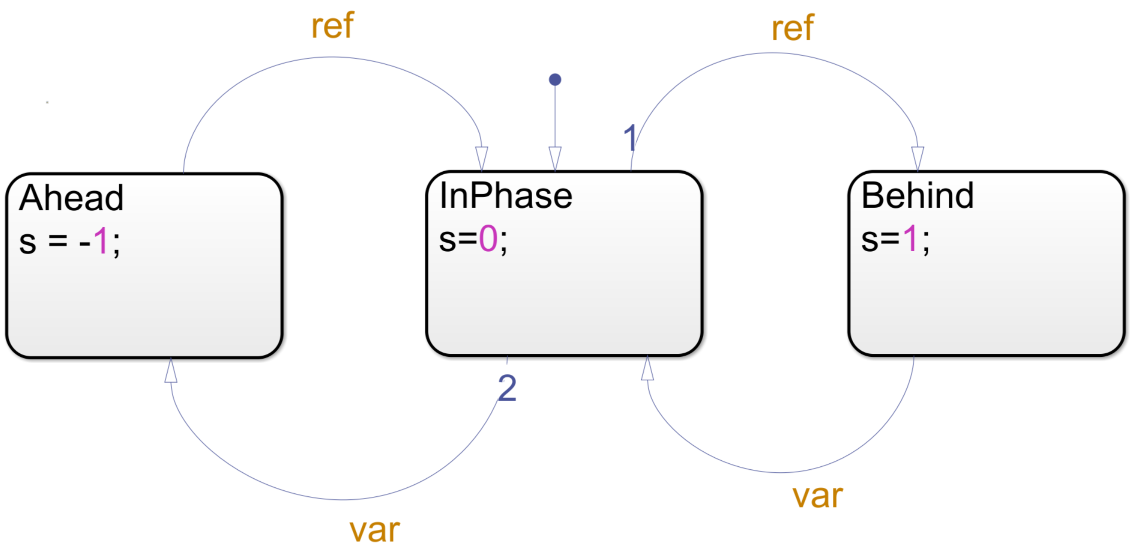


Figure 3.56: Simulink model of PFD Chart

The DPLL block has an additional Enable block to turn on and off the DPLL. At the start-up the DPLL is turned off to not calculate any wrong solutions. After the DPLL gets the enable signal from the signal extracting part the loop filter starts to calculate. Firstly the reference signal f_rect_ref gets compared with the from NCO produced and delayed signal NCO_in. For that both signals get into the PFD block like in Figure 3.55 where a Chart block is included. This Chart block, Figure 3.56, works as follows: If both frequencies are in phase the output is 0. If the reference signal, ref = f_rect_ref, gets high the PFD changes his state to behind and produces a 1 at the output as long as the produced signal, var = NCO_in, gets high too. The PFD changes his state to in phase again and produces a 0 at the output. If the var signal gets high first the PFD changes to the state ahead and produces a -1 at the output as long as the ref signal gets high too.[27] The output of the PFD goes to the loop filter where with aid of the before calculated loop filter variables the loop filter signal gets calculated. The topology of the loop filter is described at Chapter 2.5. The Convert blocks make the signals to fix bit signals of 26 bits like in Listing 3.11 line 28 calculated.

Figure 3.58 shows the output of the PFD with the reference signal after the Hangover function, Figure 3.53 and the produced signal of Figure 3.57 which is connected to the NCO_in input. This PFD output produces an output like in Figure 3.59. At this figure it is shown that the loop filter has to be at a value close to 300 μ to be in phase and ensure a frequency of around 4 kHz.

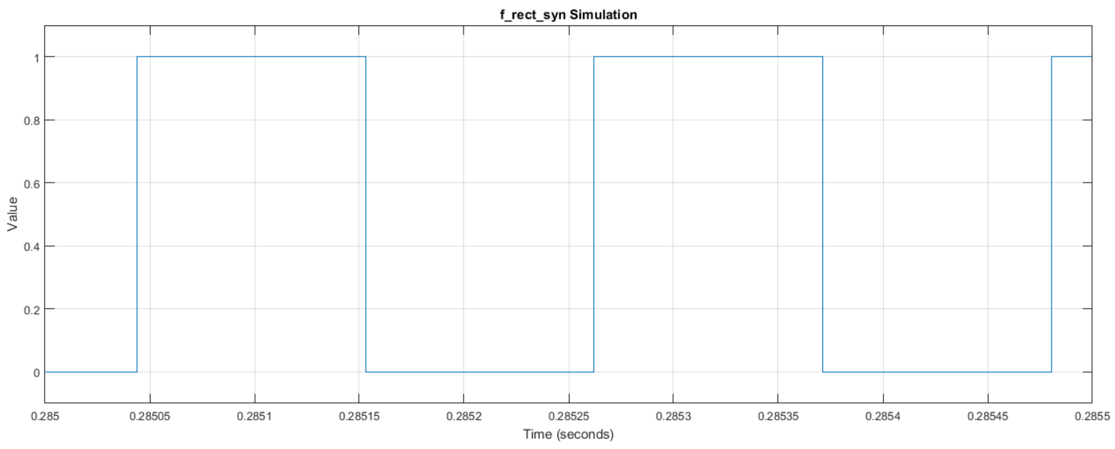


Figure 3.57: Inverted Delayed and Synchronized Signal f_rect_syn of DPLL

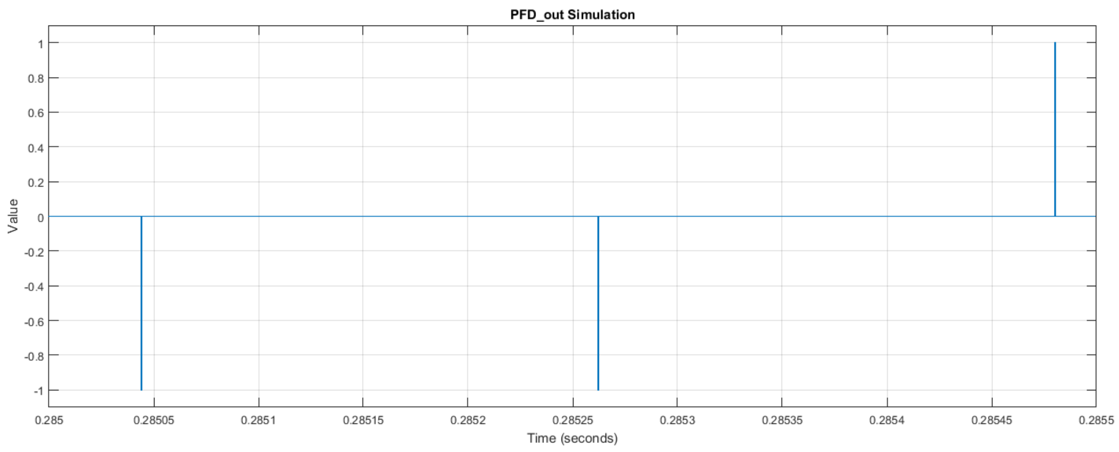


Figure 3.58: Phase Frequency Detector Signal PFD_out of DPLL

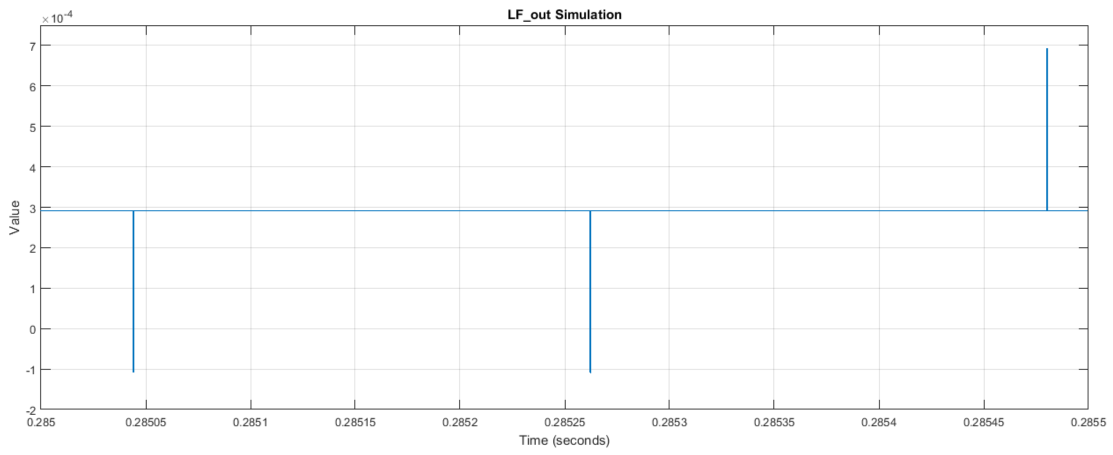


Figure 3.59: Loop Filter Signal LF_out of DPLL

3.4.3 Numerically Controlled Oscillator

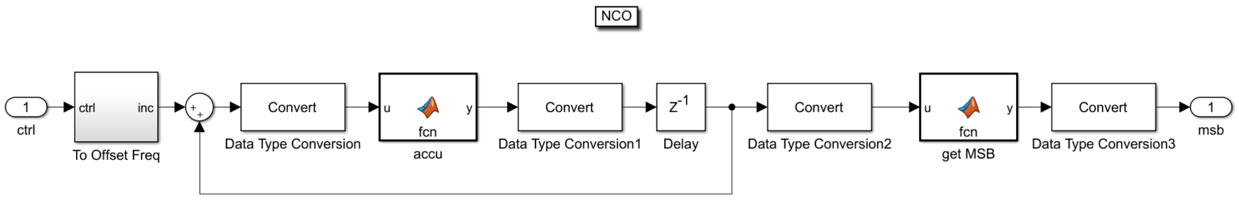


Figure 3.60: Simulink model of NCO

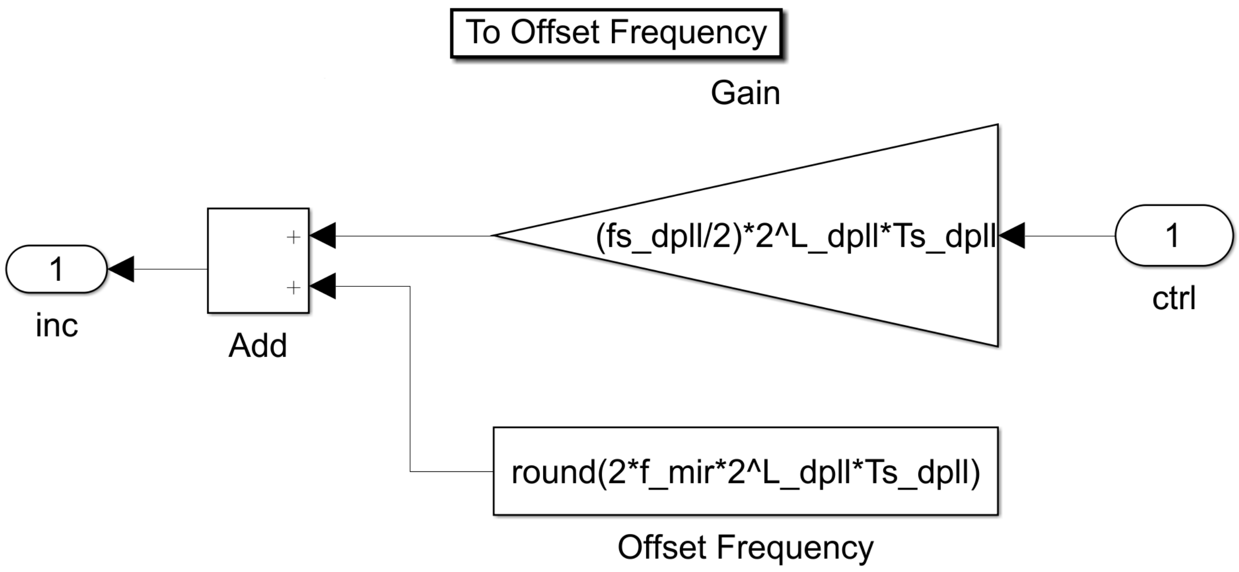


Figure 3.61: Simulink model of To Offset Frequency

```matlab
function y = fcn(u, L_dpll)
    if (u > ((2^L_dpll)-1))
        y = u - 2^L_dpll;
    else
        y = u;
    end
```

Listing 3.13: MATLAB Function Accumulator

```matlab
function y = fcn(u, L_dpll)
    if (u > (2^(L_dpll-1)))
        y = 1;
    else
        y = 0;
    end
```

Listing 3.14: MATLAB Function Get MSB

The input of the NCO, ctrl, gets either connected to the loop filter output or to the ramp_in until the DPLL gets enabled. The ramp input is described at

Chapter 3.5. This incoming signal serves as a control word and goes to the To Offset Freq Block, Figure 3.61, where it gets multiplied by a translating factor to change the output frequency of the NCO. The output of this block, inc, is the summation of this signal plus an offset frequency of 2*f_mir which is equal to 4 kHz. This offset frequency, which is approximately the expected output frequency, is necessary to keep the loop filter control signal low. The output of the To Offset Freq block gets to the Matlab accu Function after being added to the previous step result of the accu Function. This function is described at Listing 3.13 as follows: If the value of the input is higher than the size of the accumulator minus one then the result is the value of the input minus the size of the accumulator. Otherwise the output is the input. The size of the accumulator is two to the power of L_dpll, which is the amount of bits of the accumulator and declared at Listing 3.11 at line 19. Line 17 declares the resolution of the NCO which is 0.5 Hz. This resulting signal gets delayed and to the input port u of the Matlab Function get MSB - Most Significant Bit. Listing 3.14 describes this function as follows: If the value of the input signal is higher than two to the power of open bracket L_dpll minus one, close bracket, the output value is one and otherwise it is zero. In summary if the input value is higher than half of the size of the accumulator the output is one. Figure 3.62 shows the output of the NCO with an input of the loop filter of Figure 3.59.

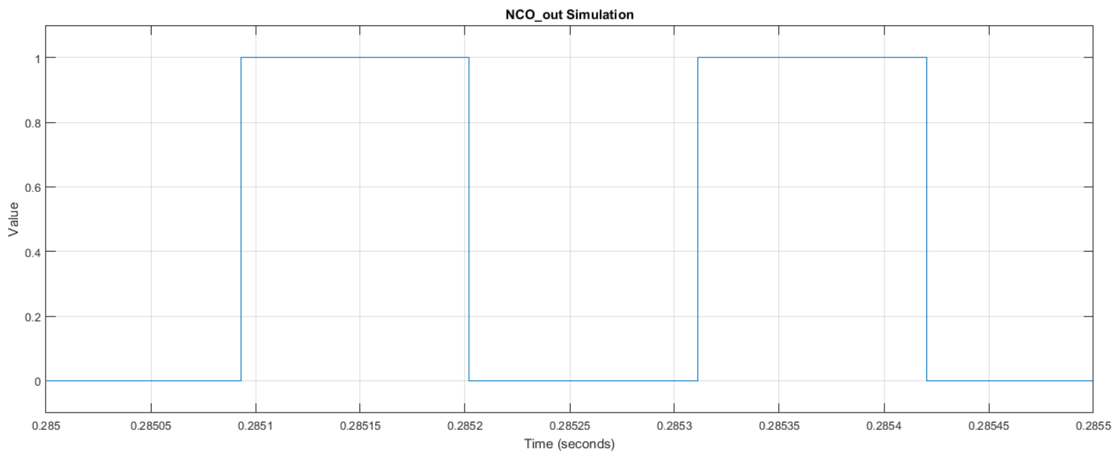


Figure 3.62: Numerically Controlled Oscillator Signal NCO_out of DPLL

3.4.4 Shaper and Inverter

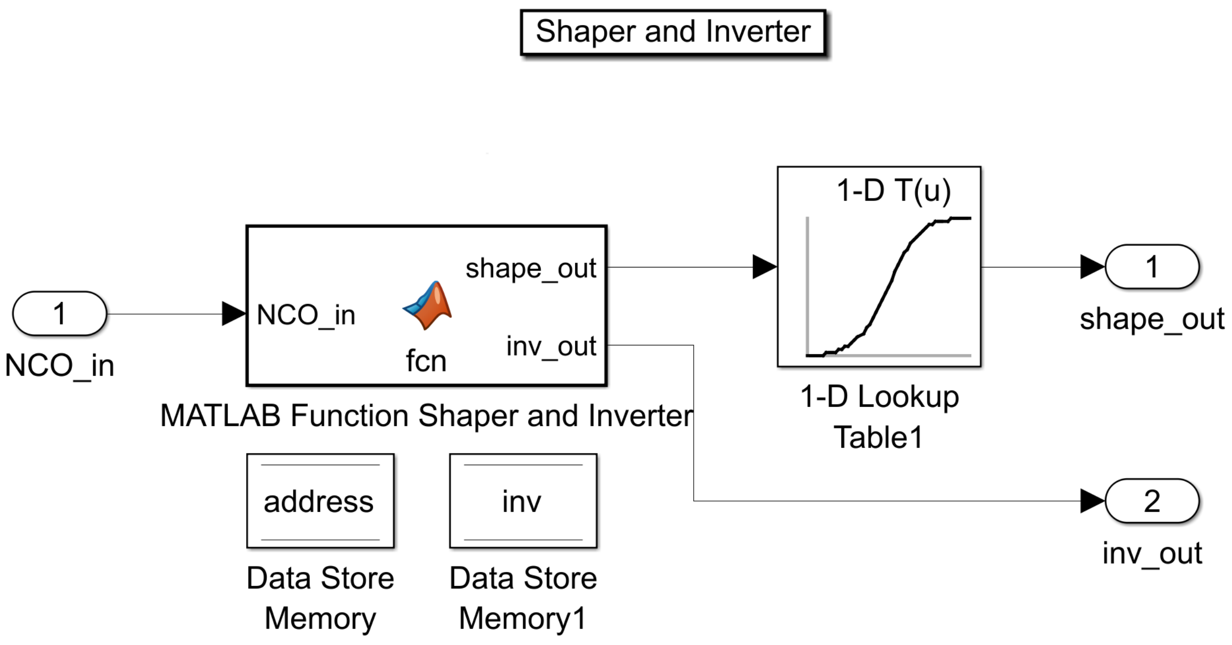


Figure 3.63: Simulink model of Shaper and Inverter

```
function [shape_out, inv_out] = fcn(NCO_in, sh_size)
    global address;
    global inv;

    if ((NCO_in == 1) && (address ~= sh_size))
```

```
        address = address + 1;
        inv = 0;
    elseif ((NCO_in == 0) && (address ~= 0))
        address = address - 1;
        inv = 1;
    end

    shape_out = address;
    inv_out = inv;
```

Listing 3.15: MATLAB Function Shaper and Inverter

The Shaper and Inverter block has two tasks to accomplish. The first task is to shape the output of the NCO to get a sigmoid similar output signal instead of a rectangular pulse signal. This happens by aid of the MATLAB Function Shaper and Inverter which is described at Listing 3.16 as follows: If the input of the function, NCO_in, is equal to 1 and temporal address not equal to the shaper size, sh_size, then the temporal address gets counted up by 1. The shaper size is set at Listing 3.11 at line 35 to 51.
The first output of the Matlab Function is the value of the address which is connected to the port shape_out. Vice versa the address gets counted down by one if the input NCO_in is equal to 0 and the address is not equal to 0. To summarize, if the input NCO_in gets 1 the output is a number between 1 and 51 which gets counted up at each time step and if the input NCO_in gets 0 then the output is a number between 50 and 0 which gets lower at each time step. This output goes to the Lookup Table which includes the form of a sigmoid like in Figure 3.64 and is defined at Listing 3.11 at the lines of 35 to 37. The resulting shaped signal goes to the output shape_out and continues to a multiplier for the voltage which is connected to the analog part and the mirror.
The second output of the Matlab Function is the inverted signal of the NCO

output. So if the NCO_in is 1 then the inv_out is 0 and vice versa. This also happens in the before mentioned Listing of the Shaper and Inverter. The reason why an inverted signal is needed is because the signal extracting part creates a rectangular pulse when the mirror crosses the 0° angle but at this point the voltage has to be turned off. Beside that fact the whole control loop needs time so an additional delay is connected between the inverted output and the PFD. Figure 3.65 shows the shaped output signal and Figure 3.57 is the inverted and delayed signal of this signal which gets compared with the signal extracted signal by the PFD. To summarize this knowledge, the DPLL always tries to synchronize the 2 signals. So if the produced second input of the PFD is a delayed signal then the produced signal is ahead the from outside coming signal and if the produced second input of the PFD is an inverted signal then the signal before the inverting is the same signal just the opposite way around.

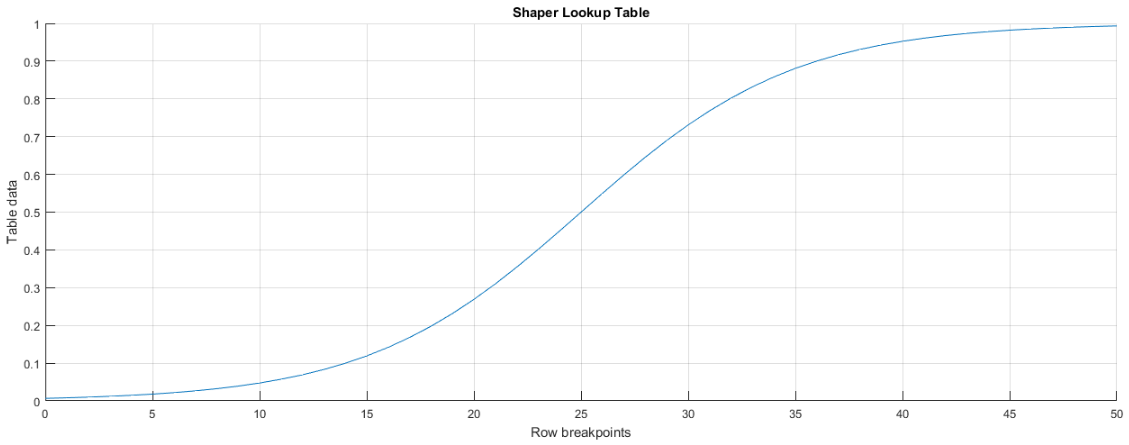


Figure 3.64: Lookup Table of Shaper

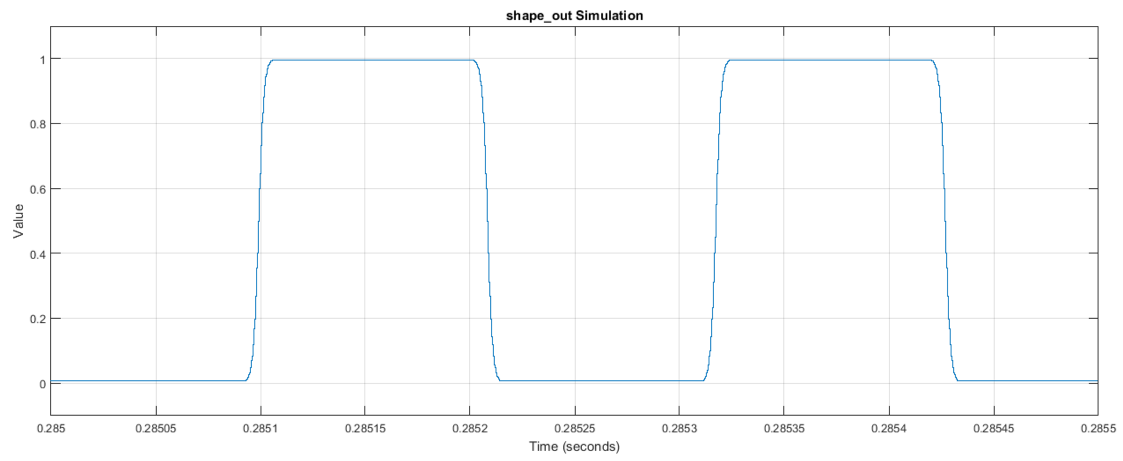


Figure 3.65: Shaped Output Signal shape_out of DPLL

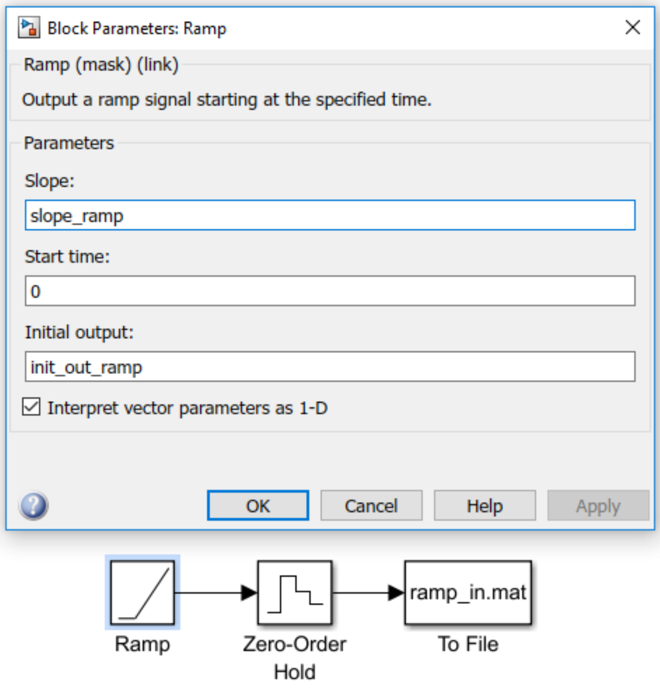
3.5 Start-up Simulation

The start-up simulation has two decisive factors: First of all the start-up frequency ramp which has to start from a higher frequency and has to drop with a certain speed and the second factor is the time of enabling the DPLL and closing the control loop.
Start-up frequency ramp: The mirror needs to start-up, beginning with a higher frequency than its resonance frequency and dropping in a certain speed to its resonance frequency. This is caused through the non-linearity of the mirror which is described by the Duffing equation in Chapter 2.2. To create the ramp_in variable from the DPLL of Figure 3.51 which is connected to the Switch block and continues to the NCO when the DPLL is not enabled a Simulink model, like in Figure 3.66, got created. This model contains a Ramp, Zero-Order Hold and To File block. The Ramp block is defined by the lines 2 and 3 at Listing 3.16. The initial output is $\frac{3}{4990}$ which is equal to 1200 Hz. But because the NCO has an offset of 4000 Hz this results to an initial output of 5200 Hz. The slope is $\frac{-8}{4990}$

which is equal to a drop of $3\ \dfrac{Hz}{ms}$. At simulation start the ramp has to be loaded to workspace from file like at line 6. Lines 9 to 12 make the ramp to a fix-32-bit variable.

```
1  %% parameter for ramp
2  init_out_ramp = 3/4990; %[5.2kHz -> 1/4990=400Hz]
3  slope_ramp = -8/4990; %[0-400ms -> 5.2-4kHz; -300Hz/100ms]
4
5  % load ramp_in from file
6  load ramp_in.mat;
7
8  % convert ramp_in to fix 32 bit
9  ramp_in_fix32 = ramp_in;
10 ramp_in_fix32.data = fi(ramp_in.data, 1, 32, 31);
11 ramp_in_fix32_bin = cellstr(bin(ramp_in_fix32.data));
12 ramp_in = ramp_in_fix32_bin;
```

Listing 3.16: MATLAB Function Shaper and Inverter


Block Parameters: Ramp

Ramp (mask) (link)

Output a ramp signal starting at the specified time.

Parameters

Slope: slope_ramp

Start time: 0

Initial output: init_out_ramp

☑ Interpret vector parameters as 1-D

Figure 3.66: Simulink model of Ramp to File

Time of enabling the DPLL: The time of enabling the DPLL has two decisive factors as well. The one is the previous mentioned frequency ramp because this is the decisive factor how fast the mirror gets to the desired resonance frequency and how big the amplitude of the mirror movement at a certain time is. Also the mirror movement starts to break after the ramp passes a certain frequency of about 4300 Hz. The second factor is the signal extracting part. To close the loop the signal extraction has to work in a proper way to ensure a correctly generated rectangular pulse signal. For that reason the amplitude of the mirror has to be high enough to guarantee a significant difference between the highest and lowest produced FM signal. Also if a white noise overlaps the FM signal the difference between these two frequencies even has to be higher to detect the rising and falling states of the signal.

Figure 3.67 shows the ramp simulation with the red vertical line at 0.22 s which highlights the time of enabling the DPLL. At this time the ramp produces with aid of the NCO and its offset a frequency of around 4500 Hz.

At Figure 3.68 the mirror movement of the whole simulation is demonstrated. The mirror starts at an angle of around 10° with an incoming frequency of 5200 Hz. The angle drops in the first 50 ms and starts to grow and to get stable afterwards again. After 0.22 s, when the control loop gets closed, the DPLL needs a few milliseconds to catch the mirror and to stabilize it again. After that the mirror controls itself to its resonant frequency and to its maximum amplitude. The whole start-up from begin of the ramp until the DPLL works stable and the mirror is at its maximum takes around 250 ms.

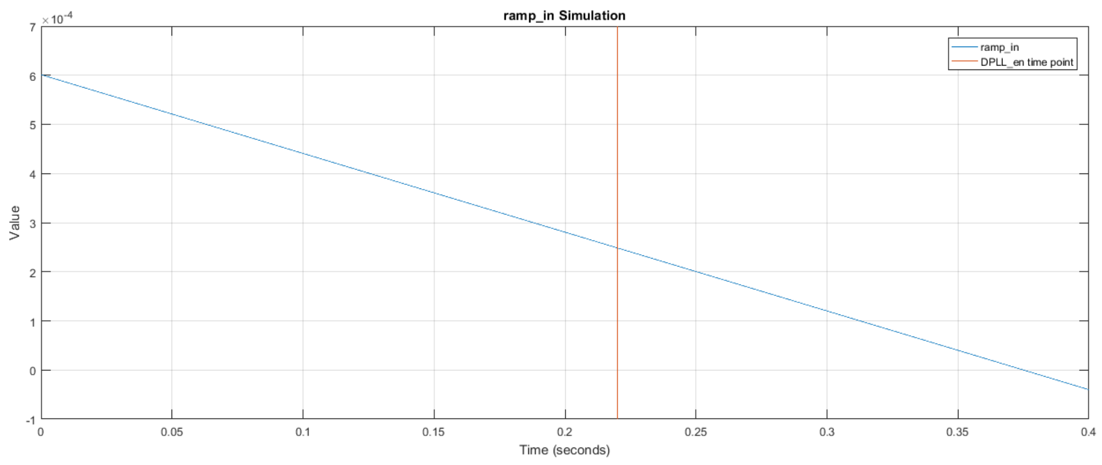


Figure 3.67: Ramp Signal ramp_in of DPLL

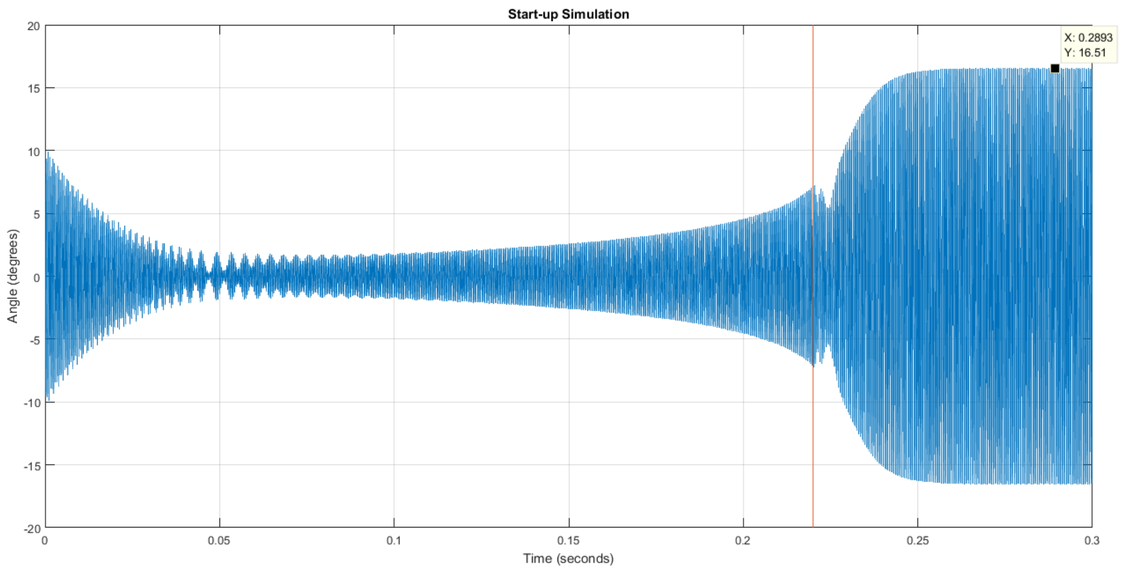


Figure 3.68: Start-up Simulation of MEMS-Mirror

3.6 Error Dependency

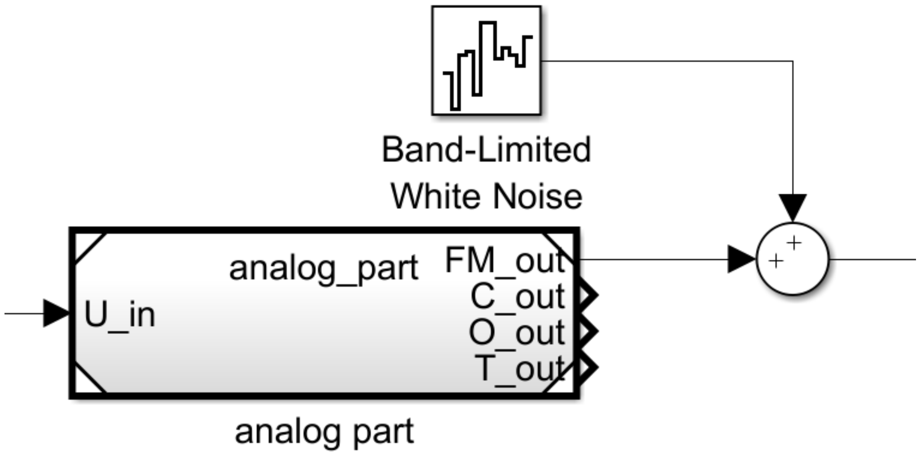


Figure 3.69: Simulink model of Analog Part and White Noise

```
%% NOISE
SNR = 25; %[dB]
Psig = 0.5; % A^2/2, A=1
np = (2*Psig)/(10^(SNR/10)*fs_sim); % noise power
```

Listing 3.17: MATLAB Code for White Noise

This chapter deals with the topic how the control loop works with an overlapping white noise and how small the SNR is allowed to be to guarantee a proper working control loop. At Listing 3.17 the noise power gets calculated and this value is included at the Simulink Block Band-Limited White Noise of Figure 3.69. Figure 3.70 shows the simulation of the angle of the mirror with a SNR of 25 dB. A zoom, around the DPLL enabling time of 0.22 s, of this simulation is shown at Figure 3.71. It is visible that the signal extracting output sqr_out has some incorrect impulses which get filtered by the hangover function of the DPLL. So the mirror can still work properly.

At a SNR of 20 dB the control loop doesn't work correctly any more and the mirror can't swing to its resonance frequency like shown in Figure 3.72. The zoom of this simulation, Figure 3.73 shows some more incorrectly produced impulses of the signal extracting part with which the control loop can't catch the mirror and isn't able to bring it to its resonance frequency.

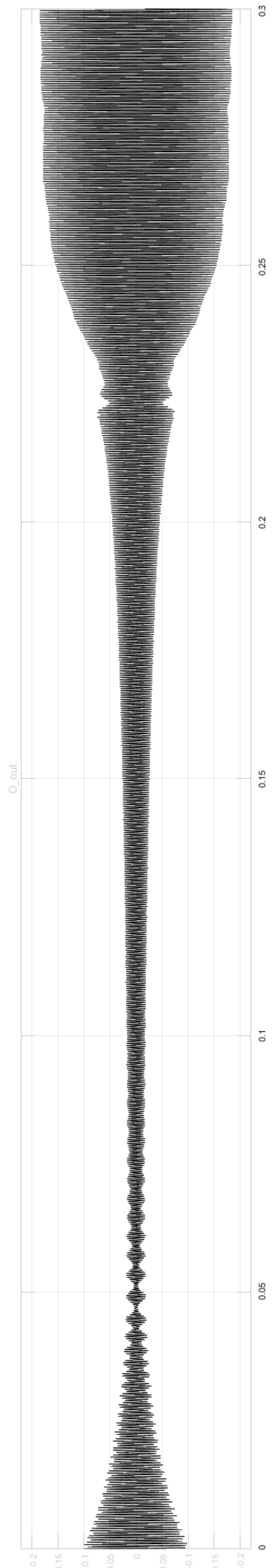


Figure 3.70: Start-up Simulation of MEMS-Mirror with 25 dB SNR

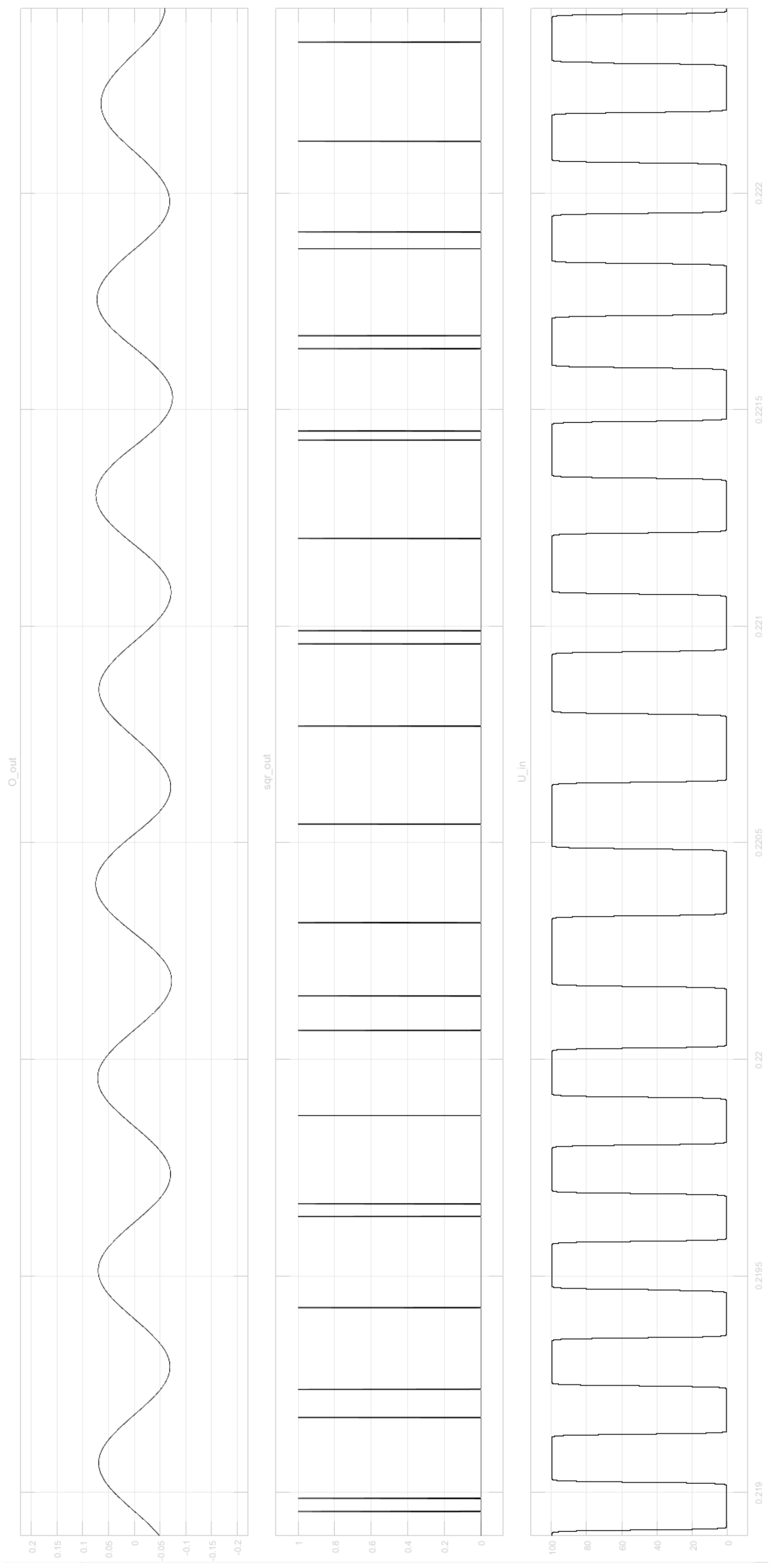


Figure 3.71: Zoom of Start-up Simulation of MEMS-Mirror with 25 dB SNR

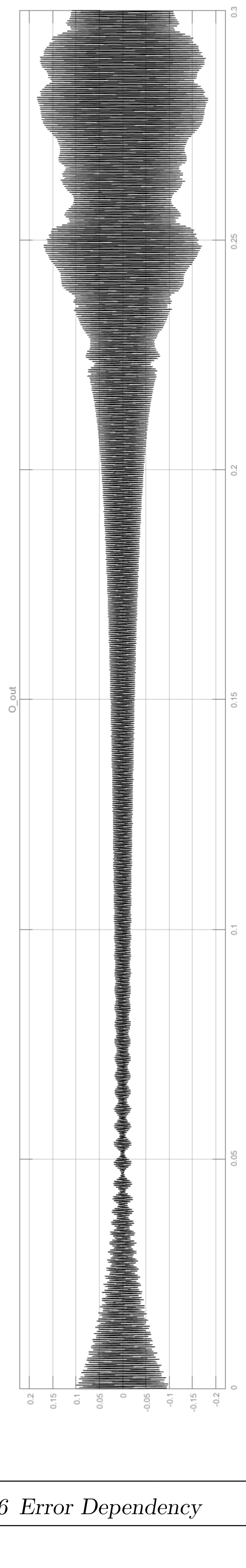


Figure 3.72: Start-up Simulation of MEMS-Mirror with 20 dB SNR

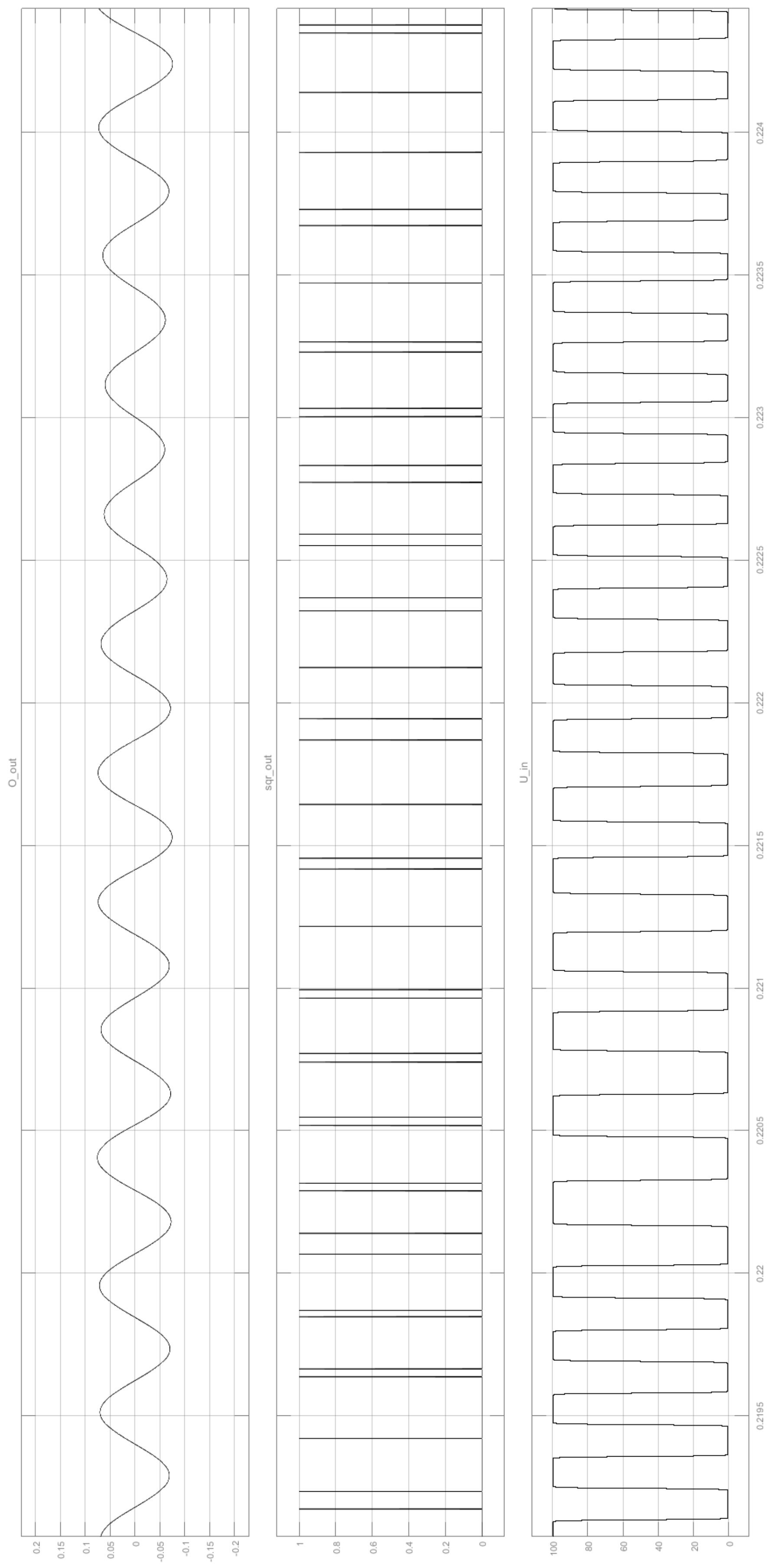


Figure 3.73: Zoom of Start-up Simulation of MEMS-Mirror with 20 dB SNR

4 VHDL and Synthesis

This chapter is about the VHDL code for the whole control loop of the MEMS mirror. For that reason a testbench got created which should behave like the Matlab Simulink Model. The following written code is synthesis-able to simulate it and afterwards, to try it on an FPGA board.

4.1 RTL Code

4.1.1 Testbench

The testbench, Listing 4.1, is at the same time the top level of the simulation. This is because the control loop is a closed loop and doesn't need any input values from outside. The entity is therefore empty like in the lines 13 to 15 shown.

At the begin of the architecture all the chosen and calculated constants are declared at the lines 13 to 41. The signals declarations to connect between components or to observe at simulation are declared from line 43 to 62. From the line 64 until the line 146 are the components declared. From the line 149 the instances of the components start until line 234. The lines 236 and 237 declare a voltage at the voltage signal u_s, which is connected to the mirror component, if the PLL generated square signal f_s is 1 and otherwise u_s is 0. The clock process, line 240 to 246, generates a clock signal for the clock period of the variable clk_period. To reset the whole simulation the reset process, lines 248 to 254, sends a negative reset signal for the first millisecond and changes to 1 afterwards. The PLL process, lines 256 to 262, enables the PLL after 220 ms like at the Matlab Simulink Model.

```vhdl
 1 --!use standard library
 2 library IEEE;
 3 --! use logic elements
 4   use IEEE.std_logic_1164.all;
 5 --! use numeric
 6   use IEEE.numeric_std.all;
 7
 8 entity mems_ctrl_tb is
 9
10 end mems_ctrl_tb;
11
12 architecture rtl of mems_ctrl_tb is
13   --constants
14   --Clock period definitions
15   constant F_CLK : integer := 84_000_000; -- 84 MHz
16   constant clk_period : time := 1 sec / F_CLK;
17   --mirror
18   constant voltage : real := 50.0; -- 50 V
19   --FM-Demodulation
20   constant counter_NBits : integer := 13;
21   constant delay_FM_DM : integer := 42; -- 500ns
22   constant counter_init : integer := 2047; -- 2^counter_NBits - 1
```

```vhdl
--moving average
constant mv_avrg_L_NBits : integer := 9;
constant mv_avrg_NBits : integer := counter_NBits + 2*mv_avrg_L_NBits + 1;
--signal extracting
constant threshold : integer := 2000000;
constant sqr_hold_NBits : integer := 5;
--clk divider
constant CLK_DIV_CNT_NBits : integer := 5; -- 2^5 = 32
constant CLK_DIV : integer := 21; -- 84 MHz to 4 MHz
--DPLL and ramp
constant DPLL_NBits : integer := 32;
constant DPLL_L : integer := 23;
constant offset_cw : integer := 8389; -- round(2*f_mir*2^L_dpll*Ts_dpll) = ↩
      2*2000*2^23*2.500000000000000e-07
constant delay : integer := 10;
constant invert : boolean := true;
constant alpha : signed(31 downto 0) := "00000000000011010001110001111111"; ↩
      -- 4.001256637061436e-04, signed fixed point
constant beta : signed(31 downto 0) := "00000000000011010001101101110001"; ↩
      -- -4.000000000000000e-04, signed fixed point
constant HANG_CNT_NBits : integer := 10;
constant hang_max_cnt : integer := 256;

--signals
--top
signal clk_s           : std_ulogic;
signal reset_n_s       : std_ulogic;
--mirror
signal u_s             : real;
signal fm_s            : std_ulogic;
--FM-Demodulation
signal dm_fm_s         : unsigned(counter_NBits-1 downto 0); -- counter_NBits-1
--moving average
signal ave_s           : unsigned(mv_avrg_NBits-1 downto 0); -- counter_NBits
--signal extracting
signal f_sqr_s         : std_ulogic;
--DPLL and ramp
signal clk_4MHz_s      : std_ulogic;
signal ramp_s          : signed(DPLL_NBits-1 downto 0);
signal pll_en_s        : std_ulogic;
signal f_s             : std_ulogic;
signal angle_s         : real;
signal capacitance_s   : real;

component clk_divider
  generic (CNT_NBits   : positive;
           CLK_DIV     : positive);
  port (
    clk_i              : in std_ulogic;
    reset_n_i          : in std_ulogic;
    div_clk_o          : out std_ulogic
  );
end component;

component mirror_first_test
  port (
    U_in               : in real;
    C_out              : out real;
    O_out              : out real;
    FM_out             : out std_ulogic
);
end component;

component fm_demodulator
  generic (delay_FM_DM    : positive;
           counter_init   : positive;
           counter_NBits : positive);
  port (
    clk_i              : in std_ulogic;
    reset_n_i          : in std_ulogic;
    FM_sqr_i           : in std_ulogic;
    DM_FM_o            : out unsigned(counter_NBits-1 downto 0)
  );
end component;

component mv_avrg
  generic (counter_NBits : positive;
           mv_avrg_NBits : positive;
           L_NBits       : positive);
  port (
    clk_i              : in std_ulogic;
    reset_n_i          : in std_ulogic;
    dm_fm_i            : in unsigned(counter_NBits-1 downto 0);
    ave_o              : out unsigned(mv_avrg_NBits-1 downto 0)
  );
end component;

component c2sqr_algorithm
  generic (threshold      : positive;
           mv_avrg_NBits  : positive;
           sqr_hold_NBits : positive);
  port (
    clk_i              : in std_ulogic;
    reset_n_i          : in std_ulogic;
    ave_i              : in unsigned(mv_avrg_NBits-1 downto 0);
    f_sqr_o            : out std_ulogic
```

```vhdl
    );
  end component;

  component ramp
    port (
      clk_i              : in std_ulogic;
      enable_i           : in std_ulogic;
      reset_n_i          : in std_ulogic;
      ramp_o             : out signed(DPLL_NBits-1 downto 0)
    );
  end component;

  component pll
    generic (DPLL_NBits      : positive;
             L               : positive;
             offset_cw       : positive;
             delay           : positive;
             invert          : boolean;
             alpha           : signed(31 downto 0);
             beta            : signed(31 downto 0);
             HANG_CNT_NBits  : positive;
             hang_max_cnt    : positive);
    port (
      clk_i              : in std_ulogic;
      enable_i           : in std_ulogic;
      reset_n_i          : in std_ulogic;
      ramp_i             : in signed(DPLL_NBits-1 downto 0);
      f_ref_i            : in std_ulogic;
      f_o                : out std_ulogic
    );
  end component;

begin
  clk_4MHz_u : clk_divider
    generic map (
      CNT_NBits      => CLK_DIV_CNT_NBits,
      CLK_DIV        => CLK_DIV
    )
    port map (
      clk_i          => clk_s,
      reset_n_i      => reset_n_s,
      div_clk_o      => clk_4MHz_s
    );

  mirror_fm_u : mirror_first_test
    port map (
      U_in           => u_s,
      C_out          => capacitance_s,
      O_out          => angle_s,
      FM_out         => fm_s
  );

  fm_demodulator_u : fm_demodulator
    generic map (
      delay_FM_DM    => delay_FM_DM,
      counter_init   => counter_init,
      counter_NBits  => counter_NBits
    )
    port map (
      clk_i          => clk_s,
      reset_n_i      => reset_n_s,
      FM_sqr_i       => fm_s,
      DM_FM_o        => dm_fm_s
    );

  mv_avrg_u : mv_avrg
    generic map (
      counter_NBits  => counter_NBits,
      mv_avrg_NBits  => mv_avrg_NBits,
      L_NBits        => mv_avrg_L_NBits
    )
    port map (
      clk_i          => clk_s,
      reset_n_i      => reset_n_s,
      dm_fm_i        => dm_fm_s,
      ave_o          => ave_s
    );

  c2sqr_algorithm_u : c2sqr_algorithm
    generic map (
      threshold       => threshold,
      mv_avrg_NBits   => mv_avrg_NBits,
      sqr_hold_NBits  => sqr_hold_NBits
    )
    port map (
      clk_i          => clk_s,
      reset_n_i      => reset_n_s,
      ave_i          => ave_s,
      f_sqr_o        => f_sqr_s
    );

  ramp_u : ramp
    port map (
      clk_i          => clk_4MHz_s,
      enable_i       => pll_en_s,
      reset_n_i      => reset_n_s,
      ramp_o        => ramp_s
    );

  pll_u : pll
    generic map (
      DPLL_NBits     => DPLL_NBits,
      L              => DPLL_L,
      offset_cw      => offset_cw,
      delay          => delay,
      invert         => invert,
      alpha          => alpha,
      beta           => beta,
      HANG_CNT_NBits => HANG_CNT_NBits,
      hang_max_cnt   => hang_max_cnt
    )
    port map (
      clk_i         => clk_4MHz_s,
      enable_i      => pll_en_s,
      reset_n_i     => reset_n_s,
      ramp_i        => ramp_s,
      f_ref_i       => f_sqr_s,
      f_o           => f_s
    );

  u_s <= voltage when f_s = '1' else
         0.0;

  --processes
  clk_process: process
  begin
    clk_s <= '0';
    wait for clk_period/2;
    clk_s <= '1';
    wait for clk_period/2;
  end process;

  rst_process: process
  begin
    reset_n_s <= '0';
    wait for 1 ms;
    reset_n_s <= '1';
    wait;
  end process;

  pll_process: process
  begin
    pll_en_s <= '0';
    wait for 220 ms;
    pll_en_s <= '1';
    wait;
  end process;
end rtl;
```

Listing 4.1: VHDL MEMS CONTROL TESTBENCH

The clock divider, Listing 4.2 is the component which divides the clock to a clock of common multiple. For that reason the clock divider has two parameters. One to declare the dividend, CLK_DIV at line 10 and the second to declare the amount of bits to represent this dividend, CNT_NBits at line 9. At Line 12 the incoming clock, clk_i, at line 13 the incoming negative reset, reset_n_i and at line 14 the outgoing clock, div_clk_o, get declared. At the lines 19 to 21 the intern signals get declared. Cnt_s is a counter, clk_s the temporally produced clock and next_start the boolean value to ensure the clock divider always starts at a falling edge of the incoming clock independent of the reset. This guarantees the proper work of it. At the process, from line 24, all signals get reset from line 27 to line 29. Line 30 begins the process for each event, falling or rising, of the clock. To ensure the correct start of the clock divider the next_start boolean has to be set first at a falling clock event, lines 31 and 32. At the next event, rising clock, the counter starts to count, lines 36 and 37, until it reaches the proposed dividend, toggles the temporal clock signal and resets the counter again, lines 33 to 35. This temporal clock signal is connected to the output div_clk_o, line 42.

```vhdl
1 --!use standard library
2 library IEEE;
3 --! use logic elements
4   use IEEE.std_logic_1164.all;
5 --! use numeric
```

```vhdl
  use IEEE.numeric_std.all;

entity clk_divider is
  generic (CNT_NBits : positive;
           CLK_DIV   : positive);
  port (
    clk_i       : in std_ulogic;
    reset_n_i   : in std_ulogic;
    div_clk_o   : out std_ulogic
  );
end clk_divider;

architecture rtl of clk_divider is
  signal cnt_s          : unsigned(CNT_NBits-1 downto 0);
  signal clk_s          : std_ulogic;
  signal next_start     : boolean;

begin
  CLK_DIV_proc: process(clk_i, reset_n_i)
  begin
    if (reset_n_i = '0') then
      cnt_s <= (others => '0');
      clk_s <= '0';
      next_start <= false;
    elsif (clk_i'event) then
      if (next_start = false and clk_i = '0') then
        next_start <= true;
      elsif ((cnt_s = (CLK_DIV-1)) and (next_start = true)) then
        clk_s <= not clk_s;
        cnt_s <= (others => '0');
      elsif (next_start = true) then
        cnt_s <= cnt_s + 1;
      end if;
    end if;
  end process;

  div_clk_o <= clk_s;
end rtl;
```

Listing 4.2: VHDL CLOCK DIVIDER

4.1.2 Analog Part

Because the analog part is the real part of the mirror, including its frequency modulation, this part got translated from Matlab to a SystemC model and inbound in a mixed signal simulation. For that reason the code isn't demonstrated in this chapter.

4.1.3 FM-Demodulation

The VHDL FM-Demodulator is exactly the same FM-Demodulator 3rd version of the Matlab Simulink Chapter 3.2.3 and is described in detail in this chapter.
The lines 9 to 11 declare the parameter for the size of the delay of the FM signal before the XOR, delay_FM_DM, the parameter for the initial value of the counter, counter_init and the parameter counter_NBits which declares the amount of bits to represent the counter and sample and hold value which is the demodulated FM signal output at the same time. The FM-Demodulator has a clock, negative reset and FM signal input, as well a FM demodulated signal output, lines 13 to 16. The signals between the parts of the FM-Demodulator are declared at the lines 21 to 26.
The reset sets all signals to 0 and the counter to its initial value, lines 31 to 36. The FM-Demodulating process starts at each rising edge of the clock, line 37. The delay_shift at line 38 shifts the vector at each step and adds the incoming FM_sqr_i bit. At line 39 xor_delayed_s gets set to xor_s to delay the XOR signal for one step. If the delayed XOR signal is 0 and the actual one is 1 then a rising is detected and rise_delayed_s gets set to 1 and the sample and hold value gets set to the actual counter value, lines 40 to 42. Otherwise rise_delayed_s is 0, line 44. If the previous step detected a rising the counter gets reset to the initial value again, lines 46 and 47. Otherwise the counter drops one step, lines 49 to 53. Line 58 is a continuous XOR of the incoming FM and the delayed FM signal. Line 59

connects the sample and hold signal with the demodulated FM output.

```vhdl
--!use standard library
library IEEE;
--! use logic elements
  use IEEE.std_logic_1164.all;
--! use aritmetics
  use IEEE.numeric_std.all;

entity fm_demodulator is
  generic (delay_FM_DM   : positive;
           counter_init  : positive;
           counter_NBits : positive);
  port (
    clk_i         : in std_ulogic;
    reset_n_i     : in std_ulogic;
    FM_sqr_i      : in std_ulogic;
    DM_FM_o       : out unsigned(counter_NBits-1 downto 0)
  );
end fm_demodulator;

architecture rtl of fm_demodulator is
  signal delay_shift    : std_ulogic_vector(delay_FM_DM-1 downto 0);
  signal xor_s          : std_ulogic;
  signal xor_delayed_s  : std_ulogic;
  signal rise_delayed_s : std_ulogic;
  signal counter_s      : unsigned(counter_NBits-1 downto 0);
  signal sh_s           : unsigned(counter_NBits-1 downto 0);

begin
  FM_DM_proc: process(clk_i, reset_n_i)
  begin
    if (reset_n_i = '0') then
      delay_shift    <= (others => '0');
      xor_delayed_s  <= '0';
      rise_delayed_s <= '0';
      counter_s      <= to_unsigned(counter_init, counter_NBits);
      sh_s           <= (others => '0');
    elsif (clk_i'event and clk_i = '1') then
      delay_shift   <= delay_shift((delay_FM_DM-2) downto 0) & FM_sqr_i;
      xor_delayed_s <= xor_s;
      if (xor_s = '1' and xor_delayed_s = '0') then
        rise_delayed_s <= '1';
        sh_s           <= counter_s;
      else
        rise_delayed_s <= '0';
      end if;
      if (rise_delayed_s = '1') then
        counter_s   <= to_unsigned(counter_init, counter_NBits);
      else
        if (counter_s = (counter_s'range => '0')) then
          counter_s <= (others => '1');
        else
          counter_s <= counter_s - 1;
        end if;
      end if;
    end if;
  end process;

  xor_s   <= FM_sqr_i xor delay_shift(delay_shift'high);
  DM_FM_o <= sh_s;
end rtl;
```

Listing 4.3: VHDL FM-Demodulator

The moving average, Listing 4.4 smooths the FM demodulated signal. The parameter counter_NBits, line 9, declares the amount of bits for the incoming signal, dm_fm_i, the parameter mv_avrg_NBits, line 10, declares the amount of bits of the moving average and the scaled incoming signal and the parameter L_NBits, line 11, declares the amount of bits for the scaling. The inputs, lines 13 to 15, are the clock, the negative reset and the FM demodulated signal. The output, line 16, is the moving average.

The two signals are the delayed and scaled moving average, line 21 and the summation of the two scaled signals, line 22, which is connected to the output.

At the reset, at line 28, the delayed moving average signal gets set to 0. At each positive clock cycle the delayed moving average gets set to the subtraction of the moving average and the scaled moving average, line 30. A right shift of a bit vector corresponds to a scaling with an integer of 2 to the power of X and X is an integer from 0 to infinite. At line 34 the incoming FM demodulated signal gets scaled and then resized to add it to the delayed moving average signal where the result the moving average signal is.

```vhdl
--!use standard library
library IEEE;
--! use logic elements
  use IEEE.std_logic_1164.all;
--! use aritmetics
  use IEEE.numeric_std.all;

entity mv_avrg is
  generic (counter_NBits : positive;
           mv_avrg_NBits : positive;
           L_NBits       : positive);
  port (
    clk_i          : in std_ulogic;
    reset_n_i      : in std_ulogic;
    dm_fm_i        : in unsigned(counter_NBits-1 downto 0);
    ave_o          : out unsigned(mv_avrg_NBits-1 downto 0)
  );
end mv_avrg;

architecture rtl of mv_avrg is
  signal delayed_s  : unsigned(mv_avrg_NBits-1 downto 0);
  signal added_s    : unsigned(mv_avrg_NBits-1 downto 0);

begin
  mv_avrg_proc: process(clk_i, reset_n_i)
  begin
    if (reset_n_i = '0') then
      delayed_s     <= (OTHERS => '0');
    elsif (clk_i'event and clk_i = '1') then
      delayed_s     <= added_s - shift_right(added_s, L_NBits);
    end if;
  end process;

  added_s           <= resize(shift_right(dm_fm_i & "0000000000000000000", ↩
        L_NBits), counter_NBits+1) + delayed_s;
  ave_o             <= added_s;
end rtl;
```

Listing 4.4: VHDL MOVING AVERAGE

4.1.4 Signal Extracting

Listing 4.5 is about the signal extracting algorithm. Line 9 declares the parameter threshold which defines the difference between the maximum input value and the actual value to toggle from rising to falling state and the difference between the minimum input value and the actual value to toggle from falling to rising state, respectively. Lines 10 and 11 declare the amount of bits of the input signal and the outgoing signal in which this is the length of the high impulse of the output.
The inputs, lines 13 to 15, are the clock, the negative reset and the incoming moving average signal. The output, line 16, is the generated rectangular pulse signal with the sample length of sqr_hold_NBits.
The signal f_sqr_s, line 21, is used as a counter to count the sample time of the outgoing high impulse. Updown_s, line 22, declares the falling and rising state and last_val_s, line 23, caches the last maximum and minimum value, respectively.
The reset, lines 28 to 31, sets all signals to 0 apart of last_val_s to a bit vector of ones. This is because the initial state is a falling incoming signal, so the last value has to be higher for the possibility to decrease it. At a rising clock signal, line 32, the real process takes place. The lines 33 to 37 describe the counter for the length of the outgoing high impulse. If the counter reaches its maximum value it gets reset to 0 again and if it is higher than 0 it gets counted up at each clock cycle. At the lines 38 and 39 the maximum incoming value gets cached at a rising signal state. At line 41 the change to a falling signal gets queried by checking if the cached maximum value is higher the actual value plus the threshold. If it is true

the state gets changed to falling, line 42 and the new cached value is the actual incoming value. The falling state, lines 45 to 52, works vice verse apart of line 51 in which the counter signal gets the value 1 to produce an rectangular impulse at the output. This happens with the aid of line 56 in which the counter signal gets queried if it is not equal 0 and if not the output is 1 until the counter is 0.

```
 1  --!use standard library
 2  library IEEE;
 3  --! use logic elements
 4    use IEEE.std_logic_1164.all;
 5  --! use aritmetics
 6    use IEEE.numeric_std.all;
 7
 8  entity c2sqr_algorithm is
 9    generic (threshold      : positive;
10             mv_avrg_NBits  : positive;
11             sqr_hold_NBits : positive);
12    port (
13      clk_i           : in std_ulogic;
14      reset_n_i       : in std_ulogic;
15      ave_i           : in unsigned(mv_avrg_NBits-1 downto 0);
16      f_sqr_o         : out std_ulogic
17    );
18  end c2sqr_algorithm;
19
20  architecture rtl of c2sqr_algorithm is
21    signal f_sqr_s    : unsigned(sqr_hold_NBits-1 downto 0);
22    signal updown_s   : std_ulogic;
23    signal last_val_s : unsigned(mv_avrg_NBits-1 downto 0);
24
25  begin
26    alg_proc: process(clk_i, reset_n_i)
27    begin
28      if (reset_n_i = '0') then
29        f_sqr_s       <= (others => '0');
30        updown_s      <= '0';
31        last_val_s    <= (last_val_s'range => '1');
32      elsif (clk_i'event and clk_i = '1') then
33        if (f_sqr_s = (f_sqr_s'range => '1')) then
34          f_sqr_s     <= (others => '0');
35        elsif (f_sqr_s > 0) then
```

```
        f_sqr_s     <= f_sqr_s + 1;
      end if;
      if ((ave_i > last_val_s) and (updown_s = '1')) then
        last_val_s  <= ave_i;
      end if;
      if ((last_val_s > (ave_i + threshold)) and (updown_s = '1')) then
        updown_s    <= '0';
        last_val_s  <= ave_i;
      end if;
      if ((ave_i < last_val_s) and (updown_s = '0')) then
        last_val_s  <= ave_i;
      end if;
      if ((ave_i > (last_val_s + threshold)) and (updown_s = '0')) then
        updown_s    <= '1';
        last_val_s  <= ave_i;
        f_sqr_s     <= to_unsigned(1, f_sqr_s'length);
      end if;
    end if;
  end process;

  f_sqr_o           <= '1' when f_sqr_s /= 0 else '0';
end rtl;
```

Listing 4.5: VHDL SIGNAL EXTRACTING

4.1.5 DPLL

The DPLL is included in Listing 4.6. The lines 9 to 17 declare the parameter DPLL_NBits for the amount of bits of the signals, L the size of the accumulator of the NCO, offset_cw the NCO offset control word, delay the size of the delay, invert the boolean to invert the signal or not, alpha and beta the loop filter constants and hang_max_cnt and HANG_CNT_NBits the length of the hangover counter and its amount of bits.

The inputs, lines 19 to 23, are the clock, the dpll enable, the reset, the incoming ramp signal for the start-up and the reference input frequency. The output, line 24, is the outgoing frequency.

The needed signal to connect in and between the processes and to connect between the in- and outputs are declared from the line 29 until the line

51. Hang_s is the signal after and hang_cnt the counter of the hangover. State_s describes the status of the PFD with aid of the enumeration type pfd_state_t. The signals hang_last_s and delayed_last_s are the on clock cycle delayed signals after the hangover and the delay to query a falling or rising edge of these signals. Delayed_state is the one clock cycle delayed state_s. Alpha_s and beta_s are the signals after the loop filter constants and lf_s and lf_delayed are the loop filter output and delayed loop filter output, respectively. LF_scaled is the loop filter signal converted for the NCO, cw_added the signal after adding that signal to the offset control word offset_cw, cw_accu_added the signal after adding those signals to the recycling accumulator signal accu_delayed and nco_s the output signal of the NCO which is connected to the output of the dpll. The signals delay_shift and delayed_s are signals which are compensating the produced time delay and the inversion of the output of the signal extracting part, respectively.

At the hangover process, lines 54 to 71, the reset sets the outgoing signal and the counter to 0. At rising edge of the clock, if the counter reaches hang_max_cnt, it gets set to 0 again, if the value is higher than 0 it gets counted up 1 and otherwise, if the outgoing signal is not equal to the incoming reference frequency signal, the counter gets the value 1 and the output gets the value of the input.

At the phase frequency detection process, lines 73 to 100, the reset sets the PFD-state to EQUAL and the delayed hangover signal and the delayed produced signal to 0. At rising edge of the clock, if the dpll is enabled, the real PFD process starts. To change from state EQUAL to UP the hangover signal has to change from last to actual value from 0 to 1 and delayed produced signal has to have the value 0. To change to the state DOWN the delayed produced signal has to change from 0 to 1 and the hangover signal has to have the value 0. To change from state UP back to EQUAL the delayed produced signal has to have the value 1 and to change from state DOWN back to EQUAL the hangovered signal has to have the value 1. Lines 96 and 97 save their actual values to have the previous values at next step to detect rising or falling edges.

At the loop filter process, lines 102 to 113, the reset sets the delayed PFD-state to EQUAL and the delayed loop filter signal to 0. At rising edge of the clock, if the dpll is enabled, the PFD-state and the loop filter signal get saved to have their values at the next clock cycle. Lines 115 to 121 declare the continuous tasks of the loop filter. At the state UP and delayed state UP beta gets subtracted from alpha and added to the previous loop filter signal. At the state UP and delayed state DOWN alpha, beta and the previous lf signal get added. At the state DOWN and delayed state UP alpha and beta get subtracted from the previous lf signal. At the state DOWN and delayed state DOWN alpha gets subtracted from beta and the previous lf signal get added. At the state EQUAL and delayed state UP and DOWN beta gets subtracted and added to the previous lf signal, respectively. At the state UP and DOWN and the delayed state EQUAL alpha gets added and subtracted of the previous lf signal, respectively. If both states are EQUAL the previous loop filter signal gets the actual loop filter signal.

At the numerically controlled oscillator process, lines 123 to 130, the reset sets the delayed accumulator signal to 0. At rising edge of the clock the accumulator signal gets saved to the delayed accumulator signal to have

its value at the next clock cycle. Lines 132 to 141 declare the continuous tasks of the NCO. First the loop filter signal gets scaled, by shifting left or right depending if the amount of bits of the signal is lower or higher than the accumulator size. Then the offset sontrol word gets added to the scaled loop filter signal or the incoming ramp signal depending on if the dpll is enabled or not. Afterwards the previous accumulator value gets added to that signal by additional respecting of an overlaps. In the end the NCO produces a 1 if the resulting value is bigger than the half of the maximum accumulator value. Otherwise it produces a 0.
At the phase locked loop process, lines 143 to 154, the reset sets the delay vector to 0. At rising edge of the clock, if the invert boolean is true, the delay vector gets shift for one position to the left side and the new LSB is the inverted NCO output bit. If the invert boolean is false, the NCO output doesn't get inverted at setting as LSB. The MSB of the delay vector is continuously connected to delayed_s signal, line 156, which is the second input of the PFD. Nco_s is continuously connected to the DPLL output, f_o.

```
 1  --!use standard library
 2  library IEEE;
 3  --! use logic elements
 4    use IEEE.std_logic_1164.all;
 5  --! use aritmetics
 6    use IEEE.numeric_std.all;
 7
 8  entity pll is
 9    generic (DPLL_NBits      : positive := 32;
10             L               : positive;
11             offset_cw       : positive;
12             delay           : positive;
13             invert          : boolean;
14             alpha           : signed(31 downto 0); --IEEE 754
```

```vhdl
            beta           : signed(31 downto 0); --IEEE 754
            HANG_CNT_NBits : positive;
            hang_max_cnt   : positive);
    port (
      clk_i           : in std_ulogic;
      enable_i        : in std_ulogic;
      reset_n_i       : in std_ulogic;
      ramp_i          : in signed(DPLL_NBits-1 downto 0);
      f_ref_i         : in std_ulogic;
      f_o             : out std_ulogic
    );
  end pll;

  architecture rtl of pll is
    --HANGOVER_proc
    signal hang_s          : std_ulogic;
    signal hang_cnt        : unsigned(HANG_CNT_NBits-1 downto 0);
    --PFD_proc
    type pfd_state_t is (UP, DOWN, EQUAL);
    signal state_s         : pfd_state_t;
    signal hang_last_s     : std_ulogic;
    signal delayed_last_s  : std_ulogic;
    --LF_proc
    signal delayed_state   : pfd_state_t;
    signal alpha_s         : signed(DPLL_NBits-1 downto 0);
    signal beta_s          : signed(DPLL_NBits-1 downto 0);
    signal lf_s            : signed(DPLL_NBits-1 downto 0);
    signal lf_delayed      : signed(DPLL_NBits-1 downto 0);
    --NCO_proc
    signal lf_scaled       : signed(DPLL_NBits-1 downto 0);
    signal cw_added        : signed(DPLL_NBits downto 0);
    signal cw_accu_added   : signed(DPLL_NBits+3 downto 0);
    signal accu_delayed    : signed(DPLL_NBits+3 downto 0);
    signal nco_s           : std_ulogic;
    --PLL_proc
    signal delay_shift     : std_ulogic_vector(delay-1 downto 0);
    signal delayed_s       : std_ulogic;

  begin
    HANGOVER_proc: process(clk_i, reset_n_i)
    begin
      if (reset_n_i = '0') then
        hang_s        <= '0';
        hang_cnt      <= (others => '0');
      elsif (clk_i'event and clk_i = '1') then
        if (to_integer(hang_cnt) = hang_max_cnt) then
          hang_cnt    <= to_unsigned(0, HANG_CNT_NBits);
        elsif (to_integer(hang_cnt) > 0) then
          hang_cnt    <= hang_cnt + 1;
        else
          if (hang_s /= f_ref_i) then
            hang_cnt  <= to_unsigned(1, HANG_CNT_NBits);
            hang_s    <= f_ref_i;
          end if;
        end if;
      end if;
    end process;

    PFD_proc: process(clk_i, reset_n_i)
    begin
      if (reset_n_i = '0') then
        state_s         <= EQUAL;
        hang_last_s     <= '0';
        delayed_last_s <= '0';
      elsif (clk_i'event and clk_i = '1') then
        if (enable_i = '1') then
          if (state_s = EQUAL) then
            if (hang_s = '1' and hang_last_s = '0' and delayed_s = '0') then
              state_s <= UP;
            elsif (hang_s = '0' and delayed_s = '1' and delayed_last_s = '0') then
              state_s <= DOWN;
            end if;
          elsif (state_s = UP) then
            if (delayed_s = '1') then
              state_s <= EQUAL;
            end if;
          else
            if (hang_s = '1') then
              state_s <= EQUAL;
            end if;
          end if;
          hang_last_s    <= hang_s;
          delayed_last_s <= delayed_s;
        end if;
      end if;
    end process;

    LF_proc: process(clk_i, reset_n_i)
    begin
      if (reset_n_i = '0') then
        delayed_state <= EQUAL;
        lf_delayed    <= (others => '0');
      elsif (clk_i'event and clk_i = '1') then
        if (enable_i = '1') then
          delayed_state <= state_s;
          lf_delayed    <= lf_s;
        end if;
      end if;
    end process;

    alpha_s          <= alpha when state_s = UP  else
                        -alpha when state_s = DOWN else
                        (others => '0'); -- conditional
    beta_s           <= -beta when delayed_state = UP else
                        beta when delayed_state = DOWN else
                        (others => '0'); -- conditional
    lf_s             <= (alpha_s + beta_s + lf_delayed);

    NCO_proc: process(clk_i, reset_n_i)
    begin
      if (reset_n_i = '0') then
        accu_delayed  <= (others => '0');
      elsif (clk_i'event and clk_i = '1') then
        accu_delayed  <= cw_accu_added;
      end if;
    end process;

    lf_scaled        <= -shift_left(-lf_s, L-DPLL_NBits-1) when DPLL_NBits < L ↩
        and lf_s < 0 else
                        shift_left(lf_s, L-DPLL_NBits-1) when DPLL_NBits < L else
                        -shift_right(-lf_s, DPLL_NBits-L+1) when lf_s < 0 else
                        shift_right(lf_s, DPLL_NBits-L+1);
    cw_added         <= resize(ramp_i, DPLL_NBits+1) + to_signed(offset_cw, ↩
        DPLL_NBits+1) when enable_i = '0' else
                        resize(lf_scaled, lf_scaled'length+1) + ↩
                            to_signed(offset_cw, DPLL_NBits+1);
    cw_accu_added    <= (cw_added + accu_delayed - 2**L) when (cw_added + ↩
        accu_delayed) > (2**L - 1) else
                        (cw_added + accu_delayed);
    nco_s            <= '1' when cw_accu_added > (2**(L-1)) else
                        '0';

    PLL_proc: process(clk_i, reset_n_i)
    begin
      if (reset_n_i = '0') then
        delay_shift   <= (others => '0');
      elsif (clk_i'event and clk_i = '1') then
        if (invert = true) then
          delay_shift <= delay_shift((delay-2) downto 0) & not nco_s;
        else
          delay_shift <= delay_shift((delay-2) downto 0) & nco_s;
        end if;
      end if;
    end process;

    delayed_s          <= delay_shift(delay_shift'high);
    f_o                <= nco_s;
  end rtl;
```

Listing 4.6: VHDL DPLL

Listing 4.7 produces the ramp signal which is necessary for the mirror start-up. The inputs, lines 10 to 12, are the clock, the DPLL enable and the negative reset signal. The output, line 13, is the produced ramp.
The two, at the lines 18 and 19, declared signals are ramp_s, which is connected to the output and ramp_cnt, which is the counter to count the clock cycles to decrease the ramp for one step.
At the ramp process, lines 22 to 39, the reset sets the ramp signal to its initial value of 2517 which corresponds to a frequency of 5200 Hz if the offset control word gets added. The ramp counter gets set to 0. At rising edge of the clock, if the DPLL isn't enabled yet, ramp_cnt gets increased by 1 until it reaches the value 398. Then ramp_s gets decreased by 1 and the counter gets reset again. This corresponds a drop of 500 Hz each 100 ms.

```
 1 --!use standard library
 2 library IEEE;
 3 --! use logic elements
 4   use IEEE.std_logic_1164.all;
 5 --! use numeric
 6   use IEEE.numeric_std.all;
 7
 8 entity ramp is
 9   port (
10     clk_i             : in  std_ulogic;
11     enable_i          : in  std_ulogic;
12     reset_n_i         : in  std_ulogic;
13     ramp_o            : out signed(31 downto 0)
14   );
```

```
end ramp;

architecture rtl of ramp is
  signal ramp_s      : signed(31 downto 0);
  signal ramp_cnt    : unsigned(8 downto 0); -- 2^9 = 512 -> 400 needed

begin
  ramp_process: process(clk_i, reset_n_i)
  begin
    if (reset_n_i = '0') then
      ramp_s         <= to_signed(2517, 32); -- ↩
            round((ramp_start-cw_offset)*2^L_dpll*Ts_dpll) = ↩
            (5200-4000)Hz*2^23*2.500000000000000e-07 = 2517
      -- round((ramp_end-cw_offset)*2^L_dpll*Ts_dpll) = ↩
            (200-4000)Hz*2^23*2.500000000000000e-07 = -7969 after 1s => ↩
            -500Hz/100ms
      -- 2516 should be after 100ns -> each 400th step
      ramp_cnt       <= (others => '0');
    elsif (clk_i'event and clk_i = '1') then
      if (enable_i = '0') then
        if (ramp_cnt = 398) then
          ramp_s     <= ramp_s - 1;
          ramp_cnt   <= (others => '0');
        else
          ramp_cnt   <= ramp_cnt + 1;
        end if;
      end if;
    end if;
  end process;

  ramp_o             <= ramp_s;
end rtl;
```

Listing 4.7: VHDL RAMP

4.2 Simulation

Figure 4.1 shows the simulation of the VHDL control loop. Figure 4.2 just shows the capacitance and the loop filter signal of the same simulation. With the aid of the capacitance lapse, which is analog to the angle lapse, it is shown that the mirror settles down to its resonance frequency and stays stable at it. The loop filter just corrects a few times and stays constant at a certain value. This simulation is completely analogous to the MATLAB Simulink simulation and proves the functionality of the code.

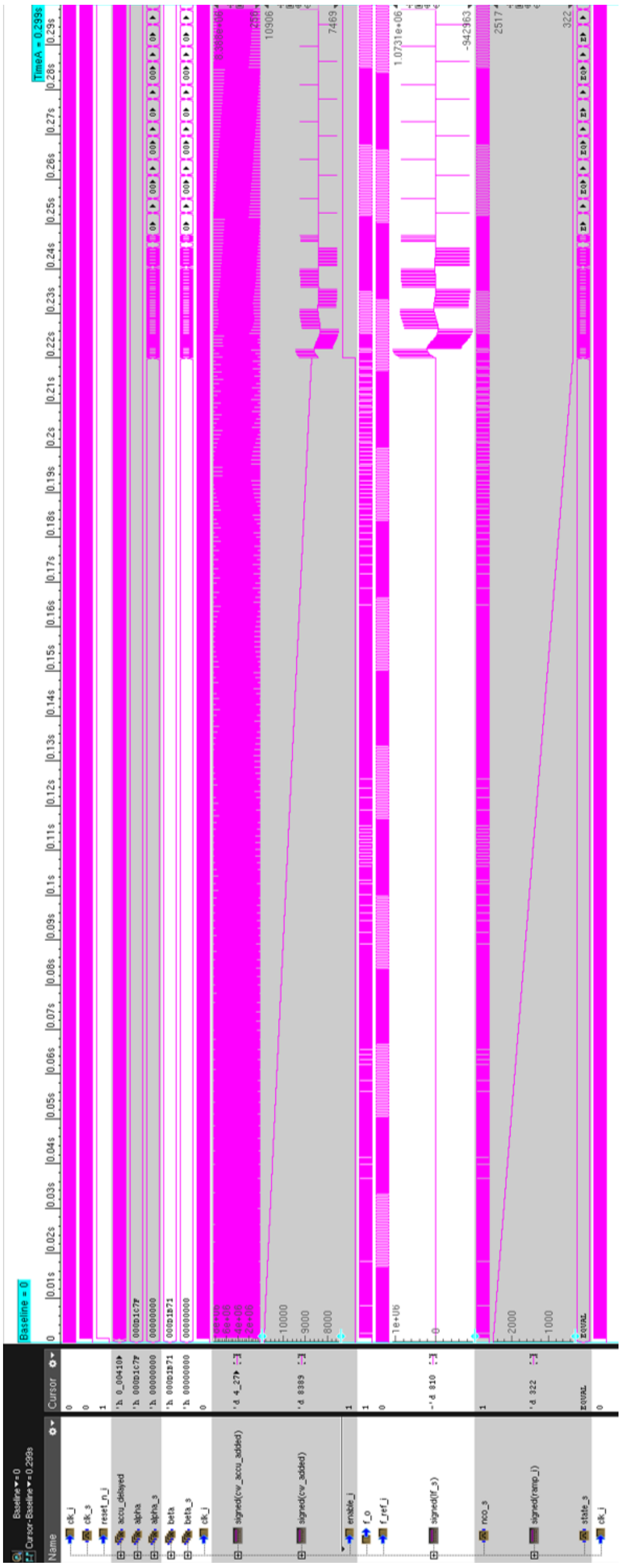


Figure 4.1: VHDL Simulation of Whole Control Loop

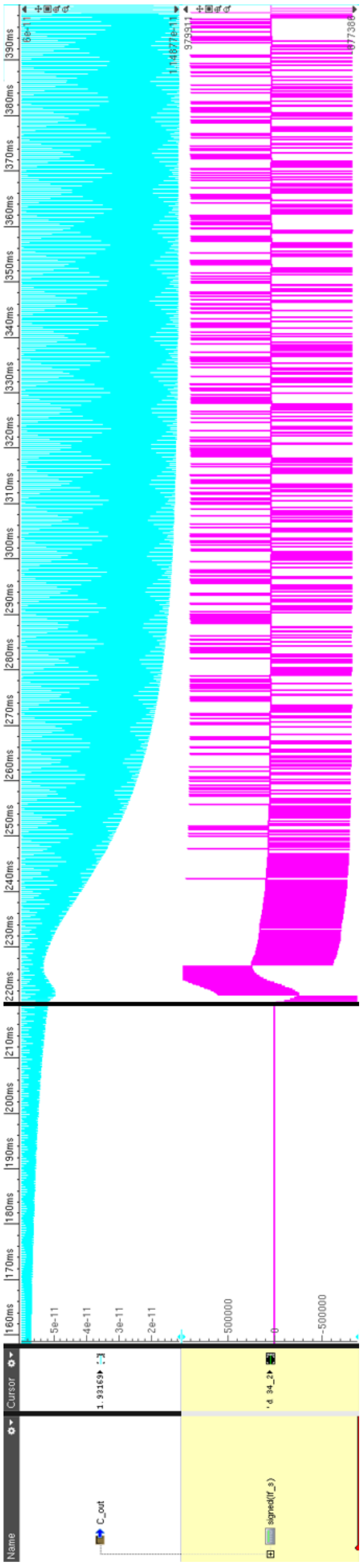


Figure 4.2: VHDL Simulation of Capacitance and Loop Filter

5 Conclusion and Outlook

In this master's thesis an all digital closed control loop got developed to swing a MEMS Scanning Mirror to its resonance frequency and to keep it there. Some non-linear attitudes of the mirror got discussed and the reason why the start-up has to drop of a higher frequency than its resonance frequency is. Three common methods for FM-Demodulation got tested and a signal extracting algorithm got created. An All Digital PLL got developed by choosing such a damping and natural angular frequency value of it. The whole control loop got tested on its error dependency by downgrading the SNR of the FM signal with an overlapping white noise. Also a VHDL code with exactly the same behaviour as the MATLAB-Simulink Model got written to simulate the control loop in a testbench. The resume is that it is possible to control the MEMS Scanning Mirror in an all digital way, though it is not the best method if resources are scarce. The best results were provided by the method which needs the most resources. In the range of this master's thesis no synthesis for the VHDL code could be done neither the FPGA implementation. Most of the time got used to implement the mathematical model of the MEMS Scanning Mirror and to find a proper FM demodulation. All in all the results are satisfying and new information of digital demodulation and All Digital PLLs are the benefits.

The outlook is to find a better way of mirror movement detection presumably by using an ADC and to use the developed All Digital PLL.

Bibliography

[1] https://futurism.com/elon-musk-lidar-tech-autonomous-vehicles
Visited on 2019/11/27.

[2] https://www.blickfeld.com/de/lidar-fur-den-massenmarkt/
Visited on 2019/11/27.

[3] https://www.springerprofessional.de/springerprofessional-de/lidar-sensorik/4307788
Visited on 2019/11/27.

[4] https://www.bussgeldkatalog.org/lidar/
Visited on 2019/11/27.

[5] https://ngin-mobility.com/artikel/welche-sensoren-gibt-es-im-auto-und-wofuer-sind-sie-da/
Visited on 2019/11/27.

[6] https://eu.usatoday.com/story/tech/news/2016/08/16/ford-baidu-bet-150m-velodyne-laser-radar/88813028/
Visited on 2019/11/27.

[7] https://en.wikipedia.org/wiki/Phased-array_optics
Visited on 2019/11/27.

[8] Yoo, H.W., Druml, N., Brunner, D. et al. MEMS-based lidar for autonomous driving. Elektrotech. Inftech. 135, 408–415 (2018) doi:10.1007/s00502-018-0635-2

[9] https://www.brandeins.de/magazine/brand-eins-wirtschaftsmagazin/2006/verkaufen/was-sind-eigentlich-mems
Visited on 2019/11/27.

[10] Karl KüpfmüllerWolfgang MathisAlbrecht Reibiger: Theoretische Elektrotechnik, Page 752, Formula 40.63, ISBN: 978-3-642-37940-6

[11] Feynman, Leighton, Sands: The Feynman Lectures on Physics, Vol. II, Formula 8.15, ISBN: 978-0465024162

[12] Ivana Kovacic, Michael J. Brennan: The Duffing Equation, Page 139, Formula 5.1.1, ISBN: 978-0-470-97786-6

[13] Oliver Schwarzelbach: Entwicklung eines mikromechanischen Drehratensensorsystems mit nichtlinearemAnregungskonzept auf der Basis des am ISIT entwickelten PSM-X2-Prozesses, Page 61, https://core.ac.uk/download/pdf/56723694.pdf
Visited on 2019/11/27.

[14] https://elektroniktutor.de/signalkunde/fm.html
Visited on 2019/11/27.

[15] Tietze Schenk: Halbleiter-Schaltungstechnik, 16. Auflage, Page 1201, Chapter 21.4.2.3 Modulation, ISBN: 978-3662485538

[16] Tietze Schenk: Halbleiter-Schaltungstechnik, 16. Auflage, Page 1204, Chapter 21.4.2.4.2 PLL-Demodulator, ISBN: 978-3662485538

[17] Texas Instruments: WhyOversamplewhenUndersamplingcan do the Job?, https://www.ti.com/lit/an/slaa594a/slaa594a.pdf
Visited on 2019/11/27.

[18] https://elektroniktutor.de/signalkunde/fm_demod.html
Visited on 2019/11/27.

[19] https://de.wikipedia.org/wiki/Koinzidenzdemodulator
Visited on 2019/11/27.

[20] https://www.mikrocontroller.net/articles/PLL
Visited on 2019/11/27.

[21] Floyd M. Gardner: Phaselock Techniques, Third Edition, ISBN: 978-0471430636

[22] https://www.bigdata-insider.de/was-ist-matlab-a-789607/
Visited on 2019/11/27.

[23] https://www.bigdata-insider.de/was-ist-simulink-a-790857/
Visited on 2019/11/27.

[24] Markus Holisch: VHDL Grundlagen, https://www.informatik.uni-ulm.de/ni/Lehre/SS03/ProSemFPGA/VHDL-Grundlagen.pdf
Visited on 2019/11/27.

[25] Pong P. Chu: RTL Hardware Design using VHDL, ISBN: 978-0471720928

[26] Wilhelm Burger, Mark James Burge: Digitale Bildverarbeitung, Eine algorithmische Einführung mit Java, 3. Auflage, Page 113, Chapter 5.4.2 Medianfilter, ISBN: 978-3-642-04604-9

[27] Karl Heinz Böhling, Dieter Schütt: Endliche Automaten II, ISBN: 978-3411007042