

# Authenticity in Music

Low-level discrete-time modelling of mixed-signal circuits

Manuscript Project 7

<b>University</b>	University of Applied Sciences Northwestern Switzerland
<b>Curriculum</b>	Master of Science in Engineering - Electrical Engineering
<b>Author</b>	Dominik Hiltbrunner
<b>Supervisor</b>	Hanspeter Schmid
<b>Date and place</b>	Windisch, January 26, 2021

### **Abstract**

A novel method to describe mixed-signal circuits as fully discrete-time systems is presented. It uses driving-point signal-flow graphs to analyse circuits which are then transformed into the z-domain while maintaining the structure of the graph. Compared to other discretization methods, this approach has the advantage that non-idealities can be modelled in their truest form. A use case is presented in which a simple, highly non-linear mixed-signal circuit is simulated on a FPGA in real-time with parameters that can be adjusted during run-time.

**Keywords:** Signal-flow graph, discrete-time modelling, non-linear modelling, FPGA

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Chosen Hardware . . . . .	2
1.2	Report overview . . . . .	2
<b>2</b>	<b>Reverse Engineering</b>	<b>3</b>
2.1	Overview on a functional level . . . . .	3
2.1.1	The eche and delay chip PT2399 . . . . .	3
2.1.2	The self-assembly kit Der Faux . . . . .	5
2.2	ADC and DAC . . . . .	7
2.2.1	Fundamentals of Delta Modulation . . . . .	8
2.2.2	Measuring the modulated current . . . . .	10
2.2.3	Measuring the delay line . . . . .	11
2.3	Summary and verification . . . . .	14
<b>3</b>	<b>Mixed-signal modelling with signal-flow graphs</b>	<b>18</b>
3.1	Model for electrical components . . . . .	18
3.2	Modelling the whole circuit . . . . .	20
3.3	A survey of discretization methods . . . . .	20
3.4	Difference between continuous-time and discrete-time filters . . . . .	22
3.5	Challenges . . . . .	24
3.5.1	Integrators and Derivatives . . . . .	24
3.5.2	Feedback loops . . . . .	25
<b>4</b>	<b>FPGA Implementation</b>	<b>33</b>
4.1	Discrete basic blocks in Simulink . . . . .	33
4.2	How real-time adjustable blocks are emulated . . . . .	33
4.2.1	Variable gain . . . . .	34
4.2.2	Variable delay . . . . .	38
4.3	Comparator logic . . . . .	40
4.4	Fixed-point Arithmetic . . . . .	42
4.4.1	Theoretical derivation . . . . .	42
4.4.2	Numerical analysis . . . . .	43
4.4.3	Error analysis . . . . .	43
4.5	Optimization . . . . .	46

4.5.1	Hardware multiplier optimization . . . . .	46
4.5.2	Reducing memory requirements . . . . .	47
4.5.3	Optimizing the numerics . . . . .	48
4.5.4	Improving timing constraints . . . . .	52
4.6	Choosing the oversampling factor . . . . .	53
4.7	FPGA Framework and synthesis results . . . . .	54
<b>5</b>	<b>Modelling non-idealities</b>	<b>55</b>
5.1	Fundamentals of non-linear modelling . . . . .	55
5.2	Application example based on temperature dependency . . . . .	56
5.3	Slewing . . . . .	58
5.4	Meaningful non-linear effects . . . . .	61
<b>6</b>	<b>Verification</b>	<b>62</b>
<b>7</b>	<b>Conclusion</b>	<b>63</b>
7.1	Limitations of the graph-based low-level modelling . . . . .	63
7.2	Strengths of the graph-based low-level modelling . . . . .	63
7.3	Open research questions . . . . .	64
7.4	Relevancy for other research fields . . . . .	65
	<b>Declaration of authorship</b>	<b>66</b>
	<b>References</b>	<b>67</b>

## Terminology, notations, and acronyms

ADC	Analog-to-digital converter
ALM	Adaptive logic module
ALUT	Adaptive look-up table
BRAM	Block RAM (random access memory)
CSD	Canonical signed digit
DAC	Digital-to-analog converter
DEM	Demodulator
DPSFG	Driving-point signal-flow graph
ECC	Error-correcting code
FPGA	Field-programmable gate array
GBW	Gain-bandwidth product
LPF	Low-pass filter
LSB	Least significant bit
LTI	Linear time-invariant
LUT	Look-up table
MOD	Modulator
MSB	Most significant bit
Op-amp	Operational amplifier
PLL	Phase-locked loop
RMS	Root Mean Square
RTL	Register-transfer level
SFG	Signal-flow graph
SR	Slew rate
SS	State-space
VCO	Voltage-controlled oscillator
VHDL	Very high speed integrated circuit hardware description language
$\vec{x}(t)$	State vector of a dynamic system
$\vec{u}(t)$	Input vector of a dynamic system
$\vec{y}(t)$	Output vector of a dynamic system
<b>A</b>	State matrix of a state-space model
<b>B</b>	Input matrix of a state-space model
<b>C</b>	Output matrix of a state-space model
<b>D</b>	Feed-forward matrix of a state-space model
$V_{DD}$	Supply voltage, i.e. most positive voltage in the circuit
$V_{SS}$	Absolute ground, i.e. the smallest voltage in the circuit
$V_{REF}$	Reference voltage used for the virtual ground, placed in the middle of $V_{DD}$ and $V_{SS}$
$f_c$	−3 dB cut-off frequency of a filter
$T_s$	Sampling period
$f_s$	Audio sampling frequency
$f_{sim}$	Simulation frequency
$H_c(s)$	Continuous-time transfer function
$H_d(z)$	Discrete-time transfer function
$\tau$	A delay

- Every variable is assumed to be a scalar. Vectors have an arrow symbol (e.g.  $\vec{x}$ ), and matrices are written in bold (e.g. **M**).
- All models of dynamic systems are assumed to be time-invariant.
- A discrete sequence is denoted by square brackets, i.e.  $x[k] = x(k \cdot T_s)$  where  $k \in \mathbb{Z}$ .

# 1 Introduction

Institutions like the Academy of Music in Basel deal with the preservation of music. The research in this field deals with the question how an old piece of music must be performed in order to sound like the original. With the rise of electronic music, digital sound processors became more popular and were widely used since the '70s. Electronic devices suffer from decay and components will inevitably cease functioning someday. Transferring the task of preserving music into modern times hence means that electrical systems must be preserved.

Today, there are three strategies for the preservation of electronic music:

- The repair and replacement of broken hardware
- The emulation of a work's hardware in software
- The portation in a new software and hardware context

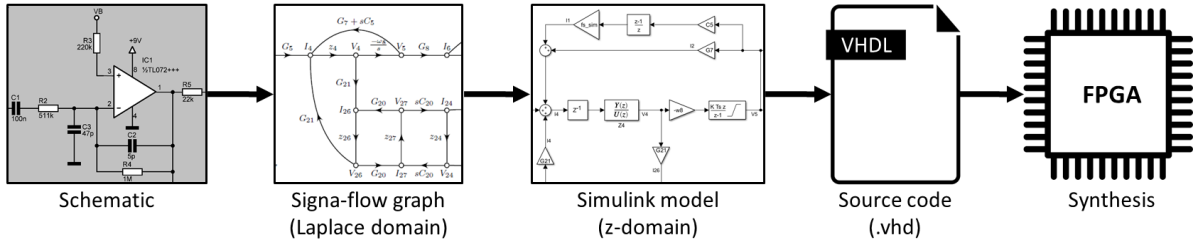
One major problem in our technologically fast-moving times is that software and hardware becomes outdated very quickly. Many components required for repair are no longer available. Replacing a broken part with an equivalent component can be difficult, especially for semiconductor devices.

Another problem arises from the fundamental, underlying question of music preservation: How close to the original work is this reinterpretation? The iconic sound of electronic music is often characterized by exploits, bugs, and glitches of its hardware. Devices are, for example, modified, driven above their limits on purpose, or even completely self-made. Simply replacing broken parts with new, available hardware cannot be a satisfying solution since those replica do not capture the characteristics of their original. Electronic music, whose quintessential sound properties are based on certain hardware, are thus considered irreplaceable nowadays.

The same problem arises when one tries to simulate a work's hardware in software. Modelling techniques for analog and digital components only capture the ideal behaviour of a circuit. Non-ideal effects such as clock jitter, harmonic distortion, temperature dependency, slewing, or saturation are difficult to imitate. Furthermore, implementing a precise model of analog components quickly results in complex systems whose computation is enormous. An authentic replica should be able to be used simultaneously together with other hardware and to be adjusted while playing. Hence, it must be possible to perform the simulation in real-time and to change parameters during run-time.

The field of simulating mixed-signal systems in a fully digital environment is not new. Recent work is available in which one not only tried to port analog and digital systems to a new hardware context, but also focused on implementing real-time simulation or real-time adjustment of analog dynamics [1]–[3]. They all have in common that they model a system on a higher level of abstraction, meaning that they look for functional blocks on a circuit, e.g. a filter, and then try to emulate this block as a whole. Non-ideal effects are then somehow imitated to match with simulation results from other, highly accurate software tools such as Spice. This approach can be very time consuming when complex, non-linear effects must be modelled. Furthermore, it is not very flexible since a change in component parameters requires a complete recomputation of the digital model.

The work in this report wants to address all those problem. We provide a novel modelling technique in which system identification is performed on a low abstraction level, that is, we handle every single component on a circuit as a separate model. Non-linear effects are then modelled for every individual component. This has the advantage that nothing else than a schematic (or net list) must be available to start with the design flow. Additionally, non-idealities are hereby implemented in their truest possible form.



**Figure 1.1:** Flowchart of the design flow.

The goal is to provide a discrete, hardware-independent description of a mixed-signal circuit that captures any kind of non-linearity. In a next step, this discrete model is implemented in hardware and compared with the original device in a black box sound test together with musicians. In an ideal case, the discrete replica delivers authentic acoustics in the sense that it cannot be distinguished from its original by professionals.

We have chosen FPGA technology since fast signal processing capabilities are required to provide a real-time model. The key tool to convert a schematic to a "low-level" model are the so-called driving-point signal-flow graphs, which will be explained later in this report. The output is a description in the Laplace domain which is then converted to the z-domain and implemented in Simulink. Finally, the Simulink model is converted to VHDL source code which is synthesized to a programmable FPGA binary. The flowchart of the complete design flow is shown in Figure 1.1.

## 1.1 Chosen Hardware

The work in this report does not only introduce the proposed modelling technique based on signal-flow graphs, but also provides a proof of concept with a concrete hardware. The chosen audio device is a mixed-signal self-assembly kit called *Der Faux* from *das musikding.de* [4]. It contains a simple sound processor of the type *PT2399* from Princeton Technology [5]. Its functionality is dependent on the external circuitry, which comes from the self-assembly kit, and produces some highly non-linear audio effects that can be adjusted with variable resistors. It is thus a suitable use case of a simple mixed-signal circuit whose characteristics mainly depend on non-ideal properties.

The model will be implemented on a *DE1-SoC* development board which contains an *Altera Cyclone V SE 5CSEMA5F31C6N FPGA* [6]. This board includes a 24-bit audio codec with variable sampling rate, and amplifiers for Line level and headphones. It can thus be easily used together with other audio equipment.

## 1.2 Report overview

This report is organized as followed: Chapter 2 analyses the chosen hardware and explains its functionality. Furthermore, some parameters which are not given in the datasheet are reverse engineered. In chapter 3, the graph-based analysis method is introduced and any hardware-independent consideration for the conversion into the z-domain is discussed. Chapter 4 shows how the general, discrete description can be implemented on a FPGA. It gives a concrete example how a real-time model can be achieved whose parameters can be adjusted during run-time. In chapter 5, it is explained how non-linear effects can be modelled. First a general theory is provided and then some specific examples for the FPGA implementation are given. Finally, the product is verified in chapter 6 and the outcome of this work is reflected in chapter 7.

## 2 Reverse Engineering

In order to model the behaviour of the echo and delay chip precisely, it is necessary to fully understand its internal functionality. Unfortunately, the *PT2399* audio chip is a black box in many aspects. The datasheet only provides a circuit in a block diagram like matter as shown in Figure 2.1. It does not explain how the analog-to-digital conversion is done nor what types of op-amps are used. Furthermore, the exact functionality of the delay line and the purpose of the *MOD* and *DEM* blocks are also unknown. The goal of this chapter is to gain a detailed insight into the *PT2399* by reverse engineering its functionality. Next, its role in the self-assembly kit *Der Faux* is analyzed.

### 2.1 Overview on a functional level

At first, the hardware shall be understood on a functional level. For this purpose, basic building blocks are identified and analyzed in order to give a brief overview about the system.

#### 2.1.1 The eche and delay chip PT2399

A better understanding can be obtained by redrawing the block diagram. Figure 2.2 shows the circuit of the *PT2399* in an alternative representation. An audio signal enters the chip at pin 16 where it is low-pass filtered by op-amp 1. The signal is then converted to a digital bitstream by the ADC which consists of a comparator (op-amp 2), the modulator block, and an integrator (op-amp 3). The bitstream is delayed by a shift register and converted to an analog signal by the demodulator block and yet another integrator (op-amp 4) before it leaves the chip at pin 12.

### BLOCK DIAGRAM

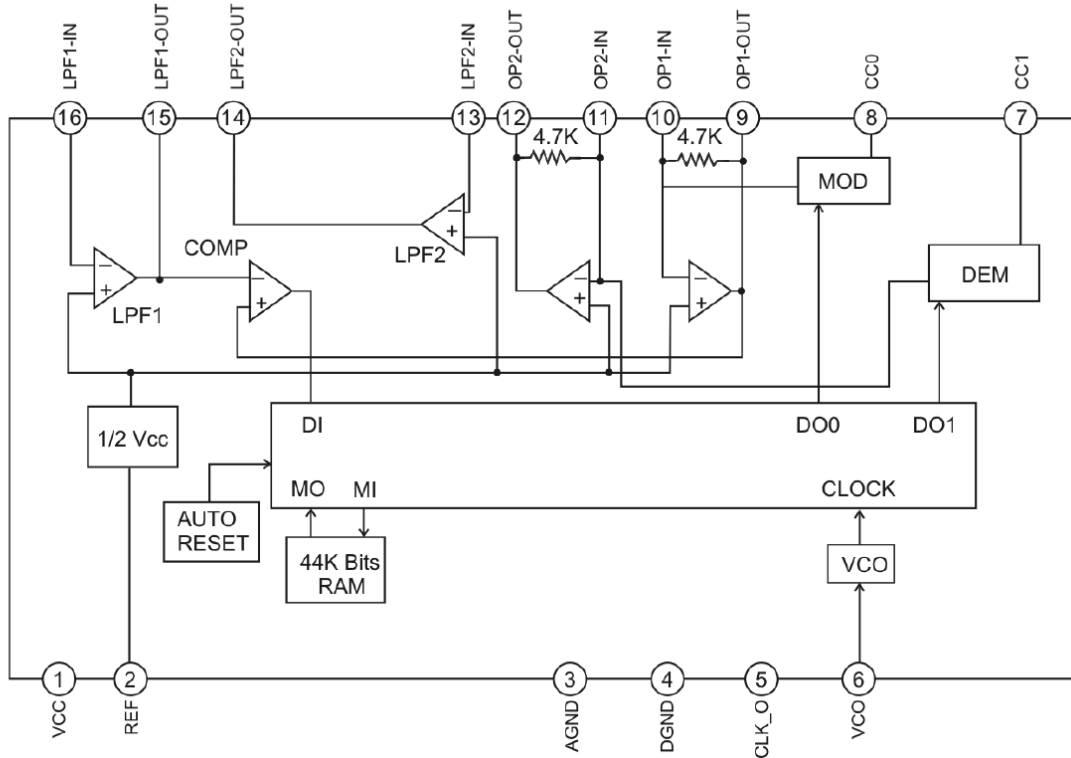
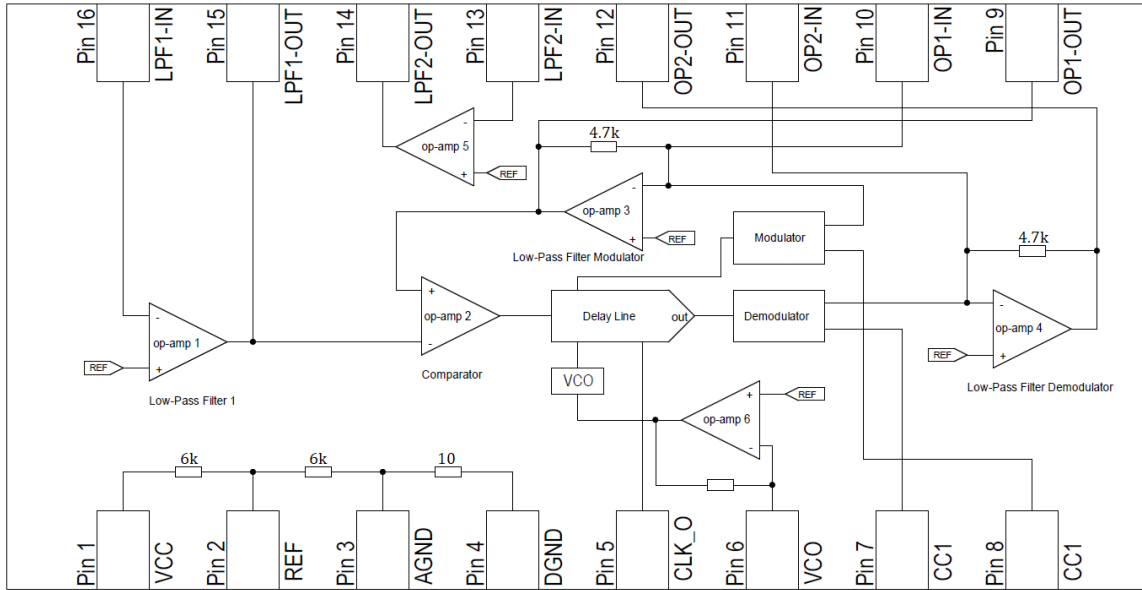


Figure 2.1: Schematic of the *PT2399* echo and delay audio chip [5].





**Figure 2.2:** Alternative representation of the PT2399's schematic.

Pin	in/out?	Name	Description
1	-	VCC	Supply voltage at 5 V. Should be stabilized with a regulator.
2	-	REF	Reference voltage at 2.5 V (virtual ground).
3	-	AGND	Analog ground. Should be shortened with pin 4.
4	-	DGND	Digital ground. Should be shortened with pin 3.
5	output	CLK_O	Clock of the VCO.
6	-	VCO	VCO frequency adjustment. This pin is connected to ground with a tunable resistor in order to change the clock frequency.
7	-	CC1	Unknown.
8	-	CC2	Unknown.
9	output	OP1-OUT	Output of op-amp 3 which is used in the modulator.
10	input	OP1-IN	Inverting input of op-amp 3. Connecting a capacitor over pin 9 and 10 is necessary to obtain the behaviour of an integrator.
11	input	OP2-IN	Inverting input of op-amp 4.
12	output	OP2-OUT	Output of op-amp 4. Connecting a capacitor over pin 11 and 12 is necessary to obtain the behaviour of an integrator.
13	input	LPF2-IN	Inverting input of op-amp 5.
14	output	LPF2-OUT	Output of op-amp 5. This op-amp is for free use.
15	output	LPF1-OUT	Output of op-amp 1.
16	input	LPF1-IN	Inverting input of op-amp 1. This op-amp is used as a low-pass filter in front of the ADC. (anti-aliasing filter)

**Table 2.1:** Purpose of each pin in the PT2399 audio chip.

The exact functionality of the ADC and DAC process is explained in section 2.2. If the chip shall produce an echo, then the output signal at pin 12 is fed back to pin 16 by the external circuitry. Op-amp 5 is not connected internally and can be used for whatever purpose the designer wants. Notice that op-amp 1,3,4, and 5 require external components to operate correctly. From now on, this report will always refer to the component numbering in Figure 2.2. Finally, Table 2.1 summarizes the purpose of each pin.

### 2.1.2 The self-assembly kit Der Faux

A complete circuit diagram of the signal path, including the internal components of the *PT2399* chip, is provided in Figure 2.3. Functional building blocks have been marked by dashed lines and their important parameters are given. Between the building blocks there are RC elements which serve as DC decoupling. Their impact has not been considered when simulating the parameters of the active filters.

The signal path starts at op-amp 7 where a band-pass filter removes any DC components as well as non-audio frequencies. Parts of the signal are feed forward directly to op-amp 8 which is the last amplifier before the signal leaves the circuit. Another part of the signal enters the *PT2399* where it is low-pass filtered before being digitized and delayed as explained in section

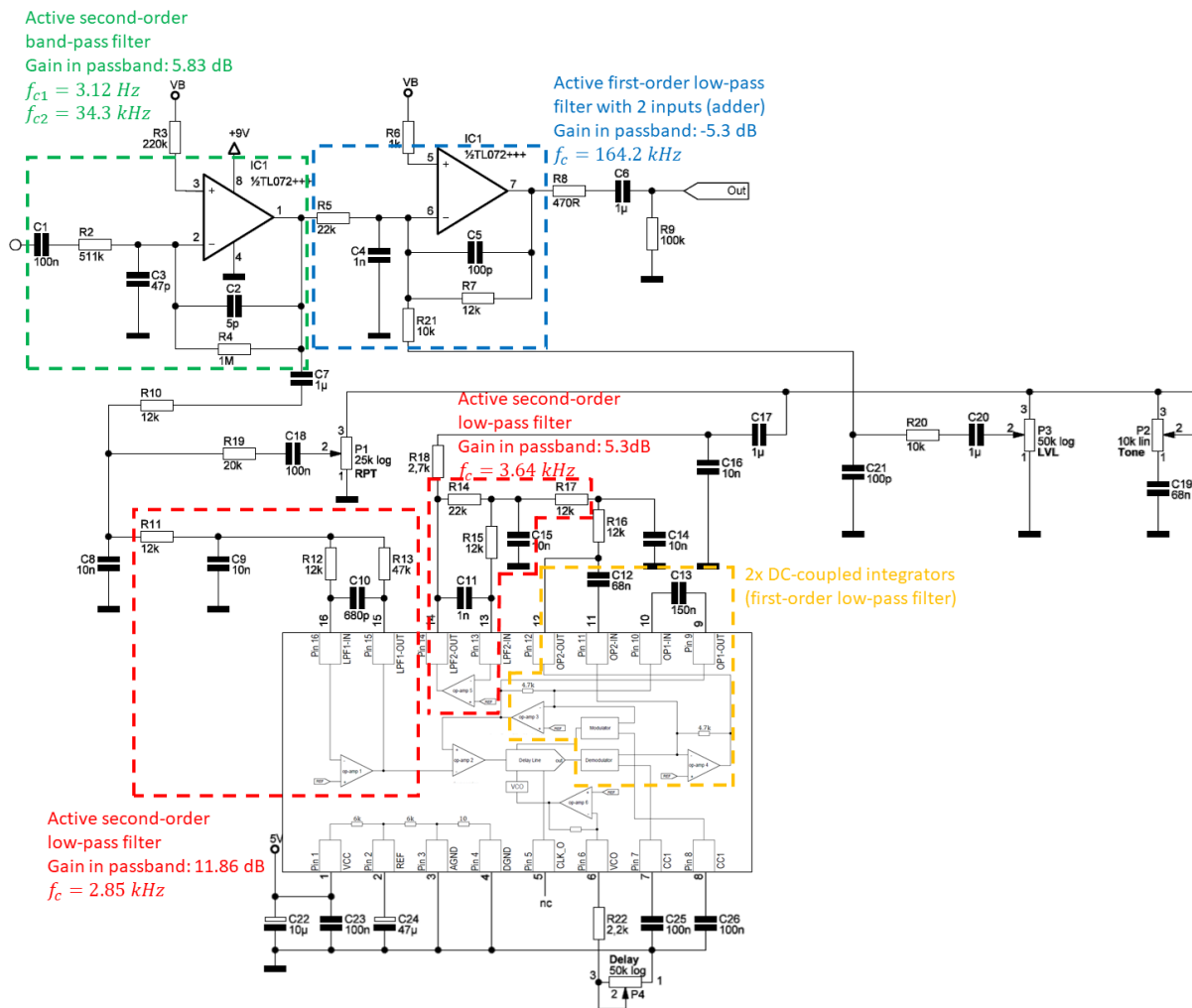


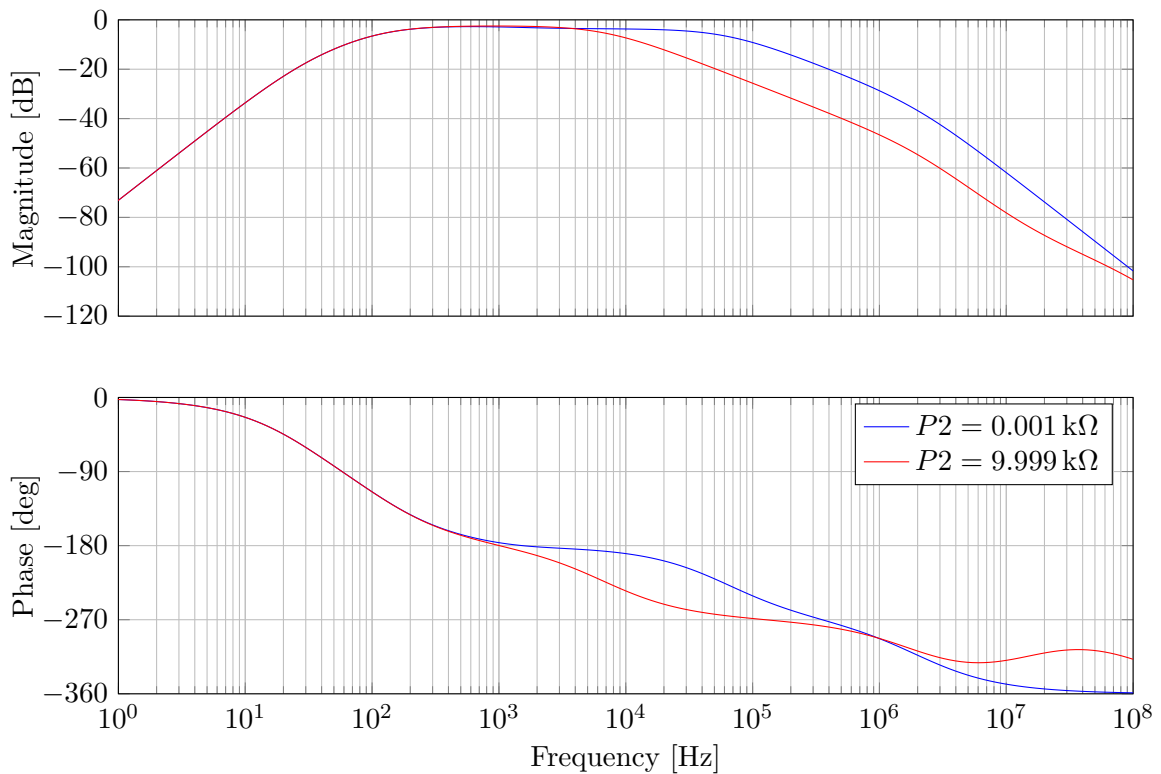
Figure 2.3: Schematic of the signal-relevant components in *Der Faux*.

2.1.1. Op-amp 5 is configured as a second-order low-pass filter and used to further filter the signal that leaves the *PT2399*.

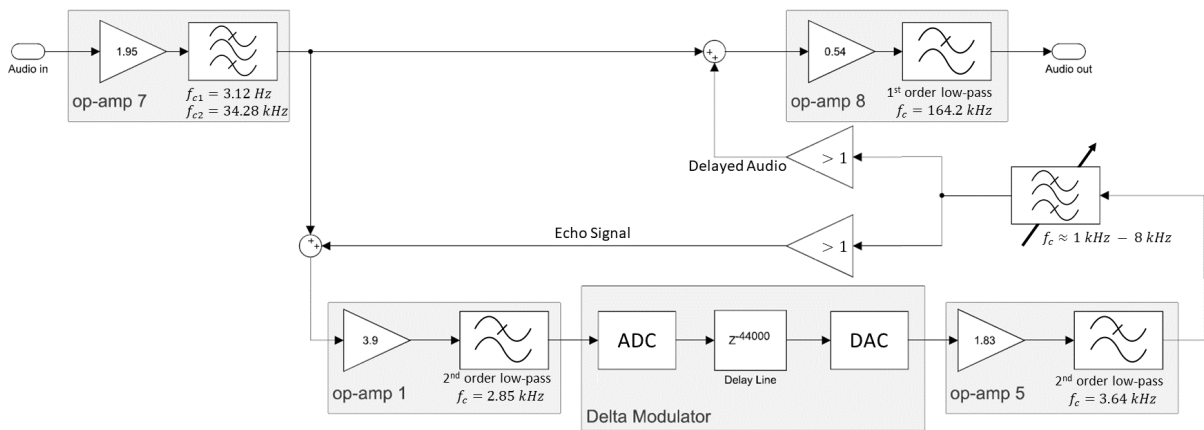
Now the variable resistors come into play. Some parts of the signal are fed back to the input to create an echo. This is controlled by potentiometer *P1* labelled as *Repeat*. *P3* controls what fraction of the signal is added by op-amp 8 together with the original audio signal. It is the *Level* regulator, but the name is a bit misleading since it actually only controls the audio level of the signal leaving the *PT2399* rather than the overall volume. *P2* is the *Tone* control and its purpose is to change the sound pattern of the circuit. Marking a functional block is a bit difficult since *P2* only changes the impedance to ground of a certain node. Finding its transfer characteristic can be done by evaluating the transfer function from the output of op-amp 5 to the output of op-amp 8. To equalize the influence of the low-pass filter formed by op-amp 8, capacitors  $C_4$  and  $C_5$  are set to zero. Additionally, op-amp 8 has infinite gain (ideal component). Next, in order to correct the gain, *P3* is set to a very low value and the whole transfer function is multiplied by the inverse of the gain which is given by the circuitry of op-amp 8. A bode plot comparing the transfer characteristics for small and large values of *P2* is shown in Figure 2.4.

Technically speaking, it is an adjustable band-pass filter where the variable resistor changes the attenuation of high frequencies. Higher values correspond to more damping in the stopband. The  $-3$  dB cut-off frequency is approximately shifted from 1 kHz to 8 kHz.

Finally, *P4* controls the voltage of a voltage-controlled oscillator (VCO) which in turn controls the delay through the shift register. The length of the shift register is fixed at 44 kbit. Different delay times are achieved by changing the clock of the register. Summarizing the functional description of the circuit, a block diagram as in Figure 2.5 can be derived.



**Figure 2.4:** Bode plot showing the transfer characteristics of the *Tone* control for minimum and maximum values of *P2*.

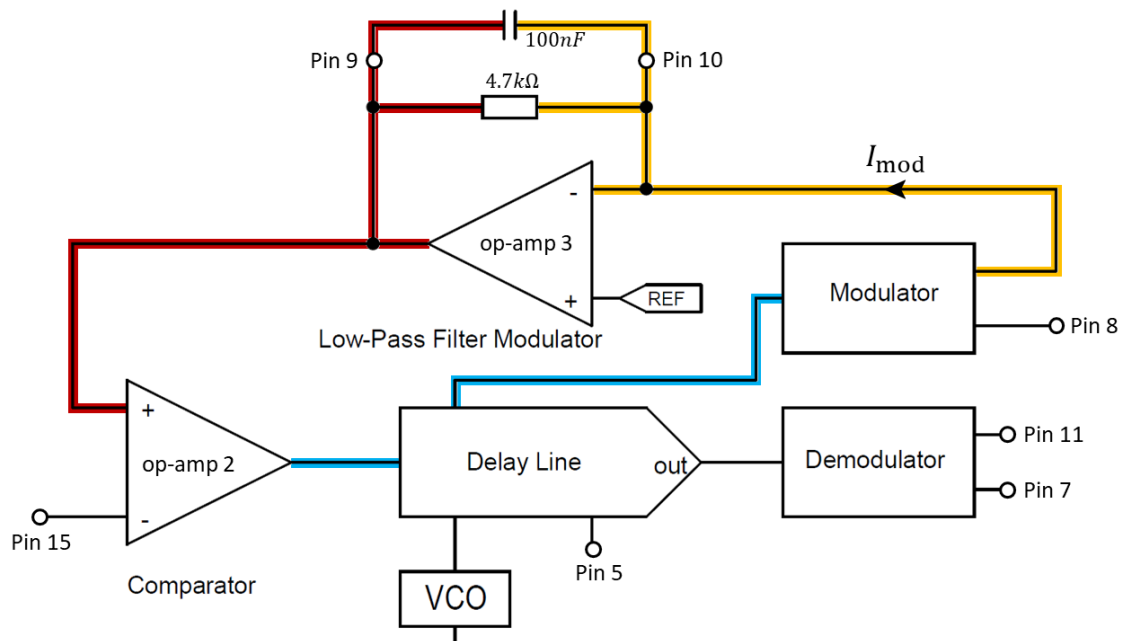


**Figure 2.5:** Block diagram of *Der Faux*.

## 2.2 ADC and DAC

The analog-to-digital conversion starts with a comparator (op-amp 2) which produces a 1-bit representation of the audio stream. It compares the low-pass filtered audio signal with a feedback signal. This feedback is an analog representation of the previously converted audio stream, produced by a modulated current signal which is integrated by op-amp 3. The whole process is illustrated in Figure 2.6.

The output of op-amp 2 is the digital representation of the audio (blue highlighted net). Op-amp 3 forces the yellow net to stay on the reference voltage at 2.5 V. A current that is proportional to the logic level of the digitized audio is applied and causes a voltage drop over the feedback components of op-amp 3. Since op-amp 3 acts like an integrator due to the feedback capacitor, the voltage at its output (red net) is an analog representation of the audio stream. In other



**Figure 2.6:** Delta modulator of the PT2399.

words, its output is an approximated version which tracks the course of the original signal. The *Modulator* block, whose functionality is not explained in the datasheet, is thus nothing else than a voltage-to-current converter which produces an output current that is related to the logic levels.

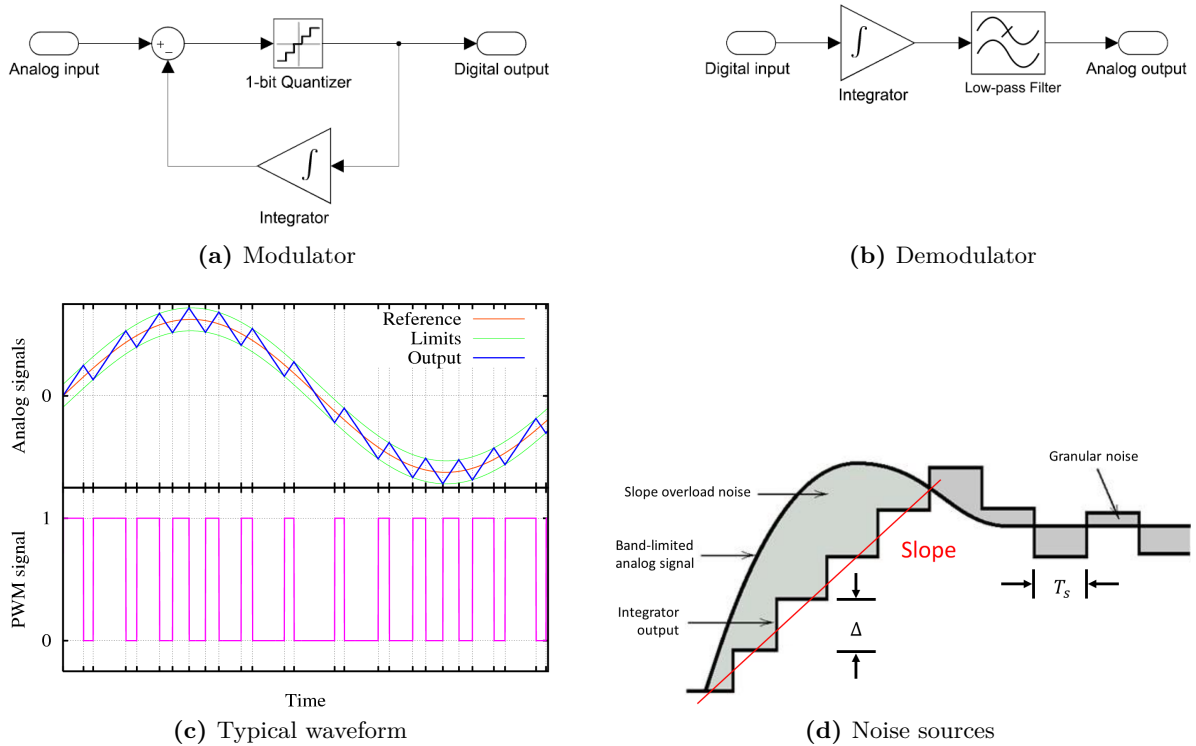
This concept of analog-to-digital conversion corresponds to so-called delta modulation, a technique used for AD-conversion in systems where high-quality is not important. Some fundamentals about delta modulation are explained in the next section before unknown system parameters are backward engineered through measurements.

### 2.2.1 Fundamentals of Delta Modulation

Delta modulation is the simplest form of differential pulse-code modulation [7]. Another term would be pulse-width modulation. Here, simple refers to the fact that the difference is encoded in a 1-bit data stream. To build a delta modulator, all that is required is a 1-bit quantizer and an integrator, as illustrated in Figure 2.7(a).

The digital data stream is used to control the integrator, i.e. it tells the integrator whether its output should increase or decrease. By that, the resulting output tracks the original signal and can be used for comparison. A typical waveform is given in Figure 2.7(c) where the upper plot shows the input and feedback signal, and the lower plot shows the decision of the quantizer.

Demodulation is simply done by smoothing the modulated signal, which is created in the same way as in the feedback loop of the modulator. Depending on some system parameters, the integrated signal jitters more or less around the original signal. If it is low-pass filtered again,



**Figure 2.7:** Concept of delta modulation, showing the modulator (ADC) in (a), the demodulator (DAC) in (b), a typical waveform in (c) [8], and a visualization of the noise sources in (d).

then the original waveform can be restored. Figure 2.7(b) shows the required components for demodulation.

Delta modulation focuses on simplicity rather than accurate AD-conversion. There are two fundamental noise sources caused by this method. Both are characterized by the step size  $\Delta$  of the integrator, that is, the maximum change in amplitude between consecutive samples. If  $\Delta$  is too small, then the modulated signal cannot follow the input signal if its derivative is too large. This phenomenon is called *slope overload*. If  $\Delta$  is large, then the variation for slow or constant signals is high, resulting in what is called *granular noise*. Both effects are illustrated in Figure 2.7(d). As often in technical systems, there is a trade-off between them. Relaxing one noise source increases the other and vice versa. However, granular noise is much less problematic since, due to the averaging by the low-pass filters, it should partially cancel out over time. The condition to avoid slope overloading can be derived analytically. Suppose we apply a sinusoidal input signal in the form  $A \sin(\omega t)$ , its maximum deviation is given by

$$\max \left\{ \frac{d}{dt} A \sin(\omega t) \right\} = A\omega. \quad (2.1)$$

This slope must be smaller or equal to the maximum slope of the integrator. Hence,

$$A\omega \leq \frac{\Delta}{T_s} = \Delta f_s \quad (2.2)$$

must be fulfilled in order to avoid slope overload. It should be noted that this type of noise is a major source of distortion and thus highly affects the audio characteristics of the circuit. In order to reproduce the original sound pattern, it is necessary to model the delta modulator precisely. In our case, the 1-bit quantizer is the comparator and the integrator is a first-order low-pass filter. Both are known in their exact functionality assuming that the operational amplifiers are ideal. But in total there are three open questions to be answered.

Firstly, we do not know the physical representation of the digital logic levels, i.e. what voltage/current can be measured in the blue net of Figure 2.6. It is likely that the comparator produces an output signal close to the supply voltages. The voltage levels for logical 1 and 0 could therefore be 5 V and 0 V. However, we do not know that for sure and also cannot measure it since this net is not accessible by any pin.

Secondly, one must know the current  $I_{\text{mod}}$  that is produced by the modulator block.  $I_{\text{mod}}$  is the most important parameter since it controls the trade-off between the noise sources, i.e. it determines the step size  $\Delta$  which the feedback signal can take per sample interval.

Thirdly, there is one unknown part not discussed yet. According to the datasheet, the modulator block is also connected to the shift register. From a technical point of view this is not necessary, i.e. delta modulation requires no delay after AD-conversion. In Figure 2.1, the *MOD* block is fed by another output of the delay line than the *DEM* block. This leads to the assumption that the delay for the modulator is much shorter. In order to copy the behaviour of this ADC as precisely as possible, it is necessary to find out how long this delay is.

A final note before finishing this section: Delta modulation requires oversampling in order to work properly. It might be the case that sampling is done with the same frequency as the oscillator which drives the shift register. In this scenario, the sampling frequency and thus the integration time are variable. The step size per fixed time interval remains the same, i.e. the steepness is unaffected, but the step size per sample becomes different. In other systems, this is done on purpose to address the problem of slope overload because one can control the step

size. This is then called adaptive delta modulation. The problem in our system is that we want to copy this process in a fully digital environment with fixed sampling frequency. Imitating this effect might be difficult because one has to forward samples as a function of the delay time for the discrete model.

### 2.2.2 Measuring the modulated current

The first and second problem can be solved together. Measuring the logic levels is not possible, but fortunately this is not necessary. All that matters is the correct value for the current  $I_{\text{mod}}$ . In other words, we can choose whatever representation we want as long as we choose the voltage-to-current factor of the modulator block accordingly. The easiest choice is  $+1\text{ V}$  for logical 1 and  $-1\text{ V}$  for logical 0. In that case, the conductance  $G_{\text{mod}}$  of the modulator block is simply

$$G_{\text{mod}} = \frac{V_{\text{R10}}}{R_f} = \frac{V_{10} - V_9}{R_f} \quad (2.3)$$

where  $V_9$ ,  $V_{10}$ , and  $R_f$  are the voltage at pin 9, pin 10, and the resistance in the feedback of op-amp 3, respectively. We can measure this by applying a known signal at pin 15, e.g.  $0\text{ V}$  ( $V_{\text{SS}}$ ), and measure the voltage drop over the feedback resistor. If the inverting input of the comparator is always zero, then its output will be logical 1 continuously, resulting in a steady current that corresponds to the physical representation. Same can be done for logical 0 by applying  $5\text{ V}$  ( $V_{\text{DD}}$ ) at the input. The setup for this measurement is shown in Figure 2.8.

Additional resistors, e.g. a  $470\ \Omega$  resistor, can be included in the feedback of op-amp 3 to artificially create more measurement points. That way the voltage drop can be measured for different feedback impedances. Several chips have been measured and the result is shown in Figure 2.9.

The voltage drop across  $R_{10}$  turned out to be almost perfectly constant over the whole frequency range. Variations were only observed across different chips. Therefore, the modulation current  $I_{\text{mod}}$  and thus the step size per fixed time interval are frequency-independent. The overall average is  $-0.866\text{ mA}$  for logical 0 and  $1.052\text{ mA}$  for logical 1. Notice that the relation is slightly asymmetric, i.e. the steepness of the op-amp's output is not the same for rising and falling edges.

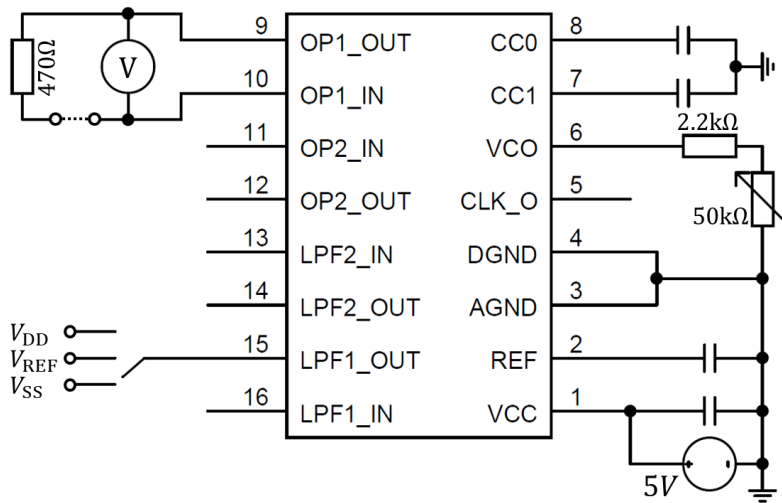
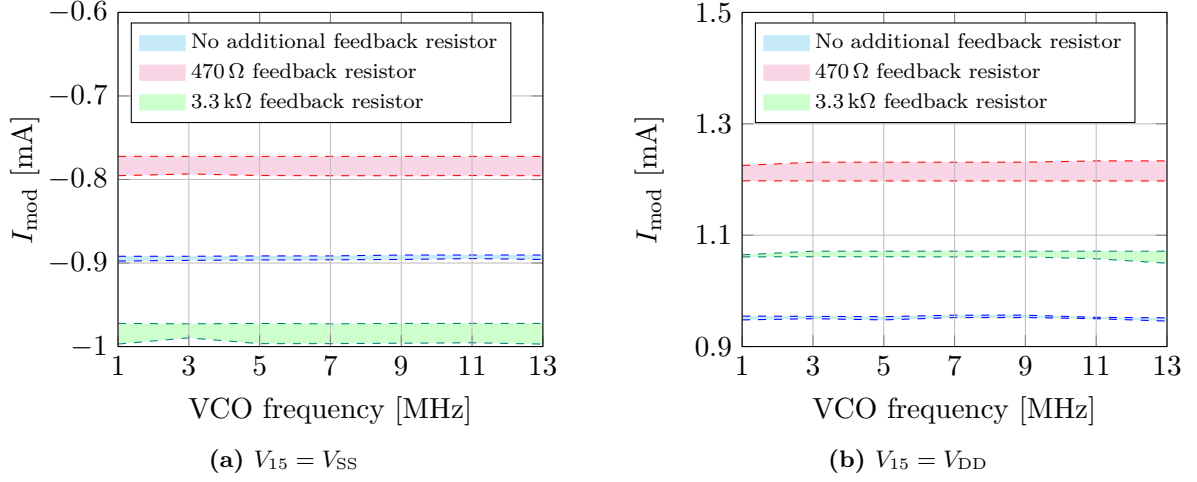


Figure 2.8: Setup for measuring the modulation current.



**Figure 2.9:** Measurement of the modulation current showing the range of the minimum and maximum for different feedback impedances versus the oscillator frequency.

Suppose that the logic levels are represented with  $\pm 1$  V, then the voltage-to-current factor of the *MOD* and *DEM* block are approximately given by the mean value, hence

$$G_{\text{mod}} = G_{\text{dem}} \approx 0.96 \text{ mA/V}. \quad (2.4)$$

### 2.2.3 Measuring the delay line

The delay in the *PT2399* is produced with a 44 kBit long shift register. Different delay times are achieved by changing the clock frequency of a voltage-controlled oscillator whose input voltage is controlled by op-amp 6. Connecting a resistor on pin 6 to ground will change the gain of the op-amp and thus adjust its output voltage, forcing the VCO to produce another frequency. In our example board, the resistance at pin 6 is a series connection of a 2.2 k $\Omega$  resistor and a 50 k $\Omega$  potentiometer.

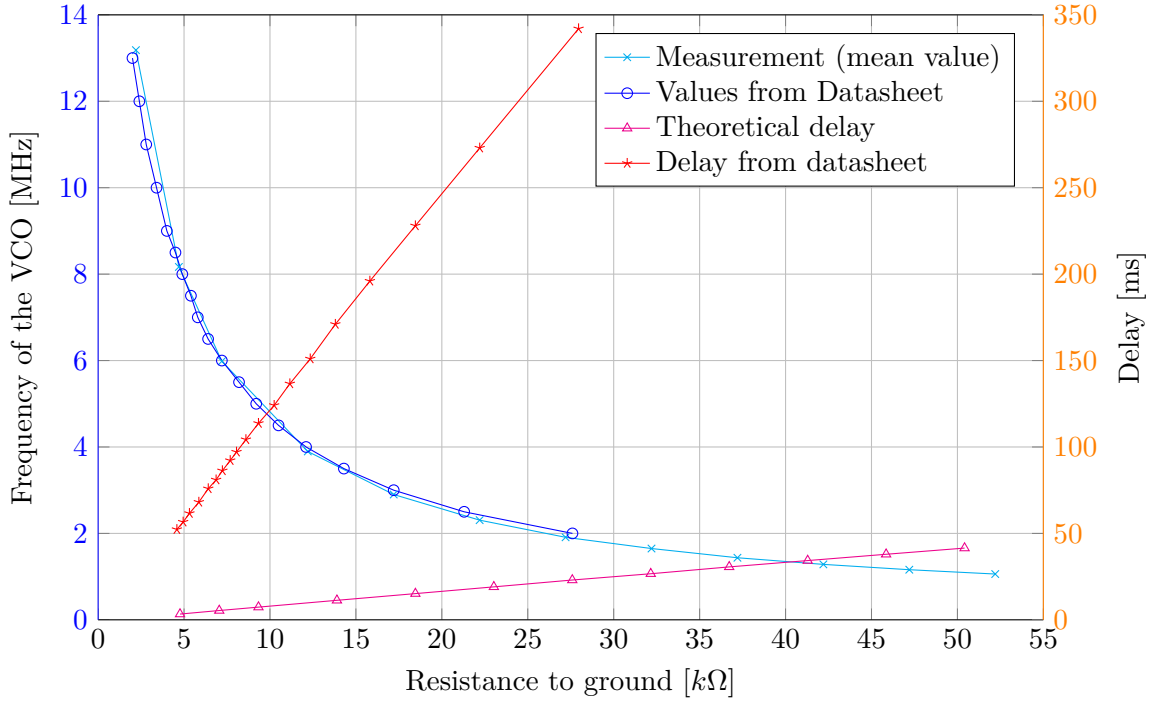
The datasheet provides a table in which the resulting delay times and VCO frequencies are given. Nevertheless, several measurements with different chips have been performed and compared with the values from the datasheet. The result is shown in Figure 2.10.

Notice that the datasheet only provides values up to 27.6 k $\Omega$ . The maximum value of 52.2 k $\Omega$ , which is possible with our configuration, is far beyond this limit. However, the relationship seems linear according to the measurement in this region although it is not specified in the datasheet. Another interesting thing is the fact that the delay times in the datasheet do not match the theoretical values. If the delay line is directly connected to the VCO's clock, then the delay time  $\tau$  is given by

$$\tau = \frac{1}{f_{\text{VCO}}} \cdot 44000. \quad (2.5)$$

The resulting delay according to the datasheet is much larger, 15.5 times larger to be precisely. Using linear regression yields a slope of 0.7658 ms/k $\Omega$  for the theoretical delay, and 11.397 ms/k $\Omega$  for the datasheet values. They differ by a factor of 14.89. This leads to the assumption that the VCO clock is divided internally by a factor of 16.





**Figure 2.10:** Measurement of the VCO frequency versus the resistance to ground and the resulting delay times.

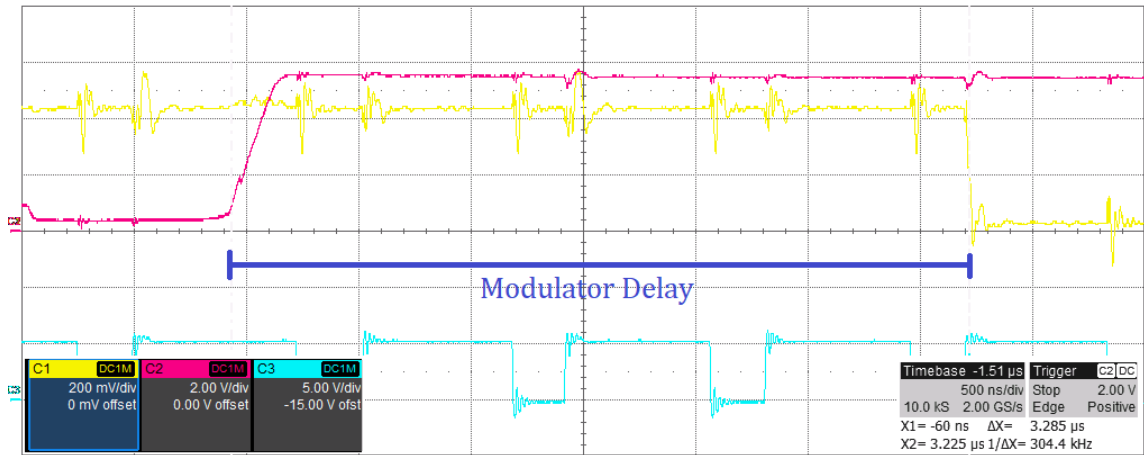
With this knowledge it is possible to measure the delay of the modulator path. The measurement setup is similar to the one shown in Figure 2.8, but this time an oscilloscope monitors the waveform of pin 15 and 10. At pin 15, a step signal is applied, e.g. from  $V_{DD}$  (logical 1) to  $V_{SS}$  (logical 0). It is then measured how long it takes until the logic level at pin 10 changes. An example of how this looks like is given in Figure 2.11.

The problem is that this measurement turned out to be very inconsistent. Performing several runs with identical setups resulted in strongly varying values. Not only are the signals rather noisy in general, the *PT2399* also suffers from strong clock jitter which makes it difficult to perform such a measurement.

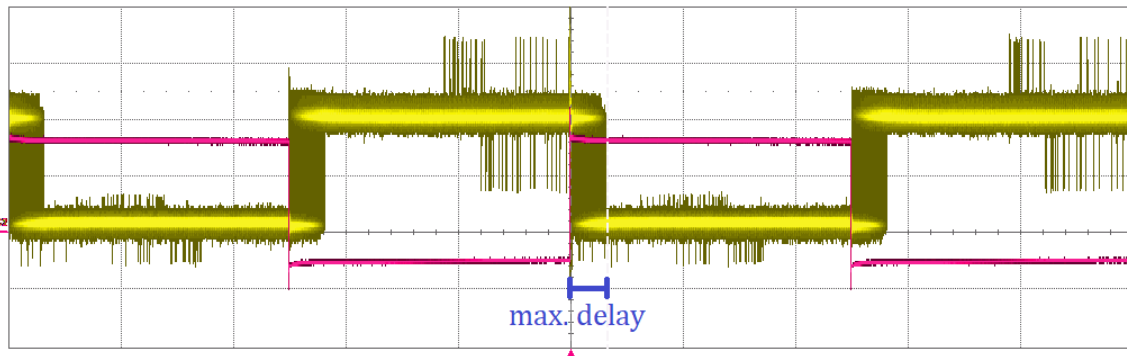
To address this problem, a square wave generator was connected to pin 15 rather than just a jumper which switches from one logic level to another. The screen of the oscilloscope then shows a blurred version of Figure 2.11. This is illustrated in Figure 2.12 where one example of such a measurement is shown.

With this approach one can measure an upper limit for the modulator delay. The measurements were converted to number of clock cycles and are shown in Figure 2.13 versus the VCO frequency. This measurement is much more consistent over several runs with different chips and turned out to be more or less independent of the input signal.

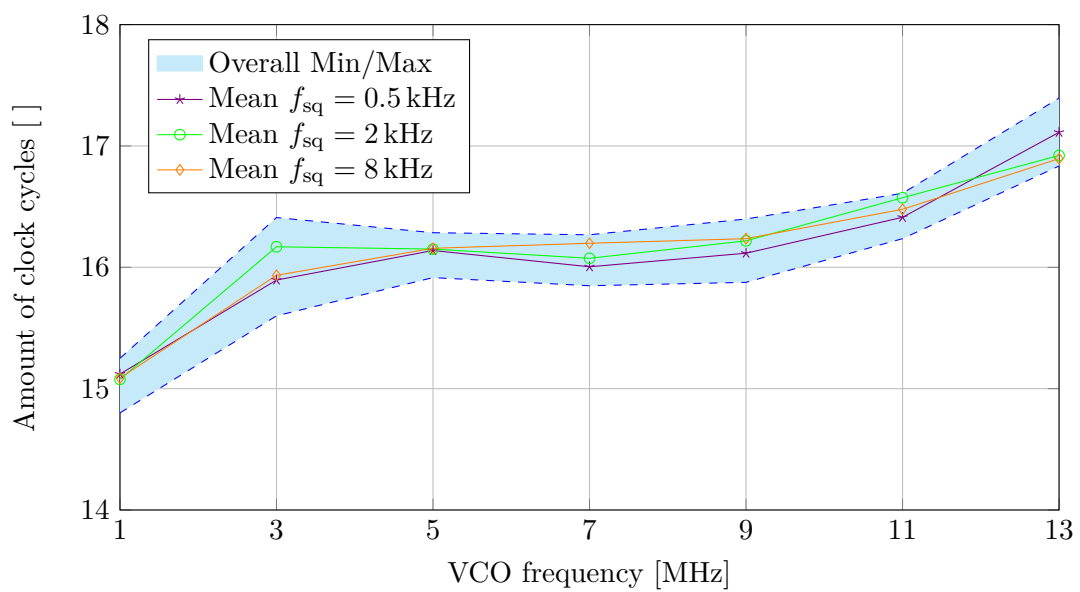
Due to the fact that the maximum delay barely exceeds 16 clock cycles, and with the knowledge that the clock is probably divided internally by 16, it is very likely that the modulator path (blue line in Figure 2.6) has a delay of at maximum one sample period. In other words, the modulator is connected to the very first memory cell in the shift register. This consistency also strengthens the hypothesis that the delta modulator operates with the same clock as the shift register.



**Figure 2.11:** Wave form of pin 5 (cyan), pin 15 (magenta), and pin 10 (yellow) when a step signal is applied at pin 15.



**Figure 2.12:** Wave form of pin 15 (magenta) and pin 10 (yellow) when a square wave generator is connected at pin 15.



**Figure 2.13:** Maximum delay of the modulator inside the *PT2399* in number of clock cycles if a square wave with frequency  $f_{sq}$  is applied at pin 15.

### 2.3 Summary and verification

With the two performed measurements, the delta modulator can be fully described. However, it is not possible to answer the question how variable integration times should be handled. To gather more evidence one should derive formulas which show how the modulator behaves if the clock frequency is adjusted.

For large gains, the transfer function of the integrator (without the *MOD* or *DEM* blocks) is given by

$$\lim_{s \rightarrow 0} H(s) = \lim_{s \rightarrow 0} \frac{V_{\text{out}}(s)}{I_{\text{in}}(s)} = \frac{-1}{sC + G}. \quad (2.6)$$

Here,  $G$  and  $C$  are the components in the feedback. Since the integrator is driven by a constant current, one can model the input signal as  $I_{\text{in}}(s) = I_{\text{mod}}/s$ . The response in the time domain is

$$v_{\text{out}}(t) = \mathcal{L}^{-1} \{I_{\text{in}}(s)H(s)\} = \mathcal{L}^{-1} \left\{ \frac{I_{\text{mod}}}{s} \cdot \frac{-1}{sC + G} \right\} = I_{\text{mod}} R \left( e^{\frac{-t}{RC}} - 1 \right) \quad (2.7)$$

where  $R = 1/G$ . For small time intervals, one can approximate (2.7) by linearization at  $t = 0$ .

$$\left. \frac{d}{dt} v_{\text{out}}(t) \right|_{t=0} = \frac{-I_{\text{mod}}}{C} \quad (2.8)$$

Using the numbering from the schematic in Figure 2.3, one can calculate the general relationship between the VCO frequency and the step size. Notice that the modulator and demodulator use different components in their feedback.

$$-\Delta_{\text{mod}} = T_s \cdot \frac{-I_{\text{mod}}}{C_{13}} = \frac{16}{f_{\text{VCO}}} \cdot \frac{-I_{\text{mod}}}{C_{13}} \quad (2.9)$$

$$-\Delta_{\text{dem}} = T_s \cdot \frac{-I_{\text{dem}}}{C_{12}} = \frac{16}{f_{\text{VCO}}} \cdot \frac{-I_{\text{dem}}}{C_{12}} \quad (2.10)$$

In (2.9) and (2.10), the assumption that the clock is divided by 16 internally is already included. Now one can determine whether slope overload occurs or not by evaluating the inequality

$$A\omega \leq \frac{I_{\text{mod}}}{C_{13}}. \quad (2.11)$$

It should be emphasized that (2.11) is independent of the sampling period. Any signal entering the delta modulator is filtered by a second-order low-pass filter with cut-off frequency  $f_c = 2.85 \text{ kHz}$ . Inserting in (2.11) yields

$$A\omega = 2.5 \text{ V} \cdot 2\pi \cdot 2.85 \text{ kHz} \approx 44.8 \text{ V/ms} \not\leq \frac{I_{\text{mod}}}{C_{13}} = \frac{0.96 \text{ mA}}{150 \text{ nF}} \approx 6.4 \text{ V/ms}, \quad (2.12)$$

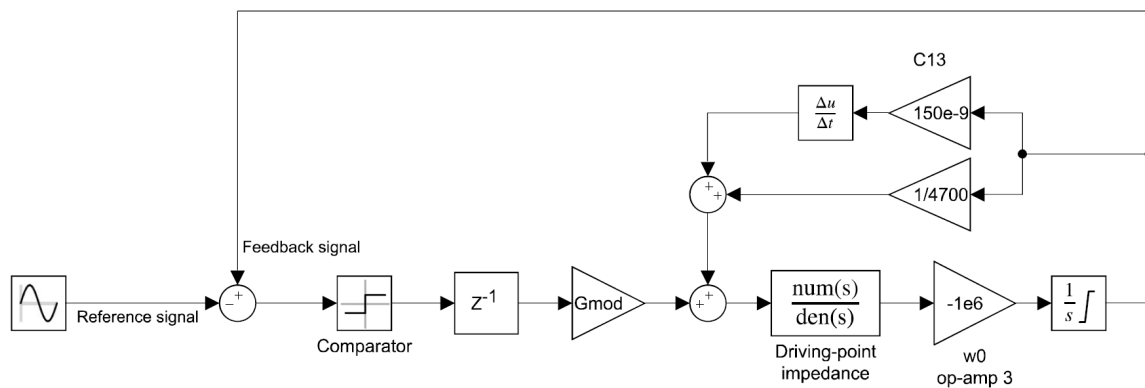
which is approximately seven times larger than the slope of the integrator. Hence, slope overload occurs! For the demodulator, the right-hand side of (2.12) is equal to  $14.1 \text{ kV/s}$  which is approximately three times smaller than the maximum slope.

One can verify this behaviour by a simple simulation. Figure 2.14 shows a Simulink model of the delta modulator. The functionality of the comparator is idealized by replacing it with the

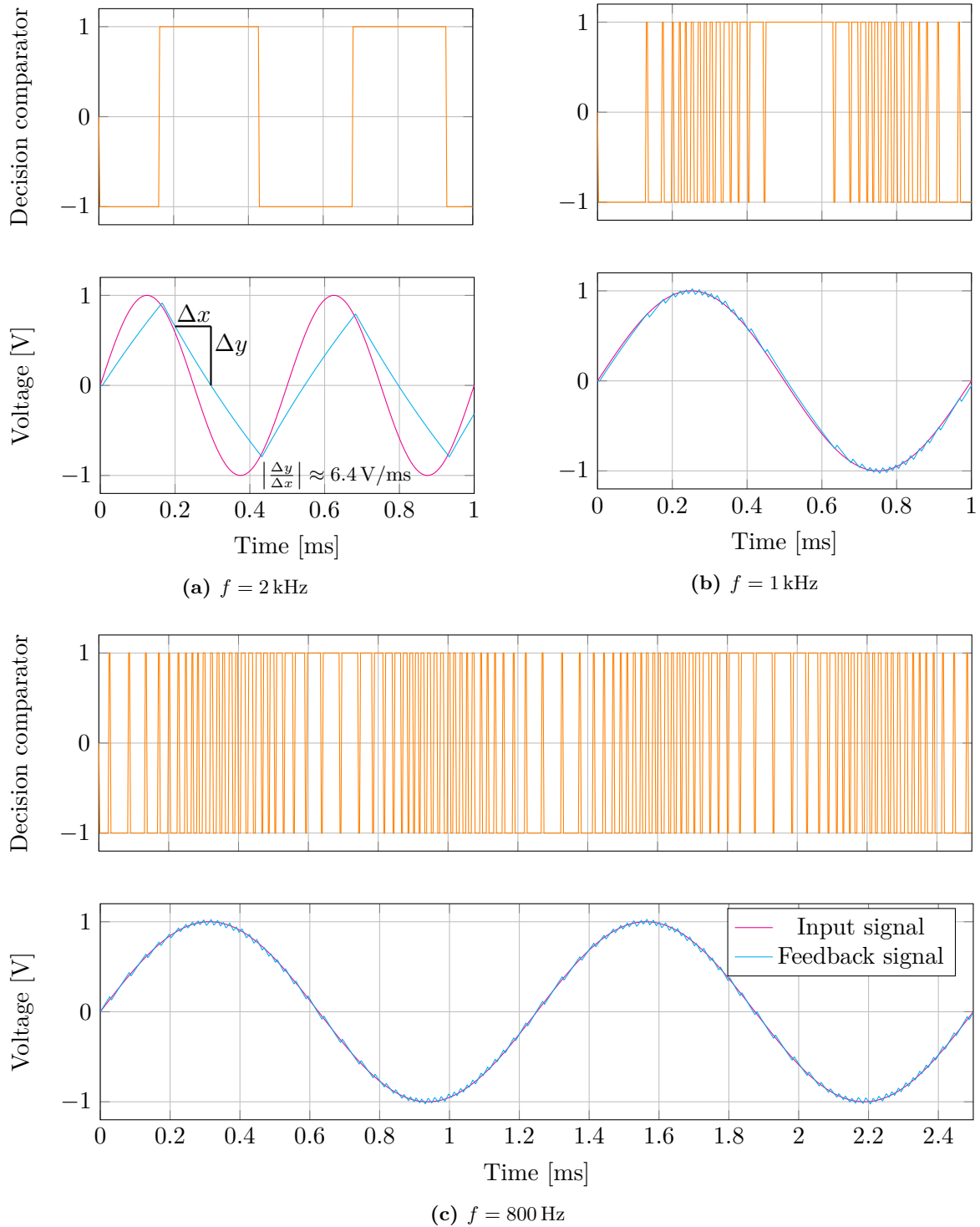
sign operator. Component values are taken from the schematic and the GBW of the op-amp was assumed to be 1 Mrad/s. This model is simulated with a fixed sampling time in order to imitate the behaviour correctly.

From (2.11), it follows that slope overload occurs approximately at  $I_{\text{mode}}/(C_{13} \cdot A \cdot 2\pi) \approx 1.02 \text{ kHz}$  if a sinusoidal signal with amplitude  $A = 1$  is applied. Figure 2.15(a) shows the comparison of the input and feedback signal if a 2 kHz signal is applied, which is above the limit to slope overload. It can be observed that the integrator is continuously increasing or decreasing because it is unable to follow the input signal and the slope  $\Delta y/\Delta x$  matches precisely the theoretical value from (2.8). Additionally, one can observe the slightly exponential waveform which corresponds to the response in the time domain from (2.7). In Figure 2.15(b), a scenario close at the border to slope overload is shown. At points where the sinusoidal signal is the steepest, it can be seen that the feedback signal cannot follow the curve. This is also visible if one looks at the comparator's decision because it stays constant as long as slope overloading occurs. Finally, Figure 2.15(c) shows the normal case where no slope overload occurs.

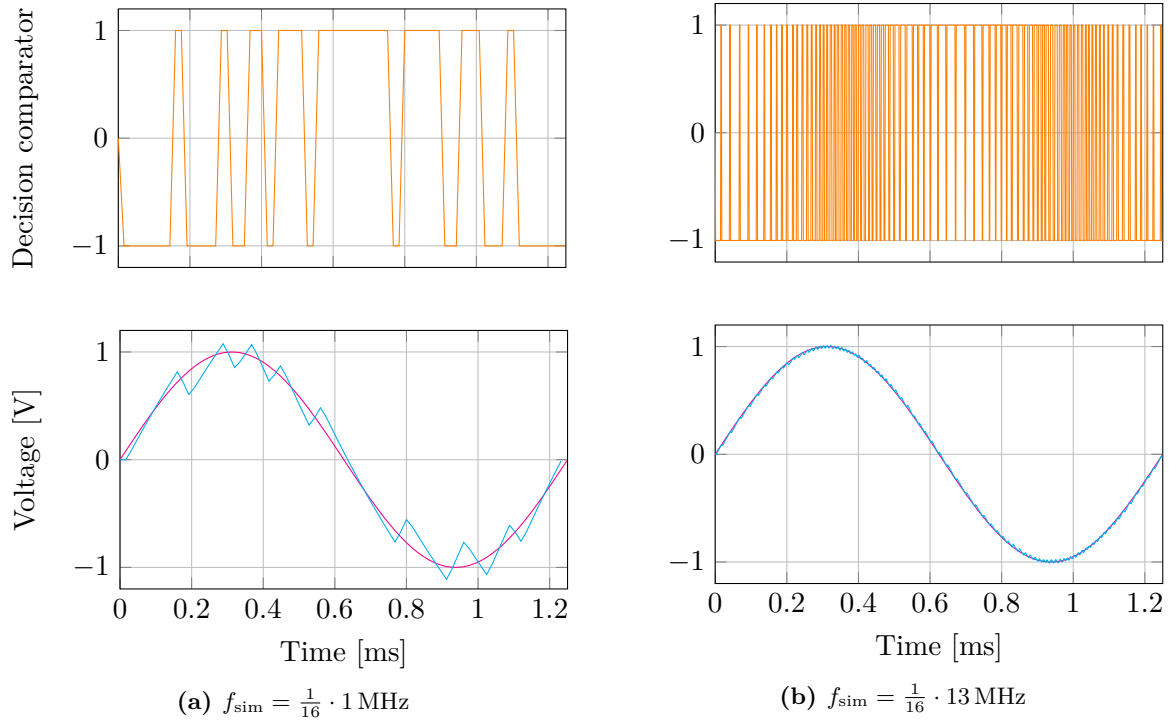
For the next simulation, the influence on variable sampling times shall be demonstrated. Recall that the delta modulator runs at one sixteenth of the VCO's frequency which ranges from 1 MHz to 13 MHz. The impact on different sampling intervals is illustrated in Figure 2.16. From (2.11) it is known that the effect of slope overloading is independent of the sampling period. Figure 2.16(a) and (b) confirm this since no slope overload occurs for a sinusoidal signal of 800 Hz. Furthermore, it can be seen that –as stated by the theory– granular noise is increased since it takes much longer until the decision of the comparator is passed to the integrator. Theoretically, this means that noise could be reduced by sampling at higher rates without changing the overall system too much.



**Figure 2.14:** Continuous-time simulink model of the delta modulator.



**Figure 2.15:** Simulink simulation of the delta modulator for sinusoidal input signals with different frequencies. (a) shows the case in which slope overload occurs, (b) shows a case close to the border of slope overloading, and (c) shows the wave forms without slope overload. The simulation frequency was fixed at  $6 \text{ MHz} \cdot 1/16$ .



**Figure 2.16:** Simulink simulation of the delta modulator with  $V_{\text{in}} = 1 \cdot \sin(2\pi \cdot 800 \text{ Hz})$  for the minimum sampling frequency in (a), and the maximum sampling frequency in (b).

### 3 Mixed-signal modelling with signal-flow graphs

A signal-flow graph (SFG) is essentially nothing more than a graphical description of an equation system. They are used to describe the flow of information in linear systems. An arrow pointing from node  $x$  to  $y$  with weight  $A$  translates to  $y = A \cdot x$  where  $A$  can be a scalar, a matrix, or a transfer function. SFGs are widely used e.g. in digital systems or control engineering, but they are seldom used to describe analog systems, though. The flow of information are the currents and voltages in a circuit which, due to the principle of superposition, are never unidirectional. Modelling them as SFGs is rather difficult and has no benefit to conventional analysis methods.

The trick is to modify the procedure to create so-called driving-point signal-flow graphs (DPSFG). They introduce auxiliary voltage sources at every node in a circuit which does not contain a real voltage source, making the superposition principle easy to evaluate. This has the advantage that modelling a circuit becomes straightforward and scales well, i.e. the effort to create a graph rises linearly with circuit complexity. Furthermore, they are multi-domain suitable, meaning that mixed-signal circuits can be represented as long as there is a linear model describing the transition from one domain to another. The exact procedure to set up a DPSFG is not explained in this report. For a more detailed view, the reader is referred to [9]–[11].

Solving a SFG, i.e. finding a certain transfer function, is done with Mason’s gain formula [12]. It is a graphical method that avoids setting up an equation system and provides a powerful tool to handle linear systems. However, it gets complicated to evaluate once many loops are present. DPSFGs create up to two loops for every passive component in a circuit. Solving them becomes unfeasible quickly without the help of software. A tool to evaluate SFGs is available in [13].

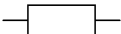
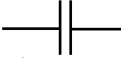
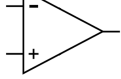

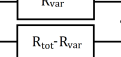
The good news is that evaluating Mason’s gain formula is not necessary. The idea is to translate a graph directly to digital components while maintaining its structure. Every continuous component (integrator, derivative, transfer function, gain, delay) is converted to its discrete-time equivalent. From now on we will call this method the *low-level approach*. If the discrete models are precisely enough then one expects that the output of the system imitates the behaviour of the analog circuit.

#### 3.1 Model for electrical components

The circuit is analysed in the Laplace domain which, by definition, provides a linear model that can be described with SFGs. Every components has its transfer function (= model) that expresses the relationship between currents and voltages.

Our test board only consists of five different components, namely: resistors, capacitors, operational amplifiers (op-amps), delays, and potentiometers (variable resistors). Their corresponding model in the Laplace domain is given in Table 3.1. Notice that there are other components in the circuit like switches or diodes, but they only affect the large-signal behaviour of the circuit and are thus not required to model the flow of information which only deals with the small-signal behaviour.

A resistor is just a multiplication with a constant. The voltage of a capacitor is given by the integration of the current and weighting the result by the capacitance. Since DPSFGs described passive components always with their admittance, i.e. the voltage-to-current relationship, the model becomes a multiplication by  $s$  which corresponds to a derivative in the time domain. A positive time shift is given by one of the properties of the Laplace transform and a variable resistor is modelled as two resistors with interchanging resistance. In other words, it is a three-terminal component that forms an adjustable voltage divider.

Component	Symbol	Model	Input	Output
Resistor $R$		$G = 1/R$	[V]	[A]
Capacitor $C$		$sC$	[V]	[A]
Op-amp		$\frac{\pm\omega_0}{s}$	[V]	[V]
Delay $\tau$		$e^{-s\tau}$	[V]	[V]
Potentiometer $R_{\text{var}}$		$G_{\text{var}}$	[V]	[A]

**Table 3.1:** Laplace domain models of the electrical components and their units.

The model for op-amps is explained more detailed since it is needed in chapter 5 to model non-linear effects. Conventional op-amps consist of a differential pair with a single-ended output and optional additional current gain. This is cumulated to a gm-block which is followed by one or more gain stages, forming a voltage-controlled voltage source. A generic op-amp model is shown in Figure 3.1. The resistor  $G$  and capacitor  $C$  model the finite output impedance and the capacitive load, respectively.

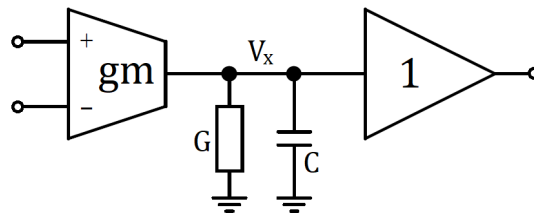
Depending on whether a signal enters at the inverting or non-inverting input, their transfer function is given by

$$\frac{V_{\text{out}}}{V_{\text{in}}} = \frac{\pm g_m}{sC + G}. \quad (3.1)$$

For frequencies above  $2\pi \cdot G/C$ , (3.1) can be simplified to

$$\frac{V_{\text{out}}}{V_{\text{in}}} \approx \frac{\pm g_m}{sC} = \frac{\pm\omega_0}{s} \quad (3.2)$$

where  $\omega_0$  denotes the gain-bandwidth product (GBW). (3.2) results in a frequency response that decreases with 20 dB per angular frequency decade. The flattening of the gain at low frequencies, which is not captured by (3.2), is usually not crucial since op-amps work correctly as long as their gain is "large". That's why this simplified model can be used.



**Figure 3.1:** Simple small-signal op-amp model.



### 3.2 Modelling the whole circuit

The DPSFG of the whole circuit is shown in Figure 3.2. This graph is a good example to show the strengths of graph-based analysis methods. Figure 3.2 gives a nice insight into the system in terms of information flow. The echo and delayed path are clearly visible as well as feedback paths of op-amps. Furthermore, the fact that all connections are bidirectional due to the principle of superposition is also illustrated in the form of loops by the graph – something that a set of linear equations do not provide.

### 3.3 A survey of discretization methods

The complete driving-point signal-flow graph of *Der Faux* describes the circuit in the continuous-time domain. In order to implement it on a fully digital system, it must be discretized. Conversion from the s-domain to the z-domain can be done by various transformations. An overview is given in Table 3.2. Notice that the names of the transformations vary strongly if different literature is considered. The names in Table 3.2 refer to Matlab's notation.

The first three transformations in Table 3.2 base on the idea that the relationship  $z = e^{sT_s}$  is approximated by another function. Neither Forward nor Backward difference are supported by Matlab's API, but their implementation is trivial since they can be broken down to a simple substitution.

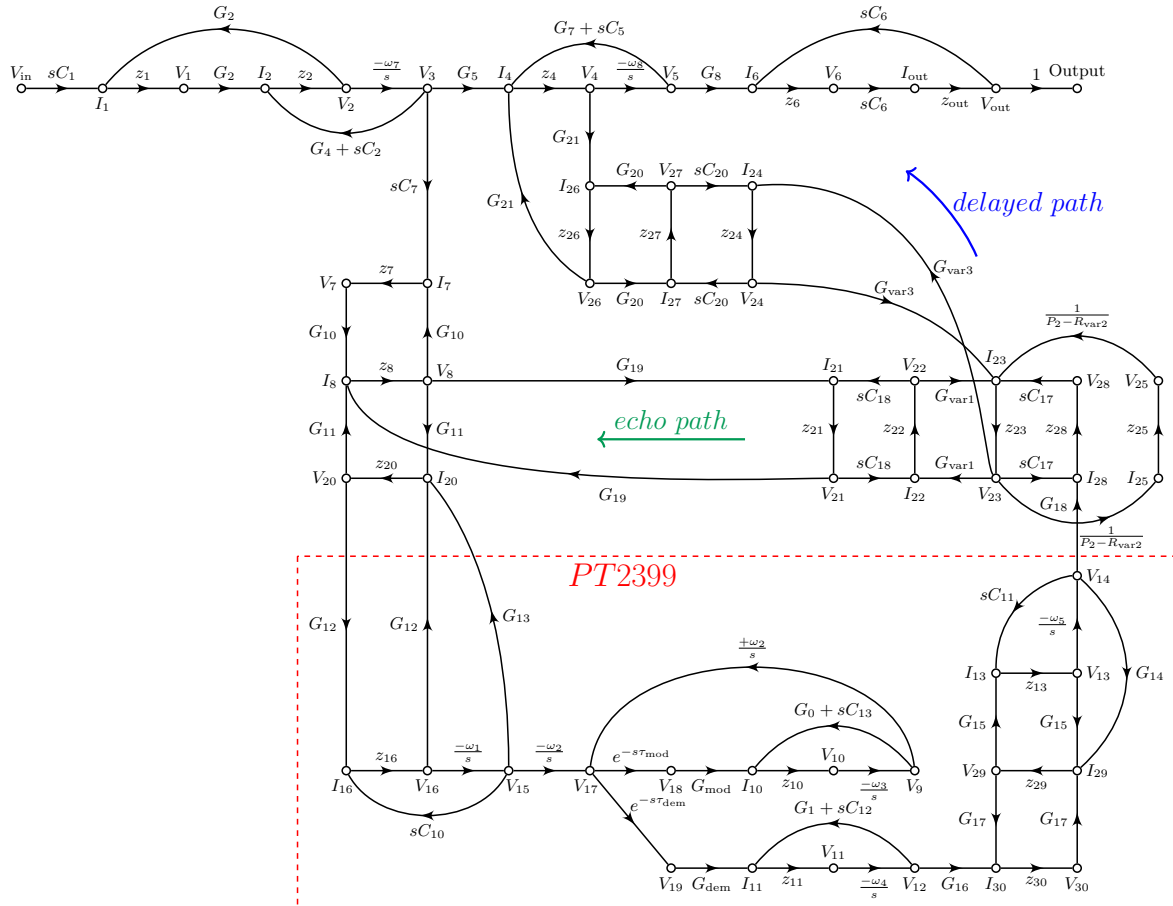


Figure 3.2: Driving-point signal-flow graph of *Der Faux*.

Discretization method	Transformation
<i>Simple substitution methods:</i>	
Forward difference	$s = \frac{z-1}{T_s}$
Backward difference	$s = \frac{z-1}{zT_s}$
Tustin's approximation	$s = \frac{2}{T_s} \cdot \frac{z-1}{z+1}$
<i>Impulse response sampling methods:</i>	
Impulse-invariant mapping	$H_d(z) = T_s \cdot \mathcal{Z} \{ \mathcal{L}^{-1} \{ H_c(s) \}  _{t=kT_s} \}$
Zero-order hold	$H_d(z) = (1 - z^{-1}) \cdot \mathcal{Z} \left\{ \mathcal{L}^{-1} \left\{ \frac{H_c(s)}{s} \right\}  _{t=kT_s} \right\}$
First-order hold	$H_d(z) = \frac{(z-1)^2}{zT_s} \cdot \mathcal{Z} \left\{ \mathcal{L}^{-1} \left\{ \frac{H_c(s)}{s^2} \right\}  _{t=kT_s} \right\}$
<i>Other methods:</i>	
Zero-Pole mapping	$H_d(z) = K' \cdot \prod_{k=1}^{N-M-1} (z+1) \cdot \frac{\prod_{k=1}^M (z - e^{\omega_{zk} T_s})}{\prod_{k=1}^N (z - e^{\omega_{pk} T_s})}$
Least-squares	-

**Table 3.2:** List of conversion methods to transform a continuous-time transfer function into a discrete-time transfer function [14].

Forward difference, also called *Euler's method*, uses the approximation  $z = e^{sT_s} \approx 1 + sT_s$  which is identical to the Taylor series. The main problem of this method is stability. It can happen that a stable continuous system is mapped to an unstable discrete-time system.

Backward difference uses the approximation  $e^{sT_s} \approx 1/(1 - sT_s)$  and does not suffer from the stability problem, i.e. a stable continuous-time system is always transformed into a stable discrete-time system. It is usually not used to transform systems from the s-domain to the z-domain, but it is used to build discrete-time derivatives because it does not rely on data in the future.

Tustin's approximation, which is often referred to as *trapezoidal method* or *bilinear transformation*, maps the whole frequency range into the unit circle. The complete left-hand side of the complex plane is located inside that circle. Therefore, it does not suffer from instability or aliasing. However, it causes frequency distortion since the mapping is non-linear. Fortunately, this problem can be addressed by appropriate filter design. What this means exactly is explained in section 3.4. If done correctly, it yields the best matching in the frequency domain which is beneficial for our project. High accuracy can be achieved by performing so-called *frequency prewarping*. If a system has important dynamics at a particular frequency, then one can preserve the frequency response at the prewrap angular frequency. Mathematically, the transformation is changed to

$$s = \frac{\omega}{\tan(\omega T_s/2)} \cdot \frac{z-1}{z+1} \quad (3.3)$$

where  $\omega$  is the prewrap frequency. In our system, one would set this frequency in the range of a few kilo hertz since the most important dynamics lie there.

The next three methods in Table 3.2 are based on sampling the impulse response of the continuous-time system. The idea is that the discrete impulse response is identical to the continuous one at the sampling intervals. They cannot be described in terms of a simple substitution

between  $s$  and  $z$ , but it is still possible to derive a closed formula for the transfer function in the  $z$ -domain. Sampling is done by applying the inverse Laplace transform on the continuous-time transfer function. Afterwards,  $t$  is substituted by  $kT_s$  where  $k \in \mathbb{Z}$ .

This is exactly what impulse-invariant mapping does, except that it additionally multiplies the result by the sampling period to address the  $1/T_s$  difference in the spectrum. In the frequency domain, it has the property that the frequency axis is periodically winded around the unit circle, meaning that there is an overlap. There is no compression of the frequency axis as for the Tustin method, but it has the limitation that aliasing occurs if the transfer function of the continuous-time system is non-zero above the Nyquist frequency.

Zero-order hold is very similar to impulse-invariant mapping. It is based on the idea that a sample of the impulse response is held constant over one period. Consequently, the discrete-time transfer function has the same step response at the sampling interval. Such a sample and hold circuit is modelled as a rectangular pulse which has the Laplace transform

$$H_{SH}(s) = \frac{1}{s} \cdot (1 - e^{-sT_s}). \quad (3.4)$$

A continuous-time system  $H_c(s)$  is then multiplied by (3.4). The rest of the zero-order transformation is straightforward and identical to the impulse-invariant transformation. This method is typically used for digitally driven systems, i.e. analog systems with a staircase input.

First-order hold is identical to the zero-order hold but uses a piecewise linear approximation of the impulse response. This requires a convolution with a triangular function, which is the reason why this method is also called *triangle approximation* or *ramp-invariant approximation*. It is more precise than zero-order hold for smooth inputs, but the resulting filter is acausal and thus often modified to what is called *delayed first-order hold*. In our case, we want to use the transformation in a precomputing step. Therefore, we do not want to build a filter and can use the acausal transformation.

Zero-Pole matching does exactly what it says. It matches the poles and zeros of the continuous-time transfer function with the discrete one. Then a constant factor  $K'$  is searched such that both systems have identical DC gain. This transformation only works if  $N \geq M$  where  $N$  is the number of poles and  $M$  the number of zeros. In the case where there are more poles than zeros,  $N - M - 1$  zeros at  $s = j\infty$  are introduced. This transformation also preserves stability but introduces aliasing since the frequency axis is repeatedly mapped around the unit circle.

Finally, there is the least-squares method which tries to minimize the squared error of the frequency response between the two domains. It is not further documented how this is done exactly. Matlab states that it yields similar results to Tustin's approximation but works with lower sampling rates, i.e. it can provide the same accuracy with less computational effort, if required.

In conclusion, forward difference, zero-pole matching, and impulse-invariant mapping are not suitable transformations due to the listed limitations. The least-squares method has no beneficial properties in our case since computational effort is not important. Zero-order and first-order hold are possible but they are mainly used if accuracy in the time domain is required. If high accuracy in the frequency domain is preferable, then Tustin's approximation yields better results. Therefore, all components of the SFG will be converted to the  $z$ -domain by Tustin's method.

### 3.4 Difference between continuous-time and discrete-time filters

In section 3.3, it was shown that the bilinear transformation, also known as Tustin's method, is the best choice to convert continuous-time filters into the  $z$ -domain. Since the whole left-hand

plane is mapped into the unit circle, this method causes some frequency distortion. Comparing the impact of the transformation at the frequency response on a theoretical basis is a bit difficult since the back transformation, i.e. the substitution  $z = e^{sT_s}$ , does not yield a polynomial in  $s$ .

In order to get an insight, the influence on poles and zeros is investigated. Any LTI system can be fully described by its poles and zeros. A general s-domain transfer function is thus given by

$$H(s) = K \cdot \frac{\prod_{m=1}^M (s - \omega_{zm})}{\prod_{n=1}^N (s - \omega_{pn})} \quad (3.5)$$

where  $K$ ,  $\omega_{zm}$ , and  $\omega_{pn}$  are the gain factor, zeros, and poles, respectively. Applying the bilinear transformation yields

$$H_d(s) = K \cdot \frac{\prod_{m=1}^M \left( \frac{2}{T} \cdot \frac{z-1}{z+1} - \omega_{zm} \right)}{\prod_{n=1}^N \left( \frac{2}{T} \cdot \frac{z-1}{z+1} - \omega_{pn} \right)}. \quad (3.6)$$

Using the relationship  $z = e^{sT_s}$  in (3.6) gives

$$\tilde{H}(s) = K \cdot \frac{\prod_{m=1}^M \left( \frac{2}{T} \cdot \frac{e^{sT_s}-1}{e^{sT_s}+1} - \omega_{zm} \right)}{\prod_{n=1}^N \left( \frac{2}{T} \cdot \frac{e^{sT_s}-1}{e^{sT_s}+1} - \omega_{pn} \right)} = K \cdot \frac{\prod_{m=1}^M \left( \frac{2}{T} \cdot \tanh\left(\frac{sT_s}{2}\right) - \omega_{zm} \right)}{\prod_{n=1}^N \left( \frac{2}{T} \cdot \tanh\left(\frac{sT_s}{2}\right) - \omega_{pn} \right)}. \quad (3.7)$$

Comparing (3.7) with (3.5), one can see that in both cases a factorized numerator and denominator is present. Thus, one can solve each individual term for zero to find the transformed poles and zeros. Denoting  $\omega_c$  and  $\omega_d$  to be an arbitrarily continuous-time and discrete-time frequency, the transformation from  $\omega_c$  to  $\omega_d$  is found as:

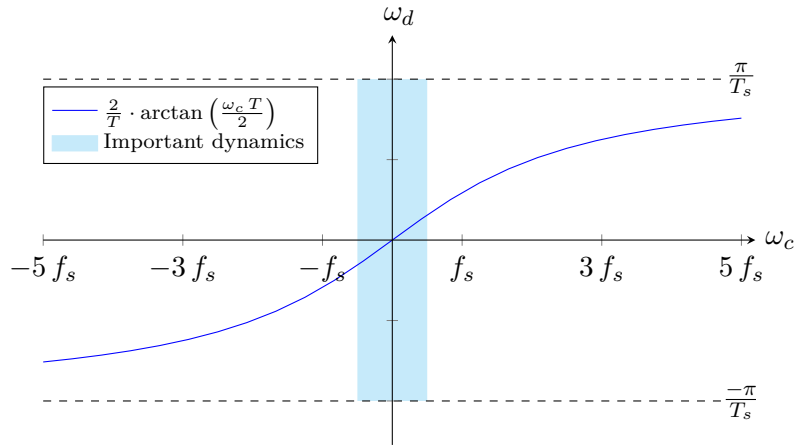
$$0 \stackrel{!}{=} \frac{2}{T} \cdot \tanh\left(\frac{sT_s}{2}\right) - \omega_c \stackrel{s=j\omega_d}{=} \frac{2j}{T} \cdot \tan\left(\frac{\omega_d T_s}{2}\right) - \omega_c \quad (3.8)$$

$$\omega_d = \frac{2}{T} \cdot \arctan\left(\frac{\omega_c T}{2}\right) \quad (3.9)$$

(3.9) is the distortion that occurs if the transformation is applied. Its domain is  $]-\infty, +\infty[$  but the function range is limited to  $[-\pi/T_s, +\pi/T_s]$  which corresponds to the wrapping of the frequency axis around the unit circle. When designing a discrete-time filter, one can use the inverse transform of (3.9) to match its behaviour with the continuous-time filter at this particular frequency. This is where the frequency prewarping from (3.3) comes from.

A sketch of (3.9) is shown in Figure 3.3. The waveform of the tangent function remains the same independent on the sampling frequency. If one wants to match the digital filter precisely, then one needs to make sure that important dynamics of the analog filter stay inside the region where distortion is low, i.e. the region for which the slope of the tangent function is close to 1. This is illustrated by the light blue area in Figure 3.3: The slimmer this region the higher the matching of the frequency response.

In conclusion, an accurate mapping from the s-domain to the z-domain can be achieved by the Tustin transformation if frequency prewarping and high sampling rates are used. The latter one comes at a price, though. The higher the Nyquist frequency is the more poles get squished to  $z = 1$ . Higher sampling rates thus require higher numerical precision.



**Figure 3.3:** Frequency distortion caused by the bilinear transformation.

### 3.5 Challenges

The procedure of converting a SFG into the  $z$ -domain by applying the Tustin transformation is straightforward. Implementing it on hardware is not, though. There are many challenges when trying to implement a graph as discrete-time system:

- A discrete model is always only an approximation of a real, continuous system. Achieving precise behaviour often requires oversampling.
- Converting a  $s$ -domain transfer function to a  $z$ -domain transfer function is not uncritical. Especially higher-order polynomials can result in inaccurate coefficients due to numerical problems. For example, poles from analog components often get mapped close to  $z = 1$  because the frequency axis is compressed into the unit circle. Depending on the resolution, poles close at  $z = 1$  form a cluster and cannot be distinguished, leading to inaccurate models or even unstable behaviour.
- By nature, analog components span up a huge dynamic range. Constant value multiplications range from  $10^6$  down to  $10^{-12}$ , requiring a large word size to represent the outcomes precisely.
- The DPSFG analysis produces many feedback loops without delays in it. Such combinatorial loops are not allowed in discrete-time systems unless they are able to solve an equation system.
- One component, namely the model for capacitors which includes a continuous-time derivative, is an acausal system and cannot be implemented directly.

Many of them are hardware-related and not explained in this section. Every hardware-dependent problem is discussed in chapter 4. Here, some general problems which apply to any kind of discrete implementation are evaluated.

#### 3.5.1 Integrators and Derivatives

In the continuous-time domain, a derivative is simply a multiplication by  $s$ . Not only is this system unstable, but also non-causal. All transformations that are based on sampling the impulse response cannot handle this system since it is impossible to sample the Dirac delta function at  $t = 0$ . Applying the Tustin transformation is possible, but results in a marginal

stable system whose poles are on the unit circle. To address this problem, a derivative can be approximated by the transfer function

$$\frac{Y(s)}{X(s)} = \frac{s}{cs + 1} \quad (3.10)$$

where  $c$  is the derivative coefficient. For small values of  $c$ , (3.10) imitates the behaviour of a derivative. Any transformation can then be applied to 3.10. Alternatively, one can describe a discrete-time derivative by the causal difference equation

$$y[k] = \frac{x[k] - x[k-1]}{T_s} \quad (3.11)$$

whose  $z$ -transform  $(z-1)/(zT_s)$  is equal to what we would get if we apply the backward difference to a real derivative. It provides good accuracy if the sampling period is small and is simple to implement. Thus, it is the preferred method to build discrete-time derivatives.

Integrators are marginal stable systems who remain stable as long as the distribution of positive and negative signals is equivalent – a property that is generally true for audio systems as long as DC components are decoupled. Their implementation is therefore non-critical.

There are three methods to approximate  $1/s$  in the discrete-time domain:

- Forward Euler:  $\frac{1}{s} \approx \frac{T}{z-1}$
- Backward Euler:  $\frac{1}{s} \approx \frac{Tz}{z-1}$
- Trapezoidal method:  $\frac{1}{s} \approx \frac{T}{2} \cdot \frac{z+1}{z-1}$

The formulations above are equal to the transformations in Table 3.2, applied on the transfer function  $1/s$ . They are related to the numerical integration methods known from calculus, i.e. the forward and backward method correspond to the left-hand and right-hand rectangular approximation. The trapezoidal method is the combination of both and provides the best accuracy. Thus, the trapezoidal method, which is equivalent to the Tustin transformation, will also be used for discrete-time integrators.

### 3.5.2 Feedback loops

Analog circuits rely strongly on feedback systems, which are visible in the form of loops in their signal-flow graph. Such loops cannot be implemented directly since it is not possible to assign a value at the input if the required output has not been computed yet. Therefore, one is forced to introduce additional delay blocks to break any combinatorial loop.

Inserting internal delays in a system can highly affect its behaviour, i.e. it changes its transient and frequency response, and influences stability. The only way to reduce this error is to use high oversampling in order to minimize the delay.

To get an insight into the problem, consider the circuit in Figure 3.4(a). It is a simple op-amp circuit whose values are identical to the first op-amp in Figure 2.3. The corresponding DPSFG is shown in Figure 3.4(b) and a discrete implementation using Simulink in Figure 3.4(c). Notice that Simulink itself can handle the loop formed by the feedback components since it solves differential equations numerically. However, other discrete systems, e.g. FPGAs, are forced to break this loop. This is illustrated in Figure 3.4(c) with the additional delay block in light grey.

First, let us analyse the problem in the continuous-time domain. A delay in the Laplace domain is represented by  $e^{-s\tau}$  where  $\tau$  is the delay in seconds. By definition, a transfer function cannot

represent delays because it can only display linear systems. Other dynamic system models such as state-space representation must be used if one wants to implement internal delays. Another option is to approximate  $e^{-s\tau}$  by a power series. This way one can describe the delay by a polynomial and analyse the system as transfer function in the Laplace domain. For hand calculations, low order polynomials are preferred since computing stability criterions is difficult for higher-order systems. An accurate method to obtain low-order polynomials is the Padé approximation [15]. It approximates a function as a fraction of power series which is generally more accurate than other well-known techniques such as Taylor series when both have the same degree. Mathematically, a Padé approximation is given by

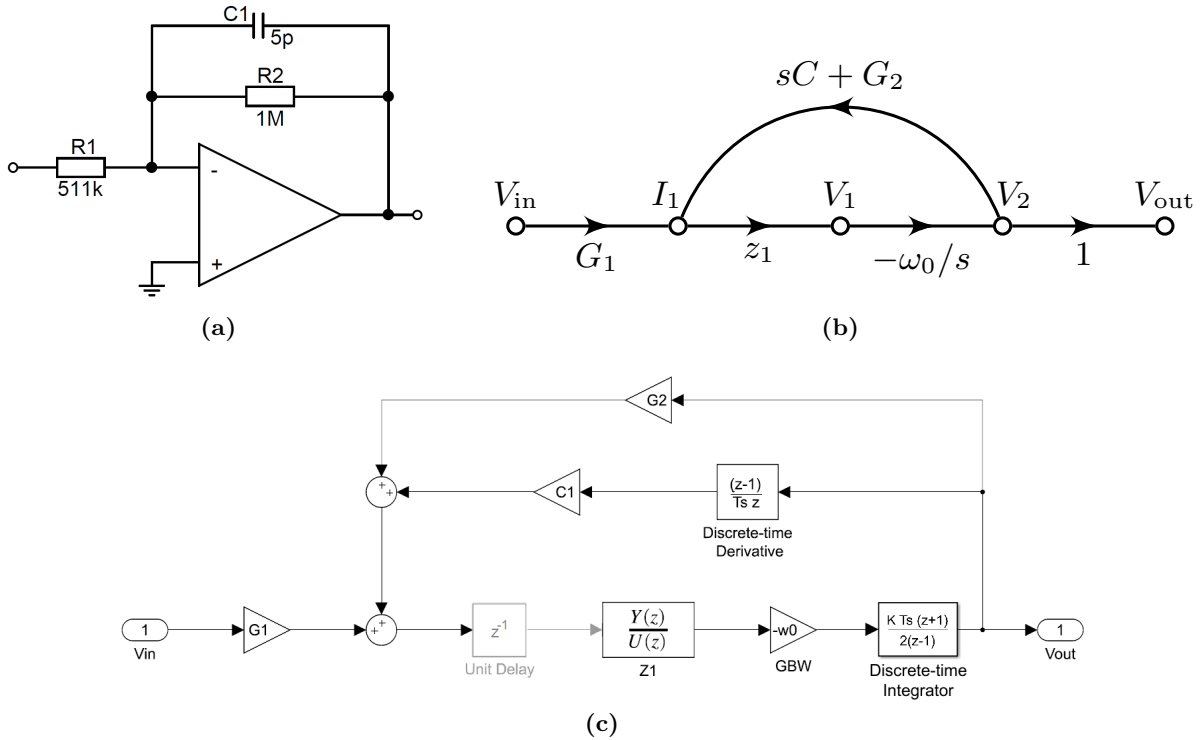
$$e^x \approx \frac{\sum_{j=0}^M a_j x^j}{\sum_{k=0}^N b_k x^k} \quad (3.12)$$

where the coefficients  $a_j$  and  $b_k$  can be taken from tables. Denoting the delay as  $\tau(s)$ , the transfer function for the op-amp circuit is

$$\frac{V_{\text{out}}}{V_{\text{in}}} = \frac{-\omega_0 G_1 \tau(s)}{s^2 C_1 + s(G_1 + G_2 + \tau(s) C_1 \omega_0) + G_2 \omega_0 \tau(s)}. \quad (3.13)$$

Using a second-order Padé approximation, the delay can be modelled as

$$e^x \approx \frac{x^2 + 6x + 12}{x^2 - 6x + 12} \quad x := -s\tau \quad \frac{(s\tau)^2 - 6s\tau + 12}{(s\tau)^2 + 6s\tau + 12}. \quad (3.14)$$



**Figure 3.4:** A simple feedback system showing an op-amp circuit in (a), the corresponding DPSFG in (b), and its discrete implementation with additional delay in (c).

Inserting (3.14) into (3.13) yields a polynomial of degree 4. After normalizing the denominator to a polynomial with leading coefficient 1, the transfer function has the form

$$\frac{b_2 s^2 + b_1 s + b_0}{s^4 + a_3 s^3 + a_2 s^2 + a_1 s + a_0}. \quad (3.15)$$

After decomposing the denominator coefficients by collecting terms with  $\tau$ , one can apply the Routh–Hurwitz stability criterion.

$$a_0 = \frac{12 G_2 w_0}{C_1 \tau^2} \stackrel{!}{>} 0 \quad (3.16)$$

$$a_1 = \frac{(-6 G_2 w_0) \tau + 12 G_1 + 12 G_2 + 12 C_1 w_0}{C_1 \tau^2} \stackrel{!}{>} 0 \quad (3.17)$$

$$a_2 = \frac{(G_2 w_0) \tau^2 + (6 G_1 + 6 G_2 - 6 C_1 w_0) \tau + 12 C_1}{C_1 \tau^2} \stackrel{!}{>} 0 \quad (3.18)$$

$$a_3 = \frac{(G_1 + G_2 + C_1 w_0) \tau + 6 C_1}{C_1 \tau} \stackrel{!}{>} 0 \quad (3.19)$$

$$a_3 a_2 a_1 - a_0 a_3^2 - a_1^2 \stackrel{!}{>} 0 \quad (3.20)$$

All inequalities from (3.16) to (3.20) must be fulfilled to form a stable system.  $a_0$  and  $a_3$  are always positive.  $a_1$ ,  $a_2$ , and (3.20) can be negative for large  $\tau$ . The value of  $\tau$  is inverse proportional to the simulation frequency, i.e.  $\tau = 1/f_{\text{sim}}$ . Suppose we sample the system with  $f_{\text{sim}} = \text{ovs} \cdot f_s$  where  $f_s = 48 \text{ kHz}$  is the audio sampling frequency and ovs the oversampling factor. One can now ask the question how large the oversampling factor must be in order to form a stable system. This is shown in Figure 3.5(a) where the Routh–Hurwitz coefficients are plotted versus the oversampling factor. The numeric values correspond to the components in the schematic and the datasheet of the op-amp (see [16]). One can observe that  $a_1$  is unstable for very large delay times (barely visible) and that the other two critical coefficients require a certain oversampling to be stable.

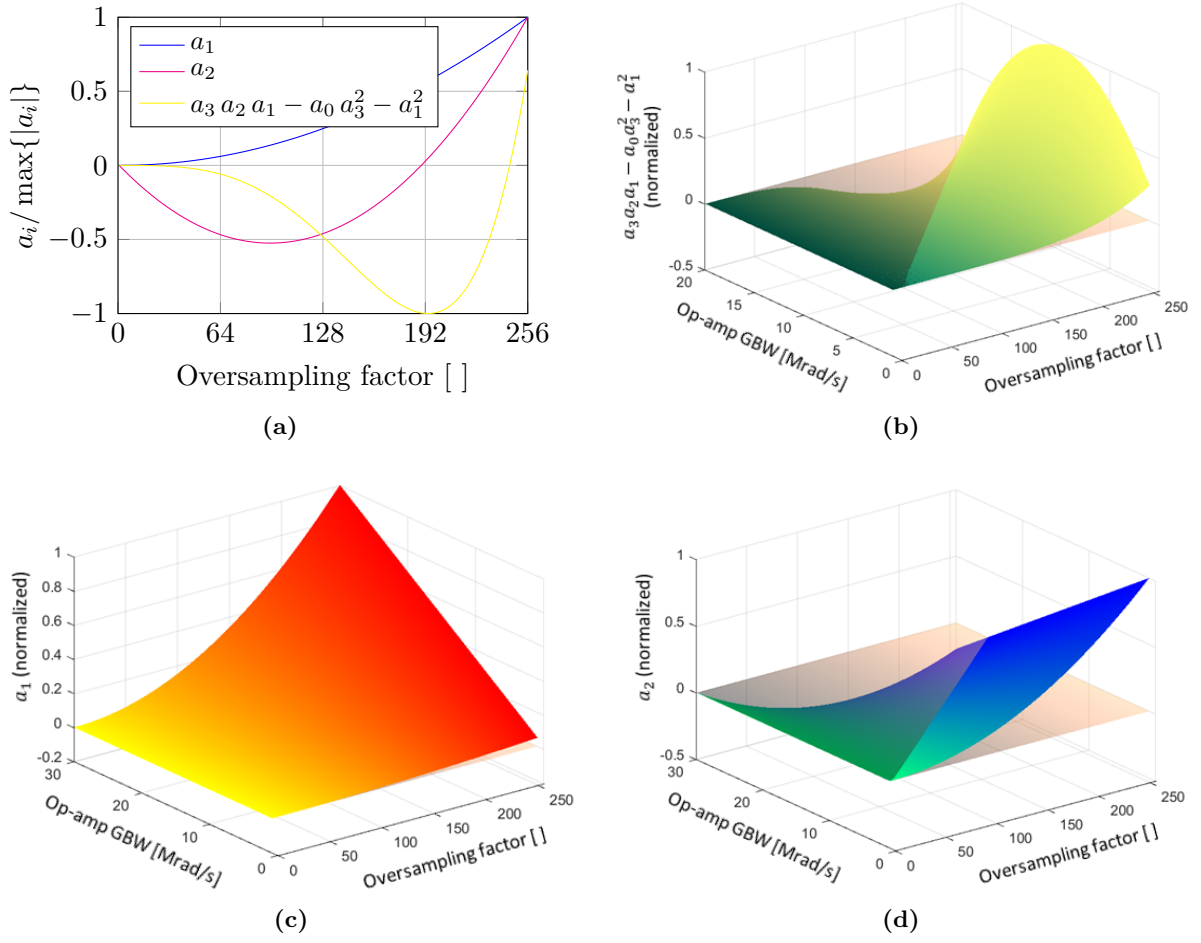
Another thing to mention is that the coefficients (3.17), (3.18), and (3.20) can become negative if the bandwidth of the amplifier is too large. This observation is important because the gain-bandwidth product of op-amps inside the *PT2399* is unknown. Loops with high gain can thus become unstable with additional delay. This effect is illustrated in Figure 3.5(b)–(d) where it can be seen that some coefficients drift towards negative values if  $\omega_0$  is large.  $a_1$  does not suffer from this problem but is still included for completeness.

So far, the analysis was based on a second-order Padé approximation of the delay function. In order to get a more meaningful insight into the problem, one should also consider higher-order approximations. Numerical methods will be used to determine the stability since symbolic analysis becomes difficult. For this purpose, the closed-loop transfer function from (3.13) was evaluated for different Padé approximations up to order five. The real part of the most critical poles, i.e. the ones that remain on the right-half plane the longest, are shown in Figure 3.6(a).

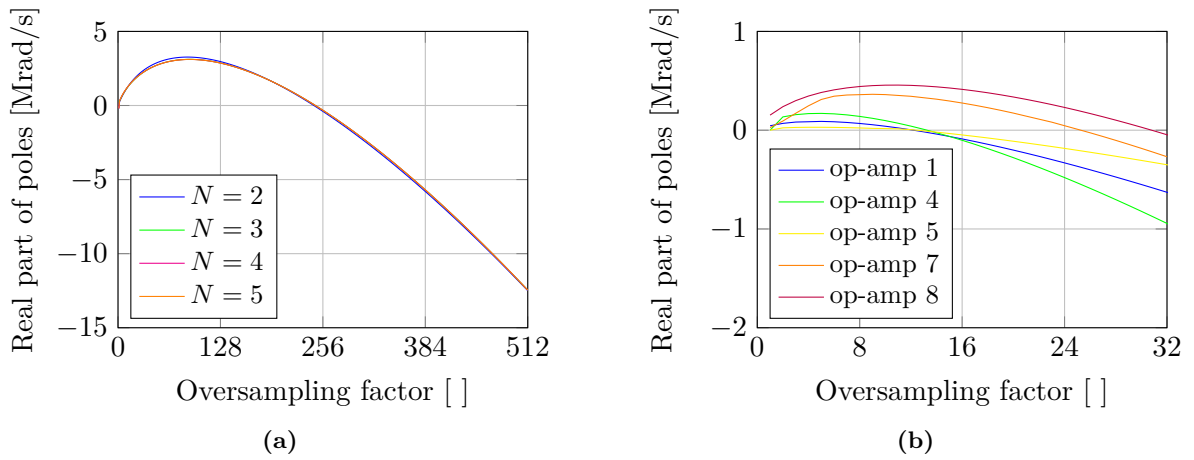
This procedure can be repeated on all functional blocks in *Der Faux* in order to answer the question which oversampling factor is at least required to make the overall circuit stable. It should be noted that the stability of individual subsystems is only a sufficient condition. Hence, this approach is an empiric method and does not guarantee overall stability.

A *functional block* refers to an op-amp and its neighbouring components, e.g. for op-amp 7, which is used to form a second-order band-pass filter,  $C_1$ ,  $C_2$ ,  $C_3$ ,  $R_2$ , and  $R_3$  are taken into





**Figure 3.5:** Routh-Hurwitz stability analysis for a simple feedback system with internal delay.



**Figure 3.6:** Behaviour of the most critical poles if the oversampling factor increases.

account. Figure 3.6(b) shows the result. The numerical evaluation of the poles was done by approximating the delay with a 4<sup>th</sup> order Padé polynomial and the GBW of the internal op-amps was assumed to be 1 Mrad/s. In general, the system is much less sensitive in terms of stability if neighbouring components are considered. For the chosen parameters, all op-amp circuits become stable at a oversampling rate of 32 or higher. Notice that this limit is strongly dependent on the bandwidth of the op-amps, i.e. large GBWs highly increase the required oversampling factor.

In total, there are three ways to address the problem of having delays in a loop. Firstly, use high oversampling until the system appears to be stable. Secondly, lower the GBW in the op-amp model to relax the stability problem. Thirdly, eliminate problematic loops by simplifying the signal-flow graph.

The first two solutions are connected as shown by the Routh-Hurwitz analysis in Figure 3.5. Which minimum sampling rates are possible depends on the hardware and is thus not discussed here. Lowering the bandwidth of an operational amplifier is feasible as long as the first pole of the closed-loop transfer function remains significantly above audio frequencies. One can estimate this point by equating the DC gain and the  $-20$  dB/dec asymptote. Using the example from Figure 3.4, the closed-loop transfer function is

$$H(s) = \frac{-\omega_0 G_1}{s^2 C_1 + s(G_1 + G_2 + C_1 \omega_0) + G_2 \omega_0} \quad (3.21)$$

and the  $-20$  dB/dec line is denoted by

$$H_{-20dB}(s) = \frac{-\omega_0 G_1}{s(G_1 + G_2 + C_1 \omega_0)}. \quad (3.22)$$

Substituting  $s = j\omega$  and equating (3.22) with  $H(0) = -G_1/G_2$  yields

$$\omega_{p1} \approx \frac{\omega_0 G_2}{G_1 + G_2 + C_1 \omega_0} \gg 2\pi \cdot 20 \text{ kHz}. \quad (3.23)$$

Solving for the  $\omega_0$  leads to an inequality constraint for the GBW:

$$\omega_0 \gg \frac{2\pi \cdot 20 \text{ kHz} (G_1 + G_2)}{G_1 - 2\pi \cdot 20 \text{ kHz} \cdot C_1} \approx 280 \text{ krad/s} \quad (3.24)$$

This calculation can be repeated for every op-amp circuit in order to find an lower bound for the bandwidth. Of course, the computation changes depending on the external circuitry. A list showing the minimum GBW such that the first pole is above audio frequencies is given in Table 3.3.

For op-amp 7 and 8, the GBW from the datasheet (3 Mrad/s) fulfills those conditions. The remaining op-amps were set to  $\text{GBW} = 1 \text{ Mrad/s}$ .

The third solution is based on the idea that critical loop, i.e. loops with low gain or phase margin, are eliminated. This is done by "cutting out" parts of the graph in which the loop is in between. For example, if one wants to remove the loop formed by the circuitry in Figure 3.4, then the continuous-time transfer function  $H_c(s)$  from  $V_{\text{in}}$  to  $V_{\text{out}}$  is calculated. Next,  $H_c(s)$  is transformed into the z-domain, yielding the discrete-time transfer function  $H_d(z)$ . The path from  $V_{\text{in}}$  to  $V_{\text{out}}$  is then replaced by  $H_d(z)$ .

This method only works when no paths enter or leave the loop except at the input and output node. To give a counterexample, the loop formed by op-amp 8 cannot be simplified because

op-amp	Min. GBW [krad/s]	Condition as inequality
op-amp 1	76.677	$\frac{G_{12} G_{13} w_1}{G_{11} G_{12} + G_{12} G_{13} + C_{10} G_{11} w_1 + C_{10} G_{12} w_1 + C_{10} G_{13} w_1} \gg 2\pi \cdot 20 \text{ kHz}$
op-amp 5	58.976	$\frac{G_{14} G_{15} w_5}{G_{14} G_{15} + G_{15} G_{17} + C_{11} G_{14} w_5 + C_{11} G_{15} w_5 + C_{11} G_{17} w_5} \gg 2\pi \cdot 20 \text{ kHz}$
op-amp 7	999.758	$\frac{G_2 G_4 + C_2 G_2 w_7 + C_1 G_4 w_7}{C_1 G_2 + C_2 G_2 + C_1 G_4 + C_3 G_2 + C_1 C_2 w_7} \gg 2\pi \cdot 20 \text{ kHz}$
op-amp 8	406.268	$\frac{G_7 w_8}{G_5 + G_7 + G_{21} + C_5 w_8} \gg 2\pi \cdot 20 \text{ kHz}$

**Table 3.3:** Minimum required gain-bandwidth product of each operational amplifier to ensure that the gain of the closed-loop transfer function at audio frequencies is sufficiently large.

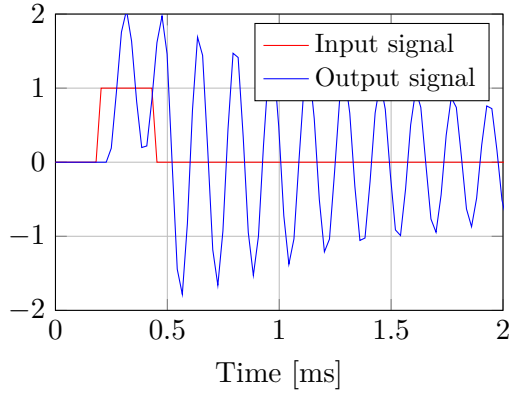
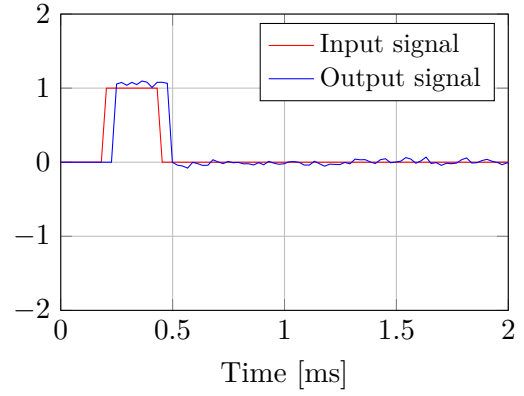
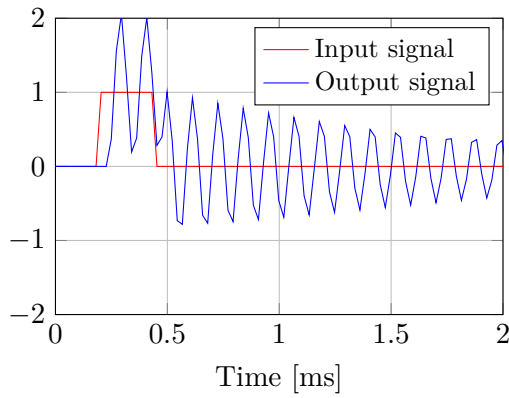
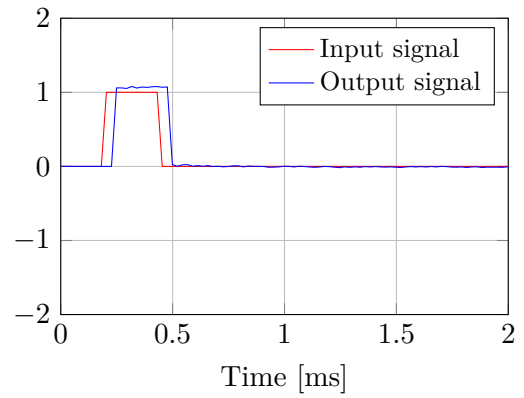
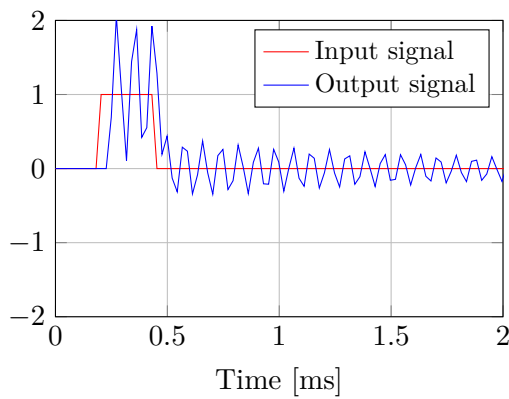
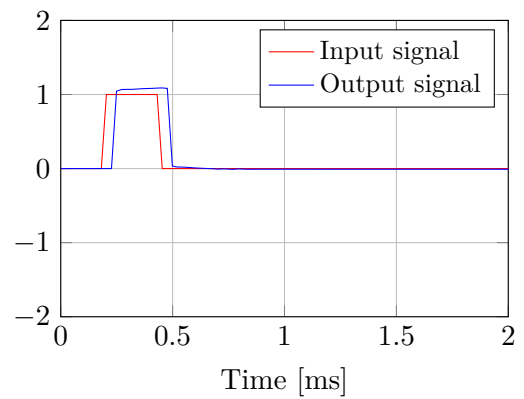
there is a path leaving from node  $V_4$  (see Figure 3.2). In total, there are three areas where this method is applicable, namely from  $V_{\text{in}}$  to  $V_3$ , from  $V_{19}$  to  $V_{14}$ , and from  $V_5$  to  $V_{\text{out}}$

Layed out as a circuit, a discrete-time transfer function still consists of loops. But it is not necessary to break those loops (they already are) and they are much less critical in terms of stability. This can be easily seen by simulation. Figure 3.7 shows the comparison of a pulse response between the untouched circuit and the circuit in which all 3 paths mentioned above were simplified. Unstable behaviour occurs in terms of oscillations which shows up as an audible sound between 3 kHz and 4 kHz. The problem is relaxed for higher oversampling rates, but it can be observed that applying simplifications improves stability even more.

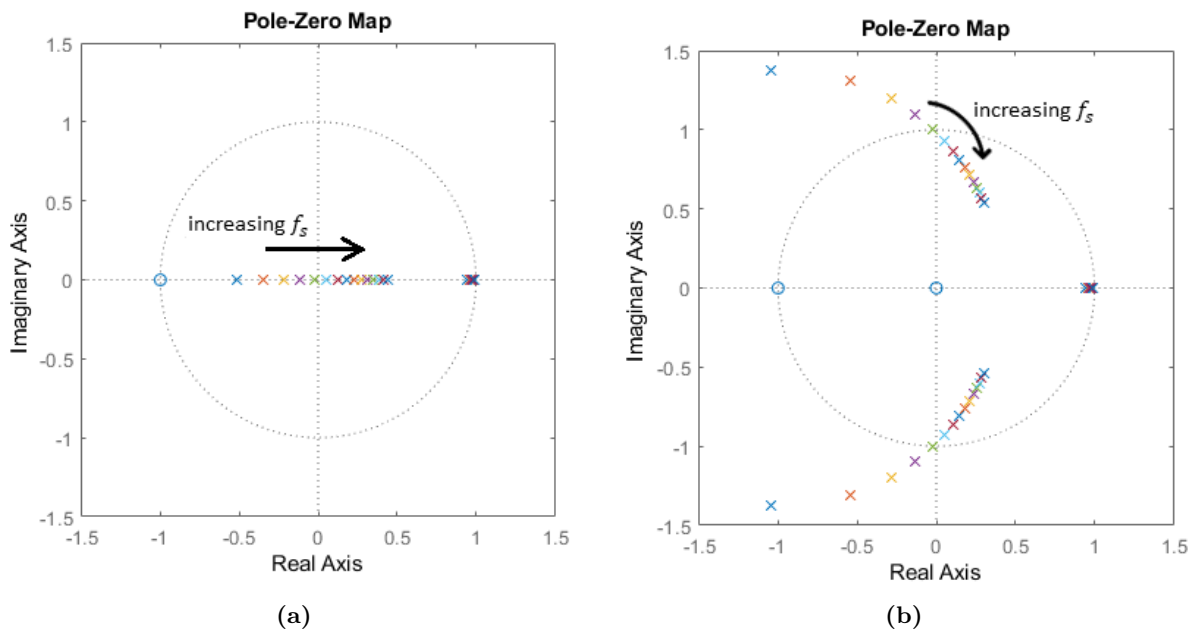
Why this works can be shown by comparing the behaviour of the system achieved by transforming (3.21) into the z-domain with the system in Figure 3.4(c). Consider the complex z-plane showing the zeros and poles of both systems in Figure 3.8. Both systems were simulated using the numerical values from Figure 3.4(a) with  $f_s = 48 \text{ kHz}$  and the oversampling factor was swept from 64 to 512. As usual, many poles get mapped close to  $z = 1$ , but additionally one can observe that the non-simplified system, i.e. the one in which each path is transformed into the z-domain individually, has complex conjugate poles whose absolute value is larger than 1 for low oversampling rates. Thus, it is unstable for certain configurations whereas the simplified system is not.

The disadvantage of this approach is that one loses the ability to model non-ideal effects in their true nature. Modelling non-linear effects becomes difficult once the devices are somewhat hidden inside a transfer function which represents the functionality of many components together as a system. Therefore, one must decide whether more stability or precise non-linear modelling is more important. This question will be answered at the end of chapter 5. Another disadvantage is that the spreading of coefficients becomes larger for higher-order transfer functions, which increases the requirements on the numerical accuracy.

Theoretically, it is possible to apply more simplifications to the SFG if required. Every time there is a series connection of two passive components, they form two loops in the SFG which can be collected to a single loop. But since such double loops are only created by passive components, no high-gain loops can occur. Thus, this simplification is not used.

(a) Without simplifications:  $f_{\text{sim}} = 32 \cdot f_s$ (b) With simplifications:  $f_{\text{sim}} = 32 \cdot f_s$ (c) Without simplifications:  $f_{\text{sim}} = 64 \cdot f_s$ (d) With simplifications:  $f_{\text{sim}} = 64 \cdot f_s$ (e) Without simplifications:  $f_{\text{sim}} = 128 \cdot f_s$ (f) With simplifications:  $f_{\text{sim}} = 128 \cdot f_s$ 

**Figure 3.7:** Influence on the stability when simplifications are applied to the DPSFG. A short unit pulse was applied at the input with  $f_s = 48$  kHz.



**Figure 3.8:** PZ-maps of a simple discrete-time op-amp circuit, showing the behaviour for a simplified system in (a) and a non-simplified one in (b) when the sampling frequency is increased.

## 4 FPGA Implementation

The most important part in this project is to provide a functional use case in which the graph based discrete-time model is implemented on hardware. As already mentioned in the previous chapter, this comes with many challenges and a lot of architecture-dependent considerations must be made. This chapter will discuss all kind of problems and how they are solved when implementing the DPSFG from Figure 3.2 on a FPGA.

Coding is done in VHDL, but code will not be written by hand. Instead, Simulink's *HDL Coder* is used to automatically generate VHDL code based on a Simulink model. Due to the fact that everything is modelled on a low abstraction level, i.e. only basic building blocks are used, DPSFGs are very code generation friendly. Setting up a Simulink model that produces synthesizable VHDL code can therefore be done with relatively low effort. Thus, many aspects in this chapter are explained using Simulink simulations with fixed step size solvers.

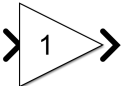
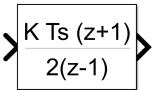
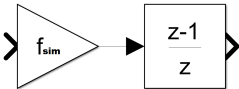
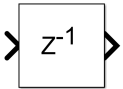
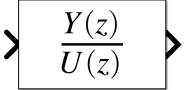
### 4.1 Discrete basic blocks in Simulink

Not every building block in Simulink is supported for automated code generation. Only blocks included in the *HDL Coder* library can be used. This leads to some restrictions, e.g. every block that uses a division internally cannot be used. An overview is given in Table 4.1. Of course, more blocks than the listed ones will be used to emulate special parts of the circuit.

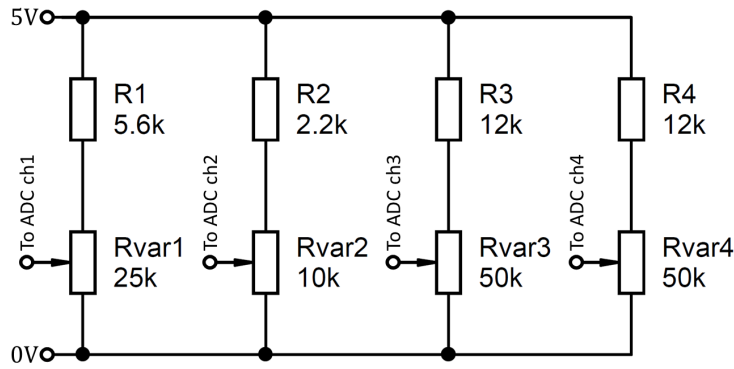
### 4.2 How real-time adjustable blocks are emulated

In total, *Der Faux* has four potentiometers which affect the circuit's behaviour. They will be called  $P_1, P_2, P_3, P_4$  from now on and the numbering refers to the schematic. On a functional level, there are two types of effects that have to be modelled.  $P_1 - P_3$  are variable gains in our DPSFG model.  $P_4$  controls the clock frequency of the VCO and adjusts the delay.

The FPGA hardware will also use real potentiometers in order to give the same look and feel, but their value, i.e. their resistance according to the position of the rotating contact, is passed

Component	Simulink blocks	Note
Gain		Used for resistors, capacitors, and op-amps.
Discrete-time Integrator		Must be set to <i>Integration</i> mode.
Discrete-time Derivative		Cannot be built directly since it requires a division by $T_s$ . It must be constructed by a difference block and a multiplication with the inverse of the sampling time.
Unit Delay		Used to break loops and long logic paths.
Discrete-time Transfer function		Used for driving-point impedances, acquired through Tustin transformation.

**Table 4.1:** Basic discrete-time building blocks in Simulink that are supported for code generation.



**Figure 4.1:** Small circuit to convert four potentiometers into digital values.

digitally to the FPGA. This is simply done by converting the current position by the *LTC2308* ADC provided by the *DE1-SoC* board.<sup>1</sup> The *LTC2308* has eight ADC channels, four of which will be used as single-ended inputs. In this configuration, the voltage range is 0 V to 4.096 V. If potentiometers with the same resistance as the original hardware and resistors from the E24 series are used, then a circuit as the one shown in Figure 4.1 has to be used in order to comply with the given voltage range.

Up to 12Bit resolution are provided by the ADC. They represent the current position of the rotating contact, i.e. from fully open to completely closed. This digital value must be further processed by the FPGA to convert it into a physical property.

#### 4.2.1 Variable gain

The first three potentiometers control the gain of an arrow in the DPSFG. A second aspect, which is not directly visible, is the influence on certain driving-point impedances, namely  $Z_{22}$ ,  $Z_{23}$ ,  $Z_{24}$ , and  $Z_{25}$ . Changing the resistance of a potentiometer changes the coefficients of these transfer functions.

Implementing a variable gain is simple if a real hardware multiplier is used. A signal (first input) is multiplied by a number stored in a register (second input) whose value corresponds to the set resistance. Recall that DPSFGs use admittances rather than impedances. Thus, the relationship between the digital value from the ADC and the conductance is non-linear; it is inversely proportional. Computing the reciprocal of a number can be done by several algorithms with digital logic. However, they come with certain disadvantages such as being iteratively, meaning that they have to use a faster clock internally, requiring a floating-point unit, or using much FPGA recourses. Hence, one wants to avoid computing the reciprocal. Since the set of all possible conductance values is limited by the ADC resolution anyway, it is beneficial to use look-up tables.

One has to be careful which interval of resistance values is used. The *LTC2308* is a mid-tread quantizer with  $2^N$  levels and  $2^N - 1$  decision thresholds. The question is to which resistance should the highest and lowest quantization level correspond to? The problem here is that the variable resistance comes in two forms, namely  $1/R_{\text{var}}$  and  $1/(P - R_{\text{var}})$  where  $P$  denotes the maximum possible resistance of the potentiometer. Neither the lowest nor the highest value are allowed because this would yield a division by zero. Therefore, the following representation is used:

<sup>1</sup>Notice that only *DE1-SoC rev.E* or newer boards use this type of ADC.

The whole range is quantized into  $N + 2$  levels, i.e.

$$R[k] = k \cdot \frac{P}{2^N + 1}, \quad k = 0, 1, \dots, N + 1. \quad (4.1)$$

This yields a step size of  $P/(2^N + 1)$  rather than  $P/2^N$  which would normally be used. A look-up table then stores the inverse of the calculated values, but the first and last entry in (4.1) are omitted. This symmetrical approach has the advantage that the same look-up table can be used for both cases, i.e. it can represent  $1/R_{\text{var}}$  and  $1/(P - R_{\text{var}})$  at the same time. For the latter one, all that has to be done is to reverse the order of the indices. The following Matlab code snippet should clarify the procedure:

```

1  P = 50e3; % Max. value of poti
2  N = 12;   % Resolution of ADC
3
4  step = P / (2^N+1);
5  range = 0:step:P;
6  LUT = 1./range(2:end-1); % LUT values for 1/Rvar
7  LUT_rev = fliplr(LUT);   % LUT values for 1/(P-Rvar)

```

**Listing 4.1:** Matlab code to compute the look-up tables for variable gains.

Additionally, this also ensures that a division by zero never occurs. Technically, one can store the look-up table values only once on the FPGA. But since it is not possible to read multiple entries at once, one has to use a clock that is twice as fast to read out the values and multiplex between the normal and reversed indices.

Threatening variable coefficients of a transfer function in the  $z$ -domain is the more difficult part. A driving-point impedance in the laplace domain is always a first-order transfer function in the form

$$Z = \frac{1}{sC + G_{\text{var}}} \quad (4.2)$$

where  $C$  and  $G$  are representative for the sum of any capacitance and conductance seen by that node. The subscript  $\text{var}$  should emphasize that this sum becomes adjustable. Transforming (4.2) into the  $z$ -domain by applying the Tustin transformation from Table 3.2 yields

$$z_0 = \frac{Y(z)}{X(z)} = \frac{T_s z + T_s}{(2C + G_{\text{var}} T_s)z + G_{\text{var}} T_s - 2C}. \quad (4.3)$$

By multiplying the nominator and denominator by  $z^{-1}$ , (4.3) can be rearranged to

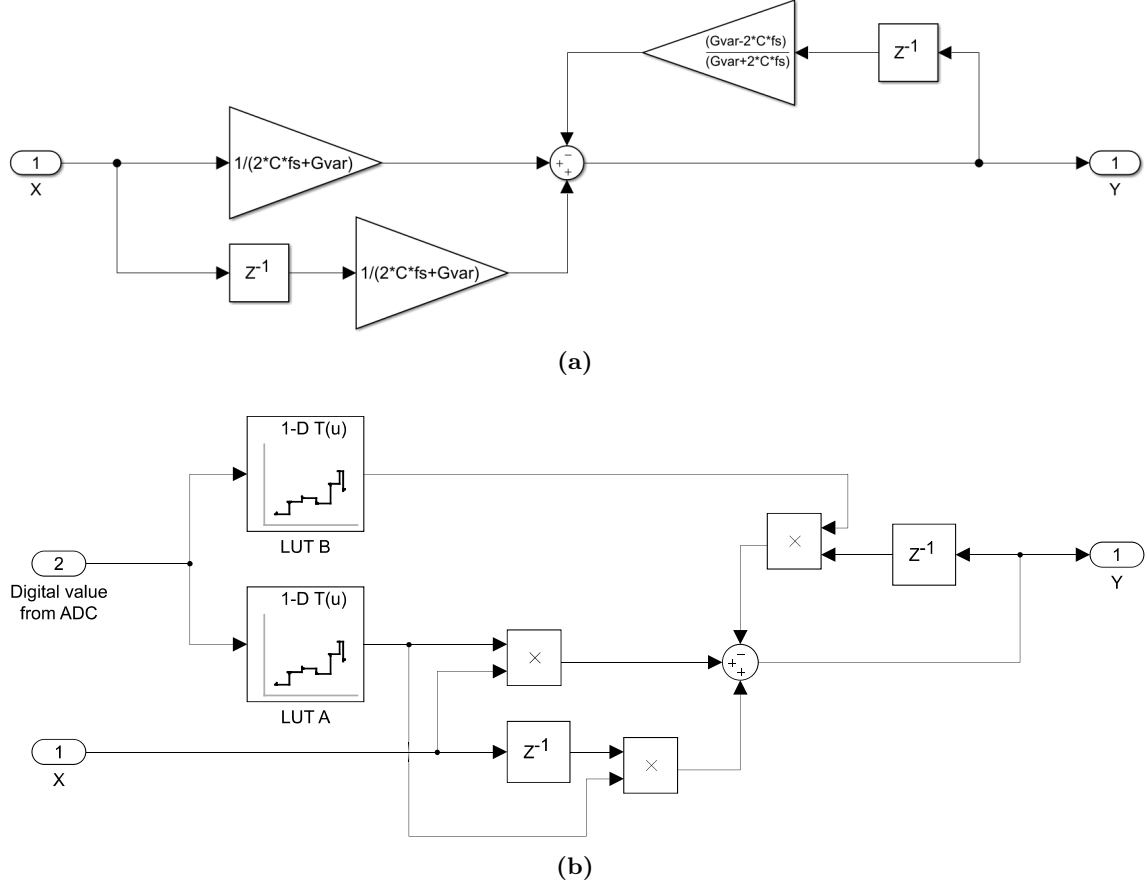
$$X(z) \cdot (T_s + T_s z^{-1}) = Y(z) \cdot (2C + G_{\text{var}} T_s + [G_{\text{var}} T_s - 2C]z^{-1}) \quad (4.4)$$

which corresponds to the following difference equation:

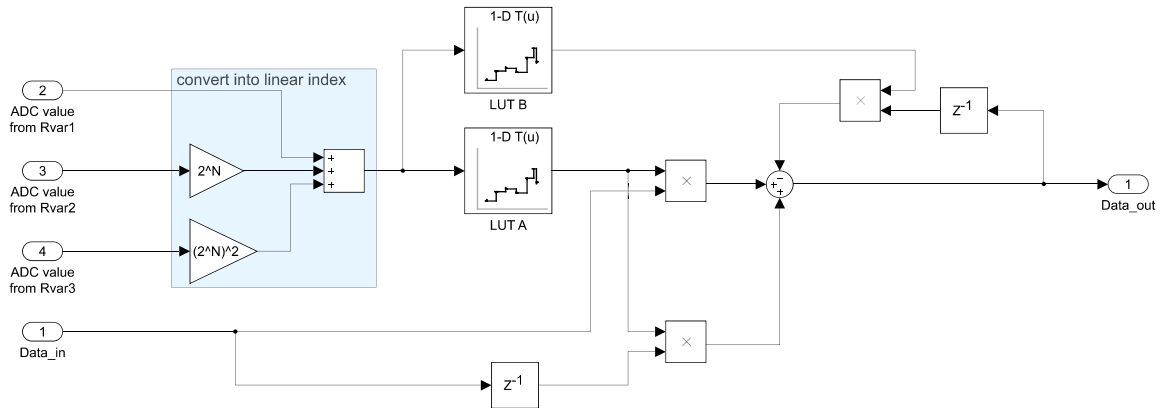
$$T_s x[k] + T_s x[k - 1] = (2C + G_{\text{var}} T_s) y[k] + (G_{\text{var}} T_s - 2C) y[k - 1] \quad (4.5)$$

The circuit that elaborates this equation is shown in Figure 4.2(a). Notice that the sampling time  $T_s$  has been replaced by the sampling frequency  $f_s$ . Again, this relationship is non-linear in the sense that  $G_{\text{var}}$  appears in the denominator which would require a division. Therefore,





**Figure 4.2:** Circuit of a driving-point impedance consisting of one capacitor  $C$  and one variable resistor  $G_{var}$ , implemented in the  $z$ -domain with the Tustin method in (a), and implemented using look-up tables in (b).



**Figure 4.3:** Circuit of the driving-point impedance  $Z_{23}$ , demonstrating how a 3D look-up table can be implemented using conventional 1D look-up tables.

two additional look-up tables are required for each driving-point impedance that consists of a variable resistor. The resulting circuit is given in Figure 4.2(b).

What if a driving-point impedance has multiple variable resistors? In the case of  $Z_{22}$ ,  $Z_{24}$ , and  $Z_{25}$  this is rather easy to solve. All three nodes have in common that they see only one potentiometer at once. Consider node 24 to make a concrete example. Its driving point impedance is given by

$$Z_{24} = \frac{1}{s C_{20} + 1/R_{\text{var3}} + 1/(P_3 - R_{\text{var3}})}. \quad (4.6)$$

Here one can take advantage of the symmetrical notation from (4.1), meaning that the conductance of  $G_{\text{var3}} = 1/R_{\text{var3}}$  at index  $k$  is equal to the reversed value  $1/(P_3 - R_{\text{var3}})$  at index  $(\text{end} - k)$ . In other words, no complicated computing is required in order to calculate the look-up tables. One can use the results from the existing tables. This is again understood best with the help of some Matlab code:

```

1  C20 = 1e-6;           % Value of capacitor C20
2  N = 12;              % Resolution of ADC
3  fs_sim = 48000*150;   % Simulation frequency
4
5  LUT_Z24a = zeros(1,2^ADC_res); % Prepare look-up tables
6  LUT_Z24b = zeros(1,2^ADC_res);
7  for k=1:2^ADC_res
8      LUT_Z24a(k) = 1/(LUT_Gvar3(k) + LUT_Gvar3_rev(k)+2*C20*fs_sim);
9      LUT_Z24b(k) = (LUT_Gvar3(k) + LUT_Gvar3_rev(k)-2*C20*fs_sim)/...
10                     (LUT_Gvar3(k) + LUT_Gvar3_rev(k)+2*C20*fs_sim);
11 end

```

**Listing 4.2:** Matlab code to compute the look-up tables for driving-point impedances based on existing look-up tables for variable gains.

Here, the variables `LUT_Gvar3` and `LUT_Gvar3_rev` are the look-up tables for the variable gain based on the calculations from Listing 4.1. Unfortunately, from an implementational point of view, one cannot profit from it because it is not possible to bypass the division. In the end, using multiple look-up tables might be more efficient than using existing ones with division.

Consider Table 4.2 to get an insight. The resource utilization for logic modules (ALUT), registers (Reg.) and block RAM in Bits (BRAM) with 16 Bit, 32 Bit, and 64 Bit word length is given. Notice that those values have to be considered as an estimation because they show the utilization before the fitting process, i.e. no optimization has been used yet.

Simulink supports two architectures for division. *Linear* means that a "/" symbol is put into the VHDL code. In other words, the synthesis tool decides how to implement the division. *Shift and Add* computes the quotient by performing multiple shift and add operations. It is more accurate – according to the documentation – but requires much more resources. The size of a look-up table is dependent on the ADC resolution, i.e. every bit doubles the amount of entries in the table. It is unlikely that the full resolution of 12 Bit will be used because it is not necessary to quantize the potentiometer position into 4096 levels. To provide a more realistic scenario, the resource utilization for a 6 Bit look-up table is also given in Table 4.2. Comparing the number of logic modules used, one can see that even a 12 Bit LUT is as big as a division block. Considering the numbers for a 6 Bit LUT, there is no doubt that implementing multiple LUTs is more efficient than using division.

Word length	Division (linear)			Division (Shift & Add)			1D LUT (12 Bit res.)			1D LUT (6 Bit res.)		
	ALUTs	Reg.	BRAM	ALUTs	Reg.	BRAM	ALUTs	Reg.	BRAM	ALUTs	Reg.	BRAM
16 Bit	412	0	0	400	822	32	1327	0	0	25	0	0
32 Bit	1351	0	0	1732	2662	816	2639	0	0	41	0	0
64 Bit	4613	0	0	5926	9190	3857	5046	0	0	68	0	0

**Table 4.2:** Comparison of resource utilization between look-up tables and division blocks for various word lengths, table sizes, and division block architectures. Results are taken from Quartus Prime 18.1 without optimization for a specific architecture.

Very critical is  $Z_{23}$ . It is the node at which all three potentiometers are connected to. Mathematically, the driving point impedance at that node is a function with three independent variables, yielding a  $2^N \times 2^N \times 2^N = (2^N)^3$  look-up table. In other words, every bit more resolution leads to  $2^{3(N+1)}/2^N = 2^3 = 8$  times more values that need to be stored. An eightfold increase for each bit blows up the resource requirements dramatically. Thus, not the full resolution can be used to represent resistor values. Implementing a 3D look-up table can be done by adding together ADC values with a corresponding offset as shown in Figure 4.3.

#### 4.2.2 Variable delay

In the real circuit, a variable delay was realized by changing the clock frequency which drives a delay line (shift register) of fixed size. Changing the frequency at runtime is, of course, not possible in a discrete FPGA implementation because this would consequently require to recompute all transfer functions in the z-domain. Therefore, the variable delay must be implemented by a different approach.

Creating a variable delay can be done in two ways, both of which have their benefits and drawbacks. Therefore, both concepts are introduced and compared afterwards.

*Solution I: Using a ring buffer with adjustable read pointer*

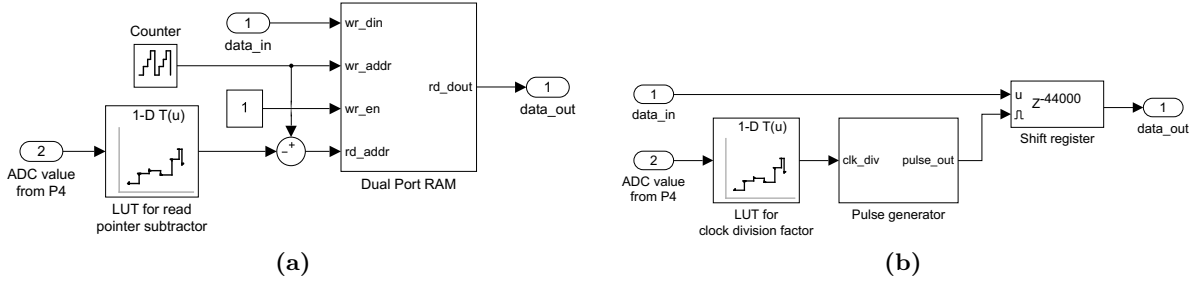
The first approach is based on the idea that the whole history of audio samples is stored inside a ring buffer. Essentially, this is a dual-port RAM block with two pointers, one for the read address and one for the write address. The write address is generated with a  $N$ -bit counter where  $N$  is the bit width of the RAM address space. Once the highest possible value is reached, an overflow will occur and the start of the RAM is written again. The read pointer on the other hand is generated by subtracting an integer value from the counter. This integer represents the delay in number of clock cycles and will be changed according to the potentiometer's position. We will call this idea *pointer method* for later references. The whole concept is shown in Figure 4.4(a). The look-up table converts the ADC value from P4 to the corresponding number of samples. The variable delay in this approach occurs due to the changing distance between the read and write pointer.

This approach is very memory consuming, which is one of the disadvantages. The size of the buffer must be large enough to store enough samples such that the maximum delay can be represented. Denoting  $\tau_{\max} \approx 603$  ms as maximum delay, the required memory in Bits is

$$N_{\text{bits}} = \lceil \tau_{\max} \cdot f_{\text{sim}} \rceil. \quad (4.7)$$

If the sampling frequency is large, then (4.7) becomes large too. Therefore, the ring buffer should be implemented on block RAM (BRAM) or even external DRAM.

The interesting question is, what happens during the transition, i.e. when the delay is changed during run-time? If the delay is decreased, then the read pointer skips certain addresses and

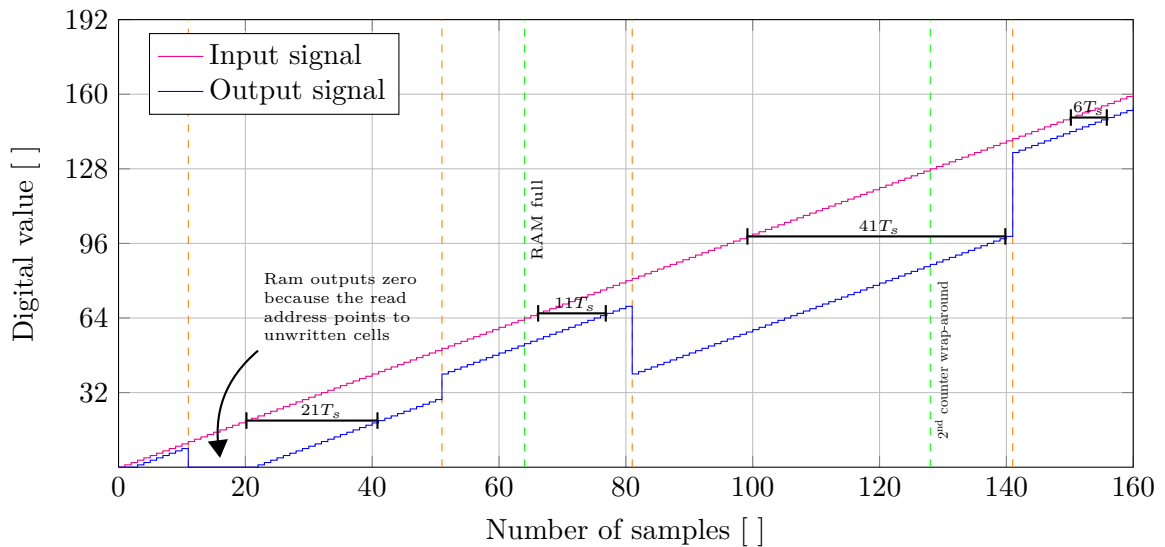


**Figure 4.4:** Concept for implementing a variable delay in a FPGA using a ring buffer with adjustable read pointer in (a), and a shift register with controllable enable port in (b).

jumps to a newer sample. If the delay is increased, then the pointer jumps to an older sample and outputs previously stored samples again. The output will be zero if the counter has not reached its maximum value once in that case. All three effects are illustrated in Figure 4.5.

The input signal is just a counter going up. The RAM is assumed to be reset and uses a 6 bit address which can store 64 samples therefore. Orange dashed lines indicate locations at which the delay is changed, i.e. where potentiometer  $P4$  was adjusted. After the first delay change, the read pointer reads from the last addresses of the RAM which have not been written yet, yielding zeros at the output. At 64 samples (green dashed line), an overflow occurs and writing on the RAM starts over again.

In general, this method does not imitate the behaviour of the real hardware precisely. *Der Faux* uses a shift register of fixed size whose samples are shifted at different rates depending on the VCO's frequency. Changing  $P4$  during runtime outputs already stored samples slower or faster whereas the approach described above skips certain samples or outputs old ones. As a consequence, the effect of a shifting pitch cannot be reproduced by the FPGA, which is another downside. The advantage of this approach is that it has a very high resolution, i.e. delay times can be represented very precisely since the only limit is the resolution of the ADC.



**Figure 4.5:** Simulation of the variable delay by using a dual-port RAM block.

*Solution II: Using a shift register with controllable enable ports*

The second approach uses a register bank formed by 44'000 flip-flops whose *enable* port is connected to a pulse generator. This essentially divides the main clock into lower sampling rates which can be used to create a variable delay. An illustration is given in Figure 4.4(b). Here, the LUT converts the values from the ADC to a clock division factor which corresponds to the delay times. It is given by

$$N_{\text{clk}} = \left\lfloor \frac{f_{\text{sim}} \cdot \tau}{44'000} \right\rfloor. \quad (4.8)$$

The pulse generator is simply a counter which counts from  $N_{\text{clk}} - 1$  down to zero. Every time it reaches zero, a pulse is generated. Thus, we will refer to it as *pulse method*. Of course, it is also possible to map this approach into the RAM rather than using flip-flops. In that case, the circuit in Figure 4.4(a) is used, but the subtrahend is a constant (44'000) and the write enable port is controlled by the pulse generator as well as the write pointer counter.

The main advantage of this method is that it is a faithful recreation of the original hardware in the sense that samples are shifted through a memory of fixed size with variable speed. Consequently, the effect of shifted pitches can be reproduced as well as the fact that the sampling time becomes variable. If the decision of the comparator is only passed after  $N_{\text{clk}}$  clock cycles, then this has the effect that the integration time of the delta modulator is changed, which is very close to what the original hardware does. Additionally, it is very memory efficient.

The downside is its resolution. One is restricted to integer multiples for the clock division factors in a synchronous design. Especially at low delay times the resolution is bad, meaning that the step to the next larger delay time is much larger than what would be possible by the ADC resolution. Another problem is that the minimal division factor is 2. Depending on the sampling frequency, very short delay times cannot be represented. Both problem can be addressed if a very fast base clock is used or when the whole shift register process is moved to another clock domain. The latter one is problematic since the need of clock synchronization circuits introduce additional delay which is not desired in the modulator path. In the simplest case, a clock synchronizer consists of three flip-flops where the first one is connected to clock domain A and the other two to domain B. Thus, a total of seven cycles of delays are introduced, four of which in the faster domain and two in the slower. Additional delay in the modulator highly increases granular noise since the decision of the comparator lags behind by several clock cycles.

From a theoretical point of view, the pulse method is superior to the pointer method. However, due to the limitations mentioned above the pulse method might not be possible to implement – at least not without some restrictions – whereas the pointer method can be used for sure since its only real limitation are the available RAM resources of the FPGA. If needed, one can still use external RAM if the internal BRAM is not sufficient.

### 4.3 Comparator logic

Recall from chapter 2.2.2 that the physical representation of the 1-bit datastream is unknown. It is likely that logic levels are equal to the output voltage that the comparator generates. However, one is not interested in this representation since all that matters is the modulated current which flows into the integrator. Therefore, the comparator logic can be implemented in an idealized way.

The input signals from the delta modulator (pin 15 and pin 9 in Figure 2.6) are compared to each other. If the non-inverting input is larger, then  $+G_{\text{mod}}$  is assigned,  $-G_{\text{mod}}$  otherwise. Mathematically,

$$V_{\text{out}} = \begin{cases} 0 & V_p = V_n \\ +G_{\text{mod}} & V_p > V_n \\ -G_{\text{mod}} & V_p < V_n \end{cases} \quad (4.9)$$

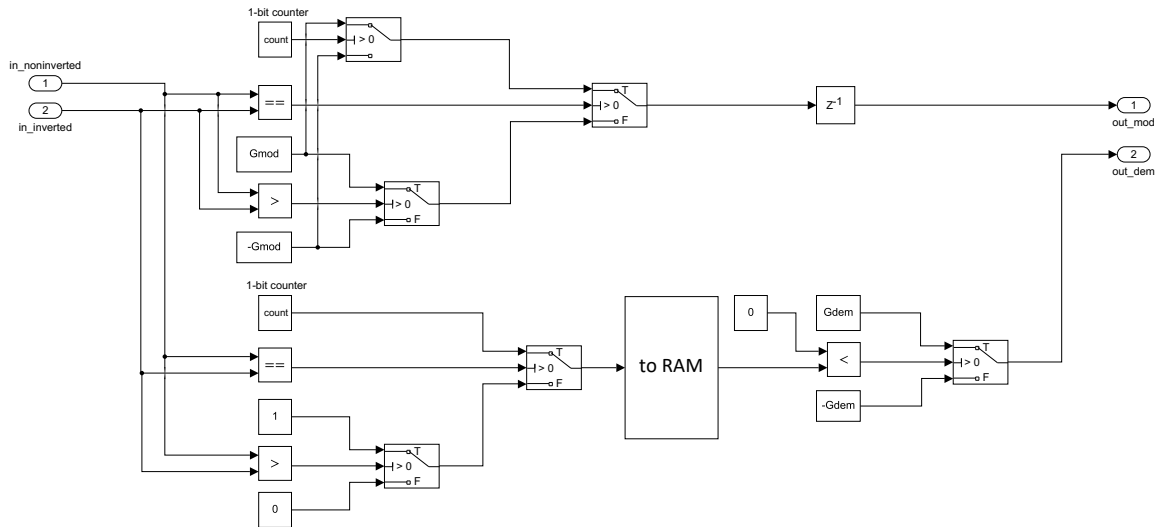
where  $V_n$  and  $V_p$  are the inverting and non-inverting input terminals, respectively.

In the case of the demodulator path, the logic is slightly different. Fortunately, it is not necessary to store the full word size inside the RAM. It is sufficient to store a boolean value which corresponds to the decision. After the RAM, some digital logic checks whether the sample is true or false and outputs  $\pm G_{\text{mod}}$ .

What if the input signals are exactly equal? In the real world, this does not happen due to noise. Consequently, the comparator is forced to make a decision and outputs an alternating sequence of zeros and ones. At least, this is the most likely scenario statistically. In a discrete-time simulation with discrete numbers, it is possible that signals are equal. For example, at the start-up when there are no samples stored yet, the feedback signal will be zero and so is the input signal.

One can imitate the behaviour of the analog world by additionally checking whether the input signals are equal. In that case, an alternating sequence of  $\pm G_{\text{mod}}$  is generated for the modulator, and  $\pm 1$  for the demodulator. Figure 4.6 shows the corresponding circuit that implements all those functionalities.

The switch blocks are the hardware-equivalent to an *if statement*. They connect the upper path if the condition of the middle terminal is met, otherwise the lower path is connected. A simple 1-bit counter can be used to generate the alternating sequences in the case of equal signals. This architecture is very efficient in terms of resources because the digital logic required to compare signals and output fixed numbers is small, and the RAM only needs to store a minimum amount of information per sample; namely a single bit.



**Figure 4.6:** Digital logic of the comparator which generates the bitstream for the RAM and the modulated currents.

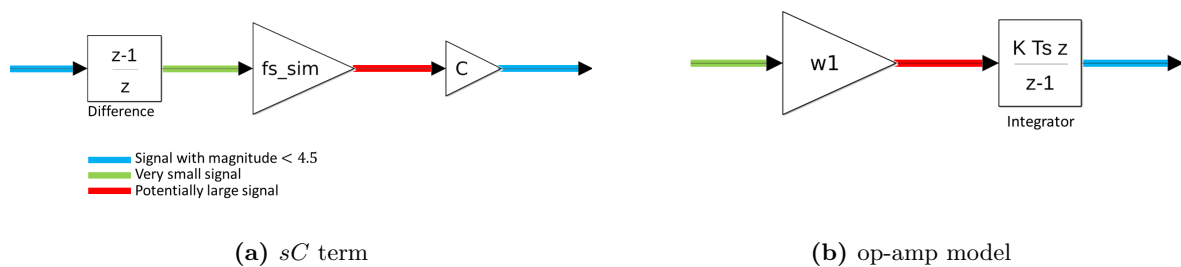
## 4.4 Fixed-point Arithmetic

By default, Simulink uses floating-point numbers for its simulations. Floating-point numbers are difficult to implement on a FPGA since they require a special arithmetic unit which takes a lot of the available resources away. Thus, it is desired – if not necessary – to use fixed-point arithmetic. The challenge here is the huge dynamic range of analog components. The largest multiplications in the circuit are caused by the op-amp models and lie in the range of  $10^6$  whereas the smallest ones, which are due to capacitors, are around  $10^{-12}$ . These are 18 (!) orders of magnitude. In theory, it requires  $\log_2(10^{18}) \approx 60$  bits to represent this dynamic range. The problem becomes even worse since integrators accumulate very large or small values. Additionally, multiplying a 60 Bit signal requires a temporary 120 Bit register. It is unlikely that a model with this complexity fits on a conventional FPGA without reducing the word length. Therefore, it is necessary to find out which word size is required in order to run the simulation without over- or underflows.

### 4.4.1 Theoretical derivation

Digital audio in consumer devices is usually represented with 16 bit resolution where 15 bits are fraction digits. The samples are normalized to the interval  $[-1, 1)$ , meaning that the maximum level which enters the circuit is a sequence of ones<sup>2</sup>. This resolution is much smaller than what is required internally. There are several critical parts in the model that might produce very small or very large numbers.

Theoretically, it is not possible that any signal exceeds the limit given by the electrical characteristics. With a power supply of  $V_{DD} = 9\text{ V}$ , the largest possible swing is  $\pm 4.5\text{ V}$ . Thus, no more than  $\lceil \log_2(V_{DD}) \rceil = 4$  integer digits are required. However, internal signals can exceed this limit. For example, consider a  $sC$  term in the DPSFG. The discrete implementation of this path is a discrete-time derivative which consists of a difference block and a division by the sampling period (= multiplication with sampling frequency), followed by a multiplication with  $C$  as illustrated in Figure 4.7(a). If a audio signal with magnitude  $\leq 4.5$  enters this path, a temporarily large signal appears due to the multiplication with  $f_{\text{sim}}$ . After the gain block with a value of  $C$ , the signal enters its normal range again. In other words, while the dynamic range of both signals might be approximately the same, their position of significant bits differs by a lot.



**Figure 4.7:** Discrete models of some electrical components and their expected numerical behaviour.

<sup>2</sup>Strictly speaking, the highest possible value is 0.999969482421875 since the integer bit is also the sign bit.

Another problematic part are the integrators produced by the op-amp model as shown in Figure 4.7(b). A signal entering its input is multiplied by the  $GBW$  of the op-amp, which is in the range of  $10^6$ , and then accumulated. Signals at the input nodes of op-amps are normally very small due to the large gain of the feedback. However, it might still be possible that large values occur here. Notice that the integrator blocks include an additional multiplication by  $T_s$ , which decreases the magnitudes of the signals that are accumulated highly.

Determining the resolution in the fraction digits can be estimated by considering the precision of digital audio. The smallest possible value is  $2^{-15} = 30.517578125 \cdot 10^{-6}$ . This is also the smallest possible difference between two samples and thus the smallest value a difference block would produce. The smallest multiplication in the circuit is caused by  $C_2$ . Hence, the resolution in the fraction bits is

$$\left\lceil -\log_2 \left( 2^{-15} \cdot 5 \cdot 10^{-12} \right) \right\rceil = 53 \text{ bit.} \quad (4.10)$$

Of course, when signals with higher resolutions are used internally then even smaller numbers can emerge. However, it is unlikely that this will affect the sound of the audio signal. In conclusion, it is difficult to determine the behaviour of the model on a theoretical basis. Therefore, a numerical analysis using simulation results should be performed.

#### 4.4.2 Numerical analysis

The required word length can be found as follows: Drive the model with maximum signal levels and set all variable resistors to their minimum value, i.e. close to zero. This ensures that the maximum amount of signal is fed back and added at the output. Additionally, set the delay to a short value to ensure that the demodulator produces an output. Then observe each signal and investigate its dynamic range and resolution.

Theoretically, the hardware can be driven with signal levels of  $\pm 4.5 \text{ V}$ . This is the limit given by the power supply of  $9 \text{ V}$ . However, since the audio enters the model digitally, it is not possible that a value larger than  $\pm 1$  is applied. In order to generate the largest possible swing of all signals, a sequence of  $+1$ , followed by a sequence of  $-1$ , followed by an alternating sequence of  $\pm 1$  to produce the highest possible difference between two consecutive samples should be applied at the input. If a simulation with this signal runs without overflows, then the resolution is sufficient. Be aware that this evaluation provides no information about the required number of fraction bits.

Simulink offers an option where it sets the comma for the fixed-point arithmetic automatically, given the word length. Performing the above test showed that no less than 63 Bit resolution is required until no error occurs. 63 bits are huge and do not fit on a conventional FPGA due to the complexity of the model. Thus, it is necessary to optimize the arithmetic which will be explained in chapter 4.5.3.

#### 4.4.3 Error analysis

With a finite resolution available for each signal, there is always a quantization error around. This error occurs when a real-world value cannot be expressed by the available resolution, or when a fixed-point number is converted to another fixed-point number with lower resolution. Typically, rounding is done to the nearest representable number in negative direction (= floor rounding) since this is what digital logic does.



Suppose that the quantization error  $\Delta = 2^{-\text{FractionLength}}$  is a uniformly distributed random variable. By rounding towards the next smaller integer, the maximum error is  $-\Delta$ . The expected (mean) error of a signal  $x$  is

$$\bar{\Delta} = \frac{1}{\Delta} \int_0^{\Delta} x \, dx = \frac{\Delta}{2}. \quad (4.11)$$

The RMS error on the other hand is given by

$$\Delta_{\text{RMS}} = \sqrt{\frac{1}{\Delta} \int_0^{\Delta} x^2 dx} = \frac{\Delta}{\sqrt{3}}. \quad (4.12)$$

With (4.12) one can calculate a signal-to-noise ratio as

$$\text{SNR} = 20 \cdot \log_{10} \left( \frac{X_{\text{RMS}}}{\Delta_{\text{RMS}}} \right) = 20 \cdot \log_{10} \left( \frac{\Delta 2^N / (2\sqrt{2})}{\Delta / \sqrt{3}} \right) = (6.02 \cdot N - 4.26) \text{dB} \quad (4.13)$$

which is the well-known 6 dB per bit result, meaning that the SNR is improved by 6 dB with every added bit.

Now let us consider another rounding method where a number is rounded to the nearest representable number, meaning that it is rounded up or down depending on which way results in a lower error. The expected and RMS error are:

$$\bar{\Delta} = \frac{1}{\Delta} \int_{-\Delta/2}^{\Delta/2} x \, dx = 0 \quad (4.14)$$

$$\Delta_{\text{RMS}} = \sqrt{\frac{1}{\Delta} \int_{-\Delta/2}^{\Delta/2} x^2 dx} = \frac{\Delta}{\sqrt{12}} \quad (4.15)$$

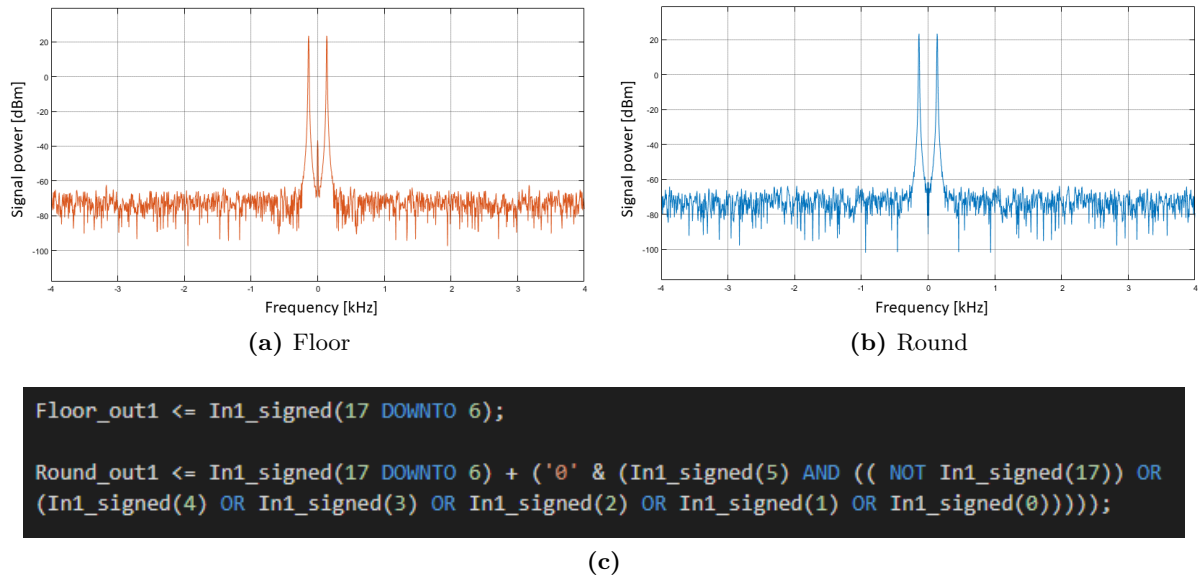
And the SNR is

$$\text{SNR} = 20 \cdot \log_{10} \left( \frac{\Delta 2^N / (2\sqrt{2})}{\Delta / \sqrt{12}} \right) = (6.02 \cdot N + 1.76) \text{dB} \quad (4.16)$$

which also yields an improvement by 6 dB per bit. Comparing (4.14) with (4.11) one can observe that the floor method yields a DC bias for uniformly distributed signals. This bias is an undesired effect that is visible in the spectrum. In Figure 4.8(a) the spectrum of a sinusoidal signal with magnitude 1 and frequency 137 Hz was recorded. The signal was quantized to 12 bit fixed-point arithmetic using the floor rounding method. It can be seen that a peak at DC is present whereas the spectrum in Figure 4.8(b) does not show this artefact. Here, the same test was performed but the real rounding was used. This is an advantage compared to floor rounding.

In terms of RMS error the real rounding in (4.15) also improves the system's noise figure compared to (4.12). However, this advantage comes at a price. The digital logic required to implement this rounding is much larger. Figure 4.8(c) shows the generated VHDL code when a 18 bit signal is converted to a 12 bit number.

Table 4.3 summarizes different rounding methods that are supported by Simulink. Notice that the values are only valid for uniformly distributed signals, i.e. evenly distributed positive and



**Figure 4.8:** Comparison of rounding methods showing the spectrum for floor and real rounding in (a) and (b), and VHDL pseudo code in (c).

negative values. For audio, it is assumed that this assumption holds true. Also notice that this table is incomplete, there are more possible methods available that affect a model differently if input signals are non-uniformly distributed.

Real rounding requires much more logic compared to floor rounding. Hence, there is a trade-off between noise and resource requirements. Real rounding should be preferred as long as there are enough resources available. On the other hand, the available resources could be used to implement a larger word length rather than a rounding logic to improve the noise figure by reducing the quantization error. Furthermore, the circuit includes many DC-decoupling elements in the signal path which get rid of the DC peak in the spectrum. Therefore, real rounding will not be used.

Rounding Method	Description	Cost (Resources)	Overflow possible?	Max. error	Mean error	RMS error
Floor	Round to nearest representable number in negative direction.	None	No	$-\Delta$	$\Delta/2$	$\Delta/\sqrt{3}$
Ceiling	Round to nearest representable number in positive direction.	Low	Yes	$+\Delta$	$\Delta/2$	$\Delta/\sqrt{3}$
Round/ Nearest/ Convergent	Round to nearest representable number.	High	Yes	$\pm\Delta/2$	0	$\Delta/\sqrt{12}$

**Table 4.3:** Comparison of different rounding methods [17].

## 4.5 Optimization

In this section, several optimization techniques are discussed and evaluated. The goal is to reduce the resource and memory requirements of the system in order to fit it on a conventional FPGA as well as improving timing constraints which allows the model to operate at a faster sampling rate.

### 4.5.1 Hardware multiplier optimization

Hardware multipliers are the most important, but also most limited resource on a FPGA. High-performance FPGAs typically have a few hundred 18x18 multipliers. For example, our Cyclone V FPGA support up to 174 hardware multipliers [18]. To give an upper limit, high-level Cyclone 10 chips have close to 400 multipliers.<sup>3</sup> The DPSFG analysis creates up to two multiplication blocks for each component in the circuit. A direct implementation exceeds the amount of available multipliers. Therefore, it is essential to reduce the number of multipliers. There are two fundamental approaches to achieve this: Firstly, share the multipliers through multiplexing. Secondly, implement the multiplication with a combination of shift and add operations. The latter one becomes quite easy once one factors remains constant, which is the case in our model. There are two concepts how this can be implemented.

*Method I: Canonical Signed Digit optimization*

The Canonical Signed Digital (CSD) optimization is used to represent a constant as a sum of numbers with base 2 [19]. A number  $x$  is thus given by

$$x = \sum_{n=0}^N x_n \cdot 2^n \quad (4.17)$$

where  $N$  is the word length in binary representation. The coefficients  $x_n$  are either  $-1$ ,  $0$ , or  $1$ . The CSD algorithm always searches for the representation with the minimum number of additions.<sup>4</sup> For example, the number 231 in CSD form is written as

$$231 = \sum_{n=0}^N x_n \cdot 2^n = -1 \cdot 2^0 + 1 \cdot 2^3 - 1 \cdot 2^5 + 1 \cdot 2^8 = -1 + 8 - 32 + 256. \quad (4.18)$$

Since every part on the right-handed side of equation (4.18) is a power of 2, the multiplication with another number  $y$  becomes a shift operation. Using the same example we can write a multiplication with another number  $y$  as follows:

$$\begin{aligned} 231 \cdot y &= (256 - 32 + 8 - 1) \cdot y \\ &= (y \ll 8) - (y \ll 5) + (y \ll 3) - (y \ll 0) \end{aligned} \quad (4.19)$$

Here,  $\ll$  denotes the left-shift operation. The CSD algorithm requires  $N$  additions in the worst case. Notice that this procedure is not a iterative method by default. If the timing constrains are met, then it is possible to implement the whole operation at once. The advantage of this technique is the fact that it is a purely pre-processing step, meaning that a multiplication can be

---

<sup>3</sup>There are special industrial FPGAs like the Agilex F series from intel with over 10'000 multipliers, but such top-level products are out of scope due to high costs.

<sup>4</sup>Signed fixpoint arithmetic uses 2's complement where addition and subtraction are the same thing.

directly written in this form by the VHDL code. Matlab's hdl coder supports this optimization for constant multiplication blocks [20].

*Method II: Factored Canonical Signed Digit optimization*

CSD minimizes the number of additions and subtractions by searching for a representation which has as many zeros as possible. This approach can be extended to Factored Canonical Signed Digits (FCSD). Here, the constant is first factorized before it is converted to CSD representation. If all factors together have less none-zero digits than the original number, then this type of description improves the performance. Since both CSD and FCSD can be precomputed, it is possible to tell which one optimizes the design best. Simulink can automatically search for the better representation for each constant multiplication.

#### 4.5.2 Reducing memory requirements

The BRAM used to emulate the shift register is already highly optimized by the approach introduced in section 4.3. However, the pointer method from section 4.2.2 is very memory consuming and requires more optimizations.

The architecture from Figure 4.6 implies that a single bit is stored at each address. Depending on the used hardware, this might be impossible. Memory blocks often have a fixed or minimum word length per address. Bypassing this limitation can be done by collecting samples to one word. Every bit more collected in this word halves the required address space. Implementing this requires some digital logic and introduces additional delay, which is not a problem in our case.

Which word length is suited best? FPGAs are a bit special because they have no minimum word size per address, but only certain supported memory configurations. RAM is organized in blocks of 10 kBit, which is the reason why they are called *M10K* blocks. To be more precise, there are 10'240 bits per M10K block. All possible RAM configurations are listed in Table 4.4.

The reason why there are multiple word length options is because of error correction (ECC). For example, if the width is set to 4, then not all 10'240 bits are used. Instead, there is one additional bit left per address which is used for error correction. Errors in individual audio samples are not of importance and ECC is thus not mandatory. Therefore, the best configuration is the one where each M10K block is fully used and the word length is the shortest in order to minimize the additional delay. Hence, the 2048 x 5 Bit configuration is used.

A second optimization is based on the fact that FPGAs often have an odd amount of RAM resources. Our Cyclone V FPGA from the 5CSEA5 series has 397 M10K memory blocks, yielding

Number of addresses (depth)	Word length (width)
256	x40 or x32
512	x20 or x16
1024	x10 or x8
2048	x5 or x4
4096	x2
8192	x1

**Table 4.4:** List of possible memory configuration for Cyclone V devices [18].

a total of  $397 \cdot 10\,240 \text{ Bit} = 4'065'280 \text{ bits}$ . By nature, the counter used to generate the write pointer counts up to a power of two. As a result, only the next lower power of 2 from the total memory could be used. This can be fixed by inserting some digital logic which ensures that the counter only increments up to the highest needed address. This limit is given by

$$n_{\text{samples}} = \tau_{\text{max}} \cdot f_{\text{sim}} \quad (4.20)$$

where  $\tau_{\text{max}}$  is the maximum delay. It is approximately 603 ms according to the measurements from chapter 2.2.3. The required bit width for the RAM addresses is therefore

$$n_{\text{RAM}} = \lceil \log_2(n_{\text{samples}}) \rceil. \quad (4.21)$$

It is important to note that (4.20) is divided by the amount of collected samples if that optimization is used. The modified RAM structure with the two optimizations is shown in Figure 4.9.

First, five 1-bit samples are collected to one word by performing a left-shift operation. The result is summed up and passed to the RAM every 5<sup>th</sup> cycle. Consequently, the ring buffer runs with a five-times slower clock. This is indicated by the colors which illustrate different clock domains.

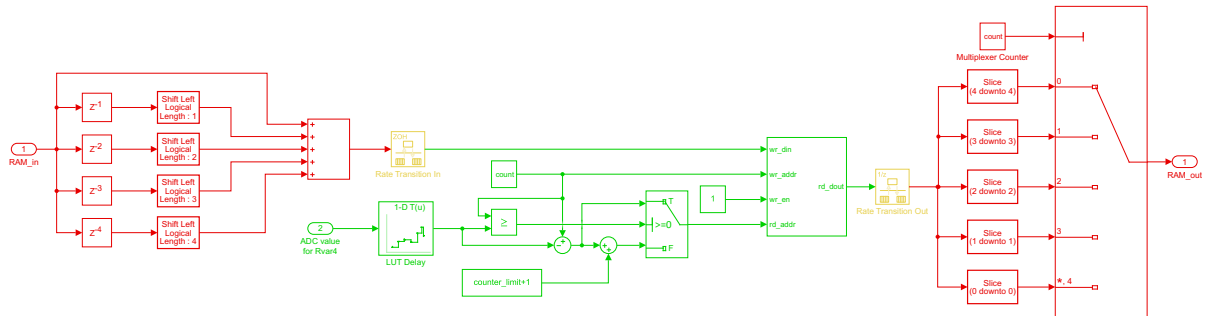
The highest address is given by the variable `counter_limit`, i.e. the read pointer counter increments up to this value before it performs its wrap-around. Also shown in the green part is the look-up table which converts the values from the ADC to the corresponding delay in number of samples. The relationship is given by

$$\tau[k] = \frac{\tau_{\text{max}} - \tau_{\text{min}}}{2^N} \cdot \frac{f_{\text{sim}}}{5} \cdot k + \tau_{\text{min}} \quad , k = 0, 1, \dots, 2^N. \quad (4.22)$$

After the ring buffer, single bits from the 5-bit word are extracted and passed to the comparator logic by multiplexing between the individual bits.

### 4.5.3 Optimizing the numerics

From all improvements made on the design, optimizing the fixed-point arithmetic is the most important one because it is crucial in order to fit the design into the FPGA. As it was shown in section 4.4, no less than 63 Bit word length are required to run the simulation successfully.



**Figure 4.9:** Optimized RAM structure inside the delta modulator showing the ring buffer with some additional logic to collect 5 samples into one word and to generate a pointer that performs its wrap-around at any arbitrarily number.

The idea of the optimization is to set the word length and fraction size of each signal individually such that it has minimum length and sufficient resolution. To do so, the numerical behaviour is investigated with the help of Simulink's *Fixed-Point Tool*. First, a meaningful test case has to be set up. As in section 4.4, a sequence of ones, minus ones, and alternating  $\pm 1$  is applied at the input. The sequence is then repeated by additionally multiplying every sample with  $2^{-15}$  in order to produce the smallest possible values. This is important because not only should overflows be avoided, but also the necessary resolution of each signal should be found. At the same time, the ADC values for the potentiometers  $P1$  to  $P3$  are changed between their minimum and maximum value.  $P4$  stays constant either at a low value to ensure that the demodulator produces an output shortly after the first few samples, or at a large value such that it runs into saturation. The actual optimization process is iterative and works as follows:

- Step 1: Each signal is overwritten by double-precision floating-point data type.
- Step 2: Run the simulation and observe the numerical behaviour of each signal, that is, its magnitude and dynamic range.
- Step 3: Define a safety margin, e.g. 20 %.
- Step 4: Set either a default word length or fraction length. The fixed-point tool now applies the default setting to each signal and optimizes the other parameter while accounting for the defined safety margin.
- Step 5: Run the simulation again with the chosen fixed-point data types and observe if any over- or underflows occur. If not, save the model. If yes, repeat at step 3.

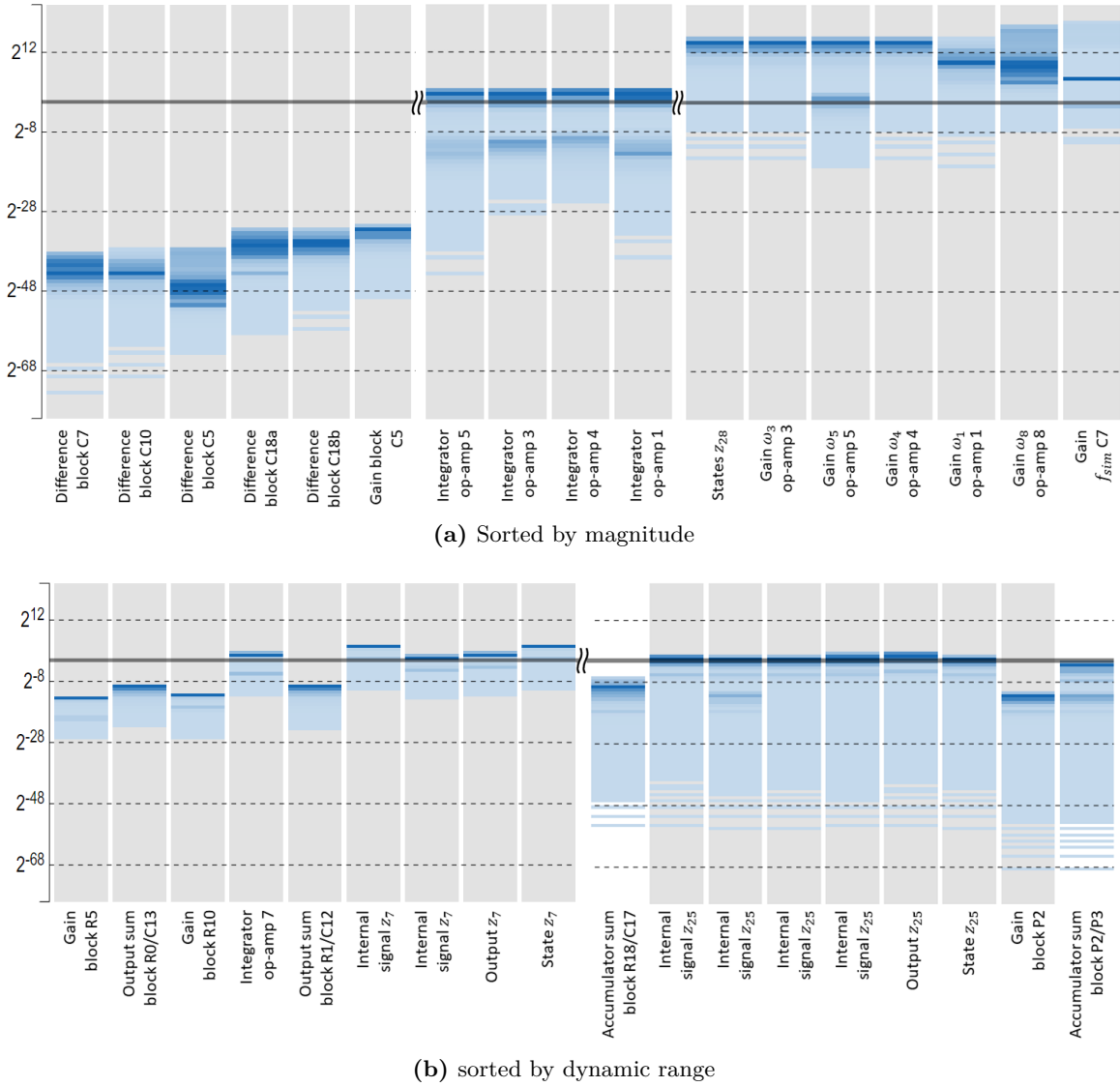
There are a few things to mention when performing such an optimization:

- The outcome is only as good as the test bench. Just because no overflow occurred in the test case does not guarantee that the data types are sufficient for any input. That is the reason why it is important to set up a good test case.
- The fixed-point tool only optimizes output signals of each blocks. Internal signals such as intermediate results from accumulation are still selected inherit. Also, coefficients of transfer functions or any other constant from the workspace must be optimized manually.
- There are a few signals that are allowed to produce an overflow. Those are: any associated signal of the read and write pointer, all integrators from op-amps (they saturate when the signal reaches the supply voltage), and the data type conversion at the output where the internal audio signal is converted to 16-bit audio.
- Any block that works at the RTL level, e.g. bit-shift or slice operations, must be excluded from the optimization since they do not work correctly when overwritten with floating-point numbers.

Figure 4.10 shows step 2 of the optimization process. In Figure 4.10(a), the smallest and largest signals in terms of magnitude are shown. The grey background shows the theoretically possible range, which is entirely covering the histogram because floating-point numbers are used. No grey background means that the data type of this signal is chosen inherit. The blue lines are appearing values during simulation. Darker color indicates that this value occurred more often. Same is valid for 4.10(b) where the signals are sorted by their dynamic range.

Looking at some examples in the histogram, one can observe that very small signals are produced by difference blocks and large ones by the op-amp models. This is somewhat related to what was predicted by the theoretical deviation in section 4.4.1. From 4.10(b) one can read out the highest resolution a signal can take, showing that up to 21 orders of magnitude are covered by a single signal.

After optimization, the histograms form a picture like the one in Figure 4.11(a) which shows some arbitrarily chosen signals. The grey area now surrounds the possible range (blue lines)



**Figure 4.10:** Investigation of the numerical behaviour of the model.

closely, indicating that the datatype has been chosen in a way such that the whole dynamic range is covered. The overlapping grey region on top is the safety margin. Figure 4.11(a) shows how the optimization works in the best case, i.e. when it is possible to cover the dynamic range of signals entirely. In turn, one can see what happens if the optimization does not work well in Figure 4.11(b) and (c) where the chosen integer and fraction bit lengths are too large and too small, respectively. If the resolution is insufficient, then some of the fraction bits are cut away, causing a quantization error.

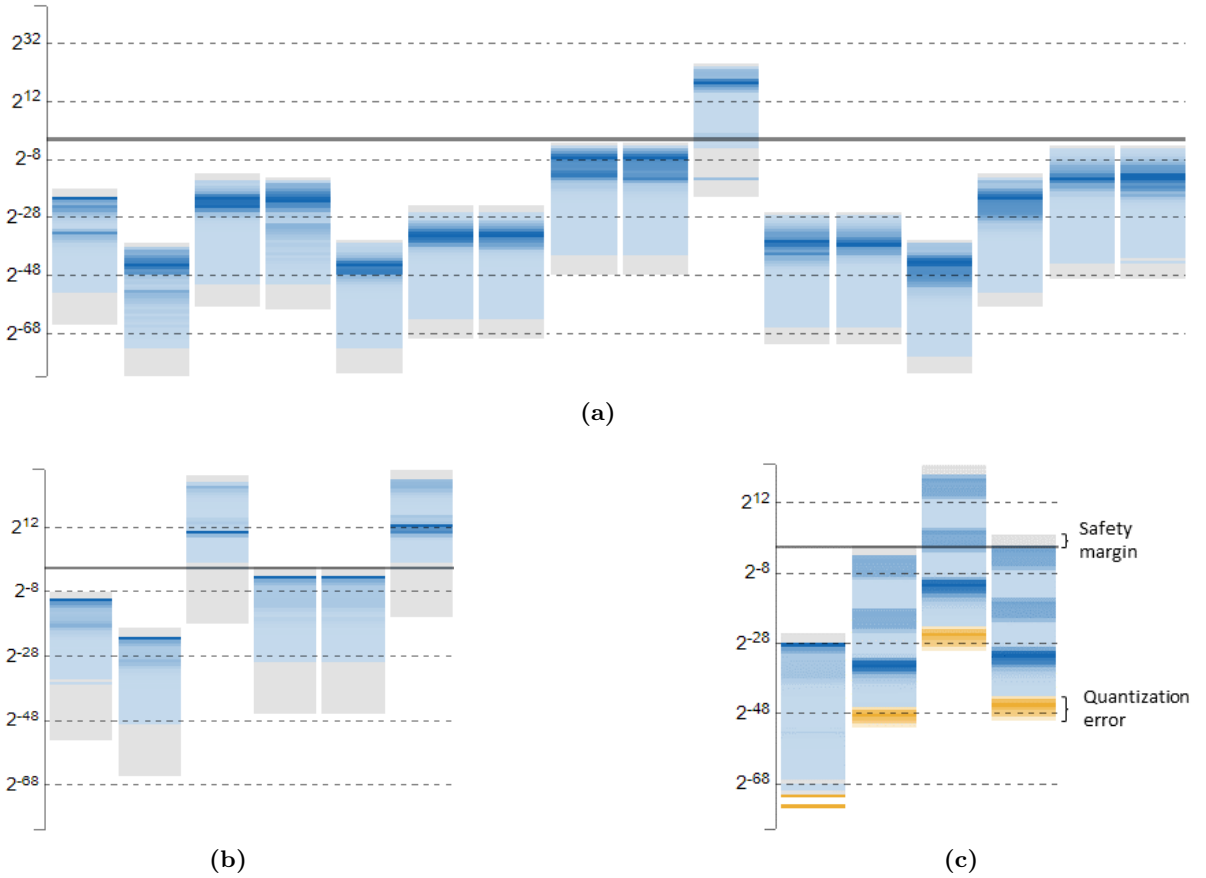
One could perform multiple iterations in which the word size of signals with large dynamics is enlarged, but this leads to a tremendous increase in resource utilization. The problem is that it is very difficult to estimate if the design still fits into the FPGA when the word length is not fixed or at least limited. Furthermore, bits with such a low value might not be of importance for the resulting audio. However, this is also difficult to assess because the relationship between the system's behaviour and quantization errors can be complex.

As mentioned before, other system parameters such as transfer function coefficients or numbers in look-up tables must be optimized manually. This is done by introducing a relative error  $\varepsilon_r$  defined as

$$\varepsilon_r = \frac{|x - \tilde{x}|}{x}. \quad (4.23)$$

Here,  $x$  denotes the real-world value of a coefficient and  $\tilde{x}$  is the approximate fixpoint representation. The word size of the fixed point variable is increased iteratively until the quantization error is less or equal than the relative error  $\varepsilon_r$ . Again there is a limitation given by FPGA resources: The larger the coefficients are the more resources are required for logical operations or for storing the numbers in LUTs.

In conclusion, choosing an appropriate data type for each signal is difficult since it is impossible to cover the whole dynamic range due to resource limitations. The search for the best data types was based on listening tests and is of heuristic nature. A good result that still fits on the FPGA was found at an average word size of 46 Bit and a relative error of 0.001. The resolution for the look-up tables was set to 5 Bit, i.e. only five MSBs out of the 12 bits from the ADC were used. This is not a lot since it corresponds to 32 position levels only. The problem is the driving-point impedance  $Z_{23}$  which, as explained in section 4.2.1, increases by an eight fold with every bit. To illustrate this in numbers, the entity for  $Z_{23}$  alone occupies 13% of the available



**Figure 4.11:** Histogram of various signals after the fixed-point optimization showing a good result in (a), a bad result with too many bits in (b), and a bad result with insufficient resolution in (c).



resources. The LUT for the variable delay based on  $P4$  is an exception since its size grows with the usual  $2^n$  relationship. Hence, its resolution could be set to 8 bit.

#### 4.5.4 Improving timing constraints

Whenever VHDL code is synthesized to digital logic, a worst-case timing analysis is performed which calculates the highest possible clock frequency. This upper limit is given by the most critical logic path, i.e. the path which has the most successive logic elements. The clock period must be longer than the delay propagation through the longest path plus some additional time due to electrical characteristics such as setup time and routing delay.

There exist many concept to optimize FPGA circuits. Basically, they balance throughput, latency, timing, and area (resource requirements and thus costs) against each other. Improving timing can be done by various approaches.

- Insert additional registers in long logic paths.
- Parallelization, i.e. divide processes into smaller, independent operations.
- Remove unnecessary logical structures.
- Balance the distribution of registers.
- Rearrange logic paths by removing components from critical parts and placing them elsewhere.
- Use hardware multipliers for multiplications.

Almost all of those concepts cannot be applied to our graph-based model. The problem is that most signals have a direct dependency to other signals in terms of loops, and introducing additional delays in those loops is not uncritical as explained in section 3.5.2. The ones that are possible are listed below.

##### *Inserting additional registers in long logic paths*

There are a few exceptions where inserting additional registers is uncritical and can be used to relax timing constraints. Whenever there is a path in the DPSFG that has no feedback, a unit delay block can be inserted. This happens if a node has a real voltage source applied to it. In other words, at the output of every op-amp it is possible to include a delay block. This slightly increases the latency of the circuit but this is not relevant since the whole circuit is based on delaying signals anyway. Therefore, by referring to Figure 3.2, additional flip-flops were included after node  $V_3$ ,  $V_5$ ,  $V_{15}$ ,  $V_{12}$ , and  $V_{14}$ .

##### *Removing unnecessary logical structures*

Another small improvement can be made when there are several identical blocks in series, e.g. two constant value multiplications. This is the case at every capacitor since the model requires a multiplication by the capacitance, followed by a multiplication of the sampling frequency. Those two blocks can be merged together.

##### *Using hardware multipliers instead of CSD optimized constant multiplications*

Lastly, if there are enough hardware multiplier recourses one can use them to perform multiplications in critical logic paths. A hardware multiplier essentially also just performs a sequence of shift and add operations, but it is located inside a DSP block which has a lower delay as when the same logic is implemented on the conventional FPGA logic.

Table 4.5 shows the result of the worst-case timing analysis where the insertion of additional flip-flops and the merging of identical blocks has been applied. With a maximum clock frequency of 10.48 MHz, the model can run at an oversampling factor up to 214, which is quite large and considered to be sufficient, i.e. no additional optimizations on timing has to be made.

Clock	$f_{\max}$ [MHz]
Main clock	254.52
Audio codec	109.11
Faux & peripherals	10.48

**Table 4.5:** Maximum possible frequency of all clock domains in a worst-case scenario.

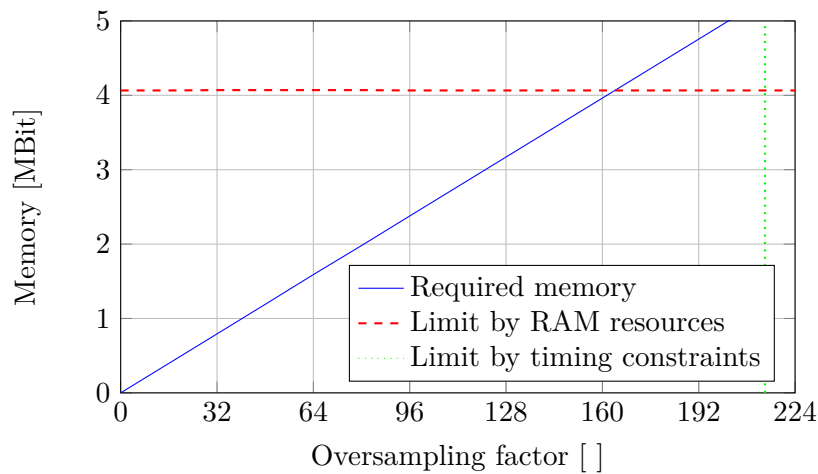
Resource	Utilization	Percentage
ALMs	28'987 / 32'070	90 %
BRAM Bits	3'170'430 / 4'065'280	78 %
DSP blocks	56 / 87	64 %

**Table 4.6:** Resource utilization summary after the fitting process.

## 4.6 Choosing the oversampling factor

There is one open design decisions to be made. Namely, which sampling frequency to use. Many properties such as stability, timing, numerical behaviour, memory requirements, resource utilization, and model accuracy are dependent on this choice. Based on many aspects, it is preferable to use as much oversampling as possible. For example, in section 3.4 it was shown that the accuracy of the bilinear transformation improves. Then in section 3.5.2 one could prove that stability is highly affected by this choice. On the other hand, section 3.4 also demonstrated that higher sampling rates require more precision in the numerics and the pointer method used to model variable delays from section 4.2.2 requires more BRAM resources if the oversampling factor is increased.

As often, there is no closed way to determine the optimal sampling rate. But there are some lower and upper bounds. One of them is the memory requirement. In the approach where the variable delay is created by storing the complete history of audio samples, the amount of needed memory bits increases linearly with the sampling frequency according to (4.7). Another limit is the worst-case timing analysis from Table 4.5. A comparison of both limits is shown in Figure 4.12. Clearly, the available BRAM resources are more critical and limit the oversampling factor to approximately 160. Notice that the BRAM limit only exists in the case where the delay is created by the pointer method, not by the pulse method. If the latter one is used, then an oversampling factor up to 214 is allowed.



**Figure 4.12:** Required memory in number of bits to achieve the full delay for different oversampling factors.

A lower bound is given by the stability constraints from section 3.5.2. If the conditions from Table 3.3 are fulfilled, then stability can be achieved approximately at  $ovs \geq 32$ . Considering the upper bound, a suitable first choice would therefore be 150 since both solutions can work with it. Additionally, this value can be represented exactly by the internal PLL and it is an integer multiple of the clock which runs the audio codec. Thus, the model will run with a simulation frequency of  $f_{sim} = f_s \cdot ovs = 48 \text{ kHz} \cdot 150 = 7.2 \text{ MHz}$ .

#### 4.7 FPGA Framework and synthesis results

Besides the discrete-time model of *Der Faux*, other components are required to operate the overall system properly. The top view of the project is shown in Figure 4.13.

The FPGA runs with a fixed clock of 50 MHz. Two clock domains are needed: a 18 MHz clock for the audio codec and a 7.2 MHz clock for all other sub-systems. Both are produced by an PLL. A reset block waits until the PLL is locked and then releases the reset of all other components.

A I<sup>2</sup>C interface is required to configure the audio codec. It is programmed to output mono audio with 48 kHz sample rate which is then passed to the *Faux* model.

The ADC interface configures the ADC via SPI and also consists of a small state machine that sequentially reads out the four ADC channels. From the 12 Bit resolution, only a fraction of the MSBs are processed. Additionally, for visualization purposes, the ADC values are passed to a 7-segment display to read off the current position of the potentiometers.

The model of *Der Faux* takes the mono audio from the codec and the ADC values as inputs and produces the audio output which is passed to both channels of the audio codec simultaneously.

Finally Table 4.6 shows the total resource utilization of the system. One can see that the CSD optimization introduced in section 4.5.1 sufficiently reduced the amount of hardware multipliers. Further, the available logic elements were driven close to their limit such that there is still some space left for small adjustments, e.g. slightly more bits in the resolution of signals or more resolutions for the LUTs.

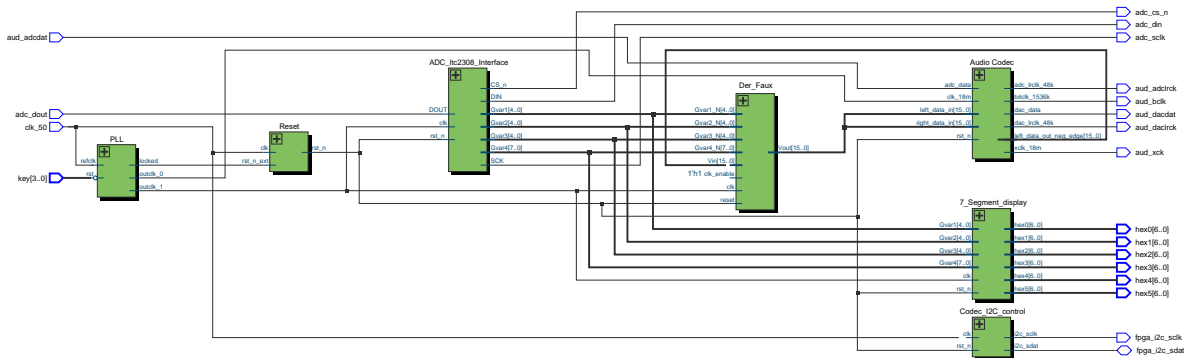


Figure 4.13: Top view of the FPGA framework.

## 5 Modelling non-idealities

By definition, signal-flow graphs can only represent linear relationships of a dynamic system. Technically, it is possible to insert non-linear blocks into a SFG in our case since we do not solve the equation system. The problem is that these functions can get complicated quickly and do not map well into FPGAs therefore. There is a theory in which dynamic systems can be extended to handle non-linear effects [21], [22]. This theory is briefly introduced in this chapter and it is shown how this approach will be used in this work.

### 5.1 Fundamentals of non-linear modelling

Every dynamic system can be represented as a system of first-order differential equations. In general, the state  $\tilde{x}(t)$  of such a system is a linear combination of the input  $u(t)$  and the previous state  $x(t)$  where

$$\tilde{x}(t) = \frac{d}{dt}x(t). \quad (5.1)$$

The output  $y(t)$  is computed as a linear combination of the previous state  $x(t)$  and  $u(t)$ . In summary, the following two equations fully describe a dynamic system:

$$\begin{aligned} \frac{d}{dt}\vec{x}(t) &= \mathbf{A}\vec{x}(t) + \mathbf{B}\vec{u}(t) \\ \vec{y}(t) &= \mathbf{C}\vec{x}(t) + \mathbf{D}\vec{u}(t) \end{aligned} \quad (5.2)$$

(5.2) is the well-known state-space representation. In an electrical circuit, the state vector  $\vec{x}$  contains the currents through components and/or the voltages of the internal nodes.  $A$ ,  $B$ ,  $C$ , and  $D$  can be matrices, scalars, or even zero, depending on the dimensionality of the signals. For a time-invariant discrete system, (5.2) can be written as:

$$\begin{aligned} \vec{x}[k+1] &= \mathbf{A}\vec{x}[k] + \mathbf{B}\vec{u}[k] \\ \vec{y}[k] &= \mathbf{C}\vec{x}[k] + \mathbf{D}\vec{u}[k] \end{aligned} \quad (5.3)$$

Suppose there are  $p$  inputs,  $q$  outputs, and  $n$  states, the dimensionality of the variables are:

$$\begin{aligned} \vec{x} &\in \mathbb{R}^n & \mathbf{A} &\in \mathbb{R}^{n \times n} \\ \vec{u} &\in \mathbb{R}^p & \mathbf{B} &\in \mathbb{R}^{n \times p} \\ \vec{y} &\in \mathbb{R}^q & \mathbf{C} &\in \mathbb{R}^{q \times n} \\ & & \mathbf{D} &\in \mathbb{R}^{q \times p} \end{aligned} \quad (5.4)$$

The idea of the non-linear extension is to compute a so-called non-linear vector function  $\vec{g} = f(\vec{h}) : \mathbb{R}^l \rightarrow \mathbb{R}^m$  where  $\vec{h} \in \mathbb{R}^l$  is a linear combination of  $\vec{x}$ ,  $\vec{u}$ , and  $\vec{g}$ . Now the computation of the state  $\vec{x}$  is extended by one term which scales and adds  $\vec{g}$  to (5.2), i.e.:

$$\begin{aligned} \frac{d}{dt}\vec{x}(t) &= \mathbf{A}\vec{x}(t) + \mathbf{B}\vec{u}(t) + \mathbf{E}\vec{g}(t) \\ \vec{g}(t) &= f(\vec{h}(t)) \\ \vec{h}(t) &= \mathbf{K}\vec{x}(t) + \mathbf{L}\vec{u}(t) + \mathbf{M}\vec{g}(t) \\ \vec{y}(t) &= \mathbf{C}\vec{x}(t) + \mathbf{D}\vec{u}(t) + \mathbf{F}\vec{g}(t) \end{aligned} \quad (5.5)$$

and the dimensionality of each variable is:

$$\begin{aligned}
 \vec{g} &\in \mathbb{R}^m & \mathbf{E} &\in \mathbb{R}^{n \times m} & \mathbf{K} &\in \mathbb{R}^{n \times l} \\
 \vec{h} &\in \mathbb{R}^l & \mathbf{F} &\in \mathbb{R}^{q \times m} & \mathbf{L} &\in \mathbb{R}^{p \times l} \\
 & & & & \mathbf{M} &\in \mathbb{R}^{m \times l}
 \end{aligned} \tag{5.6}$$

The non-linear function  $f(\vec{h})$  is typically implemented as a look-up table. Figure 5.1 shows the corresponding discrete circuit to (5.5). This circuit can be used to model any kind of non-linearity and is also suitable for our signal-flow graph based model. However, since our model uses a low-level description, many variables in (5.5) are just scalars or zero.

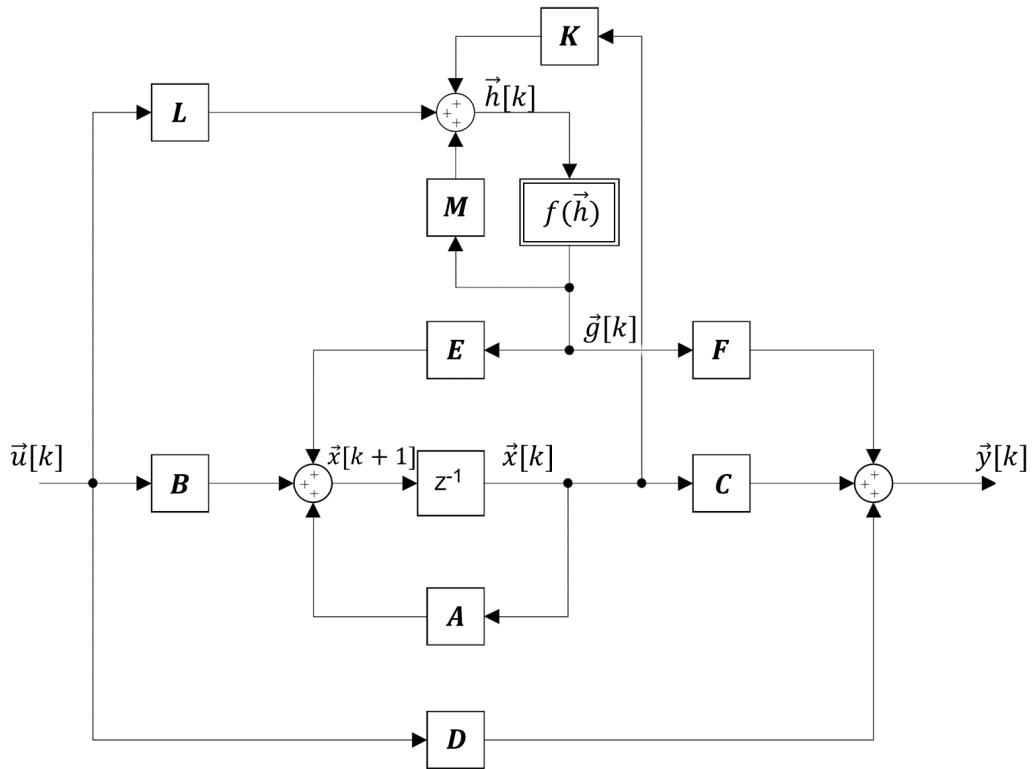
## 5.2 Application example based on temperature dependency

The use of the theory from section 5.1 will be demonstrated on a concrete example. The dependency on temperature is a good example since it is simple and generally applicable.

The change of a physical property is usually modelled with a linear relationship to the temperature change  $\Delta T$ . For example, the resistance a resistor  $R$  is given by

$$R(T) = R_0 \cdot (1 + \alpha \Delta T) \tag{5.7}$$

where  $R_0$  is the resistance at room temperature and  $\alpha$  is the temperature coefficient. It has the units  $K^{-1}$  and describes the relative change of a physical property that is associated with



**Figure 5.1:** Extended discrete-time state-space model of a dynamic system.

a given change in temperature. If a more precise model is required, then equation (5.7) can be extended to a second order polynomial or even higher.

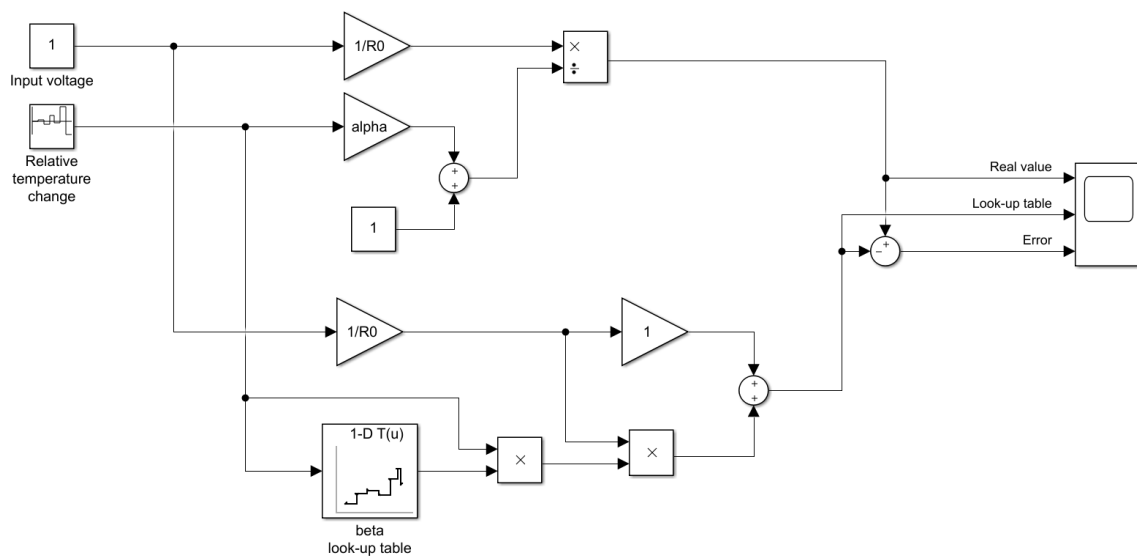
Although this approach could be directly implemented using the equations in (5.5), it cannot be used in our model since DPSFGs always deal with the conductance of resistors rather than the resistance. Inverting (5.7) yields a non-linear function that requires a division, which should be avoided when dealing with FPGAs. Thus, we want to find a new temperature coefficient  $\beta$  that describes a relationship between the conductance  $G$  and the temperature change  $\Delta T$ . Mathematically,

$$\begin{aligned} G_0 \cdot (1 + \beta \Delta T) &= \frac{1}{R_0 \cdot (1 + \alpha \Delta T)} \\ (1 + \beta \Delta T) &= \frac{1}{1 + \alpha \Delta T} \\ \beta &= \frac{-\alpha}{1 + \alpha \Delta T}. \end{aligned} \quad (5.8)$$

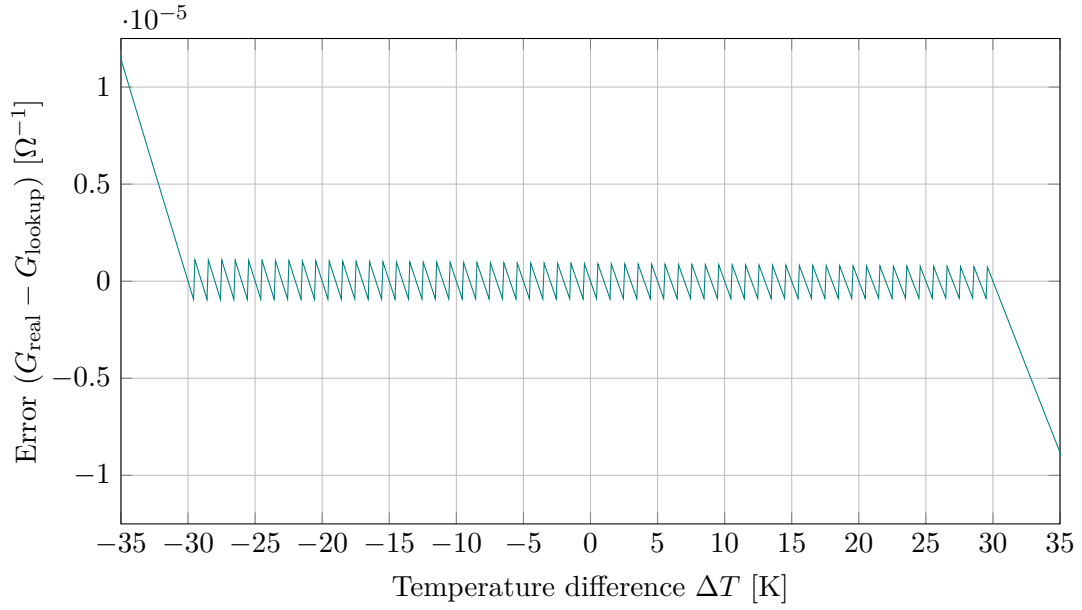
The problem here is that  $\beta$  is not a constant any more. Instead, it is a function of  $\Delta T$  in a non-linear relationship. The idea is to precompute the beta values and store them in a look-up table. For example,  $\beta$  could be discretized from  $-30$  K to  $30$  K with steps of  $1$  K. Since  $\beta$  is identical for all resistors of the same type, the same look-up table can be reused.

The current temperature would be measured by a sensor and stored inside a register. Afterwards, the relative temperature change is calculated and used as the input for the look-up table. Figure 5.2 illustrates this. This small test bench calculates the output current, given an input voltage and a temperature difference  $\Delta T$ . The upper path calculates the real value, i.e. the inversion of equation (5.7), and the lower path computes (5.8) with the help of a look-up table.

The setup was tested with  $R_0 = 1 \text{ k}\Omega$ ,  $\alpha = 0.002 \text{ K}^{-1}$ , and  $V_{\text{in}} = 1 \text{ V}$ . The look-up table contains 61 so-called breakpoint entries ranging from  $-30$  K to  $30$  K. There will be an error if the temperature difference exceeds the input space or when it lies between two breakpoints. In



**Figure 5.2:** Small Simulink test bench showing how the temperature dependency can be modelled with the help of a look-up table.



**Figure 5.3:** Error caused by the look-up table which implements the temperature dependency of resistors.

the latter case, the next closest breakpoint will be used. This leads to the characteristic curve in Figure 5.3. Notice that the error is not exactly symmetrical due to the non-linear relationship between conductance and temperature. How large the error becomes can be controlled by the range and resolution of the look-up table.

The disadvantage of this approach is the fact that two non-constant multiplications are required for each resistor. This can be optimized a bit by including the multiplication with  $\Delta T$  in the look-up table, i.e. the whole term  $(1 + \beta\Delta T)$  is stored in the tables rather than just the betas. This reduces the resources to one multiplication per component. The advantage of this method is its generality, meaning that it is generic and can be used to model any kind of temperature dependency in a system.

Referring back to the general model introduced in section 5.1, this non-linear effect can be implemented as follows. The temperature dependent resistor has two inputs: the voltage  $v_R$  and the temperature change  $\Delta T$ . Its output is the current  $i_R$ . Using the equations in (5.5) gives the following description:

$$\begin{array}{llll}
 \vec{u} = (v_R, \Delta T)^T & \mathbf{A} = 0 & \mathbf{D} = [G_0, 0] & \mathbf{K} = 0 \\
 \vec{x} = 0 & \mathbf{B} = 0 & \mathbf{E} = 0 & \mathbf{L} = 1 \\
 \vec{y} = i_R & \mathbf{C} = 0 & \mathbf{F} = G_0 & \mathbf{M} = 0 \\
 \vec{g} = v_R \beta \Delta T & \vec{h} = (v_R, \Delta T)^T & f(\vec{h}) = -v_R \alpha \Delta T / (1 + \alpha \Delta T) & 
 \end{array} \quad (5.9)$$

Notice that this assignment is not unique, i.e. there are different versions possible which yield the same result.

### 5.3 Slewing

Slewing is a characteristic of an op-amp that affects his behaviour when driven by fast changing signals. Op-amps are only capable of producing a certain steepness at their output which is

the well-known slew rate, given in voltes per seconds. One could believe that modelling this non-linear effect in our discrete implementation is simple. For example, the largest change in voltage that a op-amp can produce is given by

$$\Delta V_{\max} = T_s \cdot SR \quad (5.10)$$

where  $T_s$  is the sampling period and  $SR$  the slew rate. Now a digital logic checks whether the output exceeds this limit at every clock cycle. If yes, then the output is driven by another block which produces the voltage given by (5.10). Unfortunately, this approach is insufficient. The problem is that the settling behaviour after the slewing period changes too. Implementing in the way mentioned above leads to a perfect linear behaviour once the slewing condition is no longer true. Modelling this non-linear effect in a linear feedback loop correctly is quite difficult [23], [24].

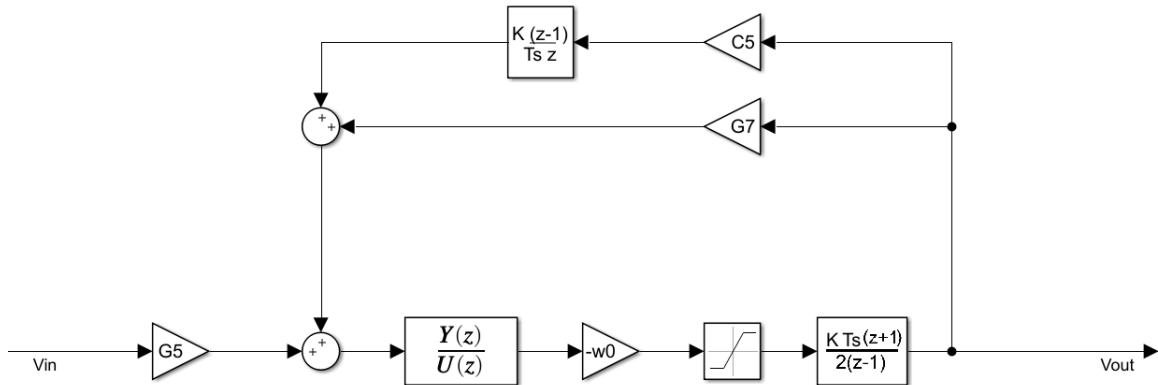
Fortunately, with our low-level modelling approach based on DPSFGs it is possible to implement this feature in its "true nature". Referring back to our model in Figure 3.1, the gm-block can produce a maximum current  $I_{\max}$ , yielding a maximum output voltage of

$$V_x(t) = \frac{I_{\max}}{C} \cdot t = SR \cdot t. \quad (5.11)$$

Inserting a saturation block that limits the output current to  $I_{\max}$  after the gm-stage would exactly produce this behaviour. Both,  $g_m$  and  $C$  are unknown. Only the slew rate  $SR$  and the GBW  $\omega_0$  are provided by the datasheet. Using (3.2) one can see that limiting  $\omega_0$  has the same effect as limiting  $I_{\max}$ . This is illustrated in Figure 5.4 which shows the discrete model of an op-amp.

The saturation limit can be found by operating the op-amp in a positive unity-gain configuration. The closed-loop transfer function is then given by

$$\frac{V_{\text{out}}}{V_{\text{in}}} = \frac{\omega_0/s}{1 + \omega_0/s} = \frac{\omega_0}{s + \omega_0}. \quad (5.12)$$



**Figure 5.4:** Discrete model of op-amp 8 in *Der Faux* including the feedback components and the saturation block to model slewing.



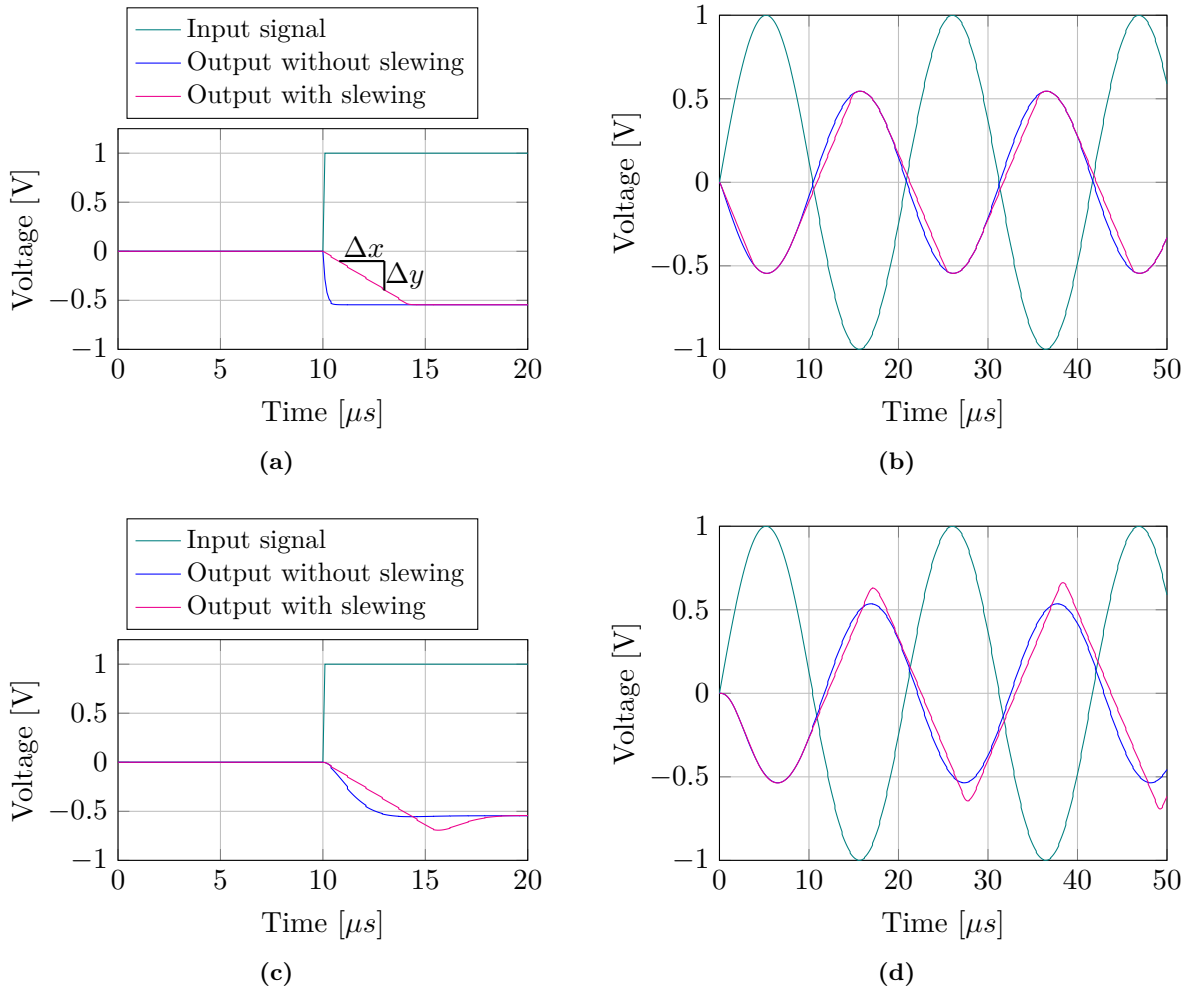
Now a step  $\varepsilon(t)$  with weight  $V_{\text{in}}$  is applied and the slope of the step response  $g(t)$  is evaluated at its steepest point, i.e. at  $t = 0$ .

$$\left. \frac{d}{dt}g(t) \right|_{t=0+} = \lim_{s \rightarrow \infty} s^2 G(s) = s^2 \cdot \frac{V_{\text{in}}}{s} \cdot \frac{\omega_0}{s + \omega_0} = V_{\text{in}} \omega_0 \quad (5.13)$$

Therefore, if  $V_{\text{in}} \omega_0 > SR$  then the op-amp is slewing. Our op-amp model starts with a multiplication by  $\omega_0$ . Hence, the limit for the saturation block is precisely equal to the slew-rate!

The verification is done by modelling op-amp 8 of *Der Faux* as in Figure 5.4. According to the TL072 datasheet [16], the parameters are:  $SR = 13 \text{ V}/\mu\text{s}$  and  $\omega_0 = 2\pi \cdot 3 \text{ MHz}$ . It should be noted that the  $SR$  of this op-amp is relatively large, i.e. the maximum frequency before slewing happens is  $SR/(A_{\text{max}} \cdot 2\pi) \approx 460 \text{ kHz}$ , which is far above audio frequencies. If, then slewing will only occur on internal op-amps e.g. due to higher order harmonic distortion. However, since the signal is low-pass filtered many times it is questionable if the slew rate model will affect the discrete implementation at all.

Nevertheless, Figure 5.5 shows the response of a unit step and sinusoidal input signal. For better visualization, the slew rate was reduced by a factor of 100. In Figure 5.5(a) and (b),



**Figure 5.5:** Discrete simulation of an op-amp with slewing model. (a) and (b) show the behaviour without capacitors in the feedback, (c) and (d) have capacitors included.

all capacitors were set to zero in order to create a first-order system with no overshoot. They show how the slope of the output is limited when the signal is changing too fast. Figure 5.5(c) illustrates the advantage of our DPSFG model. The settling behaviour is –as it would be in reality– affected by the slew rate, which is automatically captured by our model. Evaluating the steepness in the slewing period gives a value of  $\Delta y/\Delta x = 0.13 \text{ V}/\mu\text{s}$ , which is exactly the expected result.

## 5.4 Meaningful non-linear effects

So far, a general theory for modelling non-linear effects and two examples have been presented. Which (further) effects are included in the model of this work? The answer is none of them except saturation.

Saturation has not been discussed yet but it is a rather simple effect. Any active component cannot produce output voltages larger than the supply voltage. Thus, a saturation block at the output of every op-amp is included which limits the signal level to  $\pm 4.5 \text{ V}$  for op-amp 7 and op-amp 8, and  $\pm 2.5 \text{ V}$  for any internal operational amplifier.

The reason why other non-idealities are not included is because the characteristic sound of this circuit is mainly based on the delta modulator, whose functionality is imitated very precisely. Modelling other effects such as temperature dependence will not significantly improve the *authenticity*, but increase the resource requirements.

Besides the development of a functional use-case, the theoretical basis for the new low-level approach is important too. Upcoming projects can use this knowledge. As long as chosen model for an electrical component can reflect a non-linear effect, the theory from section 5.1 can be applied.

Slewing is a special case because the simple op-amp model used in this work can reflect this effect. To give an counterexample, in section 3.5.2 it was discussed that lowering the bandwidth can improve stability. A consequence of this is that the voltage variation of the virtual ground is larger, resulting in a higher non-linearity of the input stage. Our model cannot reflect this directly and the use of the extended state-space theory is necessary. Furthermore, there is a direct link between GBW and slewing as shown in section 5.3. Changing  $w_0$ , which is unknown for internal op-amps, will affect the slewing condition. Including slewing on internal op-amps is thus not meaningful and the external op-amps have such a high slew rate that slewing will only occur at frequencies far above audio frequencies. Hence, it is not included too.

As a consequence, applying simplifications to the circuit in order to relax stability constraints can be done without sacrificing important non-linear effects, which finally answers the open question from section 3.5.2.

## 6 Verification

Verification of sub-blocks and special circuits were mostly done on-the-fly in this report. Checking whether the final product fulfils the requirements is a bit difficult, though.

The problem is that neither *good behaviour* nor *authentic sound* can be defined in a technical sense, that is, something that can be measured and evaluated quantitatively. When designing audio equipment, one is typically interested in properties such as low noise and linear characteristics. Both of which are not true in this work. Test cases where those measurements are meaningful are not of interest.

The only meaningful verification is thus a blind test performed professionals. This task was left to the colleagues at the Academy of Music in Basel. Detailed results can be found in their work.<sup>5</sup> In this work, the outcome of their listening test is briefly presented.

In general, the FPGA based replica of *Der Faux* was described as "being much closer to the original in terms of sound pattern than any other preservation strategy such as purely software based simulations. But the appearance of more noise compared to the original hardware makes it clearly distinguishable."

Noise turned out to be the most disturbing effect in order to classify the FPGA as an authentic replica of the original. It was considered to be "small enough to be used in daily activities, but too large for blind tests by professionals."

---

<sup>5</sup>They are not published yet and thus not listed as a reference here.

## 7 Conclusion

On a theoretical basis, it can be shown easily that the low-level modelling technique based on signal-flow graph works. It was shown in chapter 3 how different building blocks can be converted to the z-domain using the bilinear transformation and how this affects the frequency response of filters. In chapter 4, it was shown that it is possible to implement adjustable blocks that can be tuned during runtime, providing an overall real-time simulation of a mixed-signal system using FPGA technology. Although the general procedure of the low-level approach presented in this report is straightforward, many challenges were faced when trying to implement such a model on hardware. This leads to several limitations that are now summarized.

### 7.1 Limitations of the graph-based low-level modelling

The first limitation is stability. Although a stable continuous-time model is always transformed into a stable discrete-time model by the Tustin approximation, the resulting system might still be unstable due to the necessity of breaking loops with unit delays. As it was shown in section 3.5.2 on a theoretical and empirical manner, breaking loops can be crucial in terms of stability. There are three ways to address this problem: Firstly, reduce the loop gain by lowering the GBW of op-amps. Secondly, perform massive oversampling to reduce the phase shift caused by the delay. Thirdly, eliminate loops in the DPSFG. The latter one is the most elegant one but it is not always possible. Furthermore, one sacrifices one of the most beneficial properties of the low-level approach: The ability to model non-idealities in their truest form. If a path in the SFG is collected to a single transfer function, then we arrive at the same position of previous work where system identification was done on functional blocks. In conclusion, one has to decide whether modelling non-linear effects precisely or maintaining stability is more important. In this work, it was shown in section 5.3 that non-ideal effect of op-amps have little impact on the system. Hence, high-gain loops were simplified where possible.

The next limitation is given by the available FPGA resources. Even a small circuit board like *Der Faux* produces a relatively large DPSFG. Processing signals with large dynamics requires a lot of logic elements (accumulators, multipliers, flop-flops, etc.). Together with all the look-up tables this leads to an enormous resource utilization which exceeds the capability of a Cyclon V FPGA without all the optimizations from section 4.5. Especially the fixed-point data type optimization was crucial in order to fit the design into the FPGA.

Another limitation is timing. Numerical accuracy and stability conditions both require oversampling as explained in section 3.4 and 3.5.2. The underlying structure of a SFG creates long logic paths which should not be broken. Both circumstances place high demands on timing. Constraints can be relaxed by the method shown in section 4.5.4, but there is an upper limit. As a result, the low-level approach only works for low-frequency systems. It is unlikely that HF applications can be simulated in real-time using SFGs and FPGA technology.

### 7.2 Strengths of the graph-based low-level modelling

Besides all the challenges that come with the graph-based discretization method, it also provides several strengths such as its generality and the ability to reflect non-ideal behaviour precisely.

By generality, it is meant that the workflow can be applied to any mixed-signal circuit. In other words, it is multi-domain capable. All that is required is a linear model describing the flow of information from component to component. Additionally, no knowledge about the circuitry is required to set up a signal-flow graph, i.e. it is not necessary to identify functional blocks (it is still helpful, though). Once the SFG is complete one can transform it into the z-domain. This

is an easy task since the underlying structure of the graph is preserved, i.e. no evaluation of Mason's gain rule. This then yields a hardware-independent discrete-time description which can be implemented in various ways, depending on what the engineer wants to use. Furthermore, the effort for setting up a DPSFG grows linearly with circuit complexity. Due to the fact that the discrete-time SFG only consists of fundamental building block, it is very automatic code generation friendly. A talented computer scientist might be able to fully automatize the process, given nothing more than a net list or schematic. This is, of course, no longer true once special components have to be modelled such as the delta modulator in this work.

Another advantage is the often emphasized modelling of non-linear effects. The general theory from [21] can be applied as shown in chapter 5. Moreover, one can implement non-ideal behaviour in its true nature if it is possible to reflect that effect on the model used inside the SFG. Slewing of op-amps is a good example as demonstrated in section 5.3. This is a clear improvement to previous work, but it is also limited by the resource utilization mentioned above. Non-linear functions are typically implemented using look-up tables since they occupy less resources than actual digital logic required to effectively compute those function. Once a non-linear effect becomes a multi-variant function, the look-up table solution reaches its limits. This was demonstrated in section 4.2.1 where the coefficients of a transfer functions were dependent on three independent variables. The only way to relax this problem is by reducing the resolution of the LUT which in turn lowers the precision of the model.

If the chosen linear model cannot be extended to handle a particular non-linear effect, then one can use a model on an ever lower abstraction level. For example, one could set up an op-amp on transistor level. This allows us to include non-linear effects more easily than using a simpler model together with the extended state-space model theory. This extension is very flexible in the sense that it is possible to insert a more detailed model into the SFG at any time without changing the rest of the graph.

### 7.3 Open research questions

A main complaint from the blind tests was that the FPGA emulation has too much noise compared to its original. The underlying reason for this noise has not been fully clarified yet. It could be shown that noise is only present when the system is actively processing audio samples, i.e. there is no independent noise source. A possible explanation could be numerical problems. However, many runs with Simulink's Fixpoint Tool with different optimization parameters could not resolve this problem.

Interestingly, the approach where the delay is achieved by forwarding samples in a register of fixed size with variable speed, i.e. the pulse method, suffers much more from this problem. This can be justified by the fact that delta modulation is performed with variable, but in general much lower sampling rates, causing more granular noise. On the other hand, the original hardware does this too and does not suffer from this problem. One could shown that noise is dramatically reduced if the sampling rate is artificially increased by making the length of the shift register much longer. Consequently, samples must be forwarded faster to obtain the same delay time, which is equivalent to having a higher sampling frequency.

The same observation could be made in software, i.e. not only the FPGA but also simulations in Simulink showed this behaviour. Thus, there is probably no error in the toolchain nor in the numerics. This leads to the assumption that the delta modulator in *Der Faux* might not use a clock that is 16 times slower than the VCO's clock as stated in section 2.2.3. Whether that is true and how samples are then passed to the delay line remains unknown.

At this point, one could ask if an artificially incremented sampling rate is the solution to provide an authentic replica. The answer is: maybe. The problems are the limitations listed in

section 4.2.2. Noise vanishes almost completely if the shift register is four times larger (= 4x higher sampling). With a base clock of 7.2 MHz, very low delay times can already be no longer represented since the resulting division factor is smaller than 2. Therefore, one must increase the base clock of the model or put the shift register into a higher clock domain.

Both concepts could not be implemented successfully at this stage of the project. The latter one introduces more noise due to the insertion of additional delay in the modulator path. This can be confirmed by listening tests. However, there might be a point at which very high sampling rates, which reduce noise, relaxes the overall noise figure of the system well enough. If this is possible has not been investigated yet.

Simply increasing the base clock is not sufficient unfortunately. According to formula (4.8), the maximum possible increment of the sampling rate inside the delta modulator is  $f_{\text{sim,max}} \cdot \tau_{\text{min}} / (N_{\text{clk,min}} \cdot 44'000) = 3.43$  where  $N_{\text{clk,min}} = 2$ ,  $\tau_{\text{min}} \approx 29.76 \text{ ms}$ , and  $f_{\text{sim,max}} \approx 48'000 \cdot 210$ . This number is somewhat close, but not entirely sufficient to suppress all noise. However, the main problem is a different one. Despite successful simulations in Simulink with higher sampling rates than 7.2 MHz, it was not possible to run this model on hardware although it passed the synthesis without errors. A long-term investigation of this issue could not identify the problem. Unlike the first open research question, this problem is probably linked to an error in the toolchain since it cannot be reproduced in Simulink.

## 7.4 Relevancy for other research fields

Last but not least, the relevancy of the outcome in this work shall be evaluated. In the field of simulating analog components in a fully digital environment, one often wants to copy the functional behaviour of a circuit rather than implementing a 1:1 replica. Imitating non-ideal effects is seldom required; on the contrary, they are unwanted. Music on the other hand is an exception. In this field, emulating non-ideal effects is exactly what is expected. For idealized devices, one can simply take digital filters, signal generators, or other software-based solutions. The relevancy outside the field of preserving music pieces is thus questionable.

**Declaration of authorship**

I hereby confirm that the report I am submitting is entirely my own original work except where otherwise indicated.

Student's name: \_\_\_\_\_

Place and date: \_\_\_\_\_

Student's signature: \_\_\_\_\_

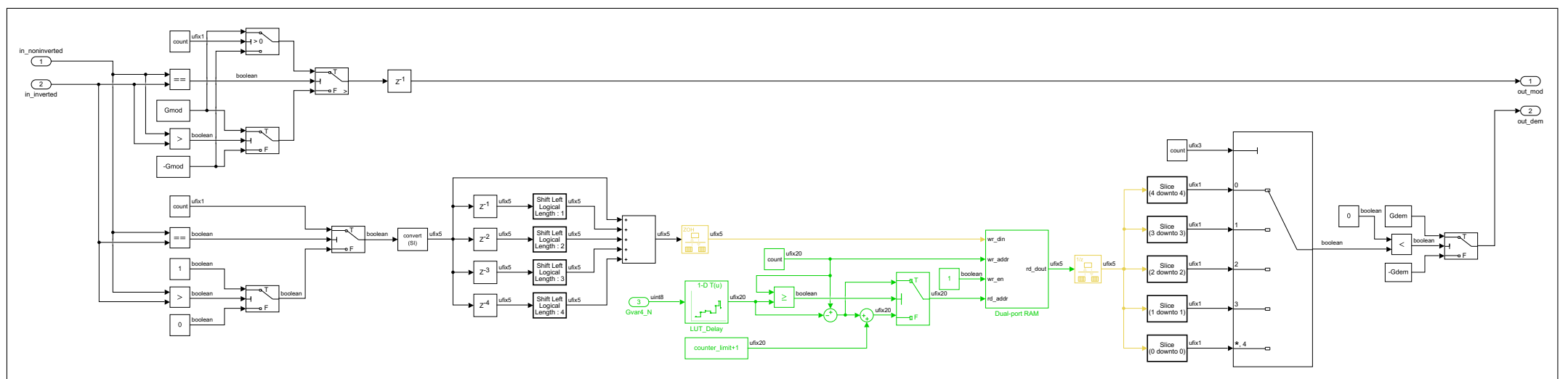
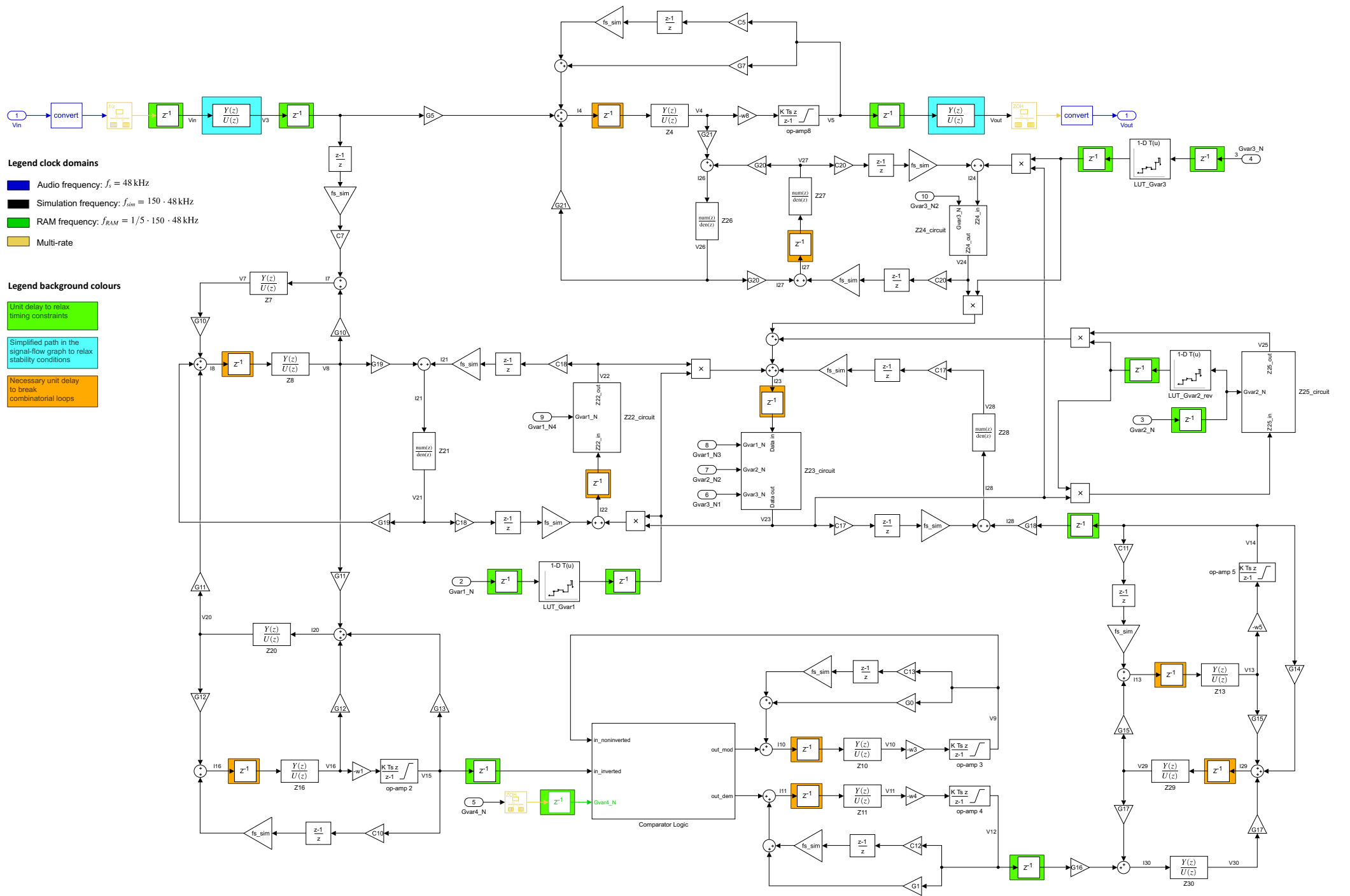
## References

- [1] S. Herbst, B. C. Lim, and M. Horowitz, „Fast FPGA Emulation of Analog Dynamics in Digitally-Driven Systems“, in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8.
- [2] P. Tertel and L. Hedrich, „Real-time emulation of block-based analog circuits on an FPGA“, in *2017 14th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2017, pp. 1–4.
- [3] R. Rabenstein, S. Petrausch, A. Sarti, G. Sanctis, C. Erkut, and M. Karjalainen, „Blocked-based physical modeling for digital sound synthesis“, *Signal Processing Magazine, IEEE*, vol. 24, pp. 42–54, Apr. 2007. DOI: 10.1109/MSP.2007.323263.
- [4] das musikding.de, *Der Faux - Delay Bausatz*. [Online]. Available: [https://www.musikding.de/3verb-reverb-hall\\_1](https://www.musikding.de/3verb-reverb-hall_1) (visited on Oct. 18, 2020).
- [5] Princeton Technology Corp., *PT2399 Echo Processor IC datasheet*, Version 1.6, Feb. 2010. [Online]. Available: [http://www.princeton.com.tw/Portals/0/Product/PT2399\\_1.pdf](http://www.princeton.com.tw/Portals/0/Product/PT2399_1.pdf) (visited on Jul. 7, 2020).
- [6] Terasic Inc., *DE1-SoC Board*. [Online]. Available: <https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=836> (visited on Oct. 18, 2020).
- [7] R. Steele, *Delta modulation systems*. New York: Wiley, 1975, ISBN: 0470821043.
- [8] Wikimedia Commons. (2011). „Principle of the delta Pulse Width Modulation (PWM)“. File: *Delta PWM.svg*, [Online]. Available: [https://en.wikipedia.org/wiki/File:Delta\\_PWM.svg#file](https://en.wikipedia.org/wiki/File:Delta_PWM.svg#file).
- [9] H. Schmid, „Circuit transposition using signal-flow graphs“, in *2002 IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No.02CH37353)*, vol. 2, 2002, pp. II–II.
- [10] H. Schmid and A. Huber, „Analysis of switched-capacitor circuits using driving-point signal-flow graphs“, in *Analog Integr Circ Sig Process*, vol. 96, 2018, pp. 495–507. DOI: <https://doi.org/10.1007/s10470-018-1131-7>.
- [11] H. Schmid, A. Huber, and L. Eichelberger, „A tutorial to switched-capacitor noise analysis by hand“, in *Analog Integr Circ Sig Process*, vol. 89, 2016, pp. 249–261. DOI: <https://doi.org/10.1007/s10470-016-0806-1>.
- [12] S. J. Mason, „Feedback Theory-Further Properties of Signal Flow Graphs“, *Proceedings of the IRE*, vol. 44, no. 7, pp. 920–926, 1956. DOI: <https://doi.org/10.1109/2Fjrproc.1956.275147>.
- [13] S. Näf and N. Wassermann, *SignalFlowGrapher*, GitHub repository, 2020. [Online]. Available: <https://github.com/hanspi42/signalflowgrapher>.



- [14] MathWorks, *Continuous-Discrete Conversion Methods*, Release 2020a. [Online]. Available: <https://mathworks.com/help/control/ug/continuous-discrete-conversion-methods.html> (visited on Oct. 27, 2020).
- [15] E. W. Weisstein, *Padé Approximant*, MathWorld—A Wolfram Web Resource. [Online]. Available: <https://mathworld.wolfram.com/PadeApproximant.html> (visited on Nov. 5, 2020).
- [16] Texas Instruments, *TL07xx Low-Noise JFET-Input Operational Amplifiers*, Jul. 2017. [Online]. Available: [https://www.ti.com/lit/ds/slos080n/slos080n.pdf?ts=1596101591961&ref\\_url=https%253A%252F%252Fwww.ti.com%252F](https://www.ti.com/lit/ds/slos080n/slos080n.pdf?ts=1596101591961&ref_url=https%253A%252F%252Fwww.ti.com%252F) (visited on Jul. 30, 2020).
- [17] MathWorks, *Rounding Modes for Fixed-Point Simulink Blocks*, Release 2020a. [Online]. Available: <https://mathworks.com/help/fixedpoint/ug/rounding-modes-for-fixed-point-simulink-blocks.html> (visited on Jan. 14, 2021).
- [18] intel, *Cyclone V Device Overview*. [Online]. Available: [https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-v/cv\\_51001.pdf](https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-v/cv_51001.pdf) (visited on Nov. 19, 2020).
- [19] R. M. Hewlitt and E. S. Swartzlantler, „Canonical signed digit representation for FIR digital filters“, in *2000 IEEE Workshop on SiGNAL PROCESSING SYSTEMS. SiPS 2000. Design and Implementation (Cat. No.00TH8528)*, 2000, pp. 416–426.
- [20] MathWorks, *Constant Multiplier Optimization to Reduce Area*, Release 2020a. [Online]. Available: <https://mathworks.com/help/hdlcoder/examples/constant-multiplier-optimization-to-reduce-area.html> (visited on Jul. 22, 2020).
- [21] D. T. Yeh, J. S. Abel, and J. O. Smith, „Automated Physical Modeling of Nonlinear Audio Circuits For Real-Time Audio Effects—Part I: Theoretical Development“, *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 18, no. 4, pp. 728–737, 2010.
- [22] D. T. Yeh, „Automated Physical Modeling of Nonlinear Audio Circuits for Real-Time Audio Effects—Part II: BJT and Vacuum Tube Examples“, *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 4, pp. 1207–1216, 2012.
- [23] C. T. Chuang, „Analysis of the settling behavior of an operational amplifier“, *IEEE Journal of Solid-State Circuits*, vol. 17, no. 1, pp. 74–80, 1982. DOI: <https://doi.org/10.1109/JSSC.1982.1051689>.
- [24] H. Zhang and G. Shi, „Symbolic behavioral modeling for slew and settling analysis of operational amplifiers“, in *2011 IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2011, pp. 1–4. DOI: 10.1109/MWSCAS.2011.6026581.

# Der Faux: Simulink Model



## Comparator Logic

