Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Leon Mayrhofer

Resource reduced ASIC CBADC digital estimation FIR filter

Master's thesis in Electronic Systems Design
Supervisor: Professor Trond Ytterdal
Co-supervisor: M.Sc. Fredrik Esp Feyling
June 2023

NTNU
Norwegian University of
Science and Technology

Leon Mayrhofer

Resource reduced ASIC CBADC digital estimation FIR filter

Master's thesis in Electronic Systems Design
Supervisor: Professor Trond Ytterdal
Co-supervisor: M.Sc. Fredrik Esp Feyling
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Preface

I would like to thank my supervisors Professor Trond Ytterdal and M.Sc. Fredrik Esp Feyling, who made it possible for me to work on this highly interesting topic. The support I got throughout the entire project phase and especially in all the meetings we had together was very motivating.

Furthermore I want to thank Dr. Hampus Malmberg, the original author of the project topic, who has shown interest in my results. Your work and the possibility to talk about it with you have been very inspiring.

Abstract

Control-bounded analog-to-digital conversion (CBADC) is a rather new concept, which features a different approach than most classic analog-to-digital converter (ADC) implementations. It implies less design restrictions on the single system components, enabling a greater design space exploration. The theoretical grounds for this concept have been set up and currently there are efforts made to create a physical implementation of it. There are two main parts, which have to be constructed. The analog system, which is filtering the analog input signal and the digital estimator, which is reconstructing a digitised approximation of the original signal.

While the analog system is the subject of other research work, this thesis helps exploring the design possibilities for the digital estimation. The design of a digital reconstruction filter with variable coefficients and the possibility to reduce the filter resource usage are presented. The system is based on a finite impulse response (FIR) filter using precomputed coefficients as data input for look-up-tables (LUTs) and adders to sum up the outputs of the LUTs to generate the final signal estimation. The size reduction is done by limiting the register sizes used for the precomputed coefficients and furthermore matching the LUT and adder data paths to the coefficient bitwidths.

In order to increase the usefulness and versatility of the proposed design and to simplify the entry into the creation of digital estimation filter implementations, a Python framework was introduced. This framework allows for automated execution of the individual design steps ranging from high level simulations to creating implementation layouts. It is closely connected to the cbadc Python package, introduced by the author of the CBADC theory. This package is used to simulate CBADCs in high level simulations and is one of the main sources of this thesis.

The resource usage (area and power) of the proposed implementations is shown and discussed. The possible savings due to reducing the design are explored. Furthermore trends for an optimal coefficient and LUT setup are established. Moreover the influence of the analog system design on the resource usage of the digital estimation filter is investigated.

This thesis shows that the design reduction has an overall positive effect on area usage and is useful for power reduction in some cases. Besides this thesis shows that in general smaller analog system configurations lead to a smaller and more efficient digital estimation filter. Unfortunately none of the variable coefficient implementations could reach the intended power target of 0,4mW for the system configuration with a bandwidth of 20MHz and a signal-to-noise ratio (SNR) of 70dB. The most efficient system had a total power consumption of 1,609mW. Due to these findings and since the design changes applied to the LUTs and adders can also be deployed in fixed coefficient configurations, some of them were included in this thesis. The most power efficient system of these configurations is able to reach the 0,4mW target with an estimated power consumption of only 0,29mW.

Sammendrag

Kontrollavgrenset analog-til-digital omforming (CBADC) er et nokså nytt konsept som nytter en annerledes fremgangsmåte enn hva de fleste klassiske analog-til-digital implementasjoner. Det innebærer mindre designbegrensninger på enkelt system komponenter, som videre tillater utforsking av et større designrom. Det teoretiske grunnlaget for dette konseptet har blitt lagt til rette og det blir for øyeblikket jobbet med å lage en fysisk implementasjon av det. Det er to hoveddeler som må konstrueres. Det analoge systemet som filtrerer det analoge inngangssignalet og den digitale estimatoren som rekonstruerer en digitalisert tilnærming av det originale signalet.

Denne avhandlingen utforsker designmuligheten for den digitale estimatoren, mens det analoge systemet er et tema for andre vitenskapelige arbeid. Designet av et digitalt rekonstruksjonsfilter med variable koeffisienter og muligheten for å redusere resursbruken vil bli presentert. Systemet er basert på et endelig impulsresponsfilter som nytter forhåndsberegnede koeffisienter som brukes som inngangs data på LUT-er og adderere for å summere utgangene til LUT-ene for å generere den endelige signalestimeringen. Størrelsesreduksjon blir gjort ved å begrense registerstørrelsene som blir brukt for de forhåndsberegnede koeffisientene, og videre ved å tilpasse LUT og adderer datastiene til bitbredden til koeffisientene.

Et Python-rammeverk har blitt introdusert for å øke nytten og allsidigheten til det foreslåtte designet, og for å forenkle oppstarten av å lage digitale estimeringsfilterimplementasjoner. Dette rammeverket tillater automatisert utførsel av de individuelle designstegene som rangerer fra høynivå simuleringer til å lage utlegg av implementasjonen. Den er sterkt knyttet til Python-pakken CBADC, som er introdusert av forfatteren av CBADC teori. Denne pakken blir brukt til å simulere CBADC-er i høynivå simuleringer og er en av hovedkildene til denne avhandlingen.

Ressursbruken (areal og effekt) til den foreslåtte implementasjonen vil bli vist og diskutert. Mulige besparelser med bakgrunn i å redusere designet vil bli utforsket. Videre skal skal det etableres trender for optimale koeffisient- og LUT-oppsett. Innflyteslen som det analoge systemdesignet har på ressursbruken til det digitale estimatorfilteret skal også undersøkes.

Denne avhandlingen vil vise at reduksjon av designet vil ha en alt i alt positiv effekt på arealforbruk og effekt i noen tilfeller. I tillegg skal denne avhandlingen vise at generelt mindre analoge systemkonfigurasjoner fører til mindre og mer effektive digitale estimatorfiltere. Dessverre kunne ikke noen av de variable koeffisient implementasjonene oppnå det tenkte effektmålet på 0,4mW for systemkonfigurasjonen med en båndbredde på 20MHz og et signal-til-støyforhold (SNR) på 70dB. Det mest effektive systemet hadde et total effektforbruk på 1,609mW. Med bakgrunn i disse funnene, og siden designendringene som ble gjort på LUT-ene og addererene kan bli distribuert i faste koeffisientkonfigurasjoner, ble noen av disse inkludert i avhandlingen. Systemet som er mest effektivt med hensyn på effekt av disse konfigurasjonene klarer å oppnå målet på 0,4mW med et estimert effektforbruk på 0,29mW.

Contents

Preface . . . . **i**
Abstract . . . . **ii**
Sammendrag . . . . **iv**
Contents . . . . **vi**
Figures . . . . **viii**
Tables . . . . **xi**
Acronyms . . . . **xiii**
1 Introduction **1**
1.1 Contributions of this thesis 3
1.2 Outline 4
2 Introduction to the CBADC concept **5**
3 Specialisation project outcome **11**
3.1 Digital estimator implementation 11
3.2 Python framework 13
3.3 Design performance results 14
4 Design changes for lower power **16**
4.1 Coefficient register 16
4.2 Look up tables 22
4.3 Adders 23
5 Python framework **25**
5.1 Design reduction implementation 25
5.1.1 Coefficient shift register size reduction 26
5.1.2 Look up table size reduction 27
5.1.3 Adder size reduction 28
5.1.4 Design considerations for the size reduced version 28
5.2 Workflow 29
5.3 Classes and configuration options 34
5.3.1 Classes 34
5.3.2 Configuration options 38
5.3.3 Workflow methods 40
6 Test environment and methodology **42**
6.1 Input signal 42
6.2 Analog system setup 43
6.3 Digital estimator configuration 43

6.3.1 Digital estimator base configuration 44
6.3.2 Design reduction stage estimation 45
6.3.3 Variable coefficient implementation sets 45
6.3.4 Fixed coefficient implementation sets 49
7 Results 52
7.1 RTL simulation 52
7.2 Synthesis & post synthesis simulations 53
7.2.1 Design reduction stage estimation 53
7.2.2 Synthesis program estimation 55
7.2.3 Variable coefficient implementations 58
7.2.4 Fixed coefficient implementations 60
7.3 P&R and post P&R simulations 63
7.3.1 Design reduction stage estimation 63
7.3.2 Post P&R synthesis program estimation 66
7.3.3 Variable coefficient implementations 68
7.3.4 Fixed coefficient implementations 71
8 Discussion and comparison 74
8.1 Comparison with RISC-V implementations 79
9 Future work 82
10 Conclusion 84
Bibliography 86
A Coefficient reconfigurability estimation 88
B Floorplan setup and layout 90
C Signal estimation and power spectral density (PSD) graph examples 95
D Post P&R area and power estimation values 102
D.1 Variable coefficient configurations 103
D.2 Fixed coefficient configurations 104

Figures

2.1 The control-bounded view on A/D conversion [1]. 5

3.1 Digital estimator with clock divider and accumulate register for downsampling [9]. 12

4.1 Lookback coefficients for N = 3, OSR = 23, bitwidth = 22, batch length = 128, LUT select input width = 2. 17
4.2 Lookback LUT bitwidths for N = 3, OSR = 23, bitwidth = 22, batch length = 128, LUT select input width = 2. 18
4.3 Lookback LUT bitwidths sorted for N = 3, OSR = 23, bitwidth = 22, batch length = 128, LUT select input width = 2. 21
4.4 Coefficient shift register module with reduced bitwidths. 21
4.5 LUT and adder modules with reduced bitwidths. 24

5.1 Visualised workflow of the digital estimator Python framework [10]. 32

7.1 Design reduction stage estimation, synthesis area. 54
7.2 Design reduction stage estimation, synthesis power. 54
7.3 Design reduction stage estimation, post synthesis PrimeTime power estimation. 55
7.4 Synopsys Design Compiler vs Cadence Genus area estimation for configurations 6.7. 56
7.5 Synopsys Design Compiler vs Cadence Genus power estimation for configurations 6.7. 56
7.6 Synopsys PrimeTime power estimation, with Synopsys Design Compiler and Cadence Genus synthesised designs for configurations 6.7. 57
7.7 Synopsys Design Compiler area estimation for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10. 58
7.8 Synopsys Design Compiler power estimation for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10. 59
7.9 Synopsys PrimeTime power estimation, with Synopsys Design Compiler synthesised designs for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10. 60
7.10 Synopsys Design Compiler area estimation for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14. 61

7.11 Synopsys Design Compiler power estimation for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14. 62
7.12 Synopsys PrimeTime power estimation, with Synopsys Design Compiler synthesised designs for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14. 62
7.13 Design reduction stage estimation, P&R area. 64
7.14 Design reduction stage estimation, P&R power. 65
7.15 Design reduction stage estimation, post P&R PrimeTime power estimation. 65
7.16 Synopsys Design Compiler vs Cadence Genus, P&R Cadence Innovus area estimation for configurations 6.7. 66
7.17 Synopsys Design Compiler vs Cadence Genus, P&R Cadence Innovus power estimation for configurations 6.7. 67
7.18 Synopsys PrimeTime power estimation, with Synopsys Design Compiler and Cadence Genus synthesised designs, P&R Cadence Innovus for configurations 6.7. 67
7.19 Cadence Innovus area estimation for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10 synthesised with Synopsys Design Compiler. 69
7.20 Cadence Innovus power estimation for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10 synthesised with Synopsys Design Compiler. 70
7.21 Synopsys PrimeTime power estimation for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10, synthesis with Synopsys Design Compiler, P&R with Cadence Innovus. 70
7.22 Cadence Innovus area estimation for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14 synthesised with Synopsys Design Compiler. 72
7.23 Cadence Innovus power estimation for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14 synthesised with Synopsys Design Compiler. 72
7.24 Synopsys PrimeTime power estimation for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14, synthesis with Synopsys Design Compiler, P&R with Cadence Innovus. 73

8.1 Area estimation comparison between RISC-V implementations from [4] and designs proposed in this thesis. 80
8.2 Power estimation comparison between RISC-V implementations from [4] and designs proposed in this thesis. 81

A.1 Comparison of the LUT data input bitwidths for different configurations variations. 89

B.1 Layout for filter with fixed coefficients, N = 3, K1/K2 = 128, LUT select input width = 4 bit, full size coefficients. 91

B.2 Zoomed in layout for filter with fixed coefficients, N = 3, K1/K2 = 128, LUT select input width = 4 bit, full size coefficients. Main and side cluster visible. 92
B.3 Zoomed in layout for filter with fixed coefficients, N = 3, K1/K2 = 128, LUT select input width = 4 bit, full size coefficients. only main cluster visible. 92
B.4 Layout for filter with variable coefficients, N = 3, K1/K2 = 128, LUT select input width = 4 bit, full size coefficients. 93
B.5 Zoomed in layout for filter with variable coefficients, N = 3, K1/K2 = 128, LUT select input width = 4 bit, full size coefficients.. 94

C.1 RTL simulation for the system with variable full size coefficients, N = 8, LUT input width = 2 bit. 96
C.2 cbadc Python package simulation PSD graph (orange) and RTL simulation PSD graph (blue) for the system with variable full size coefficients, N = 8, LUT input width = 2 bit. 96
C.3 Gate level simulation for the system with variable full size coefficients, N = 8, LUT input width = 2 bit. 97
C.4 Gate level simulation PSD graph for the system with variable full size coefficients, N = 8, LUT input width = 2 bit. 97
C.5 Post P&R simulation for the system with variable full size coefficients, N = 8, LUT input width = 2 bit. 98
C.6 Post P&R simulation PSD graph for the system with variable full size coefficients, N = 8, LUT input width = 2 bit. 98
C.7 RTL simulation for the system with variable reduced size coefficients, N = 8, LUT input width = 2 bit. 99
C.8 cbadc Python package simulation PSD graph (orange) and RTL simulation PSD graph (blue) for the system with variable full size coefficients, N = 8, LUT input width = 2 bit. 99
C.9 Gate level simulation for the system with variable reduced size coefficients, N = 8, LUT input width = 2 bit. 100
C.10 Gate level simulation PSD graph for the system with variable reduced size coefficients, N = 8, LUT input width = 2 bit. 100
C.11 Post P&R simulation for the system with variable reduced size coefficients, N = 8, LUT input width = 2 bit. 101
C.12 Post P&R simulation PSD graph for the system with variable reduced size coefficients, N = 8, LUT input width = 2 bit. 101

Tables

4.1 Lookback LUT 62 to 67 coefficient widths for N = 3, OSR = 23, bitwidth = 22, batch length = 128, LUT select input width = 2. . . 18
4.2 Lookback LUT 95 coefficient values for N = 3, OSR = 23, bitwidth = 22, batch length = 128, LUT select input width = 4. 20

5.1 Correlation between Python classes and SystemVerilog modules. . . 36
5.2 SystemVerilog assertion and testbench modules. 38
5.3 Configuration options for the Python framework. 39
5.4 DigitalEstimatorGenerator class methods corresponding to workflow stages (figure 5.1). 41

6.1 Parameters of the input signal. 42
6.2 Configuration of the analog system for which the digital estimation implementations should be designed for. 43
6.3 Digital estimator configurations matching the analog system specification, established in cooperation with Andreas Bjørsvik and Sevat Mestvedthagen. 44
6.4 Digital estimator operating frequencies. 45
6.5 Test setup for design reduction stage estimation. 45
6.6 Number of precomputed coefficients needed per digital control vector bit. 46
6.7 First configuration set for variable coefficient implementations. LUT select input width: 4; resource reduction: False 48
6.8 Second configuration set for variable coefficient implementations. LUT select input width: 4; resource reduction: True 48
6.9 Third configuration set for variable coefficient implementations. LUT select input width: 2; resource reduction: False 49
6.10 Fourth configuration set for variable coefficient implementations. LUT select input width: 2; resource reduction: True 49
6.11 First configuration set for fixed coefficient implementations. LUT select input width: 4; resource reduction: False 50
6.12 Second configuration set for fixed coefficient implementations. LUT select input width: 4; resource reduction: True 50

6.13 Third configuration set for fixed coefficient implementations. LUT select input width: 2; resource reduction: False 50
6.14 Fourth configuration set for fixed coefficient implementations. LUT select input width: 2; resource reduction: True 51

7.1 RTL SNR/ENOB values valid for all digital estimator test sets. 52
7.2 Gate level SNR/ENOB values for configurations 6.7 synthesised with Synopsys Design Compiler and Cadence Genus. 57
7.3 Gate level SNR/ENOB values for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10 synthesised with Synopsys Design Compiler. . 60
7.4 Gate level SNR/ENOB values for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14 synthesised with Synopsys Design Compiler. 63
7.5 Post P&R SNR/ENOB values for configurations 6.7 synthesised with Synopsys Design Compiler and Cadence Genus. 68
7.6 Post P&R SNR values for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10, synthesis with Synopsys Design Compiler, P&R with Cadence Innovus. 71
7.7 Post P&R SNR values for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14, synthesis with Synopsys Design Compiler, P&R with Cadence Innovus. 73

D.1 P&R area values for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10, synthesis with Synopsys Design Compiler, P&R with Cadence Innovus. 103
D.2 P&R PrimeTime power values for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10, synthesis with Synopsys Design Compiler, P&R with Cadence Innovus. 103
D.3 P&R area values for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14, synthesis with Synopsys Design Compiler, P&R with Cadence Innovus. 104
D.4 P&R PrimeTime power values for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14, synthesis with Synopsys Design Compiler, P&R with Cadence Innovus. 104

Acronyms

A/D analog-to-digital conversion. 82

ADC analog-to-digital converter. ii, xiii, 1, 2, 4–8, 43, 77, 82, 83, 85

AS analog system. ii, iii, xi, 1–9, 13, 14, 29, 30, 38, 39, 42–44, 77, 78, 82–84

ASIC application specific integrated circuit. 2, 81, 84

CBADC control-bounded analog-to-digital converter. ii, iv, vi, 1–10, 13, 25, 77, 79, 81, 82, 84, 85

DC digital control. 6, 7, 9, 30, 38, 39, 42

DE digital estimation. ii, iii, xi, 2, 5, 7–9, 11, 13, 14, 25, 26, 29, 30, 38, 39, 42–44, 77–79, 82, 83, 85

DSR down sample rate. 76

EDA electronic design automation. 31

ENOB effective number of bits. xii, 6, 7, 14, 43, 52, 53, 55, 57, 60, 63, 66, 68, 70, 73

FFT fast Fourier transform. 42, 43

FIR finite impulse response. ii, 3, 9, 11, 15, 22–25, 28, 37, 38, 42, 79, 84, 85

ISA instruction set architecture. 2

LSB least significant bit. 27

LUT look-up-table. ii, iii, viii–xii, 3, 12, 15–24, 26–28, 30, 34–37, 39, 45–51, 53–55, 58–61, 63, 64, 69–71, 73–77, 79, 80, 88–101, 103, 104

OSR oversampling rate. viii, xi, 17, 18, 20, 21, 43, 44, 77, 78, 88

P&R Place & Route. vii, ix, x, xii, 3, 34, 36, 38, 40, 41, 52, 63–68, 70, 71, 73, 75–77, 79, 82, 85, 90, 91, 95, 98, 101–104

PhD philosophiae doctor. 2, 9

PROM programmable read-only memory. 77

PSD power spectral density. vii, x, 33, 40, 41, 43, 95–101

RISC reduced instruction set computer. xiv

RISC-V Fifth iteration of a RISC architecture developed at Berkeley university. vii, ix, 2–4, 42, 44, 74, 79–81

RTL register transfer level. vii, x, xii, 3, 13, 14, 29, 31, 33, 36, 38, 40, 52, 74–76, 95, 96, 99

SNR signal-to-noise ratio. iii, v, xii, 2, 6, 7, 9, 14, 15, 29, 31, 33, 39–43, 52, 53, 55, 57, 60, 63, 66, 68, 70, 71, 73, 75–79, 82, 88, 95

SV SystemVerilog. xi, 3, 4, 13, 14, 20, 22, 25–31, 33–38, 40, 85, 95

Chapter 1

Introduction

Undeniably the world nowadays becomes more and more enriched with technology. When one thinks about personal electronic devices like smartphones, smartwatches, personal computers or the development of autonomous driving cars, high tech products are flooding the everyday live. All those systems have one thing in common, they need some sort of interface with the real world, in order to be able to capture incoming physical values, on which the next actions can be based and calculated. For example the heart rate and blood pressure sensors of a smartwatch or the collision detection sensors from autonomous cars. The big challenge is the process of converting the real world values into some form, which can be used by the electronic devices. Since post processing of captured data is normally always done digitally, these sensors are accompanied by an analog-to-digital converter (ADC), which samples and quantifies the analog values into a digital representation/approximation.

One of the major design tasks for such systems therefore is the design and implementation of accurate and efficient analog-to-digital converters. The basic working principle thereby is always the same. The amplitude of the incoming analog signal gets sampled at discrete points in time, leading to a quantification in time and amplitude. This quantified representation can always only be an approximation of the time and value continuous analog input signal. To achieve this, the ADCs use some sort of internal representation. The key point with existing ADCs is the strong coupling between the analog input signal and these intermediate representations within the ADC. The internal signals so far have been seen as direct representations of the input signal and therefore are strongly coupled to it. This implies restrictions for the design of the analog system and digital reconstruction circuit, since these components have to be closely matched. These limitations lead to the introduction of undesirable design aspects into the system, which are limiting the performance of the system.

Dr. Hampus Malmberg evaluated on the theoretical grounds of a new type of analog-to-digital converter, the so called control-bounded analog-to-digital converter (CBADC), in his PhD thesis [1][2]. The CBADC generalises this relation between the input signal and the intermediate signal. The internal signal is not seen as direct representation any more, instead it is considered to be a control signal for the analog system. This is allowing for more design freedom in the analog system and digital reconstruction filter. The structure of a CBADC resembles a well known type of ADCs, the $\Delta\Sigma$ converter. The $\Delta\Sigma$ converter samples the incoming analog input signal into an intermediate representation with typically lower bitwidth than the converter output. This bitwidth can go as low as a one single bit. The desired signal-to-noise ratio (SNR) is achieved with oversampling of the input signal. Within CBADCs a similar form of intermediate signals is created. These signals, from now on called digital control signals, are used to stabilise the by design unstable analog input system. Furthermore these signals hold the information, which are needed to reconstruct the original input signal and therefore are also passed along to the reconstruction filter.

M.Sc. Fredrik Esp Feyling is working on a physical implementation of the analog part for a CBADC [3]. In order to function as an ADC this analog system needs an implementation of a digital reconstruction filter. The step of estimating the input signal out of the digital control signals is from now on called digital estimation (DE) in this thesis. The generation of physical implementations of the digital estimation filter is the focusing point of this thesis.

There are several possible choices for an implementation of a digital estimation filter. First, a program running on an external computer could do the reconstruction. Second, a dedicated smaller processor core could be combined with the analog part of the CBADC. Third, an application specific integrated circuit (ASIC) can be used to produce the signal estimation. This thesis explores the design of the digital estimation filter as full ASIC in the style of a hardware accelerator. Andreas Bjørsvik and Sevat Mestvedthagen are focusing on the approach with a small coprocessor on the basis of the RISC-V instruction set architecture (ISA) [4]. Some results are compared later in this thesis.

The power consumption of an electric device dictates, in which environments it can be used. Devices with a fixed power supply or big batteries are normally allowed to have a higher power budget. But the percentage of those systems is decreasing more and more, since the world is shifting towards the utilisation of autonomous operating small devices. These devices only have a limited power budget available, due to the small batteries or energy harvesting techniques like solar cells, which are powering the circuit. This development leads to the point, where the power consumption of electric components like ADCs is the most critical design criteria.

1.1 Contributions of this thesis

M.Sc. David André Bjerkan Mikkelsen first touched on the implementation of a digital estimation filter for a CBADC in his specialisation project and master thesis [5][6][7][8]. His results showed that a finite impulse response (FIR) filter implementation proposes the best option in nearly all scenarios. Previously the FIR implementation was enhanced by the possibility of using interchangeable coefficients for the filter, the results can be seen in the preceding specialisation project report [9] and the code sources [10]. This thesis focuses on the reduction of used area and power both for the variable and fixed coefficient version of the filter. A low power consumption in combination with the interchangeable coefficients enables the filter implementation to be flexibly deployed in multi-sensor environments and allows for its post-production tuning. The reduction in size was achieved by limiting the bitwidth of the coefficient shift register, the look-up-tables (LUTs) and the adders at coefficient positions, where the full bitwidth is not used by the coefficient configuration. To circumvent local inhomogeneities and further reduce the design resource consumption, an algorithm to reorder the used LUTs and according coefficients was implemented.

Furthermore the also previously introduced Python working environment [10] for creating SystemVerilog implementations of digital estimation FIR filters was extended. Before it featured the creation of the SystemVerilog files based on the cbadc Python package [11] simulations. Furthermore a register transfer level (RTL) simulation was carried out on the generated SystemVerilog code, whose results are checked against the golden sample solution from the cbadc high level simulation. The new functionality now includes the usage of a second simulator to double verify the RTL simulation. Next two different synthesis programs can be employed to generate gate level netlists. From these netlists again simulations can be carried out, to verify, if the synthesis results match the given performance criteria. A power estimation, which factors in the signal activity from the simulations, can be done, in order to get an accurate metric for the anticipated power consumption. After this, the Place & Route (P&R) step can be carried out on the synthesised design. On this placed and routed design a simulation can be carried out, to record the activity of the internal gates. The last workflow step is to carry out a power estimation on the placed and routed design, featuring the recorded activity factors.

This framework was then used to evaluate the performance of different filter configurations for a specific analog system setup. These results were compared against the results from the RISC-V implementation [4], with regards to needed area and estimated power consumption.

1.2 Outline

The chapters of this thesis are structured as following. First an introduction to the concept of the control-bounded analog-to-digital converter (CBADC) is given in chapter 2, in order to convey a basic understanding of both the analog and digital part of the design. Next in chapter 3 the proposed filter design and the corresponding simulation results from the specialisation project report [9] are presented, to explain the foundation for this thesis. This thesis continues with the description of the design changes made to reduce the power consumption of the digital estimation filter in chapter 4. The next chapter 5 goes over the full functionality of the Python framework, made to automate the workflow of creating SystemVerilog implementations of the filter. The analog system, which is used to evaluate the performance of the filter design and the test setup for the filter configurations are explained in chapter 6. The results from these tests are shown in chapter 7. Next a discussion of the results and a comparison with the results from the RISC-V implementation [4] is given in chapter 8. A short prospect into possible further work is made in chapter 9. In chapter 10 this thesis is summarised by a conclusion on the status of the filter implementation, the Python framework and the possible design limitations shown by the results.

Chapter 2

Introduction to the CBADC concept

This chapter is supposed to give an overview over the concept of the control-bounded analog-to-digital converter (CBADC). The system is fully explained including the analog part of the design, since this understanding is necessary to understand the digital estimation (DE) step. The sources for this are the PhD thesis of Dr. Hampus Malmberg [1][2], the master thesis of M.Sc. Fredrik Esp Feyling [3], the specialisation project report [9] preceding this thesis, the specialisation project report [6] and master thesis by M.Sc. David André Bjerkan Mikkelsen [5].

The block diagram in figure 2.1 shows the general structure of an control-bounded analog-to-digital converter. As mentioned in the introduction, when compared to the block diagram of a $\Delta\Sigma$ converter there is not much difference in the system setup. When following the signal path from the signal input $u(t)$ to the estimated signal output $\hat{u}(t)$, there are the following system components.

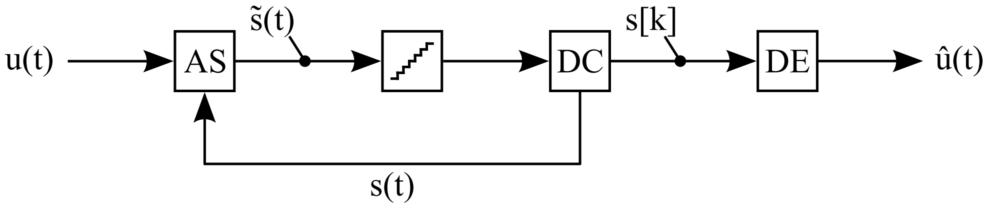

Figure 2.1: The control-bounded view on A/D conversion [1].

First comes the analog system (AS). Its task is to filter the incoming signal $u(t)$ and amplify the remaining wanted frequency range. This procedure can be seen as a bandpass, which suppresses unwanted signal components, therefore only letting the specified signal range pass and a following amplifier. This system normally consists out of several individual components (like filters, integrators, oscillators, etc.), which operate sequentially in a chain layout. Since all components operate independently of one another, each one has its own distinct state of operation. These states are called analog states from now on. The number of components present in the system therefore defines the number N of analog states. An important design aspect of the analog system is the fact that the system might be unstable without a feedback control signal. This allows for a more open design of the components, but also makes a stabilising system component for overall stable operation necessary. Hence the analog system has a second vectorised input for the control signals $s(t)$. The output of the analog system $\tilde{s}(t)$ is a vector consisting of all current analog states of the system components.

The signal $\tilde{s}(t)$ is passed to a quantifier circuit, where a time and value discrete version of it is created. This is necessary, since the further processing of the signal is done digitally. In real implementations there must be a separate quantifier for each individual analog state signal. The operating frequency of the quantifier most likely does not equal the intended Nyquist frequency of the analog-to-digital converter, but instead an integer multiple of this frequency, since the design uses oversampling of the analog states signal $\tilde{s}(t)$ to reach the intended accuracy. The targeted signal-to-noise ratio (SNR) respectively the effective number of bits (ENOB) of the system in combination with the number N of analog states determines the needed oversampling rate of the CBADC.

The quantified analog states vector $\tilde{s}(t)$ is then given to the system component called the digital control (DC) unit. This circuit observes the state changes in the analog system and identifies possible situations, which would destabilise it. Based on this observations the digital control unit is the part of the design responsible for creating the control signal vector $s(t)$, which is keeping the analog system in a stable state. Since the analog system components are independent of one another, the digital control unit has to consist of separated control units internally for each analog state, which all must output individual control signals for each analog component, hence the signal $s(t)$ is a vector. From now on the combination of the analog system, the quantifier and the digital control unit are referred to as the analog part of the CBADC.

One important detail of the control signal vector $s(t)$ is its ambiguity, when it comes to its characterisation as analog or digital signal. It is or better can be seen as both at the same time. Even though the digital control unit is a digital circuit and its output is a digital one bit signal for each control signal, the digital control vector appears as a time and value continuous analog signal for the analog system. The version of the control signal vector $s(t)$ considered digital is then written as $s[k]$, which is from now on called the digital states vector, with k being a discrete timestep. The number M of digital states is the number of bits in the digital states vector $s[k]$. It is normally identical to the number N of analog states, since as previously mentioned each analog system component needs individual stabilisation.

The digital state signal $s[k]$ is the point in the design where the CBADC differs from previously described analog-to-digital converter concepts like the $\Delta\Sigma$ converter. In the existing concepts this signal is always seen as a direct representation of the input signal $u(t)$. This view on the signal closely links the design of the analog and digital system together. Due to this close link between the system components, certain restrictions on both parts of the design are imposed. The concept of the CBADC does not see the digital state vector $s[k]$ as direct representation of the input signal $u(t)$. Instead it is seen as a system control signal, from which information about the input signal $u(t)$ can be extracted in order to calculate an approximation $\hat{u}(t)$ of the original signal. This allows for mostly independent design of the analog and digital part of the CBADC. Furthermore this makes it possible to tune the digital reconstruction filter to match the analog system after production. This can be used to counteract production variations and possible mismatch of the systems, which would normally lead to a decrease in the system SNR/ENOB.

The last part of the CBADC is the digital estimation (DE). It is responsible for carrying out the reconstruction of the input signal $u(t)$ out of digital state vectors $s[k]$ into the approximation $\hat{u}(t)$. It is important to note at this point that a single digital state is not sufficient for computing an appropriately accurate estimation $\hat{u}(t)$. It always needs a certain size of batch of digital states to reach a specified SNR. The batch consists of samples lying in the future and in the past seen from the time point of the estimation step. This leads to a certain startup time for the digital estimation and a delay between signal input and estimation output.

Dr. Hampus Malmberg has proposed general equations for the digital estimation, which form a non-linear equation system. This is due to the allowance of non uniform timesteps between sampling the analog states. When using a fixed sample time for the quantifier, the digital control and the digital estimation, these equations simplify into a linear equation system, which can be described with a few mathematical expressions. This linear digital estimation can be described in three steps, which are implemented by the following equations.

$$\overrightarrow{m}_{k+1} \triangleq A_f \overrightarrow{m}_k + B_f s[k] \tag{2.1}$$

$$\overleftarrow{m}_{k-1} \triangleq A_b \overleftarrow{m}_k + B_b s[k-1] \tag{2.2}$$

$$\hat{u}(t_k) \triangleq W^T \left(\overleftarrow{m}_k - \overrightarrow{m}_k\right) \tag{2.3}$$

Equations 2.1 and 2.2 are recursive equations, which are composed of a weighted digital state and the weighted last recursion step. Equation 2.3 sums up the recursions, while also weighting the result. The input variable k in the equations defines the discrete timestep, the approximation should be computed for. A_f and A_b are matrices defining the contribution of the last recursion steps, B_f and B_b define the contribution of the digital states for the recursion steps and W^T weights the contributions of the two recursion to the output value. In general all equations are in the complex number space, since all the matrices are complex number matrices. But in equation 2.3 all complex signal parts are eliminated, leaving a entirely real value as result. The values of the matrices correspond to the chosen analog system and digital estimation configuration. The theory for computing these matrices is described in the doctor thesis of Dr. Hampus Malmberg [1], but since details on this theory are not necessary to understand the digital estimation, its explanation is skipped in this thesis. Furthermore the cbadc Python package by Dr. Hampus Malmberg [11] provides methods for creating the matrices for simulation.

When looking at the first recursion equation 2.1, it becomes clear that it operates in the forward time direction going from older digital state samples to the timepoint, for which the approximation should be computed. Since all digital states taken into account lie in the past, this operation is from now on called the lookback recursion/calculation. When thinking about the startup of the CBADC this equation leads to the introduction of a certain startup time, which is necessary to accumulate the needed past digital state values. As long as there is not a sufficient amount of digital state vectors $s[k]$ present, the output of the approximation will have non valid values.

The second recursion equation 2.2 is quite similar to the first one, but instead of operating on digital states in the past of the approximation, it only uses samples lying in the future of the sampling time. Also the timely direction of the recursion is reversed, it operates from a future timepoint towards the sampling timestep. Since only future digital states values are used in this equation, it is be called the lookahead recursion/calculation from now on in this thesis. In a real implementation the circuit of course can not look into the future. Instead this leads to a delay between a sampling timestep and the corresponding approximation output of the CBADC. When this behaviour is compared to the one of more classical implementations like a Flash analog-to-digital converter, it becomes clear that a CBADC has a significantly higher output delay, assuming the operation at the same sampling frequency.

In theory the lookback and lookahead recursions can become infinitely large. In real implementations the laws of physics of course limit the amount of data, which an implementation can store for both the lookback and lookahead recursions. Therefore windows of digital state vectors $s[k]$ with limited sizes are used. The window sizes are called lookback length and lookahead length from now on in this thesis. Typically the digital estimation unit should be designed in a way that its accuracy performance is not lower than the possible performance of the analog part of the design. At the same time making the lookback/lookahead window sizes larger than they have to be, would waste resources, since no additional accuracy is gained. Therefore the analog system setup, the oversampling rate of the digital control unit and the targeted SNR of the approximation determine a metric for the optimal window sizes of digital state vectors $s[k]$ for the digital estimation. With this metric now also the necessary startup time can be estimated. In real implementations the rule of thumb should be that the lookback and lookahead windows have to be completely populated by valid samples, before a correct output can be produced.

When taking a closer look at the dot multiplications $B_f s[k]$ and $B_b s[k-1]$ from equations 2.1 and 2.2, a simplification can be assumed. Through the single bit per digital state nature of the digital states vector the factors of B_f and B_b actually do not need to be multiplied. The positive or negative contribution can just be added to the recursion step result. It is important to note that a zero bit in $s[k]$ is mathematically interpreted as -1 in this context. This simplification plays an important role in the design of the digital estimation unit.

In general for all possible implementations of the digital estimation Dr. Malmberg proposed two ways how it can be implemented in his PhD thesis [1]. The so called offline and online version. The offline version takes in a single fixed batch of digital state vectors and produces an output for a single timestep. Therefore this implementation is not suited for deployment in an integrated circuit. The online version takes in a continuous stream of digital state vectors and shifts the remaining previous vectors accordingly. With this implementation a continuous output stream of signal approximations can be produces. Therefore the proposed design of this thesis is implemented as online version of the digital estimation.

FIR filter digital estimation

Now the principle of the FIR filter implementation of a digital estimator is explained in further detail. Since this implementation assumes a constant sampling time and a finite lookback and lookahead length, the original formulas 2.1, 2.2 and 2.3 can be rewritten as equations 2.4, 2.5 and 2.6.

$$\hat{u}[k]_f = \sum_{l1=1}^{K1} \overrightarrow{h}_{l1}\, s[k-l1] \tag{2.4}$$

$$\hat{u}[k]_b = \sum_{l2=0}^{K2} \overleftarrow{h}_{l2}\, s[k+l2] \tag{2.5}$$

$$\hat{u}[k] = \hat{u}[k]_b + \hat{u}[k]_f \tag{2.6}$$

The most beneficial change to the equations due to these assumptions is the fact that the contribution matrices A_f and A_b for the previous recursion steps and the weighting matrix W^T can be combined with the weighting matrices B_f and B_b for the digital state vectors. The resulting matrices are $\overleftarrow{h}$ for the lookback calculation and $\overrightarrow{h}$ for the lookahead calculation. This effectively eliminates the recursion in the calculations. Additionally all values of the new matrices are fully real. This further reduces the computational effort, since no separate calculation for the real and imaginary part have to be carried out. Furthermore the equation 2.3 for combining the lookback and lookahead results can be reduced to a single addition, as it can be seen in equation 2.6.

Chapter 3

Specialisation project outcome

This chapter revisits the proposed design, the introduced Python framework and the design performance results from the preceding specialisation project [9] to give an overview over the foundation of this work. This thesis later refers to this design, when introducing the measures to lower the power consumption and discussing the extensions made to the Python framework.

3.1 Digital estimator implementation

M.Sc. David André Bjerkan Mikkelsen was the first one to work on an implementation of a digital estimator in his specialisation project [6] and master thesis [5]. His results show that a finite impulse response (FIR) filter implementation of the digital estimation with fixed point coefficients is the most power and area efficient one in almost all scenarios. Therefore the work in the specialisation project was concentrated on the FIR filter implementation.

The design used fixed hardwired coefficients so far, so it couldn't be tuned after production. This limits the usability of the design. Therefore the effort was made to include the possibility of interchanging the coefficients after production. The proposed solution to this problem is depicted in figure 3.1. This design can basically be split into an input part, a computation part and an output part.

On the input there is the downsample accumulate shift register, the coefficient shift register, the digital state vector shift register and the clock divider module. The coefficients are handed sequentially to the coefficient register, this needs to be done after startup, since then all registers will be uninitialised. The digital state vectors come in with the oversampling frequency, but it is not necessary to operate the digital estimator apart from the input handling at such a high frequency. The downsample accumulate shift register takes in digital state vectors at the oversampling frequency. Since the calculation of the estimation is run at a lower frequency (mostly the analog bandwidth frequency), this rather small

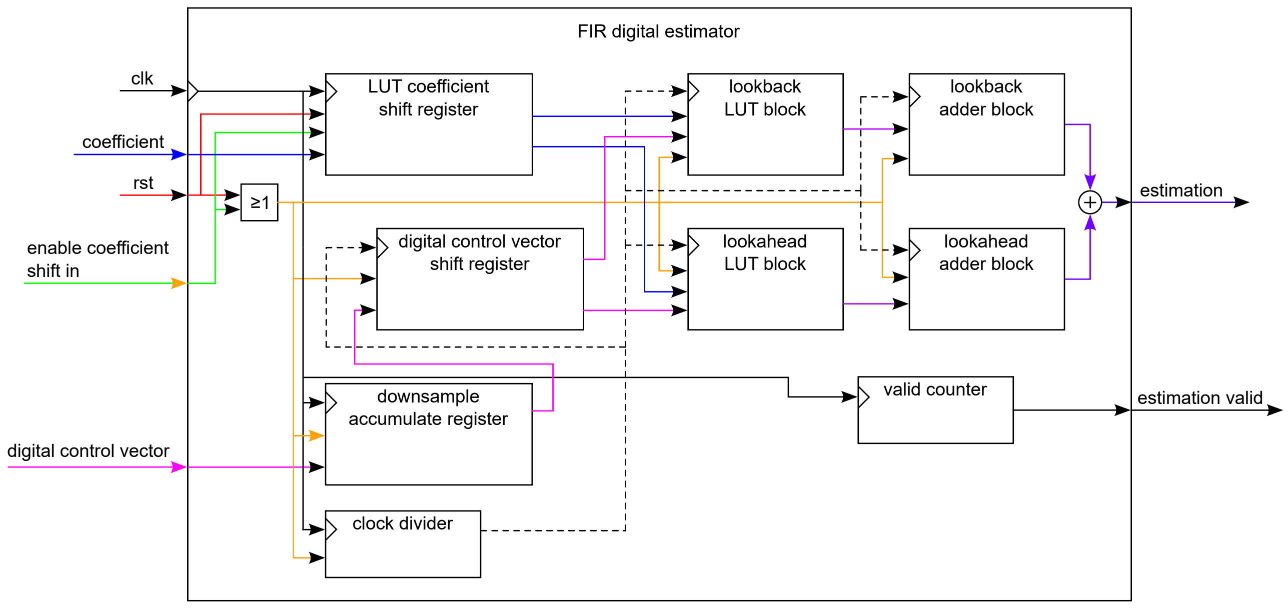

Figure 3.1: Digital estimator with clock divider and accumulate register for down-sampling [9].

register (the number of entries corresponds to the downsample rate) was introduced to not loose incoming digital state vectors and therefore precision. Power wise it is also better to operate this smaller register at a higher frequency then the overall digital state shift register. At every downsampled clock edge the entire accumulate register gets transferred over to the actual digital state register. The digital state vector shift register then holds all incoming digital states as long as they are relevant for the batch calculations. Furthermore this module distributes the digital state vectors to the according LUTs. The internal clock divider is used to reduce the clock frequency from the oversampled frequency back down to the analog bandwidth frequency. Hence only one analog circuit for creating a high frequency clock is needed in the design.

The computational part of the design consists of parallel look-up-table (LUT)s, which are used to select the correct precomputed coefficients, according to the input from the digital state vectors. The outputs of the LUTs are distributed to adders, which each add two LUT outputs together. This one stage of adders is therefore not enough to calculate the result to one single value. Several stages are needed, which each take in the results of the previous stage as input, until a single result is achieved. The needed number of stages rises with the logarithm to the base of two. When operated as synchronous adders, these stages add another delay to the estimation output. The overall calculation is split into the lookback and the lookahead calculation, leading to two very similar looking building blocks. The results of these blocks are then summed up by an additional adder.

The output part of the design basically consists of the final output for one estimation value. Besides that the small valid counter module was implemented, which gives out an indication signal, when valid estimation values are produced after startup. Like explained in the CBADC theory section, this module waits until the digital state vector register has been filled up completely and then issues the signal. A change of the filter coefficients also leads to a reset of the valid counter signal, since during this time no meaningful output value can be produced.

3.2 Python framework

The Python framework introduced in the specialisation project [9] is closely linked to the cbadc Python package by Dr. Malmberg [11]. Whereas the cbadc package focuses on the high level calculation of the analog system and matching DE and filter coefficients, the proposed framework continues with this data and creates a SystemVerilog implementation of the corresponding filter. The first major purpose of this framework is the automation of the design flow. It should be possible to easily specify several parametrisations of CBADCs, input them to the framework and it then automates the design and test workflow. This leads to a great productivity uplift, since many configurations can be specified and tested automatically. The second advantage of such a high level framework is the reduced required digital design knowledge of the engineer using the framework. This enables persons not coming from digital circuit design to create a digital estimator implementations without the need of looking into SystemVerilog code. The framework also checks some performance metrics of the design, which can be used to evaluate on the performance of the looked at filter parametrisation.

The status of the framework at the end of the specialisation project was that the design steps until the RTL simulation, including performance checks on the results, could be carried out. The design workflow is as following. First a configuration for the analog system (number of analog states, bandwidth, etc.) and the digital estimation has to be given to the framework. With this parameters the analog system will be calculated and simulated. Then the digital estimator will be generated and the output from the analog system will be given to it as simulation input. This leads to the high level simulation results, which are used by the framework as golden sample data. Then the framework starts generating the SystemVerilog code, which incorporates the results of the digital estimator generation. When the SystemVerilog code is ready, the RTL design gets simulated. The simulation results are then compared to the preciously stored golden sample high level simulation data.

The framework has a class called DigitalEstimatorGenerator. This is the main class of the framework and the one a user of the framework should mainly interact with, when wanting to create filter implementations. It has class variables for all relevant configuration options of the analog system and the digital estimation. These are intended as an input for the wanted system specification to the framework. All subsequent method calls use these parameters to customise this specific filter object. The class has methods to start the different steps of implementing the design, like simulating the analog system, generating the SV files, etc.

For the code generation of the separate modules the framework has an abstract generalised class to hold SV modules, it is called SystemVerilogModule. It brings the basic functionality to name and store the file, combined in the so called generate method. When a specific module should be implemented a child class of this base class is created. This child class then has the module specific configuration variables available over which the parametrisation is done. Furthermore it holds a template for the SV module code, which is completed according to the configuration options, when the generate method is called.

When looking at the generated SystemVerilog modules and files, the toplevel module is called DigitalEstimator. It binds together all design files and modules necessary to construct the digital estimator and establishes the connections between these modules. Another important module is the testbench used to conduct the RTL simulation, it is called DigitalEstimatorTestbench. The simulation so far was done with the Xcelium simulator from Cadence Design Systems.

It is recommended to use the DigitalEstimatorGenerator class in conjunction with the pytest environment, as it was done for the creation of the design test results in the specialisation project report. This allows the easy creation of test vectors, which are then all automatically run by the pytest program. This brings great benefits in productivity and usability of the framework.

3.3 Design performance results

This section shortly goes over the most important results gathered from the specialisation project simulations and synthesis runs. These results gave valuable insight into the possible configuration limits and preferable system configurations of the digital estimation.

A sweep over different lookback/lookahead lengths was carried out for systems with a target ENOB of 10, 12 and 15, while the coefficient data width was fixed to 64 bit. This ensures that the bitwidth is not the limiting factor for the filter accuracy. The result was that the needed length is dependant on the number of analog states and the oversampling rate in combination with the targeted SNR of the system. When looking at analog systems with the same bandwidth and wanted SNR, there is a trend visible that systems with a higher number of analog states need an increasing lookback/lookahead length. These systems then have a much larger amount of needed coefficients, LUTs and adders, due to the larger number of bits in digital states in combination with the larger batch size, resulting in a much higher needed circuit area. When looking at the power consumption, even the advantage of a lower oversampling frequency, resulting in a lower digital clock frequency for the FIR filter cannot equalise this overhead. Therefore systems with a higher number of analog states have a higher area usage and tend to have a higher power consumption.

Then sweeps over the bitwidth of the coefficients were carried out. This was done in order to determine, what the minimum needed data width is, to get the full filter resolution. The results suggest that the needed bitwidth is only dependant on the targeted SNR of the system. Thereby the needed bitwidth was around 10 bit higher then the targeted SNR of the system, with the biggest coefficient being 3 to 4 bit smaller then the specified bitwidth. This also means that systems with a higher number of analog states cannot gain an advantage by reducing the system bitwidth.

Next synthesis runs for the same system with different LUT input widths were carried out. By reducing the number of input bits per LUT, the total number of coefficients goes down, but at the same time, the number of needed LUTs and therefore adders goes up. The results show that an optimum can be expected between 1 and 4 input bits per LUT. Beyond 4 bits per LUT as it was to be expected the estimated area and power both rise quadratic with an increasing number of input bits.

Chapter 4

Design changes for lower power

In this chapter the design changes, which were implemented in order to reduce the area usage and power consumption of the system, are presented. Since some of the changes can also be applied to a fixed coefficient version of the filter, the possibility of creating fixed coefficient filters was added to the Python framework. These systems are also evaluated and compared with their variable coefficient counterparts. The optimisation applied to the design can be split into modifications of the coefficient shift register, the look-up-tables and the adders.

4.1 Coefficient register

The results from the specialisation project show two interesting facts about the distribution of the coefficients and the corresponding bitwidths over the lookback/lookahead batch. The more time steps a digital state vector is away from the sampling timepoint, the lower the needed amount of bits for its corresponding coefficient becomes. In figure 4.1 such a distribution can be seen. The second point is that even the most influential coefficients are not using the full specified bitwidth. In figure 4.2 the needed bitwidths of the LUTs corresponding to the coefficients from figure 4.1 can be seen. The set bitwidth for the coefficient calculation in the Python code is 22, the highest bitwidth in the resulting coefficients is 18. These findings suggest that it might be practicable to cut off the unused regions of the coefficients, to reduce the needed area for registers and therefore lower the power consumption.

This leads to the question, what the best way to reduce the coefficient register might be. The simple approach would be to just leave the coefficients in the order as they are and reduce the bitwidths of the corresponding registers accordingly. But when taking the shift register nature of the coefficient register and the complete distribution of coefficient bitwidths into account, this might not be practicable. Overall the size of the coefficients decreases. But due to the precalculation of the LUT coefficients the single sizes vary quite a bit, when looking at smaller portions of the coefficients. This implies a problem to the shift chain in the register. A

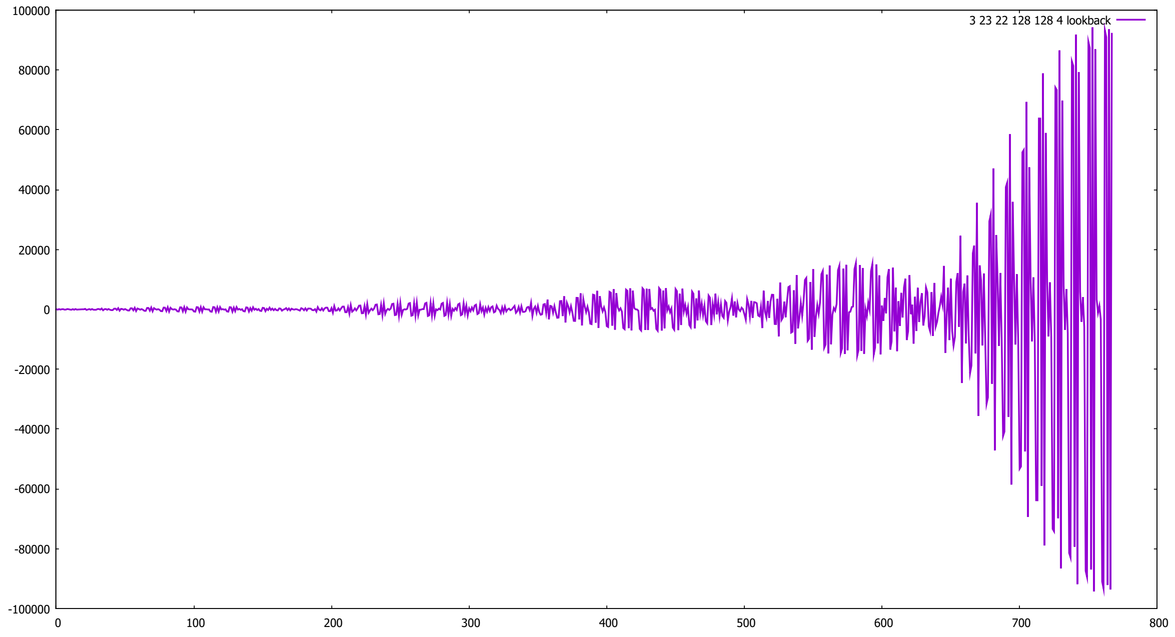

Figure 4.1: Lookback coefficients for N = 3, OSR = 23, bitwidth = 22, batch length = 128, LUT select input width = 2.

normal shift register has equally sized entries so that at every shift operation the same amount of bits is transported over to the respective next position. So far the coefficient register was designed in this way. When the single register entries now would have the exact matching bitwidths, it would not be possible to correctly shift them in any more. Looking at the situations, where a preceding coefficient has a lower bitwidth than the next one, makes this clear. An example can be seen in table 4.1. While LUTs 62, 64, 65 and 67 all need 13 bit, LUTs 63 and 66 have lower bitwidths.

An easy fix to this problem would be to reduce the bitwidth of the shift register entries as much as all following coefficients allow. When looking at the bitwidth distribution in table 4.1 again, the bitwidths of LUTs 63 and 66 need to be increased to 13 bit. But this leads to a non optimal solution in terms of possible area reduction, power reduction and headroom distribution. Some coefficients might be trimmed to a matching bitwidth, but others have an unused overhead of several bits. Especially the last point might be harmful, when looking at the reconfigurability of the system.

This leads to the idea of reordering coefficients within the coefficient shift register and subsequent trimming down to the matching bitwidths. When doing this for every coefficient according to its individual bitwidth, the negative factor of having to fill up some of the register spaces would vanish. But when looking at

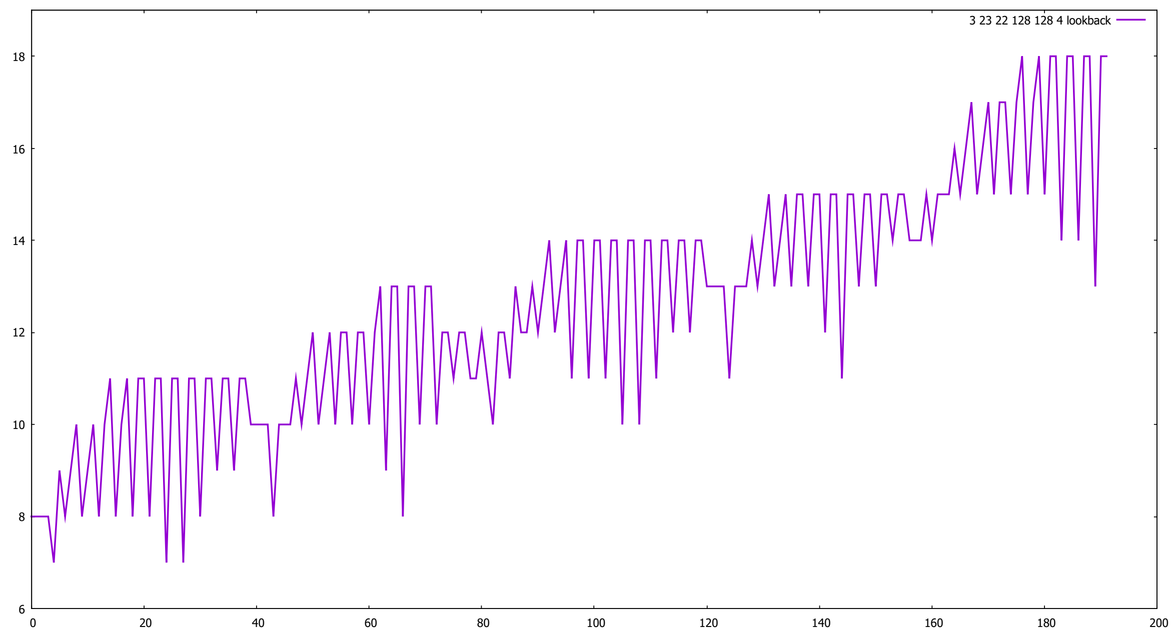

Figure 4.2: Lookback LUT bitwidths for N = 3, OSR = 23, bitwidth = 22, batch length = 128, LUT select input width = 2.

how the coefficients are precomputed (as it can be seen in the pseudo code snippet 1), other problems get introduced. The addition of several coefficients, which might have different signs, might lead to precomputation results much smaller than others. Therefore the single entries for one LUT might have some great variance in bitwidth. An example is given in table 4.2. When now thinking about the recomputation and changeability of these coefficients, it becomes clear that the coefficients with lower bitwidths would be locked to smaller values forever. This greatly reduces the reconfigurability and flexibility of the system. Therefore this is not a practicable option, it limits the filter design too much in favour of reducing the power consumption. In such a tight power budget situation a fixed coefficient implementation might be preferable.

LUT number	Bitwidth
62	13
63	9
64	13
65	13
66	8
67	13

Table 4.1: Lookback LUT 62 to 67 coefficient widths for N = 3, OSR = 23, bitwidth = 22, batch length = 128, LUT select input width = 2.

```
int[] calculate_coefficients(int h[], int lut_input_width)
  int coefficients[lut_input_width**2] = { 0 };
  for(int index = 0; index < lut_input_width**2; index++)
  {
    for(int shift_offset = 0; shift_offset < lut_input_width; shift_offset++)
    {
      if(index & (0b1 << shift_offset))
      {
        coefficients[index] += h[shift_offset];
      }
      else
      {
        coefficients[index] -= h[shift_offset];
      }
    }
  }
  return coefficients;
}
```

Listing 1: Pseudo code for LUT coefficient generation.

When now thinking about this problem that some bitwidths of coefficients within a LUT might be misleading, a new estimation metric has to be found, which is valid for all coefficients of one LUT. The biggest coefficient within a LUT is a good choice for estimating the possible bitwidth reduction for all its coefficients. With this limit for the bitwidth reduction, the flexibility of the system is preserved in most parts and the area and static power of the coefficient register can be reduced by an estimate of 42.655%, when looking at the reduction of overall bits needed for the register. Furthermore the coefficients can now be moved around in LUT blocks within the coefficient register to match the shift chain, without breaking its functionality. The reordered LUT bitwidths from example 4.2 can be seen in figure 4.3. Here no filling up of coefficients is needed between LUTs.

Even though this solution solves most of the design problems, there are still some minor difficulties left. So far the lookback and lookahead coefficients have been handled as separate arrays, but have been stored in the same shift register. But due to the reduction of the bitwidths this would mean that lookback and lookahead coefficients would mix up in the shift register. Therefore this theoretical separation has been changed to an actual separation into two different shift registers in the module. This makes it necessary to have another input, which selects to which shift register the incoming coefficients should be written. A one bit select signal is sufficient to fulfil this task. This does not introduce a noteworthy overhead to the system, which would decrease the system performance. A general abstraction of the resulting coefficient shift register module can be seen in figure 4.4.

Coefficient number	Value	Bitwidth
0	-183101	18
1	2665	12
2	-184605	18
3	1161	11
4	-185383	18
5	383	9
6	-186887	18
7	-1121	11
8	1121	11
9	186887	18
10	-383	9
11	185383	18
12	-1161	11
13	184605	18
14	-2665	12
15	183101	18

Table 4.2: Lookback LUT 95 coefficient values for N = 3, OSR = 23, bitwidth = 22, batch length = 128, LUT select input width = 4.

Due to these changes, implementing the shift chain and the distribution of the stored coefficients also became more difficult in the SystemVerilog code, leading to much more illegible code. Before, generate blocks and array indexing could be used to specify the shift chain and to assign the coefficients to the corresponding LUTs. This is not longer possible since the order of coefficients was most likely changed and SystemVerilog array declarations always have the same width for all entries. At this point the Python framework has to take over the shift chain generation and the coefficient assignment to the LUTs, since the SystemVerilog code does not have all needed information available.

This reduction also makes sense in another way. Since the LUTs will most likely be constructed out of multiplexers, the bitwidth of the biggest coefficient decides the needed bitwidth for the most multiplexers within the LUT. So making all coefficients the same size at the input of a single LUT does not introduce major drawbacks especially in terms of power consumption.

This optimisation is only applicable to a variable coefficient filter implementation, since the fixed coefficient version does not have a coefficient shift register.

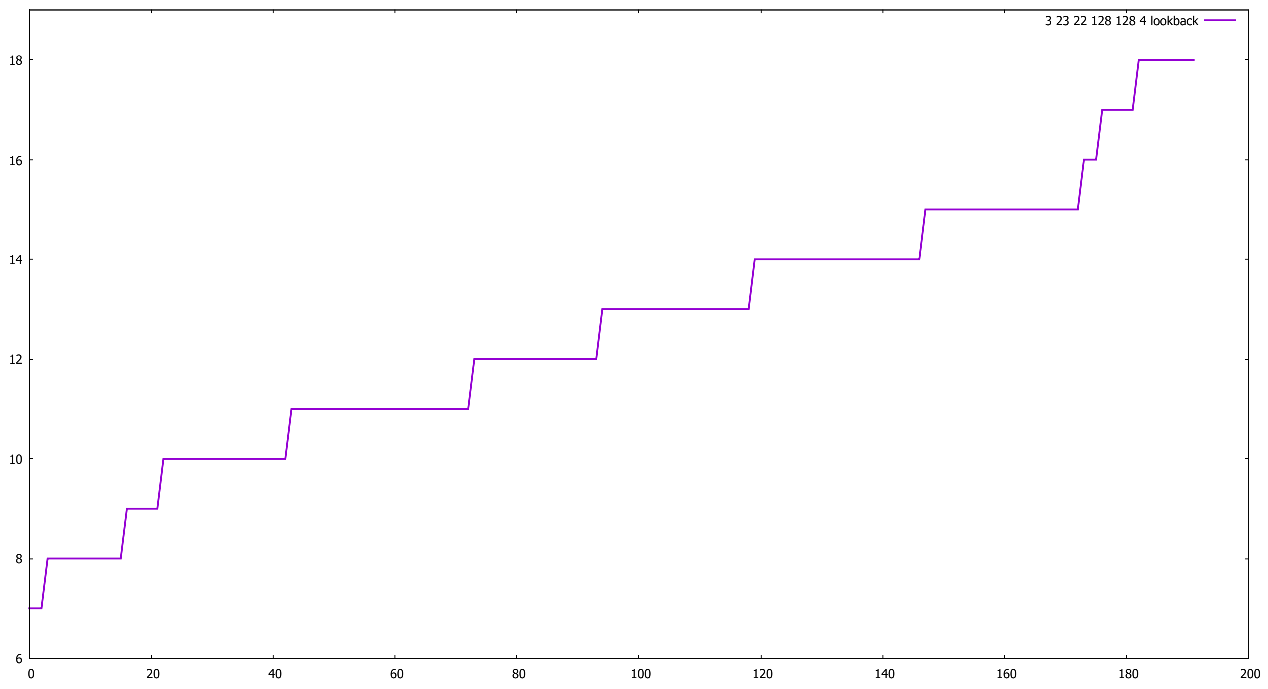

Figure 4.3: Lookback LUT bitwidths sorted for N = 3, OSR = 23, bitwidth = 22, batch length = 128, LUT select input width = 2.

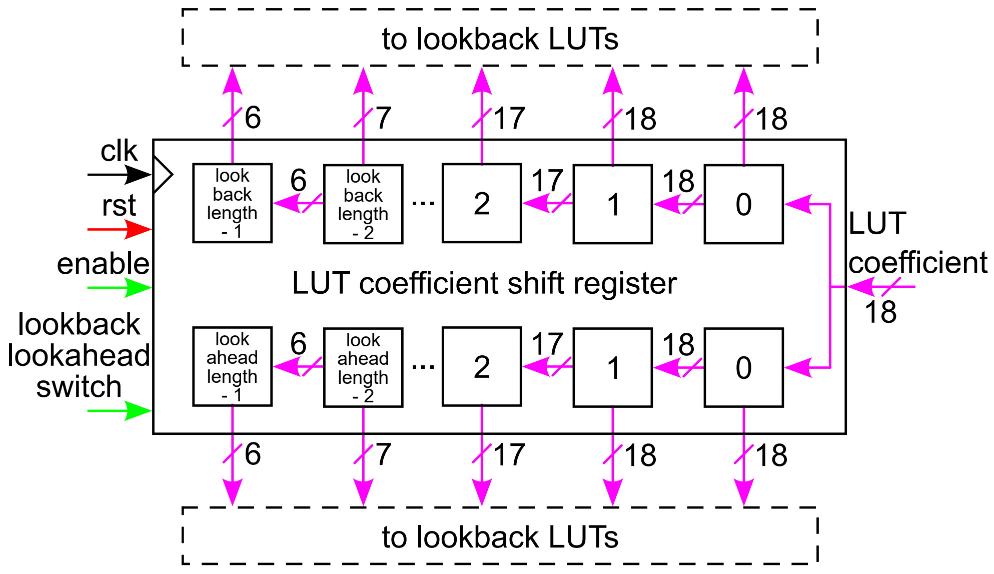

Figure 4.4: Coefficient shift register module with reduced bitwidths.

4.2 Look up tables

When looking further into the datapath of the filter coefficients in the FIR digital estimator design, the next step after the coefficient register are the LUTs. The coefficients get routed through the LUTs with multiplexers, therefore all internal datapaths need the bitwidth of the data inputs. So far the LUTs were all constructed with an equal data input bitwidth, since all coefficients had the same size. When reducing the physical register size for the coefficients, the next logical step is to match the LUTs to the assigned coefficients. This means every LUT needs its own instantiation. Hence, this procedure leads to different datapath bitwidths between some LUTs. Since this size information is part of the reduced and reordered coefficients, this step has to be carried out in the Python framework. There all necessary data is stored in an easily accessible way. Doing this in the SystemVerilog code would further complicate it and reduce the readability. The number of control input bits stays the same for all used LUTs. An example cutout of some LUTs and the according adders of a possible resulting system implementation is shown in figure 4.5.

When constructing the register to transport the results to the adders, there are two possibilities. They are dependant on weather the adders are implemented at full size or if they also get reduced in size as discussed in the next section. The first possibility leads to the assignment of the LUT results to a SystemVerilog array, where the output can just be assigned to an array cell. The second option leads to a data bus without distinguishable cells, but with individual addressable bits. Here it is needed to keep information about the individual start and endpoints of the LUT results. This information is included in the calculation from the Python framework for the reduction and reordering of the coefficients. Therefore the matching of the data bus regions between LUT outputs and adder inputs is also better already done in the Python calculations, than doing it in the SystemVerilog code. There only the result are visible.

When comparing the expected impact of this design choice with the impact of the reduction of the coefficient shift register for a variable coefficient implementation, it most likely is negligible. The coefficient register makes up the most part of the design. The internal wiring overhead of a LUT is small compared to it. In order to measure the influence of this reduction step, some filter design at different stages of design reduction are compared later in chapter 7. The more interesting aspect of this design reduction is its possible impact on fixed coefficient implementations of the digital estimator. Depending on how well the synthesis program can detect possible irrelevant bits itself, the gained performance increase may vary. The fixed coefficient design just attaches logical ones and zeros directly to the LUT inputs. It might be possible that the synthesis tools recognises, when the most significant bits are just filled with zeros for positive numbers or ones for negative numbers. The exact behaviour has to be analysed by comparing some system implementations. The results can be seen in chapter 7.

4.3 Adders

The last step in the datapath of the FIR digital estimator are the adders, which sum up all coefficient factors from the LUT outputs. In the end all coefficients have been summed up to one scalar value. In the full size coefficient configuration, the adders were configured to just use the specified bitwidth of the coefficients as width for the two data inputs and the data output. When thinking about this configuration, an overflow problem could theoretically arise, since the addition of two B bit wide numbers leads to a results with a maximum possible bitwidth of B + 1. The only thing preventing the adders in the filter from overflowing, is the composition of the number space of the fixed point coefficients and the estimation value. The cbadc Python package computes the filter coefficients in such a manner that they reflect the specified maximum analog signal amplitude. Therefore it is only possible to use up the specified bitwidth and an overflow during normal operation is impossible. During the startup phase an overflow might occur, due to invalid digital state vectors. But since the data from the startup phase cannot be used anyway, this is not a concern and it is not necessary to change the design to account for this type of faults.

A reduction of the adder bitwidths will likely reduce the overall cell count in the design and therefore lead to a reduction in area and power consumption. For the input widths of the adder, the obvious choice is to lower them down to the output widths of the LUT outputs. The higher of the two numbers therefore then ultimately decides what overall size the adder has. Therefore it is rational to group together two LUTs with ideally the same output width. Due to the reordering of the coefficients and hence also the sorting of the LUTs by size, the optimal order of LUT results is already given. For the output it is not so simple this time. It cannot be ruled out that the addition of two results exceeds the bitwidth of the bigger one. Therefore the outputs of the adders have been made one bit larger than the bigger input value is. An example of adders with non uniform bitwidth input scenarios can be seen in figure 4.5.

When now again comparing the impact of this modification with the impact of the coefficient shift register reduction, a similar prediction as for the reduction of the LUTs can be made. The influence will most likely be negligible for the overall area usage and power consumption of the variable coefficient implementation. In order to find the influence of this change, some simulation and synthesis runs were carried out and the results are presented in chapter 7. Here again the much more interesting question is, if this design change can have an effect on the area and power consumption of the fixed coefficient implementation. Assuming that the synthesis tool was able to automatically recognise that it does

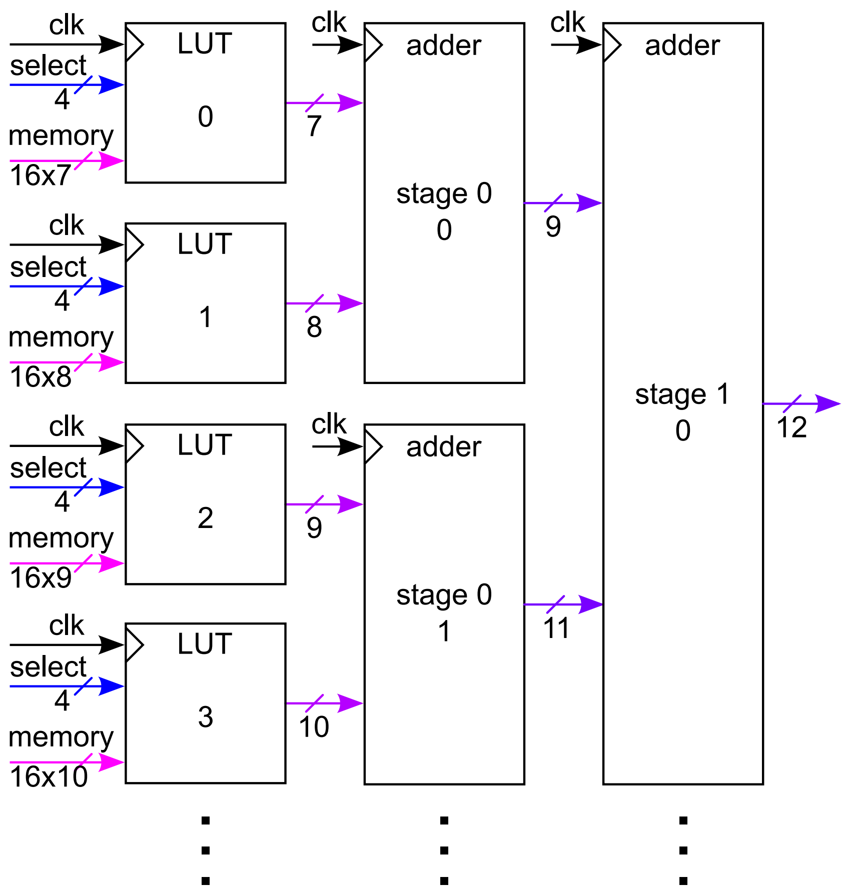

Figure 4.5: LUT and adder modules with reduced bitwidths.

not need to implement the LUTs at full size, it could also be able to automatically do this design reduction. If it could not detect the unused bits in the LUTs, it will most likely also not be able to implement this design reduction by its own. Simulations have been carried out, in order to find out, if a power reduction can be achieved for the fixed coefficient FIR filter implementation by manually reducing the bitwidths or if the synthesis algorithm is good enough to find this savings potential.

Chapter 5

Python framework

In this chapter the functionality of the Python framework is explained. The newly introduced design features for design resource reduction from chapter 4 extending the functionality briefly explained in chapter 3 are presented. The intended workflow is discussed and visualised. At the end of this chapter an overview over the most important Python classes and configuration options is given. This is intended to be a small instruction manual for the correct usage of the framework.

There are two primary goals, which the Python framework should accomplish. First it should automate the design workflow for implementations of the digital estimation FIR filter. Second it is supposed to provide an abstraction layer around the digital design aspects of the digital estimator implementation in SystemVerilog.

One big addition to the framework besides the design reduction features and the increased automated workflow is the possibility to create FIR filter implementations with a fixed set of coefficients. This feature was introduced, since simulations showed that this version of the filter features a highly attractive low area utilisation and lower power usage, when compared with the variable coefficient implementation. This of course removes one of the big advantages of the CBADC concept, the flexibility of the logically mostly separated parts of the design, providing the later possibility to tune the digital estimation. Simulations of fixed coefficient versions are also included and discussed later in order to determine, if variable coefficient variants are worth the extra design overhead.

5.1 Design reduction implementation

The proposed changes in the FIR filter design of course had to be reflected into the Python framework in order of being able to use these modifications in implementations. Formerly the Python high level simulation was carried out and the resulting filter coefficients were used in the SystemVerilog testbench to simulate a startup process with shifting in coefficients and carrying out a signal estimation. The SystemVerilog modules were adjusted by setting parameters in the modules according to the input specification. The design files itself were generic, repeated module instantiations were done with generate blocks. This method of creating modules is not viable for resource reduced implementations any more, since the single module instantiations are to non uniform. There are two feasible options to solve this problem. First, all necessary module and interconnect information could be given to the SystemVerilog code. This would lead to a lot of version specific information in the SystemVerilog code, which would make it much more illegible. Furthermore the result still would not be directly usable as generic code. The second option is to precompute every module instantiation in the Python framework in clear text and then assemble everything at the file generation. Since the use of the coefficient reduction options makes the framework essential anyway, the second option was chosen.

5.1.1 Coefficient shift register size reduction

The starting point for the resource reduction of the design starts after simulating the digital estimation in Python, extracting the lookback/lookahead coefficients and precomputing the look-up-table memory contents. These values are then allocated to their respective LUT. All precomputed coefficient values are then analysed for their resource usage by calculating the bitwidths for all of them (example in figure 4.2). Simulations have shown that for all configurations the coefficient bitwidths follow a similar pattern. Throughout the lookback/lookahead arrays the coefficients tend to become smaller and smaller going from one edge to the other. When looking closely at the bitwidths some local ripples can be found. These ripples would lead to a randomly higher resource usage as discussed previously. Therefore the now mapped to the LUTs and size measured coefficients get analysed for the highest resource usage within each LUT. This information is stored in another array corresponding to the coefficient array. With the help of the information from this new array, the coefficient array can now be sorted according to the bitwidths of the LUTs (resulting bitwidth order can be seen in figure 4.3).

The used sorting algorithm is called bubble sort [12]. Its complexity in average is $O(n^2)$, the best case complexity is $O(n)$. So normally it is not the most efficient algorithm. But since the coefficients do have a given natural basic order, it is unlikely that the worst case will be hit. The actual sorting effort will be much closer to the best case scenario. Furthermore the algorithm does not have to work on massive databases, the maximum number of values for the given test set used in this thesis is 512. The algorithm has another desirable characteristic. If two entries in the set have the same value, the relative order of these entries is maintained. Since the algorithm has to track the reordering of the LUTs, this leads to overall less LUT switching, making the resulting code more readable and intuitive. The code for this algorithm is given in listing 2.

```python
def sort_luts_by_size(lut_bit_widths: list[int]) -> list[tuple[int, int]]:
    new_lut_order: list[tuple[int, int]] = list[tuple[int, int]]()
    for index in range(len(lut_bit_widths)):
        new_lut_order.append((lut_bit_widths[index], index))
    number_of_luts = len(new_lut_order)
    for pass_number in range(number_of_luts):
        for element_index in range(0, number_of_luts - pass_number - 1):
            if new_lut_order[element_index][0] > new_lut_order[element_index +
            ↪  1][0]:
                new_lut_order[element_index], new_lut_order[element_index + 1] =
                ↪  new_lut_order[element_index + 1], new_lut_order[element_index]
    return new_lut_order
```

Listing 2: Pseudo code for LUT coefficient generation.

The newly gained LUT coefficient order is then given to the LookUpTable-CoefficientRegister class responsible for creating the SystemVerilog code of the coefficient shift register. Previously the shift register was a two dimensional array. This made it easy to implement a shift operation, since all coefficients only need to be shifted one place every clock cycle. When individually reducing the size of the coefficients, a two dimensional array can not longer be used to achieve the shifts, since the cells cannot have individual sizes. Instead two large register are now created for the lookback and lookahead coefficients, where every shift operation has to be implemented separately. The Python framework computes all the necessary shift operations and adds them to the SystemVerilog code for the shift register module. This also introduces the need for another one bit control signal, switching between the lookback and lookahead register. When two coefficients have the same size, every bit can just be shifted over normally. The interesting case occurs, when the following entry has less bit than its predecessor. Then only a certain number of bits beginning from the least significant bit (LSB) is transported over to the next entry. The remaining bit hits a dead end and is just overwritten without further transportation.

5.1.2 Look up table size reduction

Aside from the coefficient register class also the DigitalEstimatorWrapper top level module class used for connecting all the model components needs the new coefficient size and order information. Here the lookback and lookahead registers are assigned to the LUTs. But shrinking the coefficient register was only the first step in reducing the resource usage of the SystemVerilog model. As introduced in chapter 4 also the LUTs can be reduced in size by manually matching the internal data bit width to the coefficient shift register. Before this was done by using a SystemVerilog generate block for producing all the LUTs and the result assignments to the results register. Now the Python framework has to implement every module instantiation separately, due to the non uniform bitwidths. Therefore it also needs the previously computed new coefficient order and the according size information. The correct coefficients for the data input have to be picked out from the coefficient register.

5.1.3 Adder size reduction

The last step in the design reduction is the adjustment of the adders to the right bitwidth coming from the LUT outputs. Since several two input adder stages are needed to sum up all LUT output to one single value, the generation of the adders needs to be split up in two parts. The first stage only takes in LUT outputs. Therefore the size information from these outputs is needed. The adders then get their two inputs generated to the matching width. Since overflows could occur, if no carry bit was introduced in the output, it gets one bit wider than the larger of the two inputs. This new output size information has to be stored for the next stage of adders to configure the according inputs with the right sizes. When there is an odd number of values to be added in one stage the last value gets passed to the next stage with one clock cycle delay, instead of using an adder. Due to the increase of the output widths of the adders by one bit, at some point in the later adder stages the output width can become larger than the initially specified bitwidth for the coefficients. This overhead can be cut off, by using an adder with the right bit width to add up the lookback and lookahead result, thus leading to the specified overall output bit width at the design output again.

5.1.4 Design considerations for the size reduced version

The three stages of design reduction, coefficient register, look-up-tables and adders can be activated separately. The only condition for a stage to be usable is that the previous stage has to be active too. So only the following combinations are valid: 1. coefficient reduction; 2. the coefficient and LUT reduction; 3. the coefficient, LUT and adder reduction. Only the LUT reduction or the adder reduction can not be used. Also the combination from coefficient reduction and adder reduction is no allowed.

The major point of making a variable coefficient version of the FIR filter digital estimator design in the first place is the fact that it can be reconfigured for later tuning. Since the reduced design took some of the flexible away, it is of course not as flexible as before. Therefore it might be a good idea to include a certain margin in the design reduction. This can be in the form of making all of the coefficients universally 2 bit wider than they need to be. This method should significantly increase the flexibility again, while also only sacrificing a small portion of the gained resource reduction. Some similar filter implementations were evaluated for how much difference the coefficients show. The results can be seen in appendix A.1. This gives a good estimate on how big the margin has to be.

5.2 Workflow

In this section the intended workflow and the used external tools are presented. Some parts are implemented redundantly in order to be able to compare results of workflow stages to further proof the design. When shortly summarised, the workflow can be divided into the following stages: 1. Setting up the test cases. Ideally the pytest test environment is used. It makes it possible to set up test vectors, which then can be run by a single command. 2. The Python framework receives a single test from the test vector. It runs the high level simulations of the analog system and the digital estimation and saves these results as golden sample. 3. The SystemVerilog files are generated. 4. A register transfer level simulation is launched. The results are compared to the golden sample and the SNR is checked. 5. The design is synthesised. 6. The mapped design is simulated and the signal activity is recorded. With the mapped design simulation data a power analysis is performed, now featuring parasitics and the correct activity factors for all signals. 7. The design is placed and routed (P&R) 8. The placed and routed version of the filter is simulated again with recording the signal activity. 9. Another power analysis is performed using the updated signal activity factors for all signals. In figure 5.1 the steps of this workflow are visualised.

Setting up the tests

For setting up the test cases a separate file is used, which is loaded on activation of the pytest program. This file holds the test data vectors, the used class objects and the method calls to these objects. A collection of test cases is held in a list. Every individual test case is constructed as a tuple. The tuple holds values for every variable input of a test function. Therefore the construction of the tuple has to match the test function header. There have to be the same amount of entries in the tuple as there are in the function header. Furthermore the data types of the tuple entries have to match the data types of the function header and they have to be in the same order. The function is marked as parametrised so that the pytest environment takes in all the tests from the test vector. Besides it needs an unique ID to clearly identify it as separate test case. With this ID this function and the test vector behind it can be called explicitly from a program call to pytest, which then only launches this test. This also means that every test vector needs its own distinct function. To circumvent code duplication and errors due to this duplicate code maintenance, a generic function was introduced, which gets called from within every test case function. Hence the test case functions are hollow shells with calls to the generic function with the same layout. There an object of the class DigitalEstimatorGenerator is created. This object first is configured as specified in the current test case and all the necessary file directories for simulation and synthesis are set up accordingly. The generic function is also the place, where the decision is made, what test steps should be carried out. So next come the calls to the methods responsible for executing the single test steps. Most of the test steps depend on results from previous test steps, so a certain order of method calls has to be maintained.

High level calculations and simulations

The first step, which is mandatory to be carried out, is the high level simulation of the design and the generation of the SystemVerilog files with these results. For the high level simulation the analog system is simulated with the cbadc Python package based on a test input signal (exact test setup described in 6). This generates a set of analog state vectors, which is used to simulate the digital control unit. The output from the digital control unit simulation are the digital state vectors. On the one hand these are used to simulate the digital estimation filter on high level with the cbadc Python package and create a golden sample result data set. On the other hand they are stored as simulation input data for eventual later simulation runs on SystemVerilog code.

Besides from creating a golden sample simulation result data set, the simulation of the digital estimation filter also calculates the filter coefficients. Before these coefficients can be used in the SystemVerilog code, they need to be run through a precomputation step, since the look-up-tables combine several digital state bits in their inputs. The combination of the coefficients depends on the number of bits in the LUT select input. Therefore the precomputation results are unique to one LUT configuration.

If the option to reduce the design in resource consumption was chosen in the current test case, the coefficients undergo the previously in chapter 4 described calculations and modifications. When all these calculations are done, now the SystemVerilog files can be created. The input stimulus is placed in a separate file, which is used by the testbench when creating the test input to the digital estimator. The coefficient are either stored in another file, also used by the testbench to shift in the coefficients for variable coefficient implementations or they are placed directly in the digital estimator module code for fixed coefficient versions.

RTL simulation

The simulation of the generated RTL SystemVerilog implementation of the digital estimation filter is not an obligatory step in the workflow. However it is highly recommended to carry out this step. By simulating the RTL design and capturing the estimation output, the design can be checked against the high level Python implementation. This is done by directly comparing the single estimation values step by step. Furthermore the SNR of the approximated signal is calculated for both the RTL and Python high level results.

To increase the confidence in the design the RTL simulation can be launched redundantly in two different simulation programs from two different companies. The first option is the Xcelium simulator from Cadence Design Systems, Inc. The second option is VCS from Synopsys Inc. Both companies offer various electronic design automation (EDA) tools, of which several more are used in the later workflow steps. Based on the simulation results it can be decided, if the filter implementation meets the expectations and requirements or if the configuration or the design have to be altered in order to get the sought result. The next step to launch then is the synthesis of the RTL design.

Design synthesis

The synthesis of the RTL design creates a technology dependant Verilog implementation of this exact digital estimator configuration. In order to do this, the synthesis program needs all used SystemVerilog files and a technology library as input. The technology library specifies, which cells at gate level are available. The choice of library of course influences the possible outcome and performance of the synthesised design. It should be noted that the design synthesis removes all parametrisability from the design. Module boundaries are broken, internal signals and their names may disappear and all value references are removed by actual values. Therefore the synthesised design is only valid for the exact configuration chosen before the synthesis. Different filter configurations all need their own synthesised design files.

The gate level design of the digital estimator is also saved as SystemVerilog file. Besides from the mapped design two types of other files are created. The first type holds additional data from the design. This includes capacitive and inductive parasitic data for the mapped design. These files can be used in a following step for further design simulation. And then next to actual design files also a set of reports can be generated. The three most important reports are the timing, area and power reports. The timing report gives insight into possible timing violations. A timing violations occurs if a signal needs more time to travel from start to finish point than allowed by the set clock period. The headroom of signal arriving times to this period is called slack. For a design to work properly all signals need a positive slack. If signals have a negative slack, they will not arrive in time and

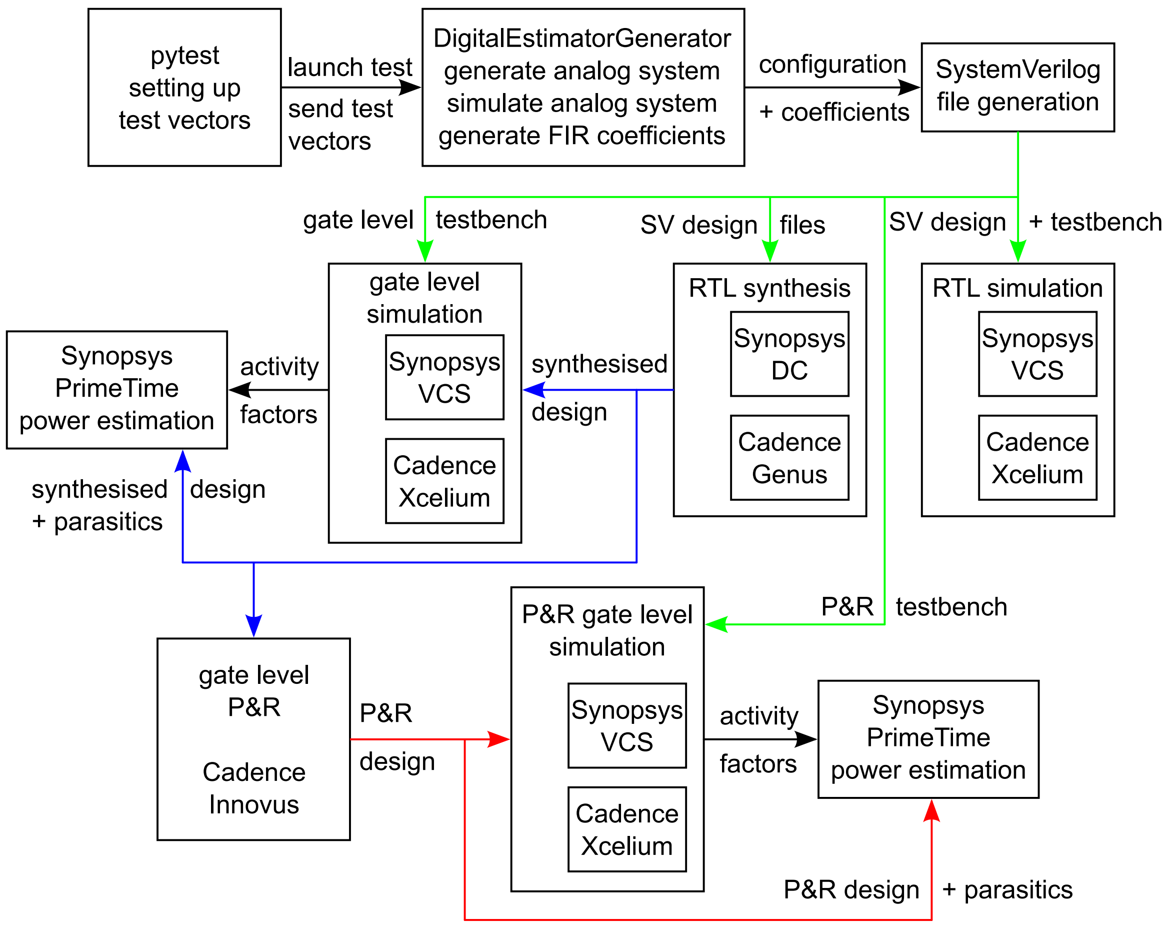

Figure 5.1: Visualised workflow of the digital estimator Python framework [10].

the parts of the design influenced by the fanout of this signal will not work as intended. The area report gives an overview over the used amount of different gates in the design and the estimated area these gates will take up in a chip. Since the required area of the design will always be a design consideration, this report can be conclusive if the design is manufacturable as intended. The power report gives an overview over the estimated power consumption of the gates in the design. Since power will also always be an important design criterion (in the case of this thesis the most important one), this report can give an estimate for the feasibility of the design. This report has a high precision, when it comes to comparing different implementations to one another. However it should be questioned for its accuracy, since it does assume a fixed and arbitrary activity factor for all signals. When a design has seldom switching parts, their dynamic power consumption will be over estimated, according to the equation for dynamic/switching power 5.1 (overestimation of activity factor α). This is likely the case for all variable coefficient implementations of the filter, since the coefficient register gets filled up once at startup and after that remains unchanged.

$$P_{dynamic} = \alpha \cdot C \cdot V^2 \cdot f \tag{5.1}$$

In this step again two different programs are available for carrying out the design synthesis. The first program is called Genus and comes from Cadence Design Systems, Inc. The second option is the Design Compiler (DC, not to be mistaken with the digital control unit) software from Synopsys Inc.

Post synthesis simulation

After design synthesis two different types of simulation of the resulting gate level implementation can be carried out. The first one is a behavioural SystemVerilog simulation, which is supposed to check if the design behaves as expected. Therefore a SystemVerilog testbench similar to the one used for the RTL simulation is created. This testbench uses the same digital state input vector as the Python high level and the RTL simulation. However this testbench cannot check any modules on their correct working with assertions any more, since the internals of the digital estimator implementation have been replaced by technology dependant gates. But it can still capture the signal approximation output from the filter. When the sample output was recorded, another power spectral density calculation can be carried out. With these results the by the gate level implementation achievable SNR can be calculated. So looking at the output itself and the calculated SNR gives an estimation, if the design matches the intended performance metrics. For carrying out this simulation there are again two choices available. They are the same programs as for the RTL simulation, Xcelium from Cadence Design Systems, Inc and VCS from Synopsys Inc.

The second thing the gate level testbench does, is to capture the switching activity of the internal signals of all the gates. With this data the second type of simulation can be launched. It is a power estimation, which now factors in the recorded switching activity for all internal gates. Furthermore the parasitic data from the synthesis is used for this simulation. Hence this power estimation is much more accurate than the power report from the synthesis tool alone. The program, which is used for this type of simulation is the static timing analysis tool PrimeTime from Synopsys Inc. These two simulations are again not necessary for carrying out further workflow steps, but they should be done in order to evaluate the design performance, before going on.

Place & route

After the successful synthesis of the RTL design, the resulting gate level netlist can be placed & routed. In this step the virtually connected gates get placed onto a physical layout and the trace connections between the gates are created. As input the gate level SystemVerilog file and the technology libraries are needed.

The output of this calculation is a design layout of the digital estimation filter, which can be used to manufacture the chip. Furthermore updated version of the gate level SystemVerilog files and the parasitic annotations are created. With these updated files it is possible to go through another round of design simulation and verification. Besides design output files, again reports can be generated. The most important ones are again the timing, area and power reports. The same things as for the synthesis reports apply to these reports. The used program is Innovus from Cadence Design Systems, Inc.

Post place & route simulation

The post place & route simulations are the same as for the post synthesis simulations. A behavioural simulation can be done either with Xcelium or VCS. The captured signal activity can be given to PrimeTime to generate another power report. This power report is most likely the most accurate estimation for the power consumption of the digital filter implementation, since after P&R there is the most accurate design implementation information available.

5.3 Classes and configuration options

In this section the most important available classes, methods and configuration options of the Python framework are briefly presented. This is supposed to be used as a small documentation for getting into the framework.

5.3.1 Classes

The classes can roughly be separated into two different types. The first type are classes, which represent SystemVerilog modules. These classes have a naming scheme, which closely resembles the module names for better readability. These classes are all child classes of the abstract parent class SystemVerilogModule. This provides unified functionality when generating the SystemVerilog files. The Python classes modelling SystemVerilog modules can again be categorised into two categories. The first category models actual design files used in the digital estimator filter design. Table 5.1 shows these classes including a small description of the function of the SystemVerilog module behind it.

AdderBlockCombinatorial.py Models the adder block module, which sums up the outputs of the LUTs. Uses fully combinatorial adders. Is only used for non size reduced implementations.

AdderBlockSynchronous.py Models the adder block module, which sums up the outputs of the LUTs. Uses synchronous adders. Is only used for non size reduced implementations.

AdderCombinatorial.py A single fully combinatorial adder with two inputs. Used for non size reduced implementations.

AdderCombinatorialReducedSize.py A single fully combinatorial adder with two inputs. The inputs can have different input widths. The output has the size of the larger input plus one bit. Used for size reduced configurations.

AdderSynchronous.py A single synchronous adder with two inputs. Used for non size reduced implementations.

AdderSynchronousReducedSize.py A single synchronous adder with two inputs. The inputs can have different input widths. The output has the size of the larger input plus one bit. Used for size reduced configurations.

ClockDivider.py Generates the downscaled clock for the internal modules in a digital estimator. It is only included in the design, if a downsample rate other than 1 is used. The counter type can be chosen between normal binary and Gray code counter.

DigitalEstimatorWrapper.py This class combines all SystemVerilog modules needed in the design and therefore is called a wrapper module, wrapping up the design. As configuration it takes in the number of analog states, number of digital states, LUT input width, the lookback length, the lookahead length, the data width, the downsample rate, the coefficient configuration and the settings for design resource usage reduction.

GrayCodeToBinary.py A module, which can convert Gray code into binary code. It is used in the clock divider module. It is only included in the design, if a downsampling rate other than 1 is used and the clock divider module uses Gray code as counter encoding.

GrayCounter.py A Gray counter, which counts up to the specified top value and then restarts counting from zero. It is used in the clock divider module.

InputDownsampleAccumulateRegister.py This models the shift register for accumulating digital control vectors between estimations, when a downsampling rate other than 1 is used. Therefore it is only needed and included when the downsampling rate is higher than 1.

LookUpTable.py A single fully combinatorial LUT. Parametrisable select input and data width. Used for both full size and size reduced implementations.

LookUpTableSynchronous.py A single synchronous LUT. Parametrisable select input and data width. Used for both full size and size reduced implementations.

LookUpTableBlock.py This models the module holding the LUT arrays for lookback and lookahead computations. Uses fully combinatorial LUTs. Only used in non size reduced configurations.

LookUpTableBlockSynchronous.py This models the module holding the LUT arrays for lookback and lookahead computations. Uses synchronous LUTs. Only used in non size reduced configurations.

ValidCounter.py This is the module indicating, if the digital estimator has accumulated enough consecutive digital control vectors for outputting a meaningful estimation (lookback and lookahead shift register are fully populated.

Table 5.1: Correlation between Python classes and SystemVerilog modules.

The second type of SystemVerilogModule child classes holds modules for design verification. These modules can be used in the SystemVerilog simulations. The most important module in this category is the testbench, used for RTL, synthesis gate level and P&R gate level simulations. The resulting SystemVerilog module needs different settings for all three simulation stages. Therefore this class has configuration inputs for setting the according simulation stage. Since the whole system is tested with this testbench, by recording the estimation output (and signal switching activity), the testbench is a system integration test for the whole implementation. The other classes in this category are modelling SystemVerilog assertions. The assertions are bundled together for the individual sub modules of the digital estimator. To differentiate between the classes modelling design modules and the ones modelling assertions, the assertion classes have an added "Assertions" at the end of the class name. The assertions are written as concurrent assertions, which means that they don't need to be called separately, they check the module internals automatically on certain entry conditions. The assertions can only be used in the RTL simulation, since after synthesis the internal structure of the RTL model is broken up into gates, removing the ability to differentiate between internal signals. Hence the generation of the testbench must factor this in, since an inclusion of the assertion modules in later simulation stages would lead to errors. In table 5.2 the assertion classes are listed and the function behind them is briefly explained.

AdderBlockCombinatorialAssertions.py This tests properties of AdderBlockCombinatorial modules. Currently no assertions are programmed, the test of this module was outsourced to the integration test. Used only in non size reduced implementations.

AdderBlockSynchronousAssertions.py This tests properties of AdderBlockSynchronous modules. Currently no assertions are programmed, the test of this module was outsourced to the integration test. Used only in non size reduced implementations.

AdderCombinatorial.py The first property checks, if resets are correctly executed so that it is ensured that the adder returns to a defined state after reset. The second property checks the computational result of the adder if the reset signal is deactivated, to ensure that the adders calculation is correct.

AdderSynchronous.py The first property checks, if resets are correctly executed so that it is ensured that the adder returns to a defined state after reset. The second property checks the computational result of the adder if the reset signal is deactivated, to ensure that the adders calculation is correct.

ClockDividerAssertions.py This module tests, if the clock divider works as intended. The first assertion checks for a proper reset. The second assertion checks the counting of the internal counter. The third assertion checks if the downsampled clock signal is generated correctly from the counter value.

DigitalEstimatorTestbench.py This module defines the system integration testbench. The digital estimator is setup according to the given specifications. Then the precomputed FIR LUT coefficients are shifted into the coefficient shift register. Now the given digital control vector stream gets simulated on the SystemVerilog digital estimator implementation. The output is stored in a file for further processing.

GrayCodeToBinaryAssertions.py This module first check for a proper reset of the Gray to binary conversion. Afterwards it checks for correct output values if new correct input values are present at the input.

GrayCounterAssertions.py The first assertion tests for a proper reset of the module. The second assertion compares if the counter is counting consecutively and resetting the output value, as soon as it reaches the defined top value.

InputDownsampleAccumulateRegisterAssertions.py At first the proper reset of the module is tested. The second test observes if the incoming values are correctly shifted within the shift register.

LookUpTableAssertions.py First the reset is checked. The second assertion checks if for every input value the correct LUT coefficient is connected to the output.

LookUpTableSynchronousAssertions.py First the reset is checked. The second assertion checks if for every input value the correct LUT coefficient is connected to the output.

LookUpTableBlockAssertions.py This tests properties of LookUpTableBlock modules. Currently no assertions are programmed, the test of this module was outsourced to the integration test.

LookUpTableBlockSynchronousAssertions.py This tests properties of LookUpTableBlockSynchronous modules. Currently no assertions are programmed, the test of this module was outsourced to the integration test.

Table 5.2: SystemVerilog assertion and testbench modules.

5.3.2 Configuration options

This section describes the configuration options present for the DigitalEstimatorGenerator class. This is the main interface for creating digital estimator implementations, either manual or with the pytest program.

path The path where the SystemVerilog files for simulations are stored. This includes the RTL, gate level and P&R simulations, as well as the gate level and P&R power estimations with PrimeTime.

path_synthesis This is the path for synthesis and P&R.

scripts_base_folder This is the folder holding the templates for scripts needed for synthesis, P&R and PrimeTime power estimation. Due to confidentiality these files cannot be included in the framework.

configuration_analog_bandwidth This sets the upper frequency limit for the analog system. Together with the oversampling/downsampling rate this also defines the digital frequency needed for the digital control unit and the digital estimation.

configuration_n_number_of_analog_states This sets the number of analog states used in the analog system. This number normally corresponds to the number of digital states in the design.

configuration_m_number_of_digital_states This sets the number of digital states used in the digital control. This number normally corresponds to the number of analog states in the design. The default value is therefore set to the same as the number of analog states.

configuration_lookback_length This sets the number of lookback values used in the digital estimation to compute an estimation of the input signal.

configuration_lookahead_length This sets the number of lookahead values used in the digital estimation to compute an estimation of the input signal.

configuration_fir_data_width This sets the bitwidth of the FIR coefficients, the adder inputs/outputs and the final estimation.

configuration_fir_lut_input_width This sets the number of bits used for the LUT inputs.

configuration_simulation_length This sets the number of tics used in the simulation. This number gets multiplied by the oversampling rate to ensure that different systems are simulated over the same amount of time and the same signal input frequency can be used.

configuration_over_sample_rate This is the ratio between the sampling frequency of the digital control and the bandwidth of the analog system.

configuration_down_sample_rate This sets the ratio between the sampling frequency in the digital control and the sampling frequency in the digital estimation. As standard this is set to the same value as the oversample rate, to match the bandwidth of the digital estimator with the bandwidth of the analog system.

configuration_counter_type With this setting the encoding of the clock down-sample counter can be chosen.

configuration_combinatorial_synchronous With this the datapath configuration of the LUTs and adders can be chosen. This can either be combinatorial or synchronous.

configuration_required_snr_db This sets the soft limit of the minimum SNR value the digital estimator should reach during testing. It does not change the design.

configuration_coefficients_variable_fixed This option selects, if the digital estimator is generated as variable coefficient or fixed coefficient version.

configuration_reduce_size_coefficients This is the first stage of design reduction possible. If using variable coefficients, the coefficient shift register is reduced. For the fixed coefficient implementation this has little to no effect at all on the design.

configuration_reduce_size_luts When selected, the LUTs are reduced to the according coefficient sizes. Only selectable if configuration_reduce_size_coefficients is selected.

configuration_reduce_size_adders When selected, the adders are reduced to the according LUT and preceding adder output widths. Only selectable if configuration_reduce_size_coefficients and configuration_reduce_size_luts is selected.

Table 5.3: Configuration options for the Python framework.

5.3.3 Workflow methods

This section gives a small overview over the methods of the DigitalEstimatorGenerator class, responsible for executing the workflow stages from figure 5.1.

generate(self) After the DigitalEstimatorGenerator class has been set to the wanted configuration, this method starts the cbadc Python package high level simulation, saves the results as golden sample and generates the SV files.

simulate(self) Starts the RTL simulation with Cadence Xcelium.

simulate_vcs(self) Starts the RTL simulation with Synopsys VCS.

synthesize_genus(self) Starts the design synthesis with Cadence Genus

synthesize_synopsys(self) Starts the design synthesis with Synopsys Design Compiler

simulate_mapped_design(self, synthesis_program) Simulates the mapped design with Cadence Xcelium. The input parameter synthesis_program selects the program, with which the design was synthesised.

simulate_vcs_mapped(self, synthesis_program) Simulates the mapped design with Synopsys VCS. The input parameter synthesis_program selects the program, with which the design was synthesised.

plot_results_mapped(file_name, synthesis_program, simulation_program) The PSD and SNR of the gate level digital estimation is calculated. The input parameters file_name, synthesis_program and simulation_program select, which simulation results should be analysed.

estimate_power_primetime(self, synthesis_program, simulation_program) The gate level power estimation with Synopsys PrimeTime is launched. The input parameter synthesis_program selects, which design should be analysed. The input parameter simulation_program sets, which recorded design activities should be used.

placeandroute_innovus(self, synthesis_program) Starts P&R with Cadence Innovus. synthesis_program selects, which design is used as input.

simulate_placedandrouted_design(self, synthesis_program) Simulates the placedandrouted design with Cadence Xcelium. The input parameter synthesis_program selects the program, with which the design was synthesised.

simulate_vcs_placeandroute(self, synthesis_program) Simulates the placed and routed design with Synopsys VCS. The input parameter synthesis_program selects the program, with which the design was synthesised.

plot_results_placeandroute(self, file_name, synthesis_program, simulation_program)
The PSD and SNR of the post place and route digital estimation is calculated. The input parameters file_name, synthesis_program and simulation_program select, which simulation results should be analysed.

estimate_power_primetime_placeandroute(self, synthesis_program, simulation_program)
The post P&R power estimation with Synopsys PrimeTime is launched. The input parameter synthesis_program selects, which design should be analysed. The input parameter simulation_program sets, which recorded design activities should be used.

Table 5.4: DigitalEstimatorGenerator class methods corresponding to workflow stages (figure 5.1).

Chapter 6

Test environment and methodology

In this chapter the utilised test environment setup is explained. This includes the used input signal, the setup of the analog system, the corresponding digital control unit and the used digital estimation FIR filter configurations, which have been selected in coordination with Andreas Bjørsvik and Sevat Mestvedthagen working on the RISC-V implementation of the digital estimation [4].

6.1 Input signal

To ensure the comparability between digital estimator implementations, a standardised input test signal was established for all test cases. The signal was chosen to be a sine wave with fixed amplitude and frequency. This makes the calculation of the signal-to-noise ratio quite easy, since a sine wave has favourable properties for a fast Fourier transform (FFT) (in the ideal case, an input sine wave would lead to only one contributing factor). The configuration of this input signal can be seen in table 6.1.

signal type	sine wave
frequency	78125,0Hz
amplitude	0,7
offset	0,0
phase	$\pi/3$

Table 6.1: Parameters of the input signal.

But not only the general signal form has an influence on the correctness of the FFT, other parameters also influence this calculation. Therefore all input signal parameters must be chosen in a way that these properties do not harm the power spectral density (PSD) calculation and resulting from that the SNR computation. The two problematic factors for the PSD calculations are window effects and aliasing of the signal. Window effects occur if the input signal does not have the right amount of data points and it therefore needs zero padding in the FFT. This effect can be avoided when the data set has a length of one element of the set $D = \{\, 2^i \mid i \in \mathbb{N}\}$. The aliasing of the signal can be avoided by fitting full sine wave cycles into the data set, which is given to the FFT computation. The input signal frequency therefore needs to be calculated in a way so that the wanted amount of full cycles fits into the given simulation length with the set sampling frequency. These signal criteria are the reason for the rather arbitrary looking input signal frequency.

6.2 Analog system setup

For this thesis a general configuration of the analog system was given. It included the specification for the analog bandwidth, the desired SNR/ENOB and the available power budget. Table 6.2 shows the given system. The SNR/ENOB values can not be reached exactly, since the number of analog states N and the OSR values are quantified as integer values. For the setup of the digital estimator implementations this means that the closest possible value was determined and used in the test vectors.

Analog bandwidth	SNR/ENOB	Available power budget
20MHz	∼70dB/∼11,34Bit	0,4mW

Table 6.2: Configuration of the analog system for which the digital estimation implementations should be designed for.

The most challenging part of the design is to meet the set power limit. It was chosen to match the power consumption of current state of the art and researched analog-to-digital converters. The specification of the number of analog states N was left open so that different configurations could be tested.

6.3 Digital estimator configuration

In this section the chosen digital estimator configuration sets are shown. These include variable coefficient implementation sets, fixed coefficient implementation sets and sets with different design reduction stages enabled.

6.3.1 Digital estimator base configuration

The given analog system leaves some freedom for the exact design of the digital estimator. The preceding specialisation project [9] explored on the lookback-/lookahead length, oversampling rate and bitwidth required so that the digital estimation does not hinder the overall system performance. In cooperation with Andreas Bjørsvik and Sevat Mestvedthagen, who are designing the RISC-V implementation of the digital estimator [4], a common configuration base for the digital estimation was chosen. The chosen system specifications can be seen in table 6.3.

Number of analog states N	Oversampling rate	Lookback (K1)/ lookahead (K2) length	bitwidth
3	23	128	22
4	15	160	22
5	12	160	22
6	9	224	22
7	8	224	22
8	7	256	22

Table 6.3: Digital estimator configurations matching the analog system specification, established in cooperation with Andreas Bjørsvik and Sevat Mestvedthagen.

These configurations set the used number of analog states N, the accordingly needed oversampling rate, the lookback/lookahead lengths and the used coefficient bitwidths. Thereby the digital frequencies, with which the digital estimation filters have to be synthesised, are defined. The according values can be seen in table 6.4. However, this still leaves some configuration options open for exploration in the conducted simulations. With these set boundaries a variety of final and complete system specifications was set up.

Number of analog states N	Oversampling rate	Digital frequency [MHz]
3	23	920
4	15	600
5	12	480
6	9	360
7	8	320
8	7	280

Table 6.4: Digital estimator operating frequencies.

6.3.2 Design reduction stage estimation

Before beginning with estimating configurations for the base set of the digital estimator implementations, the single design reduction stages have been tested. Therefore two test systems, one with variable coefficients an one with fixed coefficients, have been selected. For these systems, the configurations with no design reduction, reducing the coefficient shift register/assigning smaller coefficients, reducing the LUTs and reducing the adders have been set up. The test set can be seen in table 6.5.

Number of analog states N	Coefficient configuration	Reduce coefficients	Reduce LUTs	Reduce adders
3	variable	False	False	False
3	variable	True	False	False
3	variable	True	True	False
3	variable	True	True	True
3	fixed	False	False	False
3	fixed	True	False	False
3	fixed	True	True	False
3	fixed	True	True	True

Table 6.5: Test setup for design reduction stage estimation.

6.3.3 Variable coefficient implementation sets

When looking at test vectors testing the whole base configuration, first different digital estimator configurations with variable coefficients were explored. These configurations include variations of the LUT select input width and of course the comparison of standard implementations versus resource reduced implementations. The number of select input bits has a great influence on the overall number of coefficients needed for all LUTs. The relationship between the total number of needed precomputed coefficients, the lookback (K1)/lookahead (K2) length and the chosen LUT select input bitwidth can be seen in equation 6.1. Since the lookback (K1) and lookahead (K2) lengths don't change in this test setup, the normalised version of this equation 6.2 gives the information about how many precomputed coefficients are needed for one digital control vector bit. This therefore also gives insight into how big the coefficient registers have to be relative to one another independent of the exact configuration. The values for this calculation can be seen in table 6.6.

$$\#Coefficients_{total} = \frac{(K1 + K2) * 2^{select\_input\_width}}{select\_input\_width} \tag{6.1}$$

$$\#Coefficients_{relative} = \frac{2^{select\_input\_width}}{select\_input\_width} \tag{6.2}$$

Number of select input bits	Number of precomputed coefficients per digital control vector bit
1	2
2	2
3	$2\frac{2}{3}$
4	4
5	6,4
6	$10\frac{2}{3}$

Table 6.6: Number of precomputed coefficients needed per digital control vector bit.

However the LUT select input width does not only define the number of precomputed coefficients needed, it of course also has an influence on the total number of LUTs and adders in the design. With equation 6.3 the total number of LUTs needed for a design can be calculated. However, this time a normalisation does not make sense, since the LUTs have different resource usages with different select input bitwidths. Therefore also the comparison between the total numbers does not tell anything about the relations between the resource usages of different implementations with different select input widths.

$$\#LUTs_{total} = \frac{K1 + K2}{select\_input\_width} \tag{6.3}$$

The total number of adders in a design can be calculated with the pseudo code 3. This code can be normalised, since all adders have the same bitwidth for a full size implementation and for the resource reduced implementation this is still a good enough approximation. Then the number of adders needed in a design comes down to the used bitwidth for the LUT select input, since this value defines the number of LUTs present in the design. With a rising bitwidth of the select input the amount of needed adders becomes smaller. The extreme case would be one large LUT for all digital control vector values and therefore no adder is needed. This setup is of course not feasible, since the number of precomputed coefficients (see equation 6.1) and the one LUT would be very large. Also the timing of the circuit becomes more and more difficult with larger LUTs.

```
int getNumberOfAdders(int k1, int k2, int selectInputWidth)
{
  int numberOfLUTs = ceil((k1 + k2) / selectInputWidth);
  int numberOfAdderStages = ceil(log2(numberOfLUTs));
  int totalNumberOfAdders = 0;
  int numberOfStageInputs = numberOfLUTs;
  for(int stage_index = 1; stage_index <= numberOfAdderStages; stage_index)
  {
    totalNumberOfAdders += floor(numberOfStageInputs / 2);
    numberOfStageInputs = ceil(numberOfStageInputs / 2);
  }
  return totalNumberOfAdders;
}
```

Listing 3: Pseudo code for getting the total number of adders in a configuration.

So summarised in theory with an increasing LUT select input bitwidth the number of precomputed coefficients gets larger, the number of adders decreases and the number of needed LUTs decreases, but the resource usage per LUT rises. So for a resource usage ideal implementation it comes down to a trade-off between having more coefficients, less adders and less but more resource intensive LUTs or less coefficients, more adders and more but less resource intensive LUTs.

The results from the preceding specialisation project [9] regarding the optimum number of LUT select input bits has shown that a number between one and four can be considered resource effective for variable coefficient implementations. Everything above four leads to significantly larger designs, as the numbers from table 6.6 suggest, since the coefficient register is by far the largest part of the design. For fixed coefficient implementations the results might favour slightly larger but therefore less LUTs over smaller but more LUTs. Implementations with a LUT select input width of one bit have been generally disregarded, since the number of coefficients is the same as for system with a select input width of 2 bit and the number of LUTs and adders is higher.

With this design considerations in mind the following variable coefficient configurations have been chosen for testing. Table 6.7 and 6.8 both hold configurations with a LUT select input width of 4 bit. The first configuration set 6.7 is a normal implementation, the second configuration set 6.8 is the matching resource reduced implementation.

Number of analog states N	**LUT select input width**	**Coefficient setup**	**Design reduction**
3	4	variable	False
4	4	variable	False
5	4	variable	False
6	4	variable	False
7	4	variable	False
8	4	variable	False

Table 6.7: First configuration set for variable coefficient implementations. LUT select input width: 4; resource reduction: False

Number of analog states N	**LUT select input width**	**Coefficient setup**	**Design reduction**
3	4	variable	True
4	4	variable	True
5	4	variable	True
6	4	variable	True
7	4	variable	True
8	4	variable	True

Table 6.8: Second configuration set for variable coefficient implementations. LUT select input width: 4; resource reduction: True

Table 6.9 and 6.10 show two more configurations for variable coefficient implementations. This time the LUT select input width was chosen to be 2 bit wide. The third configuration set 6.7 is a normal implementation, the fourth configuration set 6.8 is the matching resource reduced implementation. This test setup enables the comparison between full size and resource reduced implementations as well as the estimation of the optimal LUT select input bitwidth for variable coefficient implementations.

Number of analog states N	LUT select input width	Coefficient setup	Design reduction
3	2	variable	False
4	2	variable	False
5	2	variable	False
6	2	variable	False
7	2	variable	False
8	2	variable	False

Table 6.9: Third configuration set for variable coefficient implementations. LUT select input width: 2; resource reduction: False

Number of analog states N	LUT select input width	Coefficient setup	Design reduction
3	2	variable	True
4	2	variable	True
5	2	variable	True
6	2	variable	True
7	2	variable	True
8	2	variable	True

Table 6.10: Fourth configuration set for variable coefficient implementations. LUT select input width: 2; resource reduction: True

6.3.4 Fixed coefficient implementation sets

Aside from variable coefficient implementations also fixed coefficient versions of the filter have been tested. This should give an estimation of how much more resource usage a variable coefficient implementation produces compared to a fixed coefficient one. The according test sets follow the same scheme as the test sets for variable coefficient versions. Table 6.11 and 6.12 show test vectors with a LUT select input width of 4 bit and tables 6.13 and 6.14 show test vectors with a LUT select input width of 2 bit. Therefore again the comparison between full size and reduced size implementations and the estimation of the optimal LUT select input width is possible.

Number of analog states N	LUT select input width	Coefficient setup	Design reduction
3	4	fixed	False
4	4	fixed	False
5	4	fixed	False
6	4	fixed	False
7	4	fixed	False
8	4	fixed	False

Table 6.11: First configuration set for fixed coefficient implementations. LUT select input width: 4; resource reduction: False

Number of analog states N	LUT select input width	Coefficient setup	Design reduction
3	4	fixed	True
4	4	fixed	True
5	4	fixed	True
6	4	fixed	True
7	4	fixed	True
8	4	fixed	True

Table 6.12: Second configuration set for fixed coefficient implementations. LUT select input width: 4; resource reduction: True

Number of analog states N	LUT select input width	Coefficient setup	Design reduction
3	2	fixed	False
4	2	fixed	False
5	2	fixed	False
6	2	fixed	False
7	2	fixed	False
8	2	fixed	False

Table 6.13: Third configuration set for fixed coefficient implementations. LUT select input width: 2; resource reduction: False

Number of analog states N	**LUT select input width**	**Coefficient setup**	**Design reduction**
3	2	fixed	True
4	2	fixed	True
5	2	fixed	True
6	2	fixed	True
7	2	fixed	True
8	2	fixed	True

Table 6.14: Fourth configuration set for fixed coefficient implementations. LUT select input width: 2; resource reduction: True

Chapter 7

Results

This chapter presents the results from RTL simulations, synthesis and P&R for the chosen test vectors. The gained data of all stages of the design process is shown. This includes the RTL simulation, synthesis, post synthesis simulation, P&R and post P&R simulations.

7.1 RTL simulation

All RTL simulations exactly matched the cbadc Python package high level simulations. Therefore the calculated SNR/ENOB values can be used as golden sample for estimating the performance of the synthesised and placed and routed designs. The values for the digital estimator base configurations (according to table 6.3) are shown in table 7.1. Thereby the SNR and ENOB are just two different ways of expressing the same thing. The values can be converted into each other, this is show by equations 7.1 and 7.2.

Number of analog states N	**SNR [dB]**	**ENOB [Bit]**
3	68,93	11,16
4	69,17	11,2
5	71,95	11,66
6	68,57	11,1
7	69,93	11,32
8	69,0	11,17

Table 7.1: RTL SNR/ENOB values valid for all digital estimator test sets.

$$ENOB = \frac{SNR - 1,76dB}{6,02dB} \tag{7.1}$$

$$SNR = ENOB \; * \; 6,02dB \; + \; 1,76dB \tag{7.2}$$

Both of the simulation programs available, Cadence Xcelium and Synopsys VCS, produced the same simulation results. However when looking at the overall simulation time, Synopsys VCS was noticeably faster for all configurations tested. When running larger test vectors this should be taken into account for minimising the overall runtime.

7.2 Synthesis & post synthesis simulations

In this section the results from the the carried out synthesis runs are presented. All systems have been synthesised with an industry standard 28nm process at a supply voltage of 0,8V. The looked at parameters are the estimated area usage and power consumption of the systems and the reached SNR/ENOB values of the different design configurations. The first test carried out was the estimation of the design reduction stages. This was done with both available synthesis programs. Generally both programs could be used for all other test sets as well. However, the next test was to find out, which synthesis tool gives the better performance and was used for all further tests. After these tests, first the variable coefficient configurations and then the fixed coefficient implementations are shown.

7.2.1 Design reduction stage estimation

In this section the influence of the different design reduction stages on the area and power consumption is shown. Figure 7.1 shows the area obtained from synthesis for the configurations 6.5. For the variable coefficient system a big decrease in used area can be seen, when reducing the coefficients. Going further and reducing also the LUTs and adders only makes a marginal change for the systems. The used area for the fixed coefficient versions is about the same for every reduction stage.

The synthesis power estimation can be seen in figure 7.2. The estimated power for the variable coefficient systems show a small decrease between the full size and the full reduction of the system. Synopsys Design Compiler has a glitch in the estimated power for the LUT reduction stage. The power is estimated to be several times as much, as for the other stages. For the fixed coefficient systems all reduction stages actually show an increase in power consumption. Again Synopsys Design Compiler has a glitch in the estimated power for the LUT reduction stage. Going to the PrimeTime power estimations (figure 7.3), the observations for the power consumption change a bit. The estimation for the coefficient version from Design Compiler now advertises an increase in power. Furthermore the fixed version synthesised with Genus shows a reduction in power for all stages. Both systems synthesised with Design Compiler on the LUT reduction stage could not be simulated, therefore no PrimeTime power estimation was possible. So far the results are very inconclusive with Synopsys design compiler advertising an increase in power and Genus advertising a decrease.

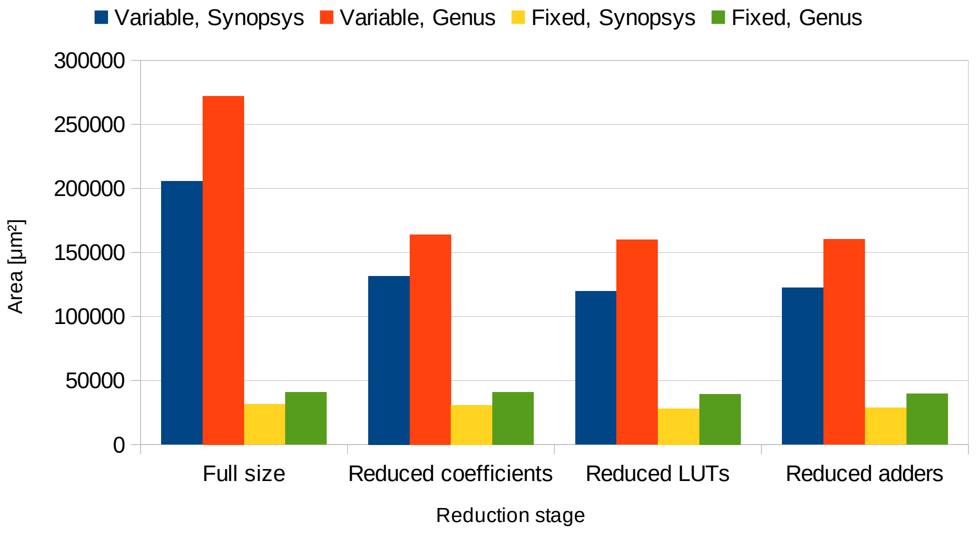

Figure 7.1: Design reduction stage estimation, synthesis area.

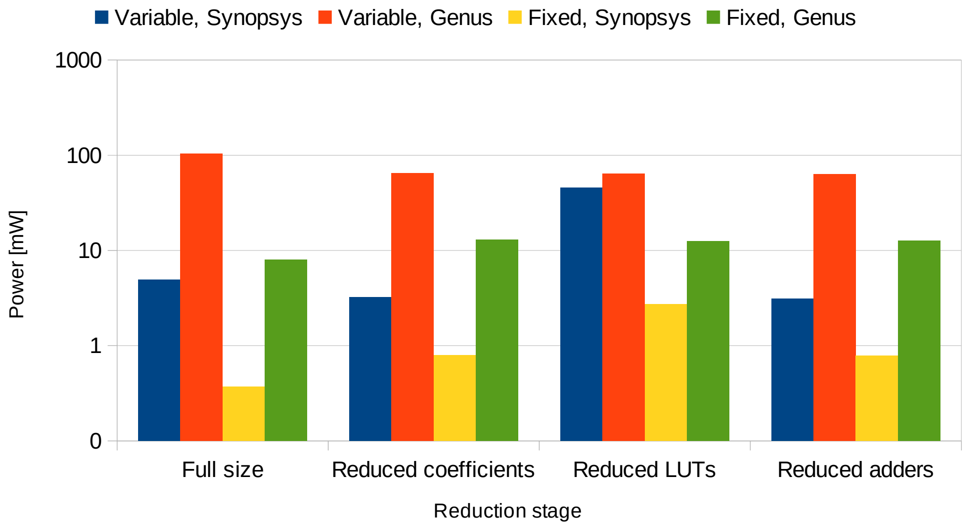

Figure 7.2: Design reduction stage estimation, synthesis power.

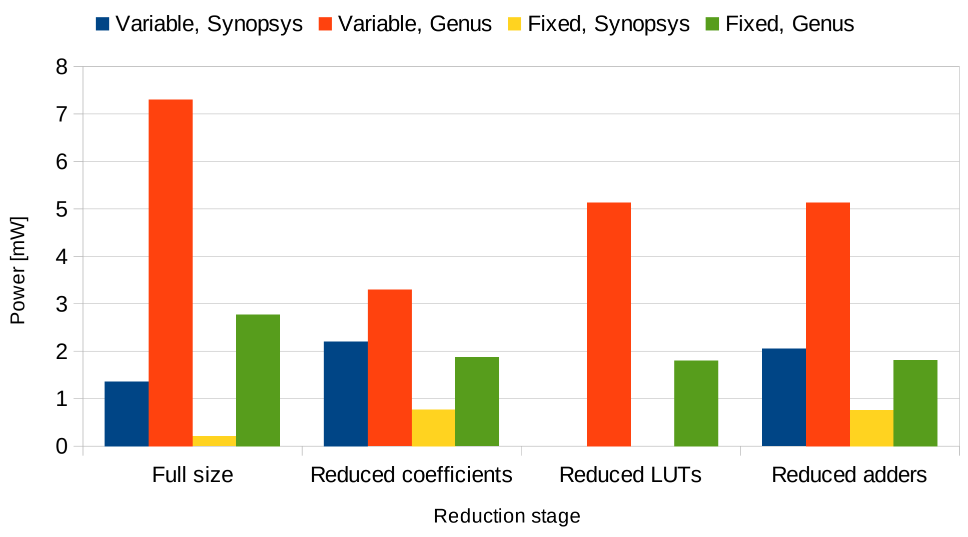

Figure 7.3: Design reduction stage estimation, post synthesis PrimeTime power estimation.

7.2.2 Synthesis program estimation

For the configuration 6.7 with full size coefficients and a LUT select input width of 4 bit synthesis runs with Synopsys Design Compiler and Cadence Genus have been carried out. This was done in order to find out, which synthesis program should be favoured for further tests. Figure 7.4 shows the estimated area. The area from Cadence Genus is about 33% larger than the one from Synopsys Design Compiler. For the power estimation, which can be seen in figure 7.5 the difference is even greater, with Genus estimating several times the power advertised by Design Compiler. The power estimation with Synopsys PrimeTime (figure 7.6) also shows a massively increased power consumption. But these results must be looked at with some restraint, since there are two rather strange values included for the synthesis with Genus. The system with $N = 3$ shows a quite low power consumption, when compared to the system with $N = 4$ and $N = 5$. Furthermore the system with $N = 6$ shows more than triple the power consumption than the system with $N = 7$, which is larger and should have slightly higher power consumption. Both Design Compiler and Genus do not manage to synthesise all systems in a way that the targeted SNR/ENOB is reached. The values can be seen in table 7.2. For the systems with the expected SNR/ENOB the single calculated estimation values also did not fully correspond to the values from the cbadc Python high level simulation.

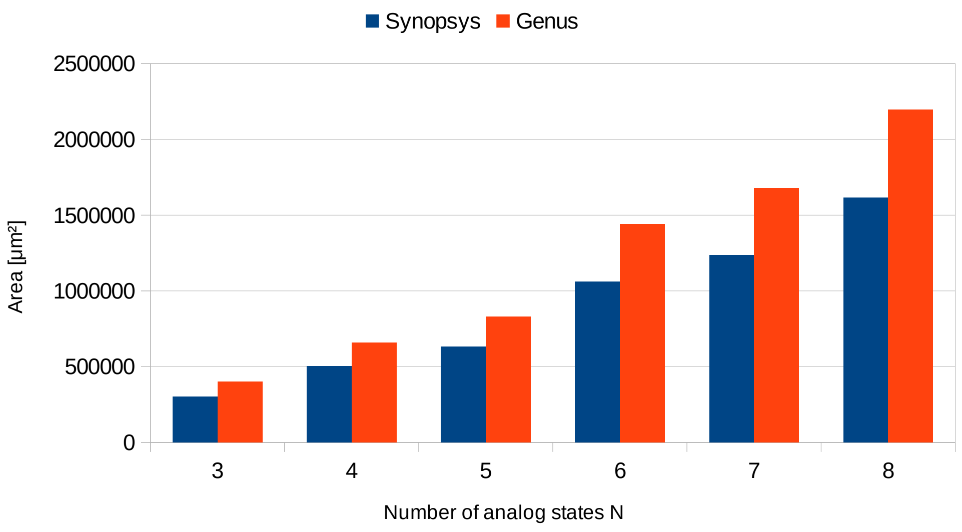

Figure 7.4: Synopsys Design Compiler vs Cadence Genus area estimation for configurations 6.7.

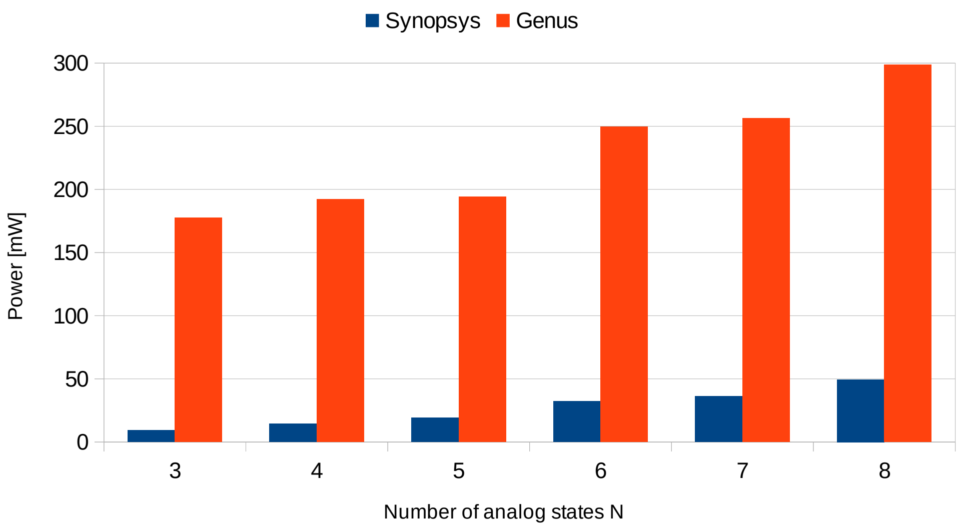

Figure 7.5: Synopsys Design Compiler vs Cadence Genus power estimation for configurations 6.7.

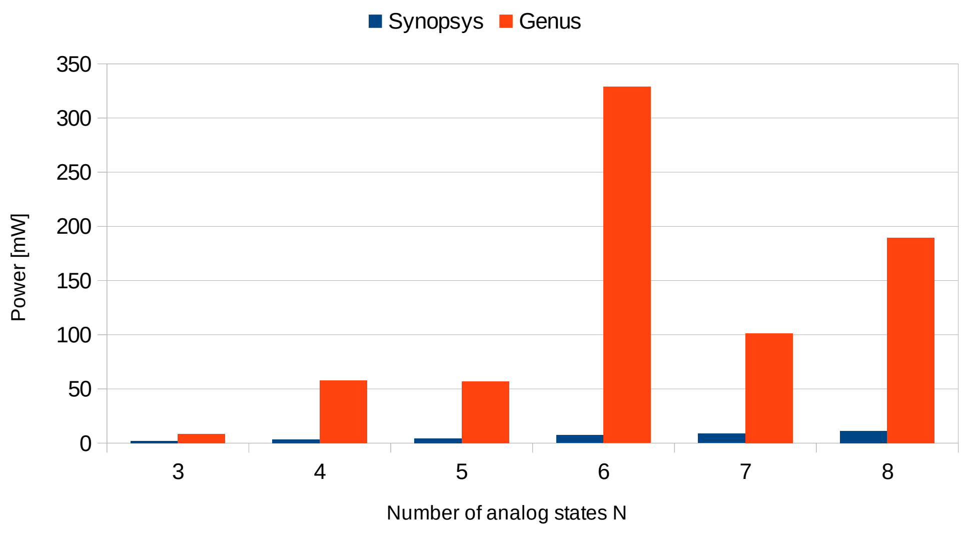

Figure 7.6: Synopsys PrimeTime power estimation, with Synopsys Design Compiler and Cadence Genus synthesised designs for configurations 6.7.

	Synopsys Design Compiler		Cadence Genus	
Number of analog states N	**SNR [dB]**	**ENOB [Bit]**	**SNR [dB]**	**ENOB [Bit]**
3	15,64	2,31	24,82	3,83
4	10,95	1,53	69,19	11,20
5	71,94	11,66	16,78	2,5
6	14,83	2,17	68,28	11,05
7	69,82	11,31	69,82	11,31
8	69,08	11,18	69,08	11,18

Table 7.2: Gate level SNR/ENOB values for configurations 6.7 synthesised with Synopsys Design Compiler and Cadence Genus.

The synthesis results suggest that Synopsys Design Compiler should be used for synthesis, since it generally produced better results for the synthesised designs and less inconsistencies than Cadence Genus. Furthermore the execution time of Design Compiler is lower than the one for Genus. Therefore the majority of synthesis runs have been performed with Synopsys Design Compiler.

7.2.3 Variable coefficient implementations

In this section the synthesis results for variable coefficient implementations are shown. As described in chapter 6 the following four test sets were used 6.7, 6.8, 6.9 and 6.10. The results for the estimated area from Design Compiler can be seen in graph 7.7. The two configurations 6.7 and 6.8 with a LUT select input width of 4 bit show the following area values. For the configuration with N = 3 (smallest configuration), the area reduction is around 33%. This factor increases with increasing number of analog states N (rising system size) up to nearly 50% for the configuration with N = 8. The two configurations 6.9 and 6.10 with a LUT select input width of 2 bit show the following area values. For the configuration with N = 3 (smallest configuration), the area reduction is around 40%. This factor increases with increasing number of analog states N (rising system size) up to around 50% for the configuration with N = 8. When comparing the configurations with a LUT select input width of 4 bit to the system with a width of 2 bit, the used area generally is around 31% lower for the systems with 2 bit. For both resource reduced implementation sets the clearly visible stepping of the area, when the number of analog states N is increased or the lookback/lookahead length is raised, is reduced. The used area in the reduced filter version therefore has a much more linear behaviour with design size increases.

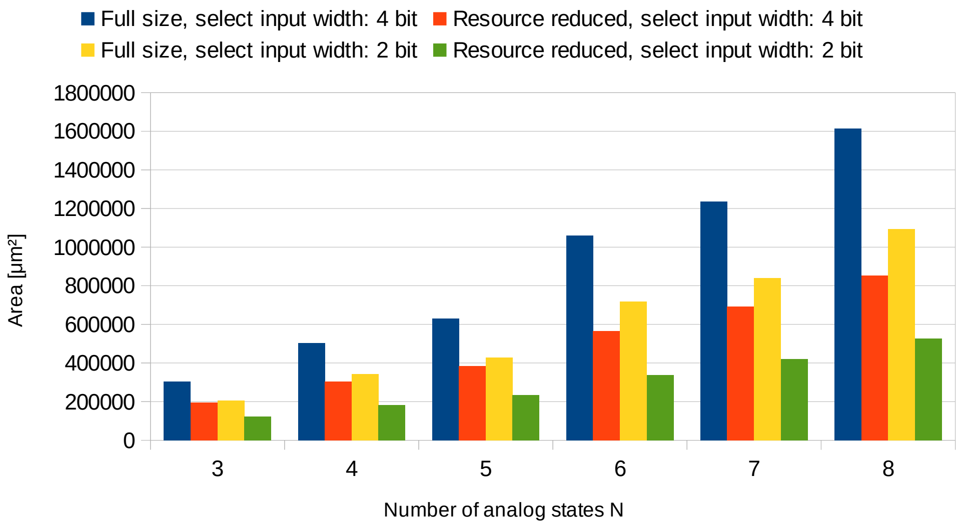

Figure 7.7: Synopsys Design Compiler area estimation for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10.

The estimated power from Synopsys Design compiler is shown in figure 7.8. This power estimation shows a general reduction by 50% for all designs from full size to resource reduced. The accuracy of this estimation however is low, since static activity factors are assumed for all system signals, like explained in chapter 5. For this reason also power estimations with Synopsys PrimeTime have been performed, using the correct activity factors obtained from post synthesis simulations. These results can be seen in figure 7.9. For configurations with a LUT select input width of 4 bit unfortunately the power estimation shows that the power reduction is not nearly as great as the area reduction for resource reduced designs. For the smallest system (N = 3) the power is nearly identically, for larger designs (N = {6; 7}), the power is reduced by around 19%. For the largest system (N = 8) the simulation could not be run successfully at all, therefore a power estimation with PrimeTime was not possible. For configurations with a LUT select input width of 2 bit the resource reduced configurations actually have a higher estimated power consumption than the standard full size implementations. The increase is between 19% and 50%.

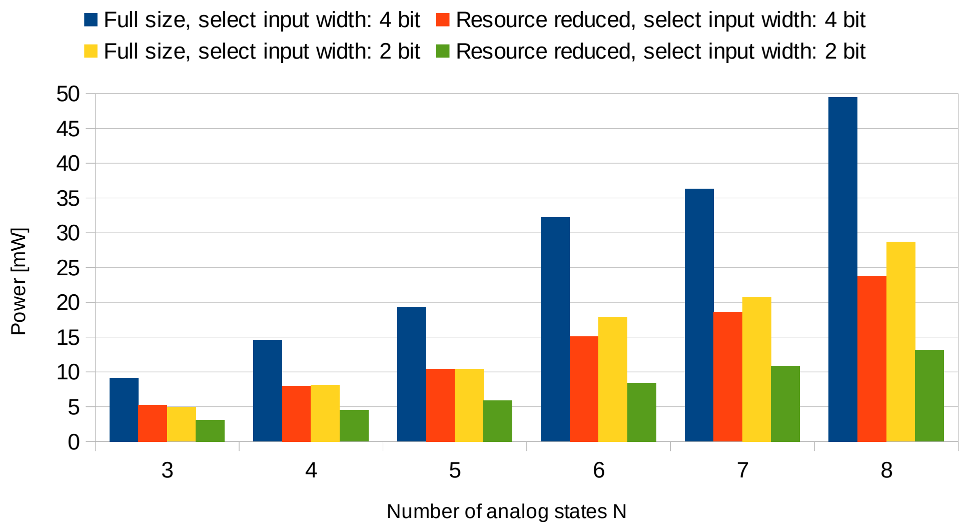

Figure 7.8: Synopsys Design Compiler power estimation for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10.

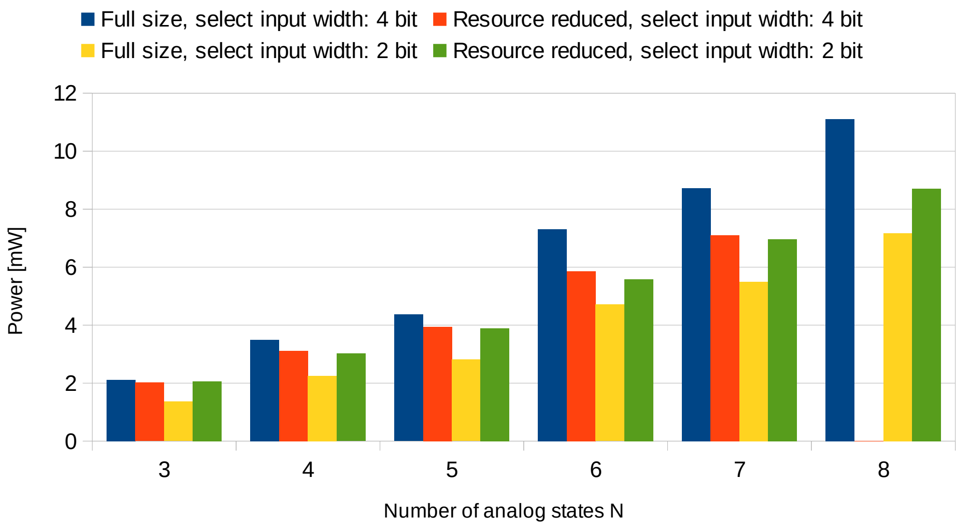

Figure 7.9: Synopsys PrimeTime power estimation, with Synopsys Design Compiler synthesised designs for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10.

When looking at the reached SNR/ENOB values obtained from gate level simulations of the synthesised designs, it can be seen that not all designs reach the intended target. As mentioned before the simulation for the resource reduced design with a LUT select input width of 4 bit and N = 8 could not be run successfully at all. The values can be seen in table 7.3

	LUT select input width = 4 bit		**LUT select input width = 2 bit**	
Number of analog states N	**Full size SNR [dB]**	**Reduced SNR [dB]**	**Full size SNR [dB]**	**Reduced SNR [dB]**
3	15,64	24,68	68,91	68,91
4	10,95	69,19	69,19	69,19
5	71,94	71,94	71,94	71,94
6	14,83	15,62	68,29	68,28
7	69,82	69,82	69,82	69,82
8	69,08	X	69,08	13,4

Table 7.3: Gate level SNR/ENOB values for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10 synthesised with Synopsys Design Compiler.

7.2.4 Fixed coefficient implementations

In this section the synthesis results for fixed coefficient implementations are shown. As described in chapter 6 the following four test sets were used 6.11, 6.12, 6.13 and 6.14. The results for the estimated area from Design Compiler can be seen in graph 7.10. The two configurations 6.11 and 6.12 with a LUT select input width of 4 bit show the following area values. For the configuration with N = 3 (smallest configuration), the area reduction is around 8%. This factor increases with increasing number of analog states N (rising system size) up to nearly 17% for the configuration with N = 8. The two configurations 6.13 and 6.14 with a LUT select input width of 2 bit show the following area values. For the configuration with N = 3 (smallest configuration), the area reduction is around 15%. This factor increases with increasing number of analog states N (rising system size) up to around 24% for the configuration with N = 8. When comparing the configurations with a LUT select input width of 2 bit to the system with a width of 4 bit, the used area is around 40% higher for the standard implementations and around 30% higher for the resource reduced implementations.

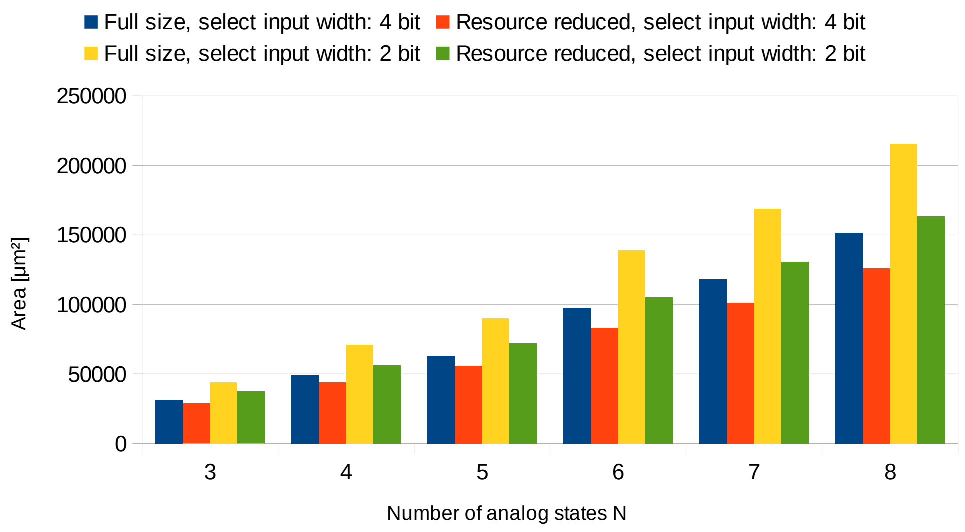

Figure 7.10: Synopsys Design Compiler area estimation for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14.

The estimated power from Synopsys Design compiler is shown in figure 7.11. This power estimation shows a general increase by around 50% for all designs from full size to resource reduced. The accuracy of this estimation however is again low, since still static activity factors are assumed for all system signals. For this reason also power estimations with Synopsys PrimeTime have been performed, using the correct activity factors obtained from post synthesis simulations. These results can be seen in figure 7.12. For configurations with a LUT select input width of 4 bit unfortunately the estimated power for the resource reduced implementations is more than tripled in comparison with the standard implementation. For the systems with a LUT select input width of 2 bit around twice the power can be seen. So at this stage, while the area usage could be reduced, no configuration for a fixed coefficient filter implementation has shown a reduction in power, instead it went up quite significantly.

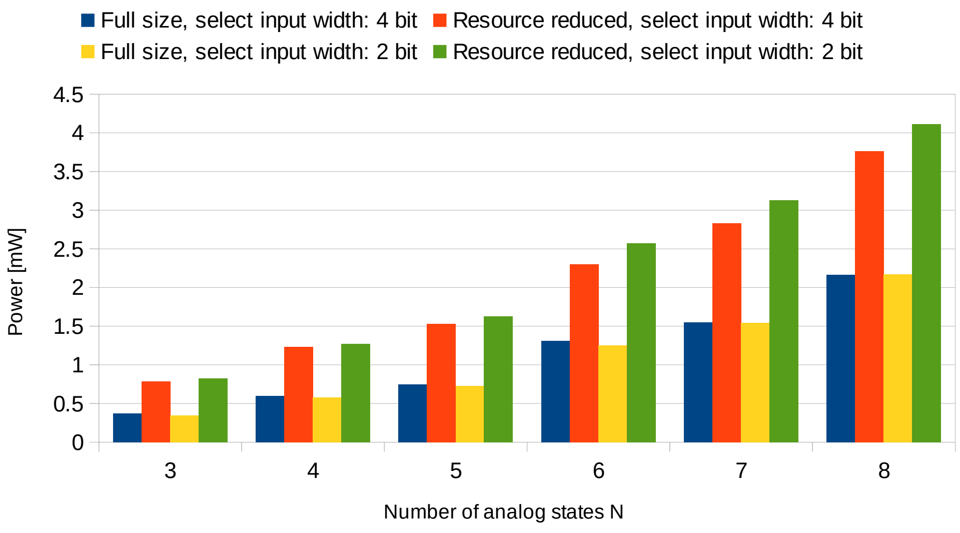

Figure 7.11: Synopsys Design Compiler power estimation for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14.

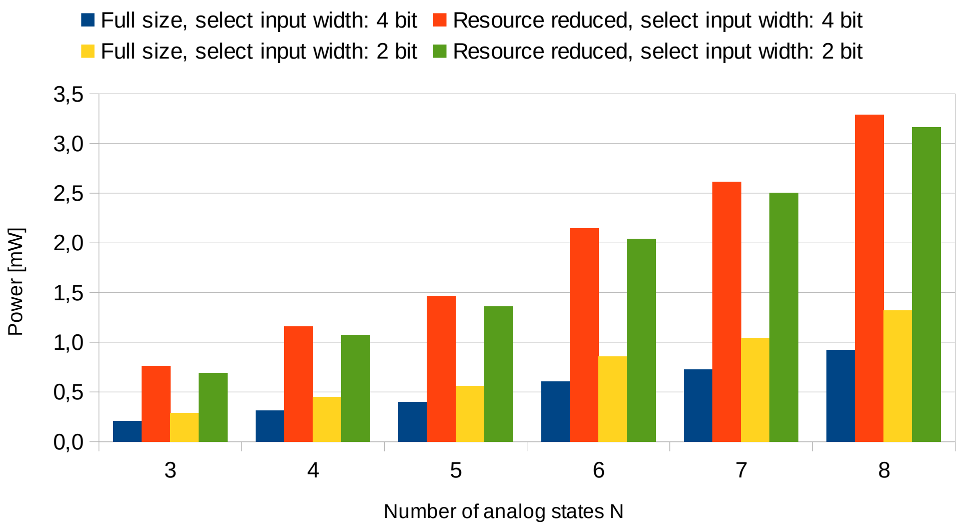

Figure 7.12: Synopsys PrimeTime power estimation, with Synopsys Design Compiler synthesised designs for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14.

When looking at the reached SNR/ENOB values obtained from gate level simulations of the synthesised designs, it can be seen that only one design (resource reduced, N = 8, LUT select input width = 4 bit) reaches the intended target. Somehow Synopsys Design Compiler had great problems with successfully synthesising the apart from the coefficient register to the variable coefficient functionally equivalent fixed coefficient designs. All obtained values can be seen in table 7.4. For the standard full size designs with N = 8, the simulations did not lead to any digital estimation at all.

	LUT select input width = 4 bit		**LUT select input width = 2 bit**	
Number of analog states N	**Full size SNR [dB]**	**Reduced SNR [dB]**	**Full size SNR [dB]**	**Reduced SNR [dB]**
3	18,44	12,64	15,09	15,39
4	8,92	34,48	11,18	10,75
5	11,1	30,64	12,03	12,55
6	2,73	6,86	6,25	17,07
7	0,19	14,78	1,72	7,08
8	X	69,07	X	54,58

Table 7.4: Gate level SNR/ENOB values for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14 synthesised with Synopsys Design Compiler.

7.3 P&R and post P&R simulations

In this section the results from the the carried out P&R runs following the design synthesis are presented. As for the synthesis results the looked at parameters are the estimated area usage and power consumption of the systems and the reached SNR/ENOB values of the different design configurations. The estimation of the design reduction stages and for which synthesis program to use continued with the evaluation of the P&R results. After these tests, again first the results for the variable coefficient and then for the fixed coefficient implementations are shown.

7.3.1 Design reduction stage estimation

In this section the influence of the different design reduction stages on the area and power consumption in the final design stage is shown. Figure 7.13 shows the area obtained from P&R for the configurations 6.5. For the variable coefficient system the results from synthesis are confirmed. A big decrease in used area can be seen, when reducing the coefficients. Going further and reducing also the LUTs and adders again only makes a marginal change for the systems. The used area for the fixed coefficient versions is about the same for every reduction stage. Furthermore the results from Design Compiler and Genus systems are very close, only showing very small differences.

The P&R power estimation can be seen in figure 7.14. As with the synthesis results, the estimated power for the variable coefficient systems show a small decrease between the full size and the full reduction of the system. The Synopsys Design Compiler glitch in the estimated power for the LUT reduction stage vanished for this simulation. The fixed coefficient systems synthesised with Design Compiler show a small decrease in power. The systems synthesised with Genus show a small increase in power consumption. Going to the PrimeTime power estimations (figure 7.3), the observations for the power consumption changed again. As for the synthesis results, the estimation for the variable coefficient version from Design Compiler now advertises an increase in power. Furthermore the fixed version synthesised with Genus now shows no significant change in power for all stages. For both systems synthesised with Design Compiler on the LUT reduction stage the glitch returned and they could not be simulated, therefore no PrimeTime power estimation was possible. Again the results are inconclusive, if enabling a reduction stage ultimately leads to a power reduction. But if one of the stages is enabled, the results mostly do not differ significantly. For this reason when following systems are described as resource reduced, all reduction stages have been activated.

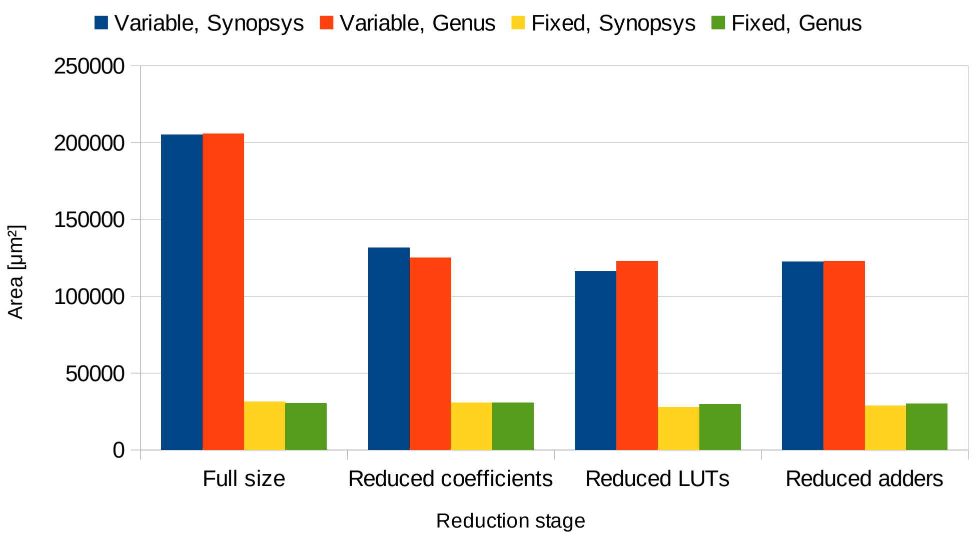

Figure 7.13: Design reduction stage estimation, P&R area.

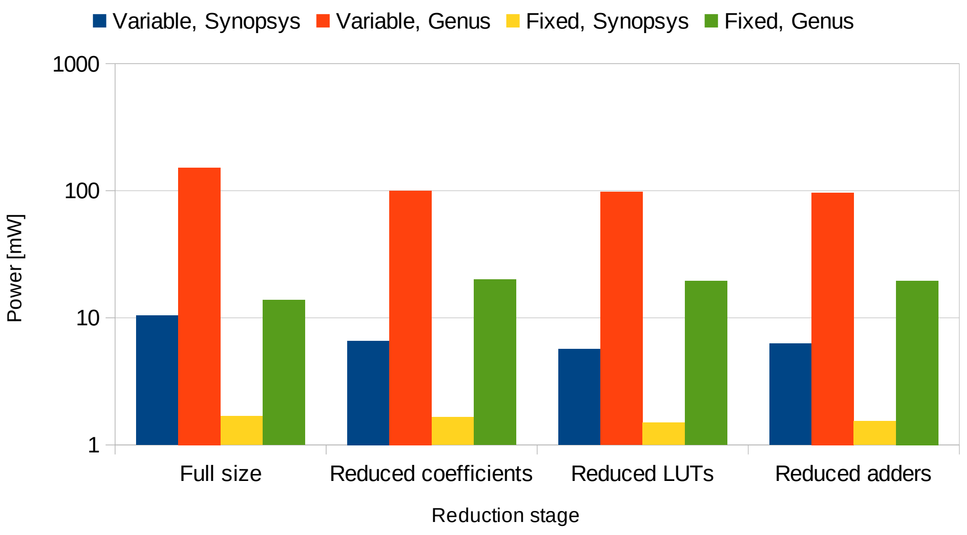

Figure 7.14: Design reduction stage estimation, P&R power.

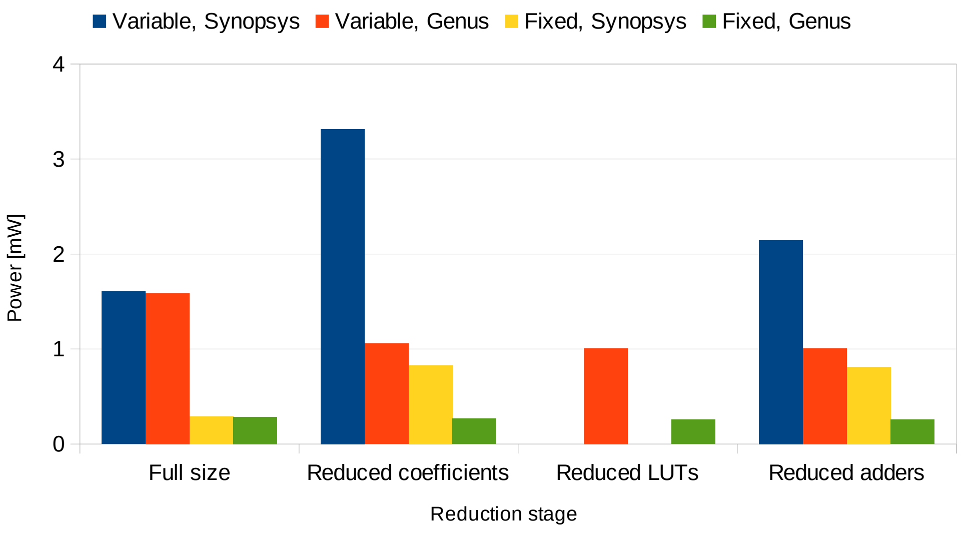

Figure 7.15: Design reduction stage estimation, post P&R PrimeTime power estimation.

7.3.2 Post P&R synthesis program estimation

The previously with Synopsys Design Compiler and Cadence Genus synthesised designs 6.7 and 6.8 are now run through P&R with Cadence Innovus. This was done in order to find out, which synthesis program should be favoured for further tests. Figure 7.16 shows the estimated area. Whereas the area for Cadence Genus designs was about 33% larger than the ones from Synopsys Design Compiler after synthesis, now the results only differ marginally. For the power estimation, which can be seen in figure 7.17 the difference still is very large, with Genus estimating several times the power advertised by Design Compiler. The power estimation with Synopsys PrimeTime (figure 7.18) now shows nearly the same power consumption for both synthesis programs. This differs greatly from the post synthesis simulations, where Genus showed a far greater power consumption. Also the two rather strange values included in the results from Genus have now vanished. The SNR calculations for Genus implementations show very similar values to the ones from the post synthesis simulations. When looking at the SNR calculation results for Synopsys Design Compiler implementations, the values now are at the expected level, even if they have been too low at the post synthesis simulations. This is a strange behaviour, which should be investigated further. All values can be seen in table 7.5. Furthermore, for the systems with the expected SNR/ENOB the single calculated estimation values still did not fully correspond to the values from the cbadc Python high level simulation or the post synthesis simulations.

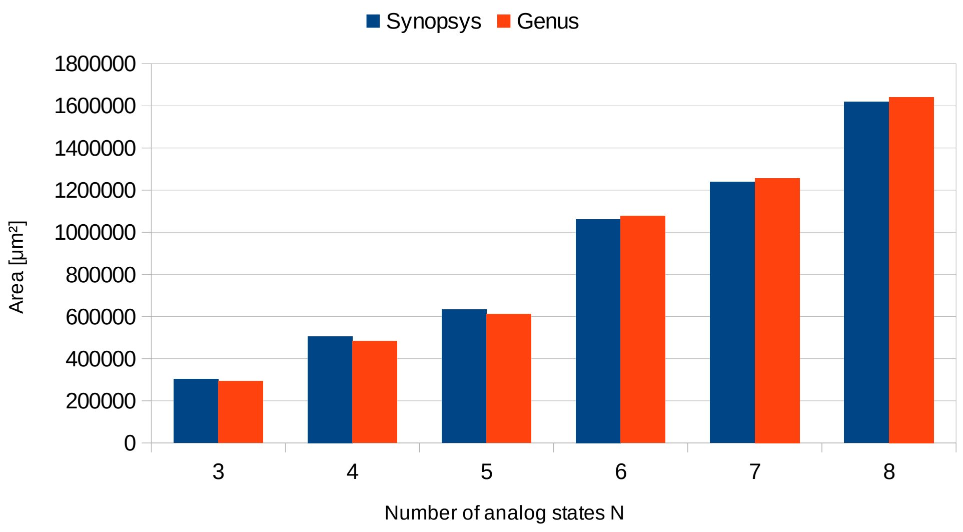

Figure 7.16: Synopsys Design Compiler vs Cadence Genus, P&R Cadence Innovus area estimation for configurations 6.7.

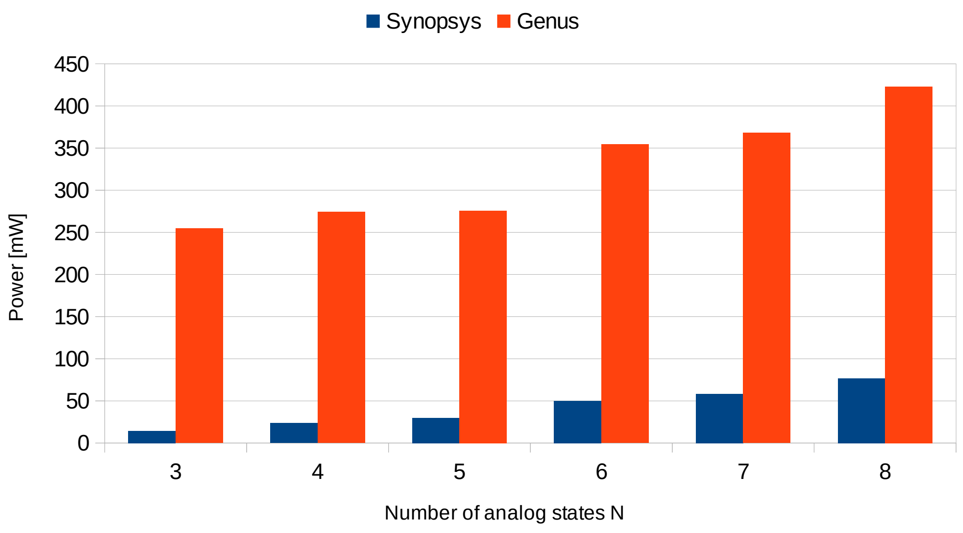

Figure 7.17: Synopsys Design Compiler vs Cadence Genus, P&R Cadence Innovus power estimation for configurations 6.7.

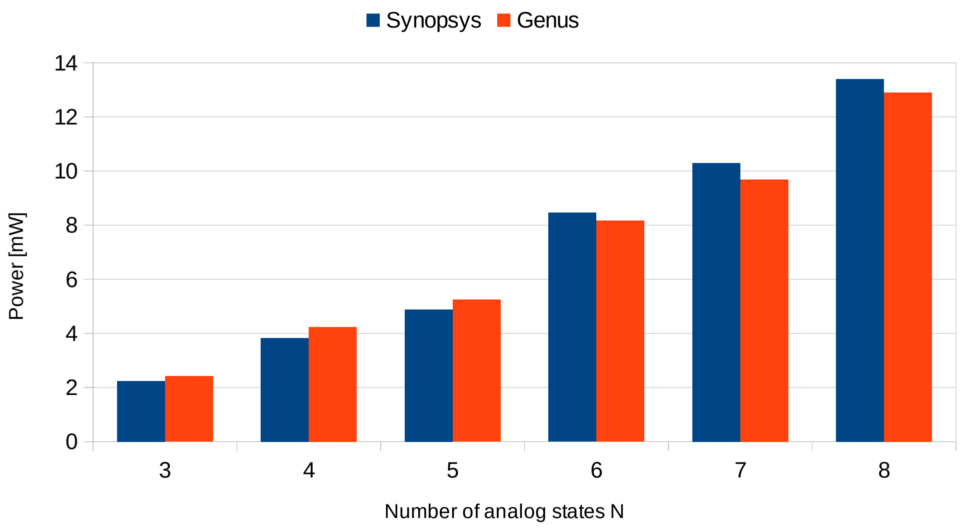

Figure 7.18: Synopsys PrimeTime power estimation, with Synopsys Design Compiler and Cadence Genus synthesised designs, P&R Cadence Innovus for configurations 6.7.

	Synopsys Design Compiler		**Cadence Genus**	
Number of analog states N	**SNR [dB]**	**ENOB [Bit]**	**SNR [dB]**	**ENOB [Bit]**
3	68,91	11,15	24,62	3,8
4	69,2	11,2	69,17	11,2
5	71,69	11,62	16,66	2,48
6	67,99	11	68,57	11,1
7	69,95	11,33	69,93	11,32
8	X	X	69	11,17

Table 7.5: Post P&R SNR/ENOB values for configurations 6.7 synthesised with Synopsys Design Compiler and Cadence Genus.

The P&R results do not lead to such a clear choice like the synthesis results. However, Cadence genus did produce much larger designs in synthesis, taking a longer time for the synthesis and the SNR values of Design Compiler implementations mostly have been correct in post P&R simulations. Therefore Design Compiler has been chosen as synthesis program and the majority of synthesis runs have been performed with Synopsys Design Compiler.

7.3.3 Variable coefficient implementations

In this section the P&R results for variable coefficient implementations are shown. The synthesis results from Synopsys Design Compiler for the configuration sets 6.7, 6.8, 6.9 and 6.10 were used as input for P&R. As an addition to the graphs, the resulting absolute area and PrimeTime power estimation values for these configuration can be found in appendix D.

The results for the estimated area from Design Compiler can be seen in graph 7.19. The estimated area is nearly the same as the one from the synthesis. Therefore the observations are the same.

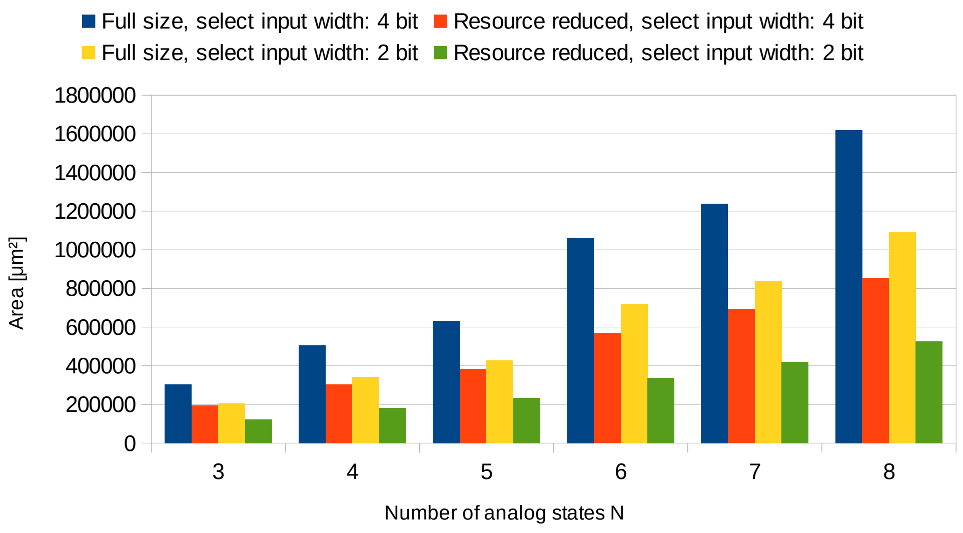

Figure 7.19: Cadence Innovus area estimation for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10 synthesised with Synopsys Design Compiler.

The estimated power from Cadence Innovus is shown in figure 7.20. Also this power estimation shows mostly the same behaviour as the synthesis power estimation. The results from the PrimeTime power estimation can be seen in figure 7.21. The results for the systems with a LUT select input width of 4 bit show the same behaviour as within the synthesis results except for being slightly higher. The more interesting results can be seen with the systems with a LUT select input width of 2 bit. For the synthesis estimation, the resource reduced designs generally showed a rise in power consumption compared to the normal ones. This has now changed for systems with a higher number of analog states N. There a small reduction of power could be achieved.

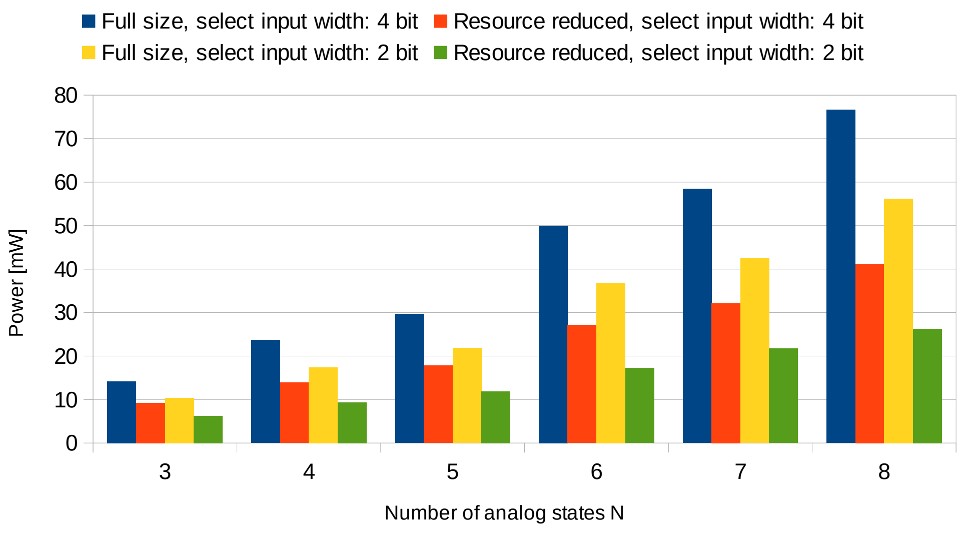

Figure 7.20: Cadence Innovus power estimation for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10 synthesised with Synopsys Design Compiler.

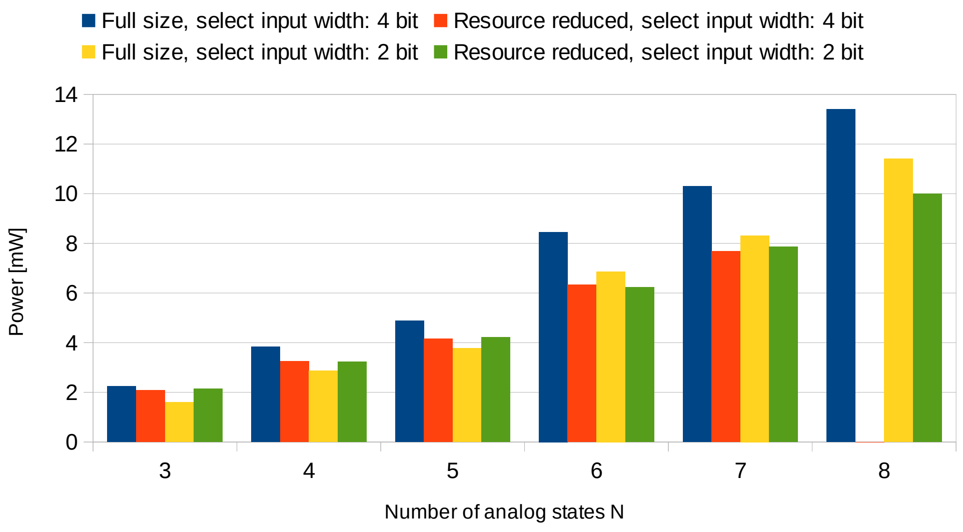

Figure 7.21: Synopsys PrimeTime power estimation for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10, synthesis with Synopsys Design Compiler, P&R with Cadence Innovus.

When looking at the reached SNR/ENOB values obtained from post P&R simulations, an interesting change can be observed. Most of the estimated values now reach the intended SNR. The only systems, which could not be simulated are the full size and resource reduced designs with a LUT select input width of 4 bit and

N = 8. All values can be seen in table 7.6

	LUT select input width = 4 bit		**LUT select input width = 2 bit**	
Number of analog states N	**Full size SNR [dB]**	**Reduced SNR [dB]**	**Full size SNR [dB]**	**Reduced SNR [dB]**
3	68,91	68,91	68,91	68,91
4	69,2	69,19	69,19	69,2
5	71,69	71,94	71,94	71,94
6	67,99	67,99	67,99	67,99
7	69,95	69,82	69,82	69,82
8	X	X	69,08	69,03

Table 7.6: Post P&R SNR values for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10, synthesis with Synopsys Design Compiler, P&R with Cadence Innovus.

7.3.4 Fixed coefficient implementations

In this section the P&R results for fixed coefficient implementations are shown. The synthesis results from Synopsys Design Compiler for the configuration sets 6.11, 6.12, 6.13 and 6.14 were used as input for P&R. As an addition to the graphs, the resulting absolute area and PrimeTime power estimation values for these configuration can be found in appendix D.

The results for the estimated area from Design Compiler can be seen in graph 7.22. The estimated area is nearly the same as the one from the synthesis. Therefore the observations are the same.

The estimated power from Cadence Innovus is shown in figure 7.23. Compared to the synthesis results, the estimated power from P&R now shows a reduction for the resource reduced designs. The power estimation results from PrimeTime can be seen in figure 7.24. Here the resource reduced designs show an increase in power consumption compared to the full size implementations. However the increase in power usage is not as high as estimated with the synthesised design. The full size systems with N = 8 show some sort of error, since the estimated power is to low compared to other systems and the results should therefore not be regarded valid.

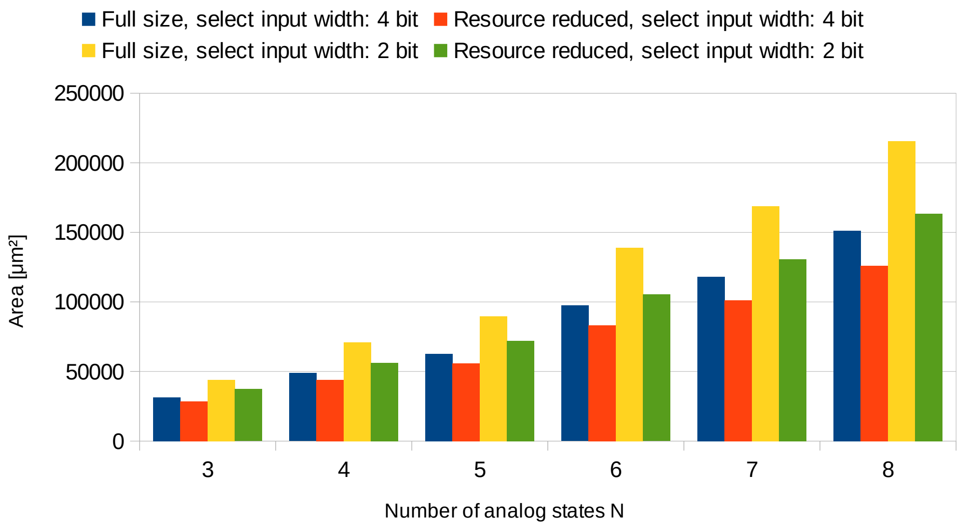

Figure 7.22: Cadence Innovus area estimation for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14 synthesised with Synopsys Design Compiler.

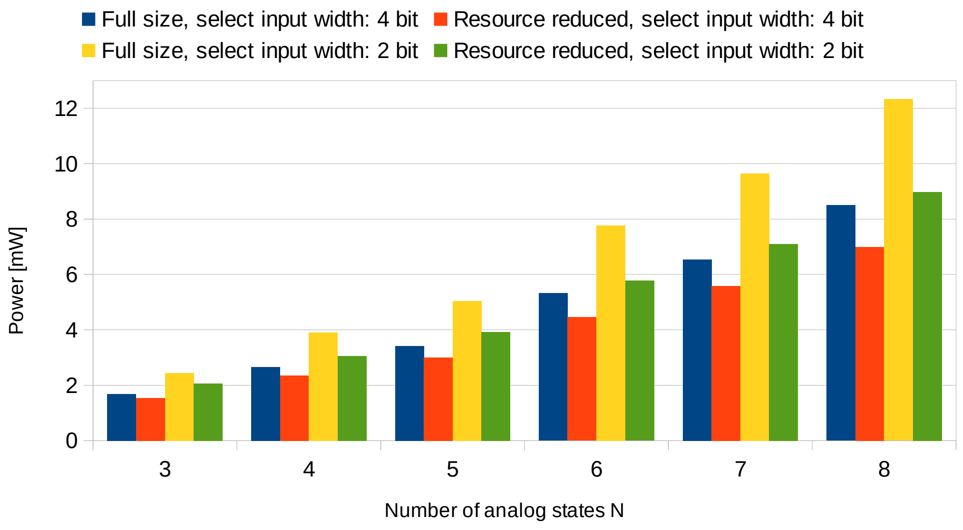

Figure 7.23: Cadence Innovus power estimation for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14 synthesised with Synopsys Design Compiler.

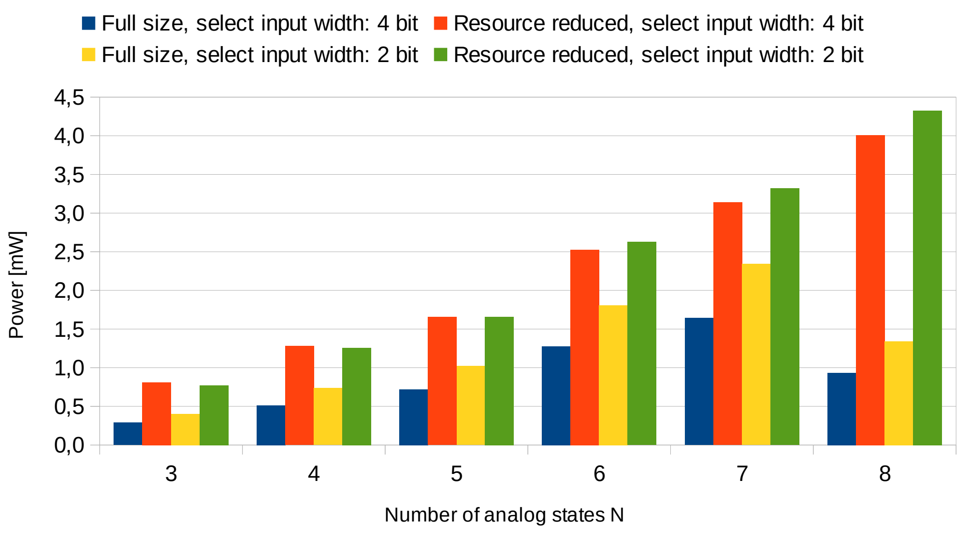

Figure 7.24: Synopsys PrimeTime power estimation for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14, synthesis with Synopsys Design Compiler, P&R with Cadence Innovus.

When looking at the reached SNR/ENOB values obtained from post P&R simulations, the same change as for variable coefficient simulations can be observed. Most of the estimated values now reach the intended SNR. And again the only systems, which could not be simulated correctly, are the full size and resource reduced designs with a LUT select input width of 4 bit and N = 8. All values can be seen in table 7.7.

	LUT select input width = 4 bit		**LUT select input width = 2 bit**	
Number of analog states N	**Full size SNR [dB]**	**Reduced SNR [dB]**	**Full size SNR [dB]**	**Reduced SNR [dB]**
3	68,91	68,91	68,91	68,91
4	69,2	69,2	69,2	69,2
5	71,69	71,68	71,68	71,69
6	67,99	67,99	67,99	67,99
7	69,95	69,95	69,95	69,95
8	X	69,02	X	69,03

Table 7.7: Post P&R SNR values for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14, synthesis with Synopsys Design Compiler, P&R with Cadence Innovus.

Chapter 8

Discussion and comparison

In this chapter the results from chapter 7 are discussed and compared to the results from the RISC-V implementation [4].

The RTL simulation results show that all system implementations behave as expected and according to the cbadc Python package high level simulation. The coefficient configuration, the number of analog states N, the filter lookback/lookahead length and the design reduction stage had no influence on the outcome of the design simulation at this stage. However this changes in later design stages, at this point the design can be considered valid.

When looking at the design reduction stage estimation, some interesting trends can be seen. The area usage results suggest that it does not matter, which stage of reduction was chosen, the area is always very similar. For variable coefficient implementations a lower area usage compared to the full size version can be seen, not so for the fixed coefficient version. The results therefore match the expected behaviour from the theoretical considerations, regarding the coefficient shift register and is overall portion in the design. Unfortunately the reduction of the LUTs and adders seems to not have a significant influence on the used area. Neither in variable nor in fixed coefficient systems. Especially in fixed coefficient systems the concept of fitting every LUT and adder component to the matching coefficient size could have made a difference for area usage, since those are the largest parts in these designs. This outcome can be an indication for several things. First the synthesis programs could be so advanced that they automatically recognise these possibilities for reducing the bitwidth in the LUTs and adders. Furthermore the reduced LUTs and adders might introduce additional design challenges due to the used number format. All designs use fixed point numbers in the two's complement form. Adding together two positive numbers is easy (the values are the same as in the normal binary form), since non occupied higher place are just zero. It therefore is possible to add together two bitwise different sized numbers without any conversion and still get the correct result. However, the situation is different for negative numbers. When two bitwise different sized numbers are to be added and the smaller number is negative, this number needs a padding with ones to be the same size as the other number. This imposes a quite hardware expensive problem, since padding the number with ones must only be done, when the number is negative. A circuit therefore has to decide if the number needs padding and apply it accordingly. When now thinking about the reduction of the LUTs, this means that these might have less internal gates, but all need a padding circuit to account for negative numbers. When also matching the adders to the LUT output widths there have to be some similar considerations to be made. The LUT outputs now might not need a padding circuit, but the adder inputs might not have the same bitwidth. Then these need number padding circuits to modify the incoming signals accordingly. Also the fact that the output of the bitwidth reduced adders has a carry bit makes the internal logic more complicated. Within the full size adders no carry bit was implemented, since overflows are not possible in normal system operation.

When looking at the more accurate PrimeTime power estimation the results strongly depends on the used synthesis program. Synopsys Design Compiler advertises a rise in power for both systems, while Cadence Genus predicts a reduction in power for the variable coefficient system and a consistent power consumption for the fixed system. The differences can be quite large, up to 300%, while the area is nearly the same for corresponding implementations. Also the changes between synthesis and P&R estimations are interesting to observe. Especially the massive reduction in estimated area and power from Genus. These factors reduce the confidence in the power estimations, even if they have been done with extracted activity factors and design parasitics.

For the estimation of the synthesis programs, the previous findings are validated. During synthesis Genus produces much larger and more power hungry systems than Design Compiler. After P&R the differences in area and for the PrimeTime power estimation are only marginal. However the estimated power from Innovus shows a huge difference between Design Compiler and Genus, even if the designs nearly have the same size. But these are not the only concerns regarding both of synthesis programs. When looking at the reached SNR values, the resulting systems from Genus show similar behaviour between post synthesis and post P&R simulations. Not all systems could produce the targeted SNR, but if they did not reach this value after synthesis, they also did not reach the value after P&R. For designs from Design Compiler the situation is different. If a simulation could be carried out successfully after synthesis, the system did reach the targeted SNR value after P&R. This leaves open several possible explanations for this behaviour. First Design Compiler might have had problems with the correct timing of the signals and Innovus was able to fix these problems during P&R with the physical cell design. Second the seemingly good design verified in RTL simulations has design elements, which are hard to synthesise correctly. Especially the clock dividing module is a candidate for this assumption. During the design phase it has been changed or slightly altered multiple times, in order to improve the system performance. The original design featured counting of both incoming clock edges. This was done to allow the downsampled clock to always have a 50% duty cycle for all configurations. However this lead to problems during synthesis and the system was therefore changed so that only full clock cycles are counted on the rising clock edges. Systems with an uneven down sample rate now feature an asymmetrical clock. But this does not lead to a change in the estimation (at least on RTL level), since the downsampled clock edge can be placed at the correct clock cycle to still match the cbadc simulation.

Going to the actual test sets, some theoretical assumptions can be confirmed. For the variable coefficient implementation the a major design aspect for producing small systems is to keep the coefficient shift register small. Therefore all configurations with a LUT select input width of 2 bit performed significantly better for area usage and at least on par with their 4 bit counterparts, since the coefficient register always is smaller by design. The increased number of LUTs and adders does not outweigh this advantage. For the resource reduction of the system, the outcome however is not so clear. While the area could be reduced (with the main part contributed by the reduction of the coefficient shift register), the power did rise for the systems with N = 3, 4, 5 and a LUT input width of 2 bit. This could be due to before mentioned higher design challenges for the LUTs and adders in these systems. Unfortunately not even the most power efficient system (N = 3, LUT select input width = 2 bit, full size coefficients) could reach the intended power target of 0,4mW. With an estimated power consumption of 1,6mW it is overshooting the target by 300%. Another interesting aspect of the systems with a LUT select input width of 2 bit is the fact that nearly all systems had a correct SNR after synthesis. These results have been carried over to the post P&R simulations. Therefore it is likely that the correct timing for smaller systems is easier to implement than for larger systems.

Looking at the systems with a fixed coefficient setup, the systems with a LUT select input width of 4 bit did perform better than their 2 bit counterparts. These results validate another assumption made earlier. A lower number of LUTs with more inputs and a therefore lower number of adders does pay of for fixed coefficient systems in terms of area usage and power consumption. When looking at the usefulness of the design resource reduction, the outcome however is not so promising. While the area usage could be reduced, the estimated power consumption does rise for all investigated systems. The design challenges as explained for the reduction stage estimation of course also apply here. The resulting reduced systems may have fewer gates, but an overall higher switching activity due to number format calculations, resulting in the observed rise in power consumption. And since there is no coefficient register for fixed coefficient systems, which could outweigh the negative effects of a higher switching activity, the manual reduction of the LUTs and adders does not propose a good design choice for fixed coefficient systems. But the power estimation also shows a desirable results. Two systems (full size, $N = 3$) did manage to meet the intended power target of 0,4mW. A concerning aspect of the synthesis results however is the fact that only one system showed the correct SNR. After P&R the systems, except for full size systems with $N = 8$, showed the correct SNR values again. This is a rather strange result, since these systems are nearly identical to their variable coefficient counterparts, of which the majority showed the expected SNR values. The LUT and adder setups are exactly the same, only the presence or absence of the coefficient shift register is different. It therefore is a surprising result that the fixed coefficient systems, which are functionally equivalent and much smaller and should be far easier to synthesise and P&R, did not reach the target SNR values on synthesis level. This is a point, which should definitely be investigated further in future research and design refinements. Then these systems have the chance to deliver the wanted performance within the set resource limits.

When comparing variable coefficient and fixed coefficient systems, both show advantages and disadvantages. The resource usage of variable coefficient implementations is far greater. The area is several times higher, the factor lies between 5,7 and 10 times, depending on the compared systems (LUT select input width, full/reduced) with the same number of analog states N. Also the power is several times higher for these systems, with a factor ranging between 4,7 and 7,7. Nevertheless the big advantage of these systems is the possibility to change out the coefficients after production. Since this is one of the big advantages of CBADCs, using fixed coefficients imposes a great restriction on the whole concept. Variable coefficient designs are therefore more suited for scientific and experimenting purposes. On the other hand fixed coefficient designs can reach a power consumption comparable to other state of the art ADC. Since the digital estimation circuit can be produced disjoint from the analog system, an idea to overcome these restrictions and get the best of both worlds might be the production of a variable coefficient tuning and a fixed coefficient final product digital estimator. The tuning circuit has no requirement to have a particular small area or low power consumption. It would be used to tune the coefficients to the given analog system. After the coefficients have been found, a fixed coefficient digital estimator with a PROM can be programmed and integrated with the analog system.

A major design consideration for the analog system and subsequently for the digital estimation is the choice of analog states N (and therefore digital states M). Since all configurations within a test set are supposed to deliver the same overall CBADC system performance, a comparison between them is possible and leads to very interesting findings. Generally, as explained earlier, the number of analog states N and the OSR contribute to the information content of digital control vectors. Therefore systems with a lower number of analog states N need a higher OSR. When using the standard cbadc Python package implementation of the digital estimator reconstruction filter, the filter order corresponds to the number of analog states N. This leads to filters with a much more aggressive out-of-band noise suppression for systems with higher values of N. To achieve this grade of filtering, a higher number of lookback and lookahead coefficients is needed. Therefore the comparison between the systems might not be entirely fair. The question, if systems with higher numbers of analog states N could operate with fewer digital state vectors/coefficients in the lookback and lookahead batches, while retaining the targeted SNR, is a topic for future research. While for the filtering aspect this might be possible, the overall information content in the stored digital state vectors might be too low. These systems have a lower OSR, what reduces the information content. Furthermore the information content of the single bits within a digital state vector are is not equal. Digital state values corresponding to later stages in the analog system component chain do have a lower contribution on the result than earlier ones. So even if the pure number of digital state values is the same for two system, the comprehensive information content might be different. Also the sizes of the digital state vector and coefficient shift registers are not the only design considerations for the power consumption. The OSR defines the digital frequency the digital estimation filter has to operate on. And of course a higher frequency leads to an increased dynamic power consumption, as the equation for dynamic switching power 5.1 proclaims. In theory this should give systems with highers number of analog states and therefore lower OSRs and frequencies an advantage for dynamic power consumption, which counteracts the higher static power. Ideally all system configurations, which are supposed to achieve the same performance (analog bandwidth, SNR), should have the same overall resource usage.

When now looking at the area and power estimation results for systems with the standard filter implementation (which has been used for all test sets), they clearly favour systems with a lower number of analog states N. For all test sets the area and power did rise gradually with a rising number of analog states N. So with these filter configurations the reduced operating frequency was not able to counteract the introduced overhead in static power consumption. But this conclusion leaves out the physical borders of the technology library. Especially the needed digital operating frequency for a given analog bandwidth might introduce severe problems for the synthesis tools. For AS in the kHz range (maybe audio applications) this is not a problem, here high OSRs can be realised. But for the analog bandwidth of 20MHz given as reference for this thesis (table 6.2), this already leads to quite high values for the digital operating frequencies. The systems with $N = 3$ have to run at 920MHz. If the analog bandwidth is supposed to be even higher, it might not be possible any more to synthesise the digital estimator design for the then needed digital operating frequency. For higher bandwidth systems there is therefore a trade-off between system size and required operating frequency. But still, the recommendation, which can be taken from these results, is that the number of analog states N should be kept as small as possible, when it is feasible to synthesise the design for the needed frequency.

These considerations for possibly too high digital operating frequencies are backed up by some synthesis runs and corresponding simulations of the system with N = 3 for different clock periods. The simulation with a system synthesised for 920MHz did not lead to the targeted SNR. However, when reducing the synthesis target frequency at some point the synthesis tool was able to correctly synthesise the digital estimation filter. Leaving the frequency at 920MHz and increasing the supply voltage of the technology library, did not produce a synthesised gate level design with the correct SNR. However this could also be a sign of a not correctly build up clock tree. Also larger systems, especially with N = 8, imposed problems for a correct synthesis, strengthening the clock tree theory. Furthermore a not fully working clock tree could explain the slight differences between the single estimation output values on synthesis and P&R level and the cbadc Python package golden sample simulation. If some values from LUT and adder outputs are delayed, the final result gets marginally changed, while at the same time not reducing the overall accuracy. The clock tree synthesis therefore is an important subject for investigation in further research, especially the downsampled clock from the clock divider module, which is crucial for the correct operation of the digital estimation FIR filter.

In all performance discussions so far only the digital estimator itself was considered. But of course for variable coefficient implementations, there has to be an external memory holding the coefficients. This introduces additional design cost. The overall longterm power consumption of the system should not be influenced too much, since shifting in coefficients is not the normal operation mode. The CBADC will spend much more time estimating values, than changing out coefficients. And during estimation the external memory does not need to be powered on, reducing the power consumption to basically zero.

8.1 Comparison with RISC-V implementations

Andreas Bjørsvik and Sevat Mestvedthagen are working on a RISC-V implementation of the digital estimation filter in their master thesis [4]. Their system of course differs from the implementation proposed in this thesis, but since the intended function is the same, a small comparison between some system implementations is done at this point. The RISC-V implementations are designed with interchangeable coefficients, therefore the comparison is limited to variable coefficient implementations. For the RISC-V systems there are two different implementation options. The first one only features a RISC-V processor, which is doing all the calculations. This system is available as full and reduced size version. The second one employs LUTs in the same way, like it is done in this thesis. Precomputed coefficients get selected by the digital state vectors in the LUTs and the RISC-V processor handles the addition operations for the LUT outputs. For the subsequent comparison the the following implementation option have been chosen. These are the implementations without LUTs in full and reduced size and the LUT version with a select input bit width of 2 bit, which is the best performing test set with LUTs. From this thesis the test sets with a LUT select input width of 2 bit as full and reduced size version have been chosen, since they are the best performing systems from this thesis.

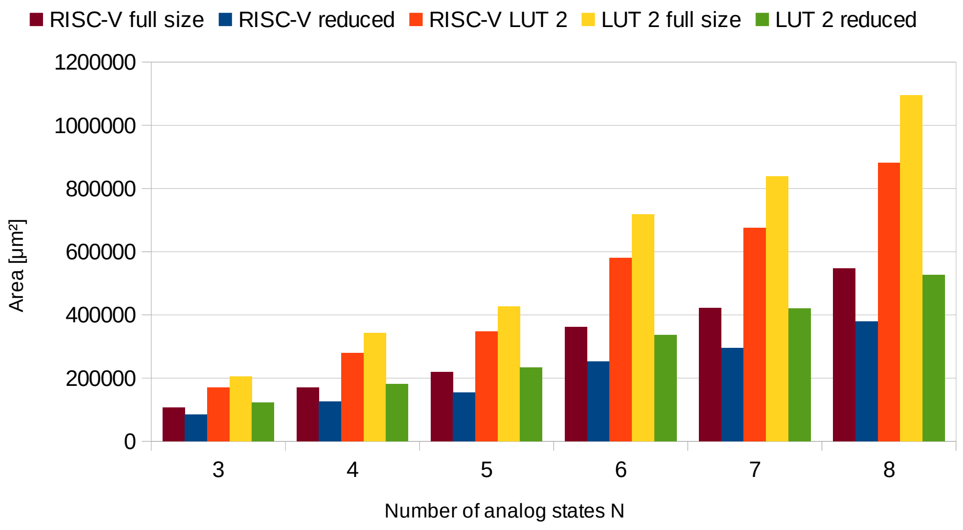

Figure 8.1: Area estimation comparison between RISC-V implementations from [4] and designs proposed in this thesis.

Before looking at the area and power estimation results, there can already be made an observation regarding the LUT setup of the systems. For the RISC-V LUT version as well as for the here proposed implementation, the best performing test set has a LUT select input width of 2 bit. It seems like the previously discussed influence of the LUT select input width on the system performance also applies to the RISC-V LUT implementation.

When looking at the area (figure 8.1), further similarities can be found. Systems with the same number of analog states N have quite similar area values, which are all well within the same order of magnitude. The overall best performing system set is the RISC-V reduced size implementation. This is shortly followed by the RISC-V full size and the resource reduced implementation from this thesis, which show a nearly identical performance. Especially the result for the RISC-V full size implementation is quite impressive, since these systems feature full size coefficients. The best implementation proposed in this thesis with full size coefficients has a bit less than double the area usage. Looking at the two system sets with LUTs and full size coefficients, the here proposed system has a by 21% to 24% increased area usage than the system from [4]. When comparing the two resource reduced test sets, the implementation proposed in this thesis

has a by 39% to 45% increased area usage. So overall the RISC-V systems from [4] have a lower area usage.
Where the in this thesis proposed implementations could not outperform the

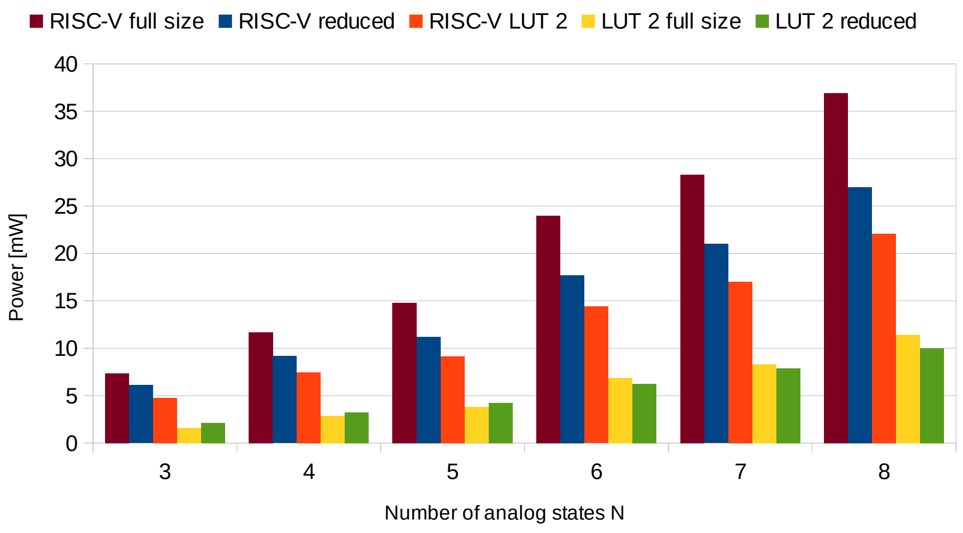

Figure 8.2: Power estimation comparison between RISC-V implementations from [4] and designs proposed in this thesis.

RISC-V implementations from [4], this situation changes for the power consumption. All values have been obtained from PrimeTime simulations, featuring extracted activity factors. The results can be seen in figure 8.2. The power consumption for the here proposed designs is about half to a fourth of the power from RISC-V implementations from [4]. Especially interesting is the circumstance that the RISC-V implementations, which had the best area performances now have the highest power estimation. This looks like a trade-off between area and switching activity for these systems. A possible explanation might be the higher needed operating frequencies of the RISC-V processor compared to the operating frequencies of the here proposed designs for the same filter configuration.

Even though the different system implementations might need some further work, this comparison between the systems gives the indication that no system design has any fatal flaw in it, which leads to a massively increased resource usage, since the area and power estimation values are on the same order of magnitude. This also means that at this point, this comparison does not lead to a final universal conclusion, which system design does perform better. It comes down to the more important design aspect and the usecase of the CBADC. If the area usage is the critical design parameter, RISC-V implementations should be favoured. If the power consumption is the prevalent design consideration, the in this thesis proposed ASIC implementations might give the better performance. And lastly the analog bandwidth might decide which system can be implemented.

Chapter 9

Future work

The research on the topic of control-bounded analog-to-digital converters should continue, since this concept proposes promising solutions to design challenges present in classic analog-to-digital conversion implementations. With the past and currently in progress scientific work on the analog system and the digital estimation, it was possible, to elevate the CBADC concept from purely theoretical descriptions up to simulated implementations. The next desirable step would be a real world implementation of a CBADC, on which physical measurements can be carried out.

Although post P&R simulations have been performed for the implementations of this thesis, they should undergo further verification to ensure their correct operation. One way for enhancing the confidence in an implementation would be the utilisation of inhomogeneous test input signals. Ideally these signals are matched to the specific application scenario of the implementations. Especially the issues with the clock division need an in depth investigation. Without a correctly working clock distribution, the signal estimation will have an unpredictable outcome. The possible failures might range from a slight reduction in SNR to the complete breakdown of the estimation.

The further reduction of the power consumption of the digital estimation should also be a primary goal for future research, since this digital part of CBADC implementations still is the prevalent power consumer. More research could be carried out on the digital estimator design itself. Also the CBADC theory for constructing coefficients could be altered so that symmetrical coefficients for the lookback and lookahead batches are created. Then the coefficient register could be shared between the batches, resulting in a size reduction of approximately 50% for this design part. Another possibility is the move from the 28 nm process to a newer and more refined technology node. This has the potential of bringing down the power consumption for the digital design parts. But certainly this introduces new challenges for the analog system design.

An interesting approach, which has been theoretically described, but for which so far no implementations exist, might be the sharing of one digital estimation circuit between multiple input signals and the according analog systems. Even if the digital estimator has a quite high resource usage, when viewed separately, this could lead to an overall well performing analog-to-digital converter design. Due to their conceptual freedom and reconfigurability, variable coefficient digital estimator implementations are well suited for a deployment in such systems. Thereby first feasibility studies and simulations could be created, exploring, if further research is worth the effort.

Although systems with a lower number of analog states turned out to be more resource efficient, this was mainly due to the chosen filter architecture. Currently there is research done, to calibrate the filter implementations and coefficients [13][14], what will likely lead to a more even resource usage for comparable systems with different numbers of analog states. However, this calibration does not have an influence on the general design of the digital estimator implementations, which were presented in this thesis, and therefore cannot be counted as an improvement of the design itself.

Chapter 10

Conclusion

This thesis has shown several possibilities for reducing the resource usage of full ASIC CBADC digital estimation FIR filter implementations. The implemented mechanisms can be deployed in variable and fixed coefficient configurations. The results of the carried out simulations can hopefully be used to decide, which system configurations lead to optimal CBADC implementations.

The goal of reducing the area usage of the digital estimator circuits could be achieved for every tested system. For variable coefficient versions also a power reduction could be achieved for most configurations. For fixed coefficient systems the power consumption went up, when using the reduction mechanisms. So for variable coefficient implementations the utilisation of the reduction mechanisms can be recommended. For fixed coefficient versions, the full size implementations should be favoured.

The comparison between variable and fixed coefficient implementations with the same performance characteristics has shown that the reconfigurability of the coefficients introduces a significant overhead in area and power. If it is possible to use fixed coefficients in a given scenario, such an implementation should be favoured over a variable coefficient one.

The results also show a correlation between the chosen number of analog states for analog systems with the same performance characteristics and the resource usage of the corresponding digital estimator implementations. With a rising number of analog states for equally performing systems, the resource usage for area and power of the digital estimator increases. Systems with a lower number of analog states are therefore preferable, when using the method for calculating the filter, which was used for this thesis. However, there is ongoing research, exploring different methods for the filter calculation, which could lead to a relativation of these results.

The introduced Python framework reduced the effort needed for creating implementations of different configurations of the digital estimation FIR filter, by automating the design steps from cbadc Python package high level simulations over the SystemVerilog file generation up to post P&R simulations. Besides that the framework also enables engineers inexperienced in digital design to create digital estimation FIR filter implementations through the introduced Python abstraction layer.

Hopefully this thesis could contribute to the ongoing research of control-bounded analog-to-digital converters and provide a roadmap for future work in the field of digital estimation filter designs in particular.

Bibliography

[1] H. Malmberg, 'Control-bounded converters,' en, Doctoral Thesis, ETH Zurich, Zurich, 2020, ISBN: 978-3-86628-697-9. DOI: `10.3929/ethz-b-000469192`.

[2] H. Malmberg, G. Wilckens and H.-A. Loeliger, 'Control-bounded analog-to-digital conversion,' *Circuits, Systems, and Signal Processing*, vol. 41, Mar. 2022. DOI: `10.1007/s00034-021-01837-z`.

[3] F. E. Feyling, 'Design considerations for a low-power control-bounded a/d converter,' master thesis, NTNU Trondheim, Oct. 2021.

[4] A. Bjørsvik and S. Mestvedthagen, 'Optimization of cbadc estimation filter algorithms for risc-v implementations,' master thesis, NTNU Trondheim, Jun. 2023.

[5] D. A. B. Mikkelsen, 'A study of control-bounded estimation filter architectures,' master thesis, NTNU Trondheim, Oct. 2022.

[6] D. A. B. Mikkelsen, 'Analyzing estimation filter implementations for control-bounded adcs,' report, NTNU Trondheim, Jan. 2022.

[7] D. A. B. Mikkelsen. 'Python scripts used in specialization project of D. A. B. Mikkelsen.' (2021), [Online]. Available: `https://github.com/GuavTek/control_bounded_filter_models/` (visited on 24/11/2022).

[8] D. A. B. Mikkelsen. 'Hdl code used in specialization project of D. A. B. Mikkelsen.' (2021), [Online]. Available: `https://github.com/GuavTek/Control_bounded_filter_HDL/` (visited on 24/11/2022).

[9] L. Mayrhofer, 'A hdl implementation of the digital estimator for the cbadc framework,' report, NTNU Trondheim, Dec. 2022.

[10] L. Mayrhofer. 'Cbadc digital estimator python framework.' (Dec. 2022), [Online]. Available: `https://github.com/leonman55/CBADC_DigitalEstimator_Framework.git`.

[11] D. H. Malmberg and M. F. E. Feyling. 'Control-bounded a/d conversion toolbox's documentation, Documentation of the cbadc python package.' (Nov. 2022), [Online]. Available: `https://cbadc.readthedocs.io/en/latest/index.html`.

[12] C. Thomas H., L. Charles E., R. Ronald L. and S. Clifford, *Introduction to Algorithms, Third Edition.* The MIT Press, 2009, vol. 3rd ed, ISBN: 9780262033848.

[13] H. Malmberg. 'Calibration tool for control-bounded analog to digital converters.' (Jun. 2023), [Online]. Available: `https://github.com/hammal/calib`.

[14] H. Malmberg. 'Python frontend for control-bounded analog to digital converter calibration tool.' (Jun. 2023), [Online]. Available: `https://github.com/hammal/calib_python`.

Appendix A

Coefficient reconfigurability estimation

In order for the coefficient reduction to be actually useful, it must tolerate certain system and the according coefficient changes. An assessment of the changes in the per LUT sorted coefficients has been done. Figure A.1 shows the resulting sorted LUT data input bitwidths necessary for the coefficients. The variations in the configurations include systems with the same performance (analog bandwidth, SNR), systems with the same configuration except for the OSR and systems with the same configuration except for the lookback/lookahead length. Larger systems of course need a larger coefficient register, more LUTs and more adders, so when compared to smaller systems, only the overlapping region is looked at.

The results show that all systems have a quite similar maximum number of bits needed, as long as the same coefficient bitwidth is set in the configuration. The differences in bitwidth rise the further the digital state vector is away from the estimation timestep k. This means the critical region for coefficients is next to the estimation timestep k. When using configurations with the same performance (analog bandwidth, SNR), the system with the highest number of analog states N and therefore the most coefficients/LUTs is dominant for all regions. When increasing the OSR also the coefficient bitwidths slightly increase. A change in lookback/lookahead length does not have a significant influence on the bitwidths.

Summing up the results, in general a reuse of a reduced system implementation is feasible. The implementation should thereby be laid out for the most resource intense system to be expected. The here shown variations for different changing configurations can be used as a first estimation metric for the maximum resource usage. To increase the probability that the implementation does not limit the coefficient tuning process a headroom for the bitwidths can be included. Two bit should be sufficient to ensure the flexibility of the system

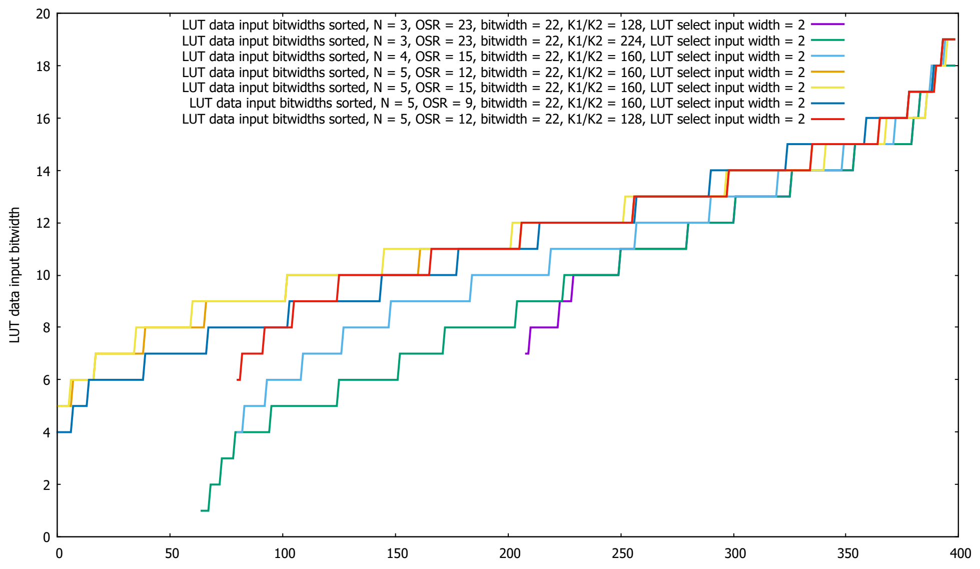

Figure A.1: Comparison of the LUT data input bitwidths for different configurations variations.

Appendix B

Floorplan setup and layout

P&R tools have certain input requirements for being able to calculate a layout. As mentioned in other chapters these are primarily the gate level design, which should be placed and routed and the technology libraries. Another input configuration is the so called floorplan. This basically defines the area the P&R tool is allowed to use for the design layout. There was no strictly defined upper limit for this thesis, although in general designs should not get astronomically large. However, the set floorplan size can influence the performance of the placed and routed design. When the size is too small, of course the P&R tool will not be able to generate a layout. If the size is too large, the P&R tool does not need to pack the gates as densely as it would be possible. In fact this can lead to a widely spread out design. Of course this increases the delay times between gates, due to increased wire lengths, which could negatively impact the design performance. If this happens, the floorplan settings given to the P&R tools need to be adjusted to the individual gate level designs. This can be done by gradually reducing the floorplan size until the P&R tool is not longer able to generate the design layout.

In order to rule out the possibility of the negative influence of too loosely placed designs, two P&R runs have been conducted and the layouts were examined for this behaviour. The floorplan size was therefore set to a size much larger than the approximately needed space for the designs. The two digital estimation filters used are the fixed and variable coefficient versions of a filter with three analog states, a lookback/lookahead width of 128, a LUT select input width of 4 bit and full size coefficients.

Pictures B.1 and B.4 show the entire layout. It is clearly visible that the designs are densely packed together. To get a close look at the actual designs, the view on the layouts was magnified. Picture B.2 shows the fixed coefficient design. A large main cluster and a smaller side cluster are visible. Picture B.3 shows only the man cluster. The magnified view on the variable coefficient design can be seen in picture B.5.

Apart from the small side cluster in the layout for the fixed coefficient version all gates have been closely placed together. But since this side cluster is not too far away from the main cluster, this should not negatively impact the design performance. For the variable coefficient design only one packed cluster exists. This shows that a too largely chosen floorplan should not impact the design performance. Therefore a fixed floorplan size was used for all P&R runs without gradually fitting it to the individual designs.

Pictures B.1 and B.4 also represent a good visualisation of the size difference between fixed and variable coefficient filters with the same performance characteristics.

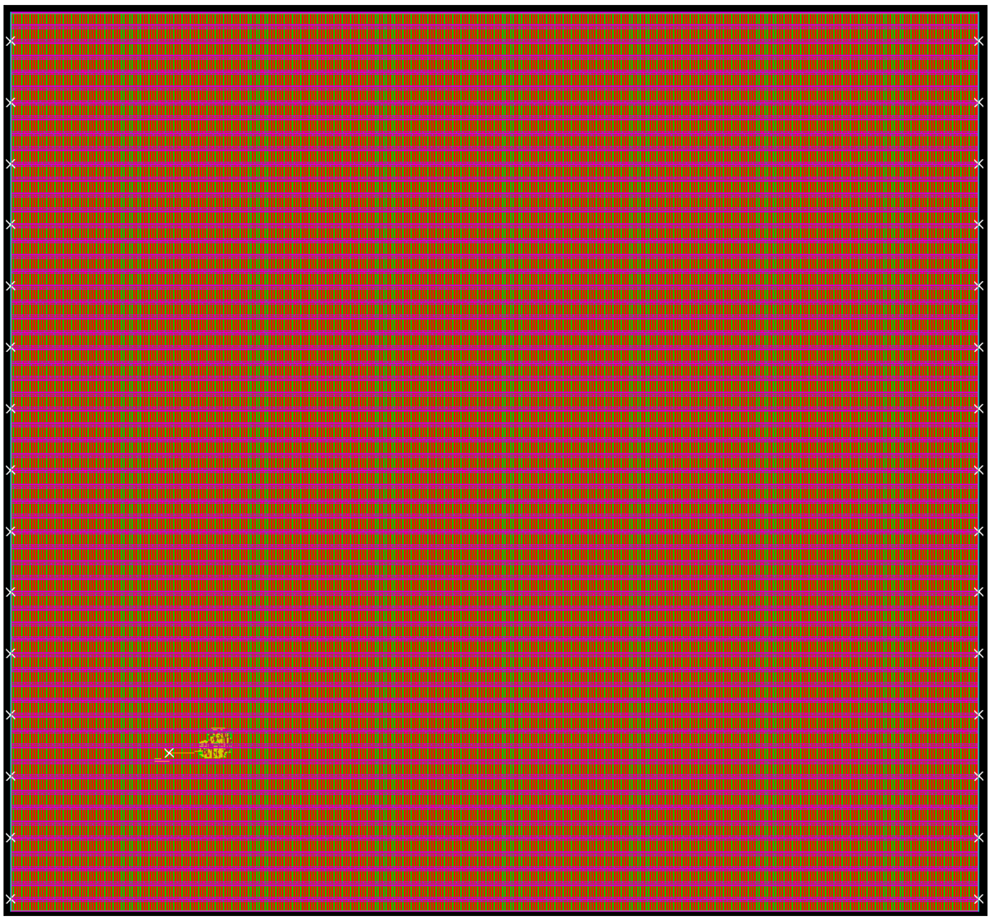
Figure B.1: Layout for filter with fixed coefficients, N = 3, K1/K2 = 128, LUT select input width = 4 bit, full size coefficients.

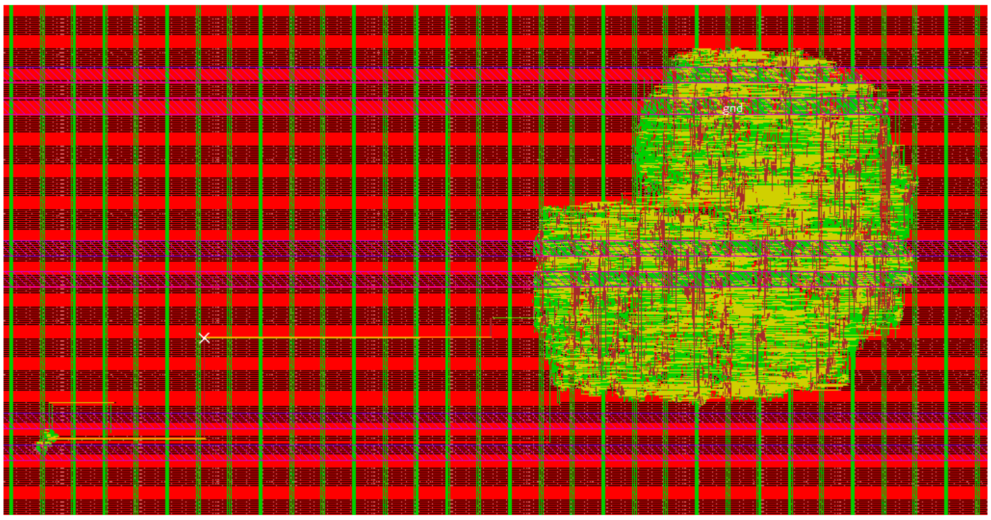

Figure B.2: Zoomed in layout for filter with fixed coefficients, N = 3, K1/K2 = 128, LUT select input width = 4 bit, full size coefficients. Main and side cluster visible.

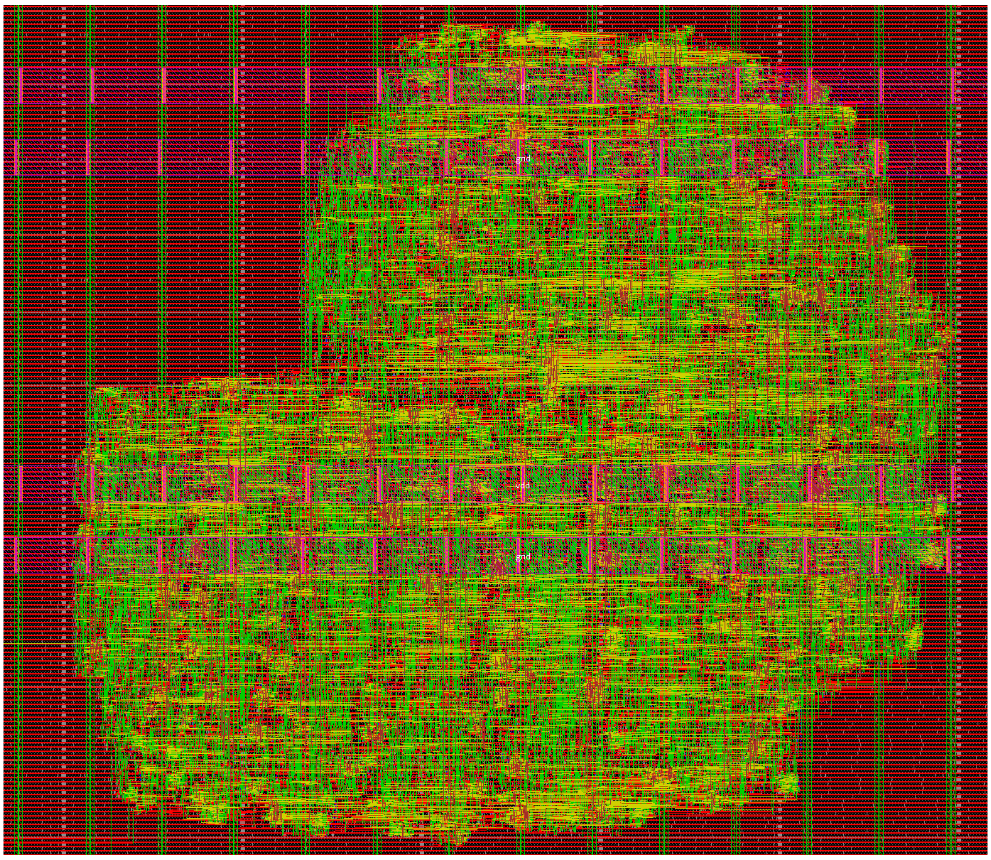

Figure B.3: Zoomed in layout for filter with fixed coefficients, N = 3, K1/K2 = 128, LUT select input width = 4 bit, full size coefficients. only main cluster visible.

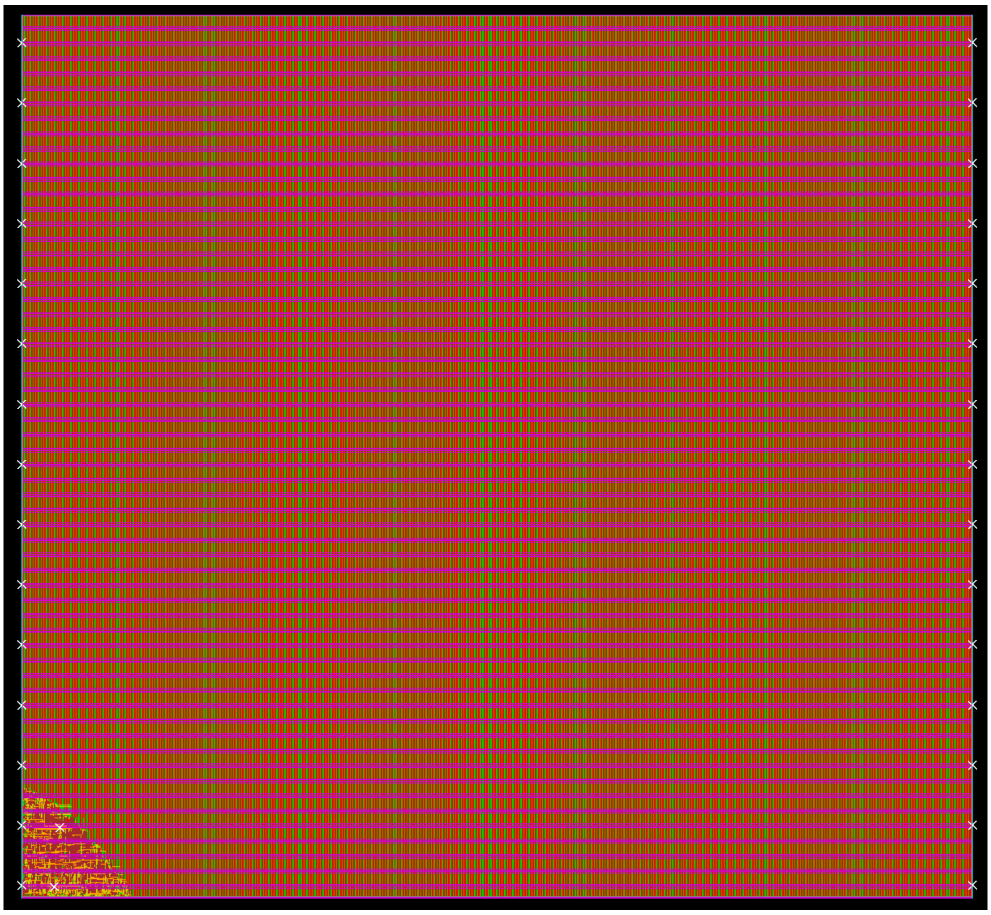

Figure B.4: Layout for filter with variable coefficients, N = 3, K1/K2 = 128, LUT select input width = 4 bit, full size coefficients.

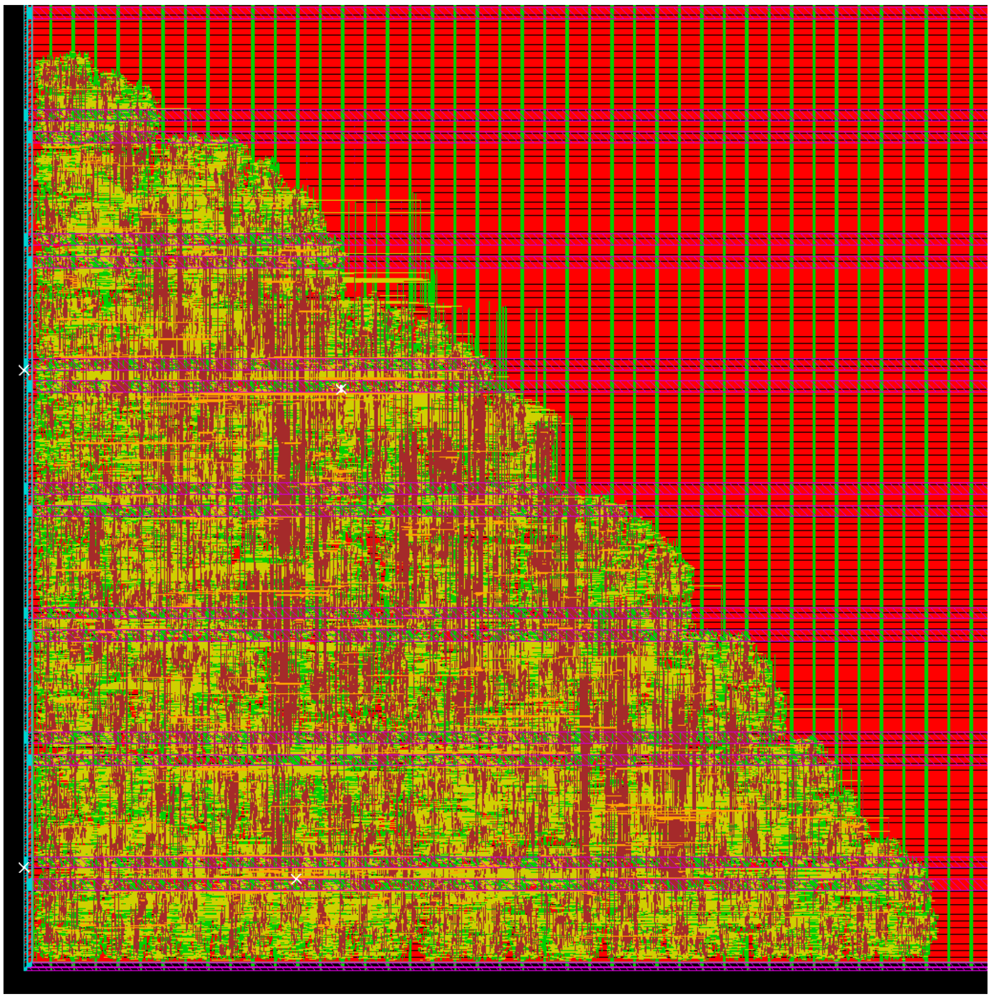

Figure B.5: Zoomed in layout for filter with variable coefficients, N = 3, K1/K2 = 128, LUT select input width = 4 bit, full size coefficients..

Appendix C

Signal estimation and power spectral density (PSD) graph examples

Here some examples of signal estimation and PSD graphs from systems from the test vectors described in 6 are shown. The first example system reaches the expected SNR in gate level an post P&R simulations. The chosen system was configured with variable full size coefficients, 8 analog states and a LUT select input width of 2 bit. Figures C.1 and C.2 show the RTL simulation results (PSD for python high level simulation included as well for reference), figures C.3 and C.4 show the gate level simulation results and figures C.5 and C.6 show the post P&R simulation results. As it can be seen in the graphs, the simulation results are the same for all design stages.

The chosen example system, which did not reach the targeted SNR values in the gate level simulation, was configured with variable reduced size coefficients, 8 analog states and a LUT select input width of 2 bit. Figures C.7 and C.8 show the RTL simulation results (PSD for python high level simulation included as well for reference). Here the SV implementation shows the same results as the cbadc Python package simulation. Figures C.3 and C.4 show the gate level simulation results. The gate level simulation shows clear signs of distortion. The input signal is still recognisable, but the single estimation values have a certain inaccuracy. This behaviour reflects into the PSD graph, which still clearly shows the peak at the input frequency, but the noise floor is significantly increased. In figures C.5 and C.6 the post P&R simulation results are shown. Here the system shows the same results as the cbadc Python package and RTL simulations again. A discussion on this behaviour was made in chapter 8.

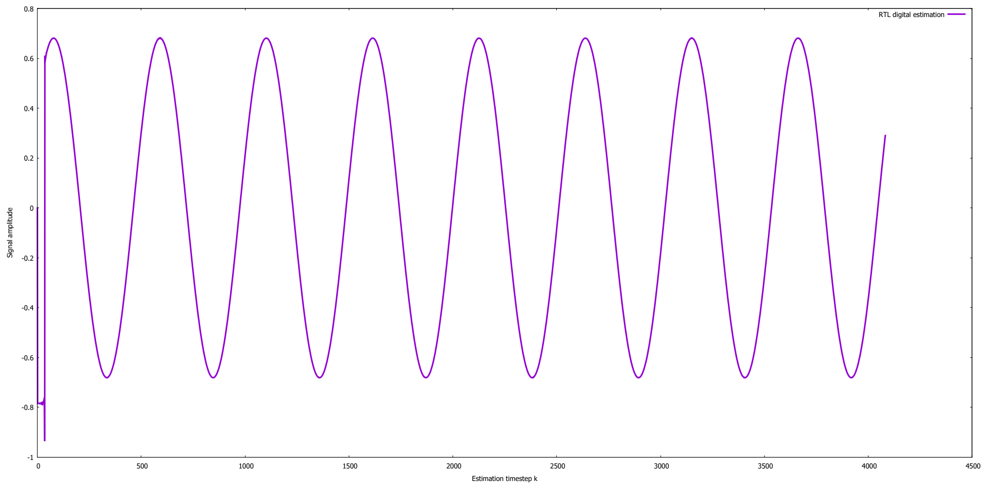

Figure C.1: RTL simulation for the system with variable full size coefficients, N = 8, LUT input width = 2 bit.

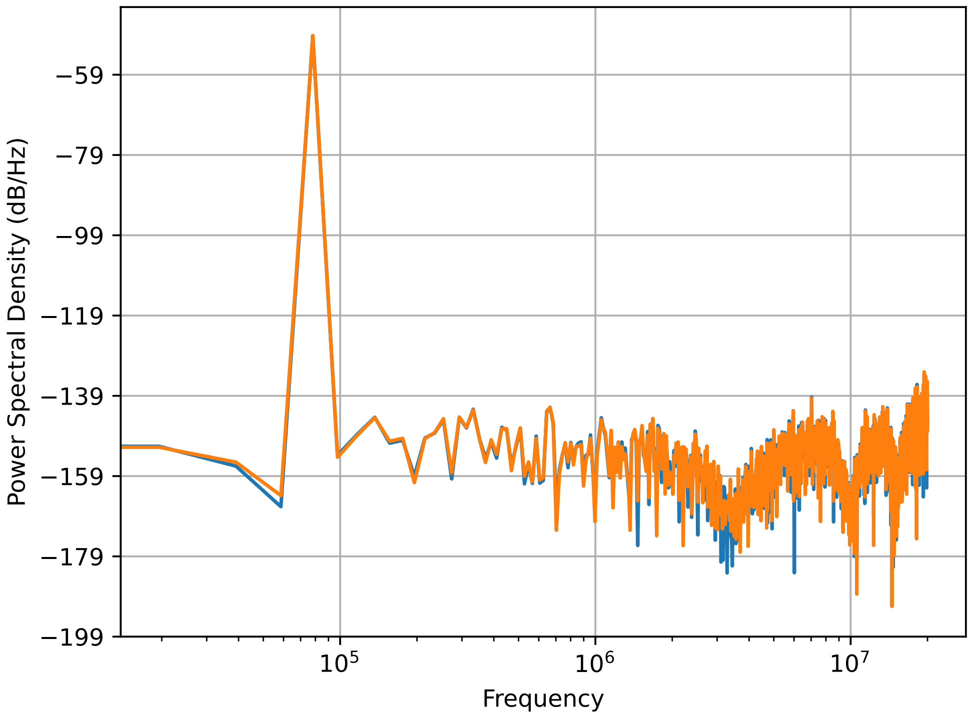

Figure C.2: cbadc Python package simulation PSD graph (orange) and RTL simulation PSD graph (blue) for the system with variable full size coefficients, N = 8, LUT input width = 2 bit.

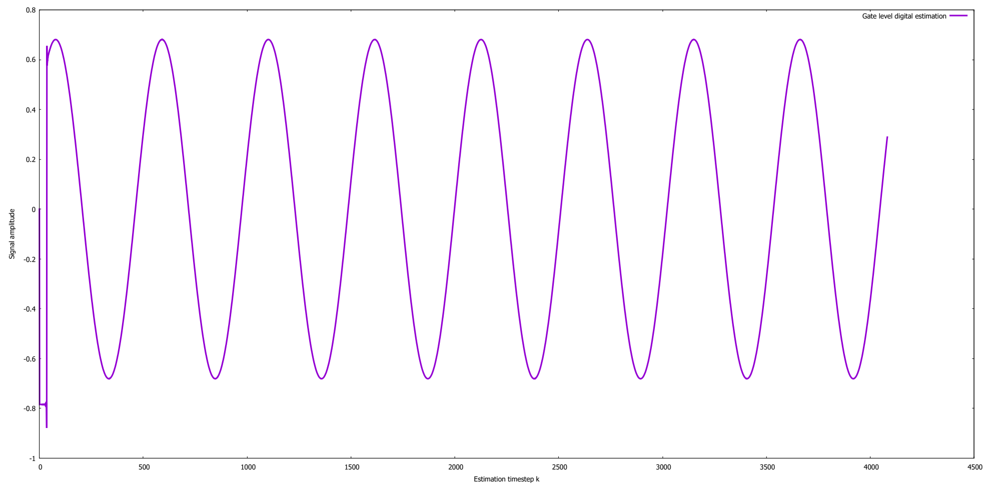

Figure C.3: Gate level simulation for the system with variable full size coefficients, N = 8, LUT input width = 2 bit.

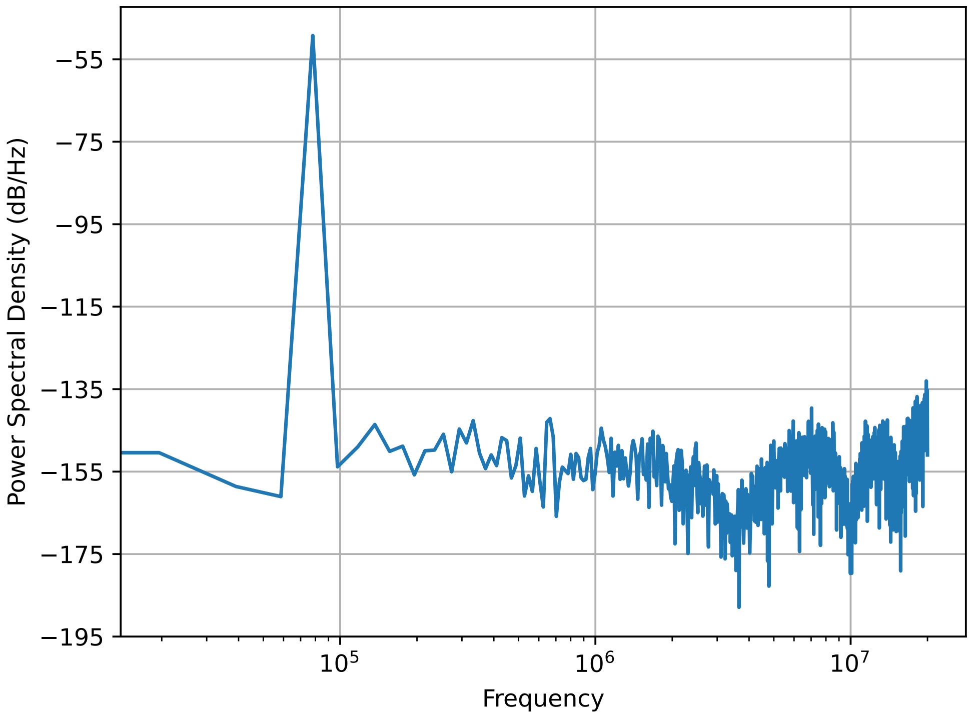

Figure C.4: Gate level simulation PSD graph for the system with variable full size coefficients, N = 8, LUT input width = 2 bit.

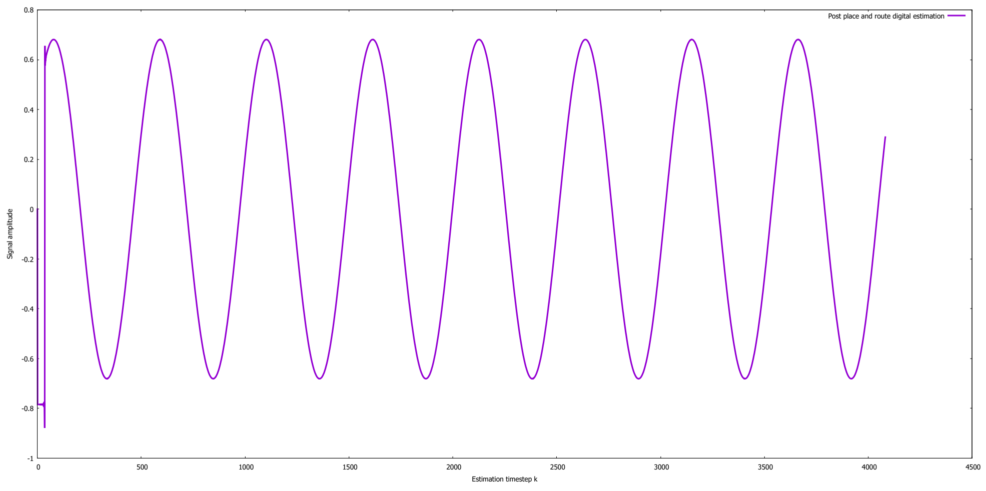

Figure C.5: Post P&R simulation for the system with variable full size coefficients, N = 8, LUT input width = 2 bit.

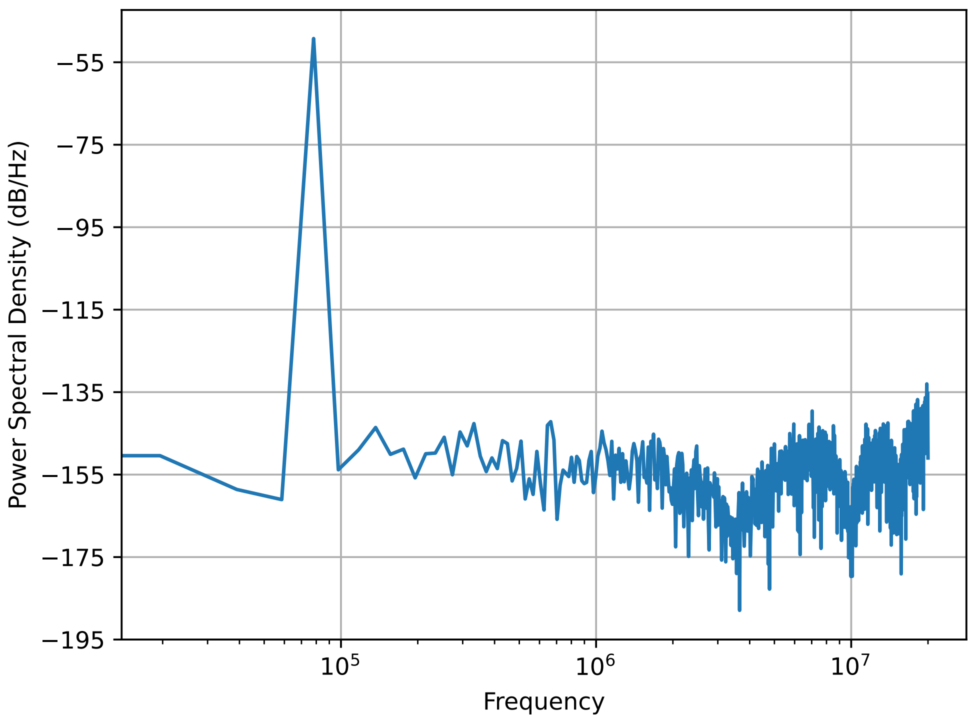

Figure C.6: Post P&R simulation PSD graph for the system with variable full size coefficients, N = 8, LUT input width = 2 bit.

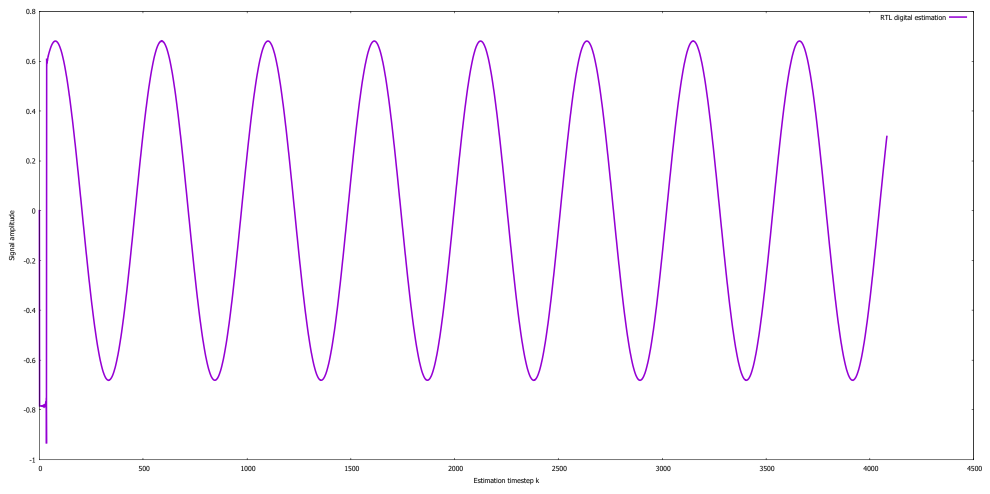

Figure C.7: RTL simulation for the system with variable reduced size coefficients, N = 8, LUT input width = 2 bit.

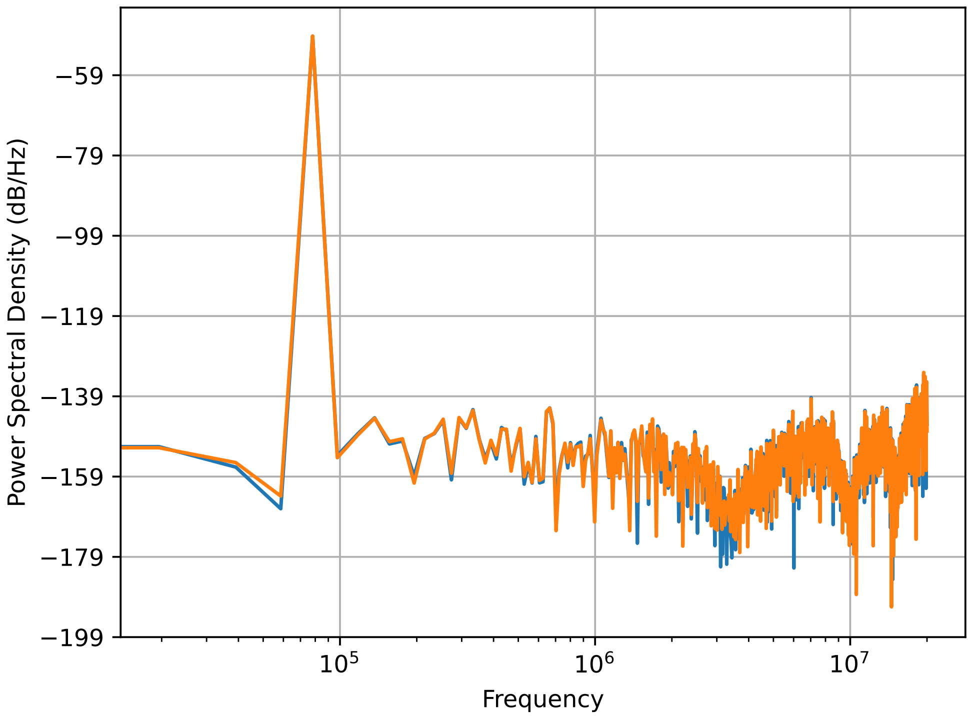

Figure C.8: cbadc Python package simulation PSD graph (orange) and RTL simulation PSD graph (blue) for the system with variable full size coefficients, N = 8, LUT input width = 2 bit.

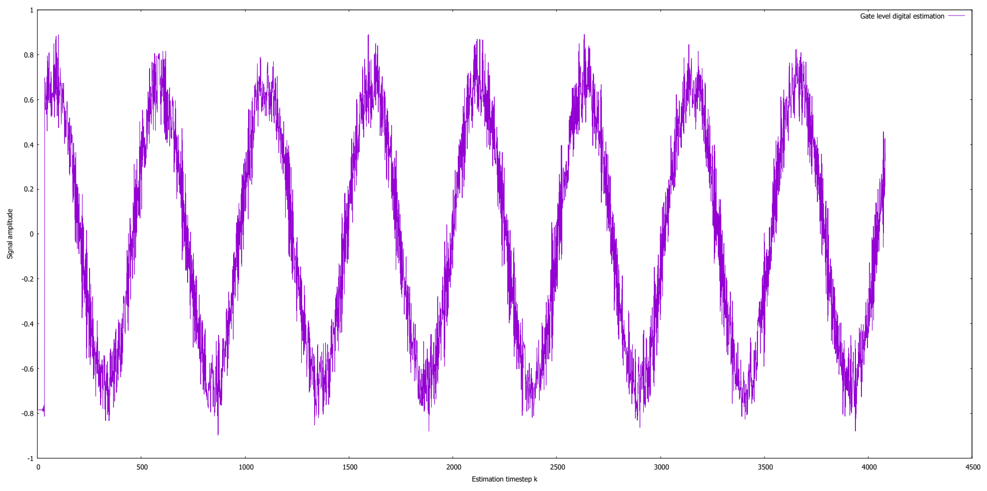

Figure C.9: Gate level simulation for the system with variable reduced size coefficients, N = 8, LUT input width = 2 bit.

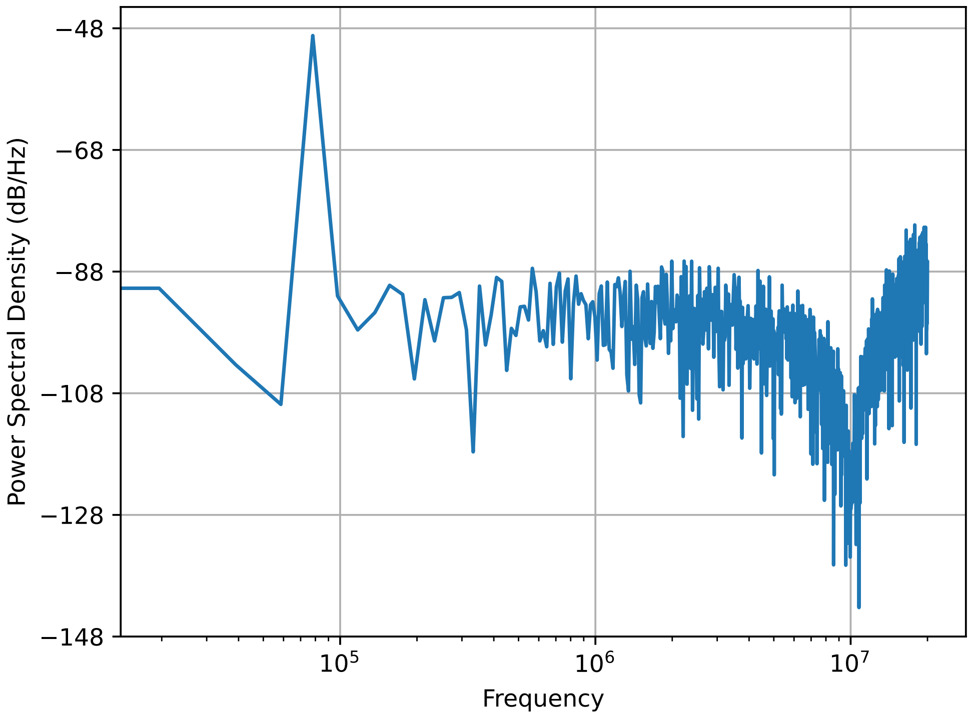

Figure C.10: Gate level simulation PSD graph for the system with variable reduced size coefficients, N = 8, LUT input width = 2 bit.

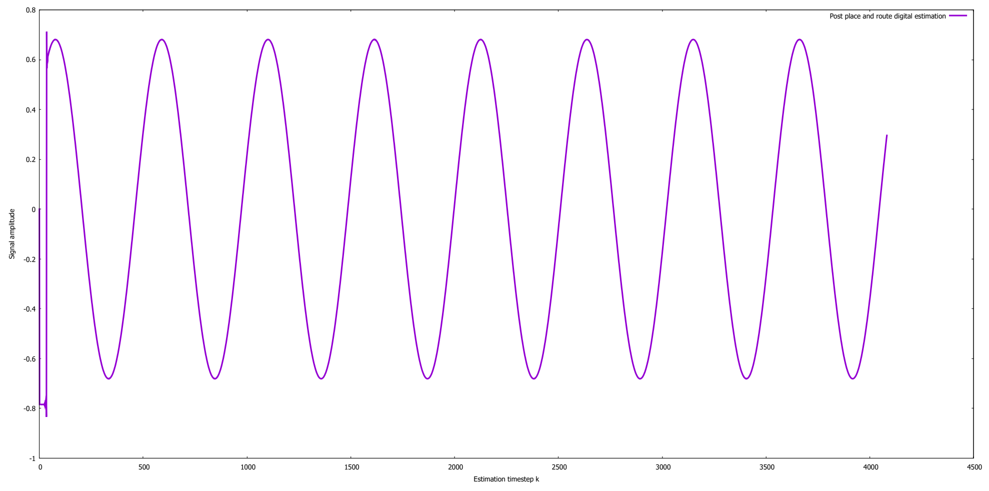

Figure C.11: Post P&R simulation for the system with variable reduced size coefficients, N = 8, LUT input width = 2 bit.

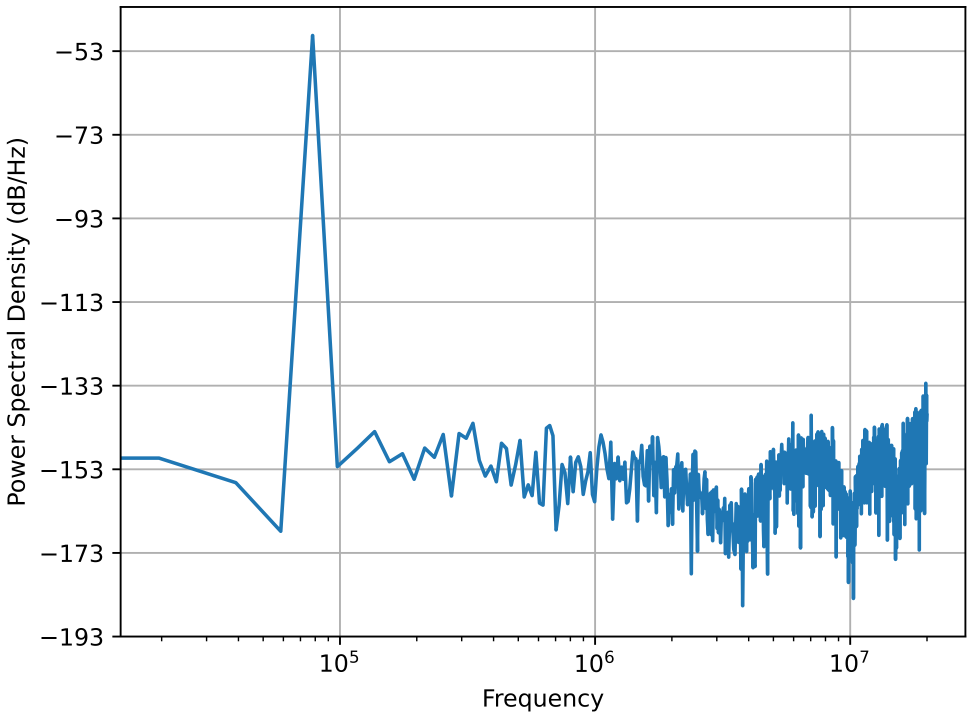

Figure C.12: Post P&R simulation PSD graph for the system with variable reduced size coefficients, N = 8, LUT input width = 2 bit.

Appendix D

Post P&R area and power estimation values

Here the absolute area and PrimeTime power estimation values for the variable coefficient configuration sets 6.7, 6.8, 6.9 and 6.10 and fixed coefficient configuration sets 6.11, 6.12, 6.13 and 6.14 corresponding to the graphs in the post P&R result section 7.3 are given.

D.1 Variable coefficient configurations

	LUT select input width = 4 bit		**LUT select input width = 2 bit**	
Number of analog states N	**Full size Area** [μm^2]	**Reduced Area** [μm^2]	**Full size Area** [μm^2]	**Reduced Area** [μm^2]
3	303480	194850	205148	122466
4	505818	303779	341904	181502
5	632188	384321	427450	233452
6	1062029	569082	718134	337052
7	1239237	693240	837855	420210
8	1618663	852051	1094224	525986

Table D.1: P&R area values for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10, synthesis with Synopsys Design Compiler, P&R with Cadence Innovus.

	LUT select input width = 4 bit		**LUT select input width = 2 bit**	
Number of analog states N	**Full size Power** [mW]	**Reduced Power** [mW]	**Full size Power** [mW]	**Reduced Power** [mW]
3	2,241	2,078	1,609	2,146
4	3,829	3,245	2,877	3,231
5	4,882	4,149	3,783	4,219
6	8,458	6,341	6,867	6,229
7	10,3	7,688	8,317	7,867
8	13,4	X	11,4	9,996

Table D.2: P&R PrimeTime power values for variable coefficient configurations 6.7, 6.8, 6.9 and 6.10, synthesis with Synopsys Design Compiler, P&R with Cadence Innovus.

D.2 Fixed coefficient configurations

	LUT select input width = 4 bit		**LUT select input width = 2 bit**	
Number of analog states N	**Full size Area** $[\mu m^2]$	**Reduced Area** $[\mu m^2]$	**Full size Area** $[\mu m^2]$	**Reduced Area** $[\mu m^2]$
3	31298	28676	44017	37369
4	48839	44091	70840	56194
5	62824	55979	89679	71973
6	97422	83115	138982	105335
7	118074	101333	168838	130619
8	151362	126128	215457	163528

Table D.3: P&R area values for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14, synthesis with Synopsys Design Compiler, P&R with Cadence Innovus.

	LUT select input width = 4 bit		**LUT select input width = 2 bit**	
Number of analog states N	**Full size Power** $[mW]$	**Reduced Power** $[mW]$	**Full size Power** $[mW]$	**Reduced Power** $[mW]$
3	0,29	0,81	0,402	0,77
4	0,514	1,278	0,736	1,257
5	0,721	1,654	1,02	1,659
6	1,277	2,525	1,809	2,627
7	1,642	3,138	2,341	3,319
8	0,934	4,01	1,339	4,327

Table D.4: P&R PrimeTime power values for fixed coefficient configurations 6.11, 6.12, 6.13 and 6.14, synthesis with Synopsys Design Compiler, P&R with Cadence Innovus.